Perlin: Scalable DAG-based distributed ledger protocol using Avalanche consensus.

Kenta Iwasaki
kenta@perlin.net

July 7, 2018

Abstract

Perlin is a directed-acyclic-graph (DAG)-based distributed ledger bootstrapping on top of the Avalanche consensus protocol, a Byzantine fault tolerant protocol built on a metastable mechanism to achieve high throughput and scalability. Natively, Perlin is armed with a decentralized compute layer, supplying it with a high-throughput engine that horizontally scales parallel computations. Perlin's novel consensus mechanism and native general-compute engine makes it the fundamental protocol for powering the global decentralized economy.

Contents

1 Introduction

Having a network of nodes that can reason in a trustless manner and reach a consensus about a set of information efficiently in terms of time and energy is a trillion-dollar problem that plagues both decentralized and distributed systems.

Several solutions that have been conceived to address this space have to do with efficiently utilizing and epidemically spreading less/more information viably across a network of nodes.

1.1 Background

In distributed ledger technology, many directed acyclic graph (DAG)-based solutions such as Hashgraph, Spectre, Avalanche, and Fantom come into consensus by weighing confidence levels based on the dissemination level of a specific set of information.

We may distinguish DAG-based solutions to resemble probabilistic consensus mechanisms that rely on the geometric integrity of how a set of vertices are bound to an edge representing a set of information.

The geometric integrity of the DAG being a black-box due to latencies inherent in the networking of multiple nodes is what defines said solutions to be probabilistic.

An alternative set of solutions which power Bitcoin, Ethereum, Monero, and EOS are comprised of game-theoretic solutions that utilize staking mechanisms, or computational complexity-driven solutions that utilize cryptographic random oracle indifferentiability mechanisms.

These mechanisms presently persist strongly in the decentralized systems ecosystem, and in recent works have been depicted as variants of the Nakamoto consensus.

Lastly, we have a set of solutions that have existed long before Nakamoto and probabilistic consensus mechanisms such as Paxos and Raft whose consensus mechanisms rely on the notion of electing a leader over a decentralized network.

Compared to these distinct consensus mechanisms, the ones time-tested and best suited for placing a consensus mechanism over a decentralized system are probabilistic with confidence thresholds that are practical and acceptable by its users.

1.2 Adoption

Proof-of-work (PoW) for one is widely adopted due to the fact that its consensus confidence threshold is determined by the amount of computing power designated within the network.

Given that the only way to maliciously attack the process in which consensus is reached is to cross the confidence threshold of 51% by dedicating more than 51% of a networks total computing resources worth of computing power into attacking the network, proof-of-work is widely accepted.

However, the computational complexity which secures the notion of proof-of-work remains a burden due to the amount of processing power, or electricity, that must be wasted for the sake of empowering such a consensus mechanism in a decentralized system.

Proof-of-Stake (PoS) works in a similar manner that is more energy-efficient, but in the process opens doors to new problems such as denial-of-service (DoS) attacks, eventual centralization, and a multitude of other issues.

These issues persist from economically transacting around virtual currency across a group of leader nodes with hopes that leader selection and governance is to be corrected and brought from a given currency's value.

The frequency to which virtual currency fluctuates as both a store of value and utility token however is what makes proof-of-stake encounter the cold-start problem in which governance as a result of initiating fluctuating value faces highly impractical uncertainty.

Interestingly enough, should a currency's value transition into a phase in which its value is relatively stable, it is economical and viable to replace its existing consensus mechanisms with a proof-of-stake mechanism.

This transition into greener consensus mechanisms can be exemplified by how Ethereum is slowly transitioning from a proof-of-work blockchain into a proof-of-stake network.

1.3 Metastability

Metastability is a principal in which consensus is reached as a result of passing over confidence thresholds derived from how information is disseminated, structured and verified across a network.

Metastability originates from quorum consensus mechanisms typically seen in high-availability databases such as Apache Cassandra and Riak KV, bootstrapped with additional structural information that assists a system in rapidly coming to a consensus.

To effectively propagate information across a network, a single node propagates information x to a quorum of k nodes out of a network of n nodes, which thereafter recursively propagates x to another quorum of k nodes.

The goal of metastability is to slowly orient batches of quorums within a network towards accomplishing a specified consensus goal of verifying and accepting/rejecting a set of information x.

In the use case of a leaderless decentralized system, having conflicting opinions about accepting and verifying information x is subjective to how one desires the magnitude of the protocol's confidence thresholds to be requiring the acceptance of $\frac{2}{3}k$-majority nodes for every single one of the entire network's $\frac{n}{k}$ quorums.

Hence, a significant difference to traditional gossip-based consensus protocols is that metastability spreads information through quorums rather than through singular nodes, persists structural information appended to x that accelerates consensus, and thus holds overall lower communication costs.

These costs engender traditional consensus protocols that have a communication cost of $O(n^2)$ with metastable consensus protocols ranging from $O(kn \log n)$ to $O(kn)$ based on confidence thresholds derived from security parameters k and n.

Given that parameters k and n do not bear any significant implications or restrictions on the geometric integrity of the network, it is practical to build a secure decentralized system by utilizing a metastable consensus protocol.

1.4 Avalanche

Avalanche is a leaderless Byzantine fault-tolerant metastable distributed ledger protocol that holds high magnitudes of confidence thresholds despite the potential presence of Byzantine adversaries.

For all information x_t that is disseminated throughout nodes under the Avalanche consensus, Avalanche additionally appends information regarding conflicts of information ancestry x_{t-1} under the geometric integrity and structure of a DAG in which t denotes indicies of ancestral order.

Forming a DAG of conflicting information in which multiple edges $\{x_t\}$ contain a vertex to a specific edge x_{t-1} is what allows for one to reason about consensus as a result of geometric integrity.

Information that remains within the system which is t-generations old within the DAG may be considered to have an extremely high confidence score, and thus is ensured to have been confidently accepted by the network.

Thus, this allows us to reason that all information that is considered to have come under consensus from the perspective of a single node in the network must have been viewed and verified by a majority of all the nodes in the network.

Hence, truly conflicting information spread by Byzantine adversaries that is not accepted by the majority of $\frac{n}{k}$ quorums would be quickly rejected by the network.

As for the case of conflicting information from honest adversaries, a reorganization of information ancestry for information x may be carried out in which a node that receives x may swap the

ancestors of x with the most viable set of ancestors.

This re-selection of ancestors is what makes the structure of the DAG accurately converge to the geometric structure of a longest-chain blockchain depicted in Nakamoto consensus, and thus additionally maximizes the likelihood that virtuous and honest information will be rapidly accepted by the network.

With the scheme denoted above, benchmarks have shown that Avalanche is able to cater for the consensus of 1626 pieces of information per second with $n = 2000$ nodes in the system.

With simulated geo-replication across more than 20 cities, consensus is facilitated for 1312 pieces of information per second for $n = 2000$ nodes with a median consensus latency of 4.2 to 5.8 seconds.

In spite of Avalanche's benchmarked high-throughput and low consensus latencies, Avalanche was originally modeled as a distributed ledger dedicated to facilitating financial transactions.

Hence, all information x would solely denote monetary transactions between two parties in a decentralized fashion.

To share the strengths of Avalanche's high-throughput and performance as a consensus mechanism with the world, we extrapolate and bootstrap Avalanche with a wide array of practical mechanisms that enable developers to utilize Avalanche as a truly decentralized and general public ledger that can empower the world's technical infrastructure.

2 Perlin

Assessing from how distributed ledgers have scaled throughout the years, Perlin introduces a different mindset as to how distributed ledgers may be utilized for the development of decentralized applications and technical infrastructure.

For example, Perlin was intentionally designed to not support Turing-complete smart contracts. Rather, should a developer wish to develop a decentralized application on Perlin, developers would instead utilize Perlin as a ledger that bypasses middlemen and brings finality and consensus into off-chain high-throughput computing systems.

Hence, rather than having Turing-complete smart contracts, Perlin provides a platform in which developers can both write and run smart contracts compiled as WebAssembly bytecode that only exist for the sake of bringing finality and consensus into an off-chain system.

More specifically, smart contracts work as either *time*, *resource*, or *signature* lock verification mechanisms redundantly computed across all nodes in Perlin's network which reward, deplete, or release funds that are in the form of mintable tokens called **perls**.

2.1 Smart Contracts

The first mechanism that Perlin bootstraps on top of Avalanche consensus is the ability to create, test, destroy, and deploy smart contracts representing time, resource, or signature locks that run as WebAssembly byte code.

All forms of locks are intended to be verifiable in a timely manner by being limited to being represented under Q instructions, as locks are redundantly computed on every node in Perlin's network. An inability to meet this aforementioned criterion results in the lock from being rejected within the system.

Time. Developers may create smart contracts that are able to release, manipulate, or reward *perls* or introduce finality to business logic for a decentralized system if the user provides a time-lock proof for a given smart contract. Time-lock proofs which have originated from a now prominent paper by Ronald Rivest, Adi Shamir, and David A. Wagner in 1996 come in the form of cryptographic

hash functions that are computationally expensive to execute, akin to several Proof-of-Work systems exhibited today.

Time-lock proofs contain a plethora of use cases, such as mitigating spam attacks (Hashcash), Sybil attacks (S/Kademlia), and especially DoS attacks for decentralized systems. Time-lock proofs may additionally be utilized to create the notion of bonds and other forms of financial contracts on top of *perls* and other virtual currencies across multiple blockchains and systems.

Resource. Developers may create smart contracts that would release, manipulate, or reward *perls* or introduce finality to business logic for a decentralized system should a user provide a resource-lock proof to said smart contract. Resource-lock proofs have recently been formalized in 2017 by Alex Biryukov and Léo Perrin; in which they provide methodology to allow for one to develop both symmetric and asymmetric cryptosystems that exemplify Proof-of-Work systems which are either time-hard, memory-hard, or code size-hard.

Resource-lock proofs may be utilized to allow for one to produce resource attestations of one's computing device, or otherwise trustlessly prove the availability and remote allocation of one's computing resources.

Signature. Developers may create smart contracts that would release, manipulate, or reward *perls* or introduce finality to business logic for a decentralized system should a single user or an entire group of users provide their cryptographic signature of a given message to said smart contract.

Signature-lock proofs may be utilized to describe multi-party financial mechanisms such as multi-signature wallets, and additionally may be used as a form of identification in a decentralized system.

2.2 Transactions

The ledger state of Perlin is modeled as a key-value store structured as a Merkle Patricia trie the contents of which are completely verifiable.

One can consider the ledger state to be represented as a history log, which is at heart a sequence of operations applied on top of a genesis state seeded by the very first transaction which emerges in Perlin's network.

Transactions in this case model the operations that emplace and modify Perlin's distributed ledger state in an entirely asynchronous yet verifiable manner.

To have ledger state become eventually-consistent with nodes in Perlin's network over time, the second mechanism that Perlin bootstraps on top of Avalanche consensus is on how disseminated transactions are modeled as verifiable operational transformations.

2.3 Node Identities

Identities. Users individually identify themselves on Perlin through asymmetric key-pairs represented as points on the elliptic curve signature scheme Ed25519. Hence, users derive their public key to be $K_{pub} = K_{priv} \times G$ where K_{priv} is an user's private key, and G is the generator of the Ed25519 curve.

Signatures. Users are required to sign messages through the Schnorr signature scheme where the signature $S(m, s, e) = H(m|s \times G + e \times K_{pub}) = H(m|r) = e$ of message m is represented through the signature components s and $r = s \times G + e \times K_{pub}$ given the Blake2b hash function $H(\cdot)$.

As a third mechanism which Perlin bootstraps on top of Avalanche consensus, Perlin enforces key-pairs used to formulate an user's identity on Perlin to be uniquely generated as a result of a Proof-of-Work puzzle.

By requiring work to be done in order to register oneself onto the network, spamming Perlin's network with a wide slew of newly minted Byzantine adversaries is computationally expensive.

Additionally, these enforced constraints on the generation of node identities makes ploys in which one chooses to enforce malicious nodes around a honest node impractical.
This addresses protections against both Sybil and Eclipse attacks in terms of routing messages throughout Perlin's network.

The **Proof-of-Work puzzle**, which is drawn from S/Kademlia is laid out as follows:

1. Generate key-pair consisting of K_{priv} and K_{pub}.
2. Compute $P = H(H(K_{pub}))$.
3. If P does not have c_1 prefixed zero-bits, go back to step 1. Else, utilize $H(K_{pub})$ derived from the generated key pair as the given node's ID M.
4. Generate a random X.
5. Compute $P = H(M \oplus X)$ where $\oplus$ is the exclusive-OR operator.
6. If P does not have c_2 prefixed zero-bits, go back to step 4. Else, the puzzle is solved with users being identified by K_{pub}, M, and X.

The time complexity of the Proof-of-Work puzzle in this case is $O(2^{c_1} + 2^{c_2})$.

The factor c_1 should be kept constant throughout the system, with factor c_2 dynamically changing with respect to how much cheaper computational resources become in the future.

For Perlin's implementation, we set both c_1 and c_2 based on tests to a value of 3 given that it takes 548ms to 2 seconds to execute said proof-of-work puzzle on average on an Amazon EC2 t2.medium instance.

2.4 Voting

Factor c_2 is dynamically voted on and set throughout the lifetime of Perlin by taking advantage of Avalanche's metastable property to learn how nodes within the network come to terms with accepting increases or decreases of the factor c_2.

This decentralized voting procedure for system parameters is implemented in code as a constraint in which nodes may propose changes towards based on their controller's opinions as to what certain system parameters values should truly be.

For example, the opinion of a node on the value of c_2 may be that the amount of instructions necessary to execute the computational puzzle should provide a time complexity which all nodes agree upon.

Such a node in that case would propose this change to be voted on by other nodes within Perlin's system.

This same mechanism may be used to tune other factors with eventual consensus among all nodes such as mining rewards, transaction validation fees, and several other system parameters.

2.5 Scalability

Through the introduction of the three bootstrapped mechanisms that are applied to Avalanche as a consensus mechanism, Perlin is able to scale in order to handle a decentralized system containing hundreds of thousands of nodes.

Although ledger state and history will grow linearly in terms of storage space over time, the

limitations emplaced on smart contracts sufficiently restrict the data which does eventually become stored on-chain.

Through the metastability principle which Avalanche follows, it is easy to eventually prune data that has unanimous consensus across the entire network.

It is additionally easier to asynchronously accept and prevent blocking operations from occurring in synchronicity while attempting to have a set of transactions come into consensus.

Through the introduction of hardened node identities through proof-of-work puzzles, it is difficult to carry out Sybil/Eclipse attacks given the computational complexities necessary to generate new identities.

Through modeling transactions as operational transforms, it is also possible to simply manage, replay, and rollback transactions that are applied onto Perlin's state.

This in turn allows Perlin to be a ledger that:

1. Requires minimal disk storage.
2. Achieves high-throughput rates in terms of consensus.
3. Attains a hardened security model in the presence of Byzantine adversaries.
4. Allows for truly decentralized governance at scale through the metastability principle.
5. Intentionally allows for truly decentralized applications that can replace and dis-intermediate traditional technical infrastructure.

3 Applications

Given the powerful nature of being able to bootstrap decentralized, trustless middlemen in the form of locks or smart contracts on Perlin, a wide array of use cases can be conceived.

In particular, we focus on one specific use case that strongly resonates with the hard limits which developers face in developing decentralized applications with current state-of-the-art distributed ledgers and blockchains.

Those particular limits being that it is impractical to write and bootstrap decentralized applications that require large amounts of random access memory, CPU time, and disk space.

In addition, this particular use case which we focus on also addresses one of the most salient problems associated with the concept of proof of work: wasteful electricity consumption.

3.1 Background

Moore's Law predicts that productivity of computing power doubles within an 18 month period. Unfortunately, this may not be fast enough to satisfy current and future demand. Leading research sectors such as artificial intelligence and drug discovery are doubling their demand for compute power within time periods as short as 3.5 months. This implies an upward trend in price for computing resources, and consequently electricity. That trend is aggravated by cloud computing oligopolies, which makes compute power even more of an economically scarce resource.

Without lower cost access to large amounts of compute power, key players in several industries and research sectors will not have access to resources required to advance the state-of-the-art technology. As a result, there may be delays in access to cures for widespread illnesses such as cancer, slowing of productivity improvements from artificial intelligence, and hampering of efforts to predict and adapt to climate change.

Moreover, advancements in research and technology would be confined to those wealthy enough to afford it. As a result, research would become concentrated in the hands of very few corporations and institutions globally.

Our first application of Perlin dis-intermediates traditional centralized computing oligopolies and replaces them with an openly accessible peer-to-peer network consisting of smartly aggregated underutilized computing resources that can be used to perform massively parallelized compute work. These resources are owned by all of us in the form of laptops and smart phones which are used only a fraction of each day.

We therefore make these resources available to those in need of computing power while protecting the privacy of both device owners and compute users.

By framing a trustless and liquid market around these resources with an economic incentive, *perls*, to buy and sell decentralized compute power, we increase the potential for allowing those in need to have access to egalitarian, trustless, and economic supercomputing at a feasible scale.

In short, our first application democratizes access to large amounts of compute power for startups, small enterprises, researchers, independent developers, students, and advocates working with sensitive data to hasten the advancement of technology.

3.2 The Network

Moving forward, any mention of Perlin refers to our very first off-chain application we are seeking to release to the general public.

Perlin is comprised of three categories of parties: *validators*, *miners*, and *customers*. Any given party may choose to belong to more than one category at any point in time.

Miners. Miners are suppliers of compute power within the network, providing customers compute power for the execution of a wide variety of computational tasks. Miners supply compute power by spawning a virtual machine instance with a specified set of resources a miner wishes to reserve for potential customers. Once a virtual machine instance is created, miners broadcast their compute supply on Perlin's market in terms of their reserved RAM capacity and CPU clock frequency. Should any computational task be open on the market requiring resource specifications equivalent to or less than a miner's provisioned resources, miners may choose to register to work on said computational task. Miners are paid out based on both how long they rent out their compute supply and how powerful their compute supply is.

Customers. Customers register computational tasks onto the network with a fixed pool of *perls* to be paid out to miners. Customers specify both the bare minimum and cumulative total set of resources required from miners looking to work on a customers submitted tasks in terms of both RAM capacity and CPU clock frequency. Customers access a miner's virtual machine instance through SSH, and are responsible for coordinating, supervising, and executing computational tasks on the virtual machine instance of a miner. Through the use of the *Perlin SDK*, customers can easily program decentralized applications and computational tasks powered by rented miner virtual machine instances using a wide number of programming languages.

Validators. Validators come to consensus and log transactions made between miners and customers through a distributed ledger. Validators are elected through a rotating quorum mechanism in which candidates from a pool of candidate validators are probabilistically more likely to be elected should they stake greater amounts of *perls* than other candidate validator. Validators benefit from taking a commission in the form of a transaction fee from every single transaction they verify and broadcast from miners or customers. Transaction fees for each transaction are determined by the customer sending the transaction, and hence are incentives for a validator to quickly validate and broadcast the transaction across the network. Elections for choosing validators out of a candidate pool of validators occur at a timely basis roughly equivalent to 1 minute.

3.3 Proof of Compute Availability

To establish a trustless computing market, a supplier must be able to prove they truly own a batch of available computational resources before delivering and selling them to another party.

3.3.1 Decentralized Clocks

Perlin's economic incentives between miners and customers heavily relies on a decentralized source of time which everyone agrees on. Miners are paid out based on the period of time they rent out their compute supply; customers pay miners based on the period of time they rent a number of miner instances.

Perlin makes use of a decentralized clock on miner virtual machine instances, a clock whose ticks are derived from the computational complexity of an asymmetric verifiable memory-hard cryptographic function termed **diodon**.

Asymmetric. Diodon's asymmetry between a prover P and verifier V is established through operating in a ring under the RSA cryptosystem. We define a semi-prime RSA modulus $N = pq$, where p and q are prime numbers only known by verifier V. Given that prover P does not know the prime number factorization of modulus N, should a prover wish to compute a checksum y which is a message x after M repeated squarings of magnitude t in ring modulus N, they would have to perform M squarings requiring $\Theta(M)$-time. Should verifier V wish to compute the checksum y, they may exploit their knowledge of p and q and calculate $y = x^e \bmod N$ in $O(1)$-time where $e = 2^{(M-1)t} \bmod (p-1)(q-1)$.

Resource-hard. Diodon's memory-hardness comes from the assumption that uniformly random accesses to segments of random access memory is computationally expensive amongst a wide range of compute devices, and that a prover P is unable to compute intermediate squarings of a message x of magnitude t in ring modulus N in O(1)-time without knowledge of the prime number factorization of semi-prime modulus N.

$$V[0] = x \quad V[i] = V[i-1]^{2^t} \bmod N \tag{1}$$

Figure 1: $V[i]$ is the i'th intermediate squaring which prover P is unable to compute in O(1)-time without knowledge of p and q.

Uniformly random access to segments of random memory is achieved through the use of an one-way trapdoor function such as BLAKE-2b denoted by $H(\cdot)$.

Given the computational complexity for prover P to compute M repeated squarings, diodon enforces uniformly random memory access by returning the L'th execution of the recurrence relation $S = H(S, V[S \bmod M])$ with base case $S = V[M-1]$ given that the zero-indexed array storing intermediate squarings V is of length M.

So long as $L \geq M$, diodon would uniformly access the array V randomly L times given the unique output pre-image predicated by $H(\cdot)$ over ring modulus N and M.

In Perlin's context, the prover P would be miners proving a reserved allocation of memory and clock frequency to verifier V which are customers.

Verifier V will generate primes p and q and send to prover P and Perlin's distributed ledger (a) the semi-prime modulus N, (b) the minimum resource requirements for prover P's allocated computational task (denoted by diodon's parameters L, M, and t), and (c) a stream of random bytes which prover P will compute diodon over to generate a proof of compute availability.

Prover P earns a payout from verifier V by submitting the proof of compute availability represented by the outputs of diodon to Perlin's distributed ledger as frequently as possible.

Time-lock. Given the computational complexity of diodon, both miners and customers may determine the CPU clock frequency of a miners virtual machine instance based on the *frequency* in which a miner is able to compute diodon with respect to the compute-hard parameter L.

Memory-lock. Given the memory-hardness of diodon, both miners and customers may determine the RAM capacity of a miners virtual machine instance due to the deterministically memory-locked nature of diodon where the minimum amount of memory required to store the intermediate squarings of array V computed in diodon is $M \times \frac{t}{8}$ bytes.

An additional paper will be provided in due time to address a wide plethora of attacks that have been taken into account for the proposed interactive protocol above through the incorporation a customer/miner reputation system.

Such attacks include: proof-proxying attacks, miner/customer sybil attacks, reputation padding attacks, minting attacks, and graffiti attacks.

3.4 Decentralized Computing

Perlin's strength in distributed computing is derived from sourcing computational power out of large batches of untapped commodity devices. To fully utilize this untapped computing supply, Perlin takes a different approach from numerous distributed computing projects by harnessing computing power in the form of virtual machine instances.

With close assessment of present-day cloud computing market prices, Perlin tackles a number of fallacies associated with distributed computing such as network latencies, Byzantine nodes, and split-brains by significantly lowering the costs associated with renting singular miner instances such that customers may viably rent large amounts of virtual machine instances.

In addition, by disincentivizing unwarranted tampering of miner virtual machine instances rented to customers, Perlin enables a a plethora of decentralized computational tasks with privacy, security, and performance being factors that are tunable with complete flexibility.

3.4.1 Privacy

Perlin provides SDKs with numerous cryptographic primitives dedicated to securing the privacy and security of both computation and data which is ran or broadcasted to miner virtual machine instances.

Should computation be mandated to be both verifiable and correct for a particular customer, a customer must trade off higher financial costs in renting more miner virtual machine instances due to latencies warranted in utilizing privacy-preserving verifiable computing techniques.

In the case of protecting the sovereignty of data distributed to miner virtual machine instances, an abundance of partially and fully homomorphic cryptography schemes with a number of privacy-performance trade-offs are available for use from Perlin such as a number of optimized variants of the Fan-Vercauteren ring-learning-with-errors scheme.

3.4.1.1 Verifiable Computation

Two properties of computation distributed among miner virtual machine instances are typically desired by customers: *verifiability*, and *correctness*. We denote a miners virtual machine instance to be the prover P, and a customer to be the verifier V.

Definition 3.1. A computation $p(x)$ for a set of inputs x is *verifiable* if the order in which p is executed on a potentially-insecure enclave hosted by prover P may be transcripted into a proof that is passable to verifier V.

Definition 3.2. A computation $p(x)$ for a set of inputs x is *correct* if the result $y = p(x)$ computed on prover P is correct after being passed to verifier V.

Perlin provides a number of verifiable computation protocols for ensuring verifiability and correctness of a computation on top of a number of sandboxed virtual machines such as WebAssembly.

Protocol 1 (Pair-wise Delegation). Verifier V sends specifications of computation p to two provers P_1 and P_2, and has both provers execute $y = p(x)$ for some inputs x.

For every intermediate step of p, both provers additionally computes $H(i, s)$ where $H(\cdot)$ is a collision-resistant hash function, i is the intermediary instruction being executed and s is the intermediate state of the virtual machine to establish a Merkle tree M that is sent to verifier V.

Should both provers differ on solution y or proof M, given that one prover is honest, a binary search in intermediate states of computation for both provers P_1 and P_2 is performed until an invalid intermediate state is discovered and corrected by verifier V.

The Byzantine prover to be determined by verifier V is thereafter blacklisted from being associated with verifier V's computational task p.

Filtering for Byzantine provers by binary-searching through and amending incorrect intermediary virtual machine states comes at amortized worst case $O(N^2)$, and average case $O(N \log(N))$ where N is the number of instructions of computation p.

Despite this protocol requiring redundancy and moderately high computational complexity, y is guaranteed to be computed correctly in a verifiable manner.

3.4.1.2 Data Privacy

Partially and fully homomorphic cryptography schemes allow for the application of arithmetic operations on ciphertext without knowledge of the originating text.

Perlin provides a number of homomorphic cryptosystems, with the most efficient being variants that have their homomorphisms and security derived from the ring-learning-with-errors (Ring-LWE) dilemma. We denote below the process behind the majority of homomorphic cryptography schemes provided under Perlin through the Ring-LWE construct.

Pre-processing. We project plain-text x into a set of polynomials X of degree less than n with coefficients modulo t into cyclotomic ring representing our plain-text space $\mathcal{R}_t = \mathbb{Z}_t[X]/(x^n+1)$. Afterwards, we define a cyclotomic ring representing our cipher-text space $\mathcal{R}_q = \mathbb{Z}_q[X]/(x^n+1)$ which is isomorphic to $\mathcal{R}_t$ where q is a product of several small prime moduli q_i s.t. $q = \prod \{q_i \mid q_i \in \mathbb{P}\}$. We derive a private key K_{priv} by uniformly sampling from finite ring $\mathcal{R}_B = \mathbb{Z}_B[X]/(x^n + 1)$ where B denotes the degree of the base number system we are working in, and generate public key $K_{pub} = (p_0, p_1) = ([-(as + e)]_q, a)$ where a is uniformly sampled from finite ring R_q, and e is sampled from a normal distribution.

Encryption. We encipher plain-text polynomials $X \in \mathcal{R}_t$ into a set of cipher-text polynomial coefficients $C = ([\lfloor q/t \rfloor m + e_1 + p_0 u]_q, [p_1 u + e_2]_q)$ where u is sampled from $\mathcal{R}_B$, and e_1, e_2 are sampled from a normal distribution by projecting X over $\mathcal{R}_q$.

Decryption. We decipher cipher-text polynomials $C = (c_0, c_1, ..., c_n)$ with private key K_{priv} by computing $X = [\lfloor c_0 + c_1 K_{priv} + ... + c_n K_{priv}^n \rceil]_t$.

Homomorphism. Given that we are performing arithmetic operations over cipher-text polynomials within cyclotomic polynomial ring $\mathcal{R}_q$, it is non-trivial that homomorphisms are established for additions and multiplications. Hence, were we to perform multiplication, given operands denoted as cipher-texts $A = (a_0, a_1)$ and $B = (b_0, b_1)$, we would result in three polynomial coefficients $C = (c_0, c_1, c_2) \in \mathcal{R}_q$ as follows:

$$
\begin{aligned}
c_0 &= [\lfloor \frac{t}{q} a_0 b_0 \rceil]_q \\
c_1 &= [\lfloor \frac{t}{q} (a_0 b_1 + a_1 b_0) \rceil]_q \\
c_2 &= [\lfloor \frac{t}{q} a_1 b_1 \rceil]_q
\end{aligned} \tag{2}
$$

Should we wish to reduce the set of resultant cipher-text C down in size, we distribute to a virtual machine instance a set of keys $K_{eval} = (E_0^i, E_1^i) = ([-(a_i K_{priv} + e_i) + B^i K_{priv}^2]_q, a_i)$ where $i \in [0, \lfloor \log_b q \rfloor]$, a_i is uniformly sampled from finite ring $\mathcal{R}_q$, and e_i is sampled from a normal distribution, and compute the reduced set of cipher-text polynomials $C' = (c'_0, c'_1, ..., c'_{k-1})$ given $C = (c_0, c_1, ..., c_k)$ as follows:

$$
\begin{aligned}
c'_0 &= c_0 + \sum_{i=0}^{\log_B q} E_0^i c_k^{(i)} \\
c'_1 &= c_1 + \sum_{i=0}^{\log_B q} E_1^i c_k^{(i)}
\end{aligned} \tag{3}
$$

with $c'_j = c_j$ for $2 \leq j \leq k-1$. By recursively applying said reduction operator, we can reduce the size of the set of cipher-text polynomials C down to $|C| = 2$ in amidst the execution of distributed computational task.

3.4.2 Performance

In spite of both distributed systems and public cloud infrastructures having inherent blackboxed network/machine latencies and instabilities, it is still possible to achieve high throughout and system availability throughout a decentralized system.

This is evident from several large enterprises maintaining highly available distributed systems such as Google and Microsoft and Amazon by making using the principles and advantages horizontal scalability brings to the table.

In other words, scaling up for greater computing throughput and availability matters more on quantity vs. quality.

Given the low costs to trustlessly securing down virtual machine instances out of commodity smart devices to a customer, Perlin stresses on scaling computational tasks by quantitatively bringing more machines to a task rather than qualitatively bringing far more beefier machines to a specified task.

This in turn is far more economical and allows for greater speed, performance and efficiency in completing a customers task on Perlin's network.

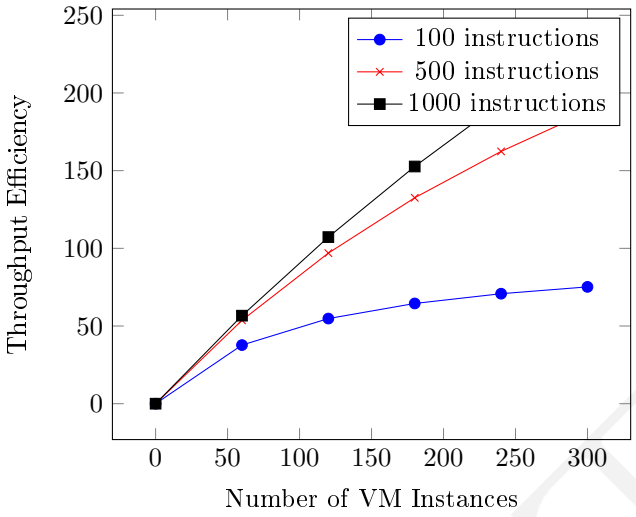

Figure 2: Number of miner virtual machine instances vs. throughput efficiency for computation p consisting of a different number of instructions. Task throughput easily goes 10× over initial throughput given approximately 8 miner virtual machine instances with single CPU processors that are equivalent in terms of average processing times assumed to be 100 nanoseconds per instruction.

The analysis to Perlin achieving high-throughput in spite of network latency is analogous to how processors that are pipelined achieve greater throughput over non-pipelined processors.

We define a computation p to consist of n instructions; to be run on a processor which can perform in parallel k instructions at once with average time per instruction being t denoted in seconds.

A pipelined processor would ideally be able to execute computation p in $(k+n-1)t$ seconds, whereas a non-pipelined processor would execute computation p in nkt seconds. Hence, the throughput efficiency of a pipelined processor vs. a non-pipelined processor is as follows:

$$\frac{nkt}{(k+n-1)t} = \frac{nk}{k+n-1} \tag{4}$$

Figure 3: The average processing time per instruction does not affect the throughput efficiency for a computational task.

Despite there being latencies inherent in distributing instructions over a pipelined processor to multiple pipeline stages, pipelining is a common technique that significantly improves CPU processor throughput given more pipelining stages.

As for Perlin, despite there being latencies inherent in distributing partitions of a computation c to multiple miner virtual machine instances, the distribution of computation is a common technique to significantly improve computational throughput given more virtual machine instances. Although network latency is an order of magnitude larger than the latencies inherent in pipelining over a processor, speedups are unbounded and are significantly affected by the number of VM instances allocated to a computational task.

See Figure 2 for a plot showcasing the significance in computational throughput given a number of remote virtual machine instances.

3.4.3 Applications

By amalgamating both performance and privacy through combining the best of both proof of compute availability and cryptographic primitives, we denote use cases for Perlin below.

3.4.3.1 Differential Programming

Numerous state-of-the-art techniques utilized in a wide variety of academic and industrial fields deal with developing mathematical models whose parameters are obtained through optimization techniques iterated over continuously differentiable objectives.

Examples of such techniques include predictive maintenance over supply chain management systems, phenotype modeling over drug discovery operations condoned by pharmaceutical companies, autoregressive time series prediction models of stock market features utilized by high-frequency trading firms and companies, and predictive user behavior modeling established by advertisement companies.

In broad terms, we can designate said techniques to all fall under an emerging academic field known as differential programming.

$$\min_{E(f(W,b,x),y')} f(W,b,x) \tag{5}$$

where $E(f)$ is a continuously differentiable objective function, $f(W,b,x)$ is a model parameterized over a set of multiplicative weight parameters W and a set of additive bias parameters b, and a dataset consisting of a set of inputs x and ground truth labels y'.

Perlin provides flexibility to its customers on partitioning the dataset (x,y'), or the model $f(W,b,x)$ across miner instances with respect to potential gains in performance and throughput achievable out of massive parallelism.

To allow for differential programming over censored data, one may simply derive gradients computed off of $E(f)$ with respect to $f(W,b,x)$ given that additive and multiplicative homomorphisms are preserved on cipher text generated by Perlin's cryptographic utilities.

Hence, with homomorphic cryptosystems, one may train differential models for numerous objectives given that gradients necessary to update parameters for model f with respect to encrypted cipher text x' is equivalent to gradients with respect to raw inputs x due to preserved homomorphisms.

Utilizing gradients $\frac{\partial E(f(W,b,x),y')}{\partial p}$ for parameter p, a customer may then apply a nth-order iterative optimization algorithm over model f to update parameter sets W and b.

$$W_{t+1} = W_t - \mu \frac{\partial E(f(W_t,b_t,x),y')}{\partial W_t} \qquad b_{t+1} = b_t - \mu \frac{\partial E(f(W_t,b_t,x),y')}{\partial b_t} \tag{6}$$
$$\frac{\partial E(f(W_t,b_t,x),y')}{\partial p_t} = \frac{\partial E(f(W_t,b_t,x'),y')}{\partial p_t}$$

where t represents the t'th iteration of a first-order optimization scheme with gradients scaled by μ being applied to a parameter p such that model error indicated by E collectively minimizes.

3.4.3.2 Distributed Systems

Distributed system components utilized in enterprises and startups such as databases, persistent messaging queues, stream processors, and container orchestrators are expensive to maintain and operate.

Monolithic variants of said components on the other hand hold a problematic issue of only being vertically scalable, and thus are more expensive to maintain.

Not only are these systems costly, but such centralization of technical infrastructure components makes achieving high-availability and fault-tolerance of one's distributed technical infrastructure subjective to the quality of computing power, hardware, and devops team one possesses.

Perlin introduces a novel application of decentralized ledgers in which it dis-intermediates centralized backbones underlying distributed systems deployed in centralized clusters such that estab-

lishing high-availability, consistency, and throughput of said systems is much cheaper and easier to achieve.

Utilizing Perlin's network, one for example may host an eventually-consistent fault-tolerant database on potentially thousands of miner nodes for low costs. Data hosted over such a massively distributed system may additionally be censored, encrypted, and operated on given Perlin's homomorphic cryptosystem implementations.

Thus, with Perlin, any enterprise or startup has the capacity to easily build fault-tolerant, highly available decentralized systems using Perlin's cryptographic utilities, parallelism utilities, and network of idle computing power.

4 Conclusion

We formalize the concept of metastability in which one can reason and come into consensus about a set of information more quickly and efficiently by equipping parties with information about how a consensus mechanism dissipates information across a network of nodes.

Through this formalization, we implement and bootstrap new mechanisms to the metastable, partially synchronous DAG-based distributed ledger protocol Avalanche to produce a novel, scalable, and practical general distributed ledger that can economically and efficiently cater for a plethora of use cases that significantly benefits both academia and numerous industries.

In doing so, we also conceptualize a practical initial use case of Perlin amalgamated with a novel proof of compute availability which suffices as the bare minimal building blocks necessary in creating a truly decentralized and trustless cloud computing market platform.