Keeping TABS

A Canon-Arbitration Algorithm for *Modified-GHOST* PoW Blockchain Protocols

Isaac Ardis

Daniel Aronoff
MIT Economics
MIT Media Lab

April 17, 2022

Abstract

Canon-arbitration algorithms are used to build consensus in decentralized state machines. The consensus characteristics of mutually implemented canon-arbitration algorithms enables a race between block authors and thereby facilitate the trust-minimized security characteristics of the blockchain state. In Proof of Work (PoW) block validation protocols, canon-arbitration algorithms typically depend on dynamic emission parameters like Difficulty, facilitating consistent canonical block emission outcomes for the network.

We consider a proposed modification to the *Modified-GHOST* algorithm in use on Ethereum.[1] This proposal introduces two novel features: a capacity for segment-specific transactions, and a positive scalar value representing a general measurement of active capital. From these features we propose a novel condition program situated in an assumed incumbent canon-arbitration algorithm.

Our proposed design is an augmentation of an existing and widely implemented consensus program. As such, it assumes and depends on the existence of other (canon-arbitration) algorithmic conditions. We inherit and generally preserve as invariants the characteristics of the incumbent canon-arbitration algorithm.

We claim and attempt to demonstrate that the proposed algorithmic condition program raises the incumbent consensus decidability rate. We show that this increased decidability rate improves chain state finality characteristics and reduces network energy waste by 50%. Further, we show that the cost to an adversary in a double-spend attack scenario rises. We detail any associated additional risk to the network, which we show to be practically nominal. We note that implementation of this design is feasible for Ethereum's primary client *go-ethereum* (`geth`) and probably feasible for most existing PoW clients.

[1] `https://ethereum.org/en/whitepaper/#modified-ghost-implementation`

1 Model Definitions

Table 1: List of Model Definitions

Model Elements	
t	Unit of time. Standard unit: seconds.
λ	Network mean block time; the average number of seconds between block timestamps.
η	Network block-message latency (measurable or assumed) between two nodes. The time elapsed between the transmission of a message containing block information and its reception.
B_i	A block with number i.
H_i	A block header with number i.
H_d	Block (and header) annotated difficulty
H_k	Block (and header) annotated TABS value
$TAB(B)$	A scalar value representing aggregated account balances for a given block B. "TAB" stands for Total Active Balance.
$TABS(B_i, B_{\mathrm{i}-1})$	A scalar value derived from the TAB value of block B_i and recursively its parent B_{i-1}. This value is annotated in a block's header. "S" stands for *synthetic* or *synthesis*.
$\sigma[a]$	Global account state σ at address a
T	A transaction
T_a	A proposed transaction field: Ancestor Hash
$\delta(T_a)$	The depth (magnitude of ancestry) of a block cited by some transaction as T_a
τ	A network-level aggregate over time or segment (assume: average) of Ancestor Hash depth δ values. Transactions *not* asserting an Ancestor Hash value can be defined as having $\delta = H_i$. Smaller values represent a greater dependence on blocks nearer a common (consensus) chain head (a greater degree of transaction/segment exclusivity), while larger values signal less exclusive transactional assertion on chain state dependence.
CS	Canon Score. A scalar value representing canon preference weight of a single block. Comparable to Difficulty.
TCS	Total Canon Score. A scalar value representing protocol canonical preference. Derivation of this value for any block relies on the antecedent calculation of that of its parent. Comparable to Total Difficulty.

2 Glossary

Table 2: Definitions of Common Terms

Glossary	
Ancestor Hash	A proposed transaction field whose value references a block by hash, specifying the existence of that block in the transaction's chain segment as a condition of the transaction's validity. Proposed. Use optional. Synonyms: Segment ID, HFC (Header Field Context) Hash. Related: Chain ID.
Canonical-Arbitration Algorithm	A program of comparative conditions resulting in the selection of one prioritized segment from any two segments. In context, the prioritized segment is used as the basis for determining a canonical chain state, and, as such, for focusing mining effort (via establishing a *parent*. Synonyms: Canonical-Preference Algorithm. Related: GHOST, *Modified-GHOST* , Inclusive Protocol, Nakamoto Protocol.
Segment-Agnostic (Transaction)	A property on a transaction such that the transaction can be applied to one or more chain histories. Related: Segment-Specific.
Segment-Specific (Transaction)	A property on a transaction such that the transaction can be applied on a subset of available chain histories. Synonyms: Segment-Exclusive, Chain Context Constrained. Related: Segment-Agnostic.
Chain State Finality	The permanence of chain state (or series of states). PoW blockchain states have characteristically 'soft' finality characteristics. PoW finality expectations increase as relative state increases; newer states are more vulnerable to change than older states.
Block Emissions	The (rate of) production of blocks. Blocks are metadata associated with chain state modification sequences.
Decidability	The frequency of ties in canon-arbitration during chain growth. Narula et al. call this *symmetry-breaking*. Decidable scenarios are characterized by eventual network-level chain state consensus. Undecidable scenarios are characterized by bifurcated (canonical) network chain states.

3 Scenarios

Table 3: Outlines of Referenced Chain State Scenarios

Scenarios	
Finality Fraud	A premeditated fraud scheme exploiting a victim's assumption of chain state which is later invalidated by the attacker. The exemplary scenario defined and reused throughout this paper assumes a single entity as a victim and a single censored transaction. In most contexts, these assumed parameters could be modified to use a plural set of victim entities and/or a plural set of censored transactions while still being considered Finality Fraud.
Finality Terrorism	Generally schematically equivalent to Finality Fraud; the difference is in the attacker's motive and an intentionally large number of victims. The exemplary scenario defined and reused throughout this paper assumes all users of a blockchain as victims and the censorship of all transactions. These parameters are extremes. In most contexts, these assumed parameters could be modified while still being considered Finality Terrorism.

4 Appendix: Ethereum Background

The parameters and logic of this paper's proposal take the existence of a system equivalent with that of Ethereum's at the time of writing. We strongly suggest that the Ethereum Yellow Paper[2] be read and understood as a precondition for interpretation and evaluation of the work of this project. Important and referenced concepts of Ethereum's protocol are discussed in this section for the the informational value of redundancy.

4.1 Ethash

Ethereum's Proof-of-Work protocol "Ethash" governs block emissions. Solutions to a constantly varying guessing game require time to discover – via trial and error – and are a notably required field in block headers. Authors of blocks providing valid solutions to these puzzles are called miners; mining is the process of searching for puzzle solutions.

A very difficult puzzle (probably requiring many guesses) is expected to take more time to solve than an easier puzzle (requiring fewer guesses).

4.2 Difficulty

Ethereum regulates its block emission rate using a header value called Difficulty (H_d). This value is used by the Ethash protocol as a parameter for puzzle solution validation. Generally speaking, the Difficulty value for some block can be thought of as the number of wrong guesses ("hashes") expected before a valid puzzle solution for a child block is found.[3] An adjustment algorithm governs the rise and fall of this value, accepting a block timestamp interval and parent difficulty as parameters; this provides a feedback loop joining block emission rates (via sequential block timestamps and thus relative intervals) with puzzle difficulty. Given a target block interval, difficulty can be adjusted dynamically, incrementally, such that the difficulty value will cause blocks to be authorable at rates approaching the target rate. In Ethereum the parameters are tuned to produce a median

[2] https://github.com/ethereum/yellowpaper/tree/fabef2531a8f8e772a4bf5be23191044d0ce3569

[3] Formally, difficulty is governed by the relation of a fixed size bit-field (called the 'nonce') in the domain of the block hash, into which a miner writes its guess, and a fixed size bit-field in the output range of the hash function, which takes on values induced by the miner's guess. The difficulty is measured by the number of required zero entries in the output bit-field, which defines the feasible output range. The size of the set of guesses that map into the feasible range expands and contracts with the size of the feasible range. For a given rate of guesses per unit of time, the difficulty induces a cumulative distribution function of the probability of a solution being found by each point in time. From this an expected interval between timestamps can be derived.

9 seconds, or about 14 seconds on average. Network block emission rates are typically modeled with a Poisson distribution.[4][5][6][7]

e.g. Given a block difficulty $H_d = 63115$ and a network target emission interval of 13 seconds, we deduce that the modeled network average hashrate is 63115 hashes / 13 seconds = 4855 hashes/second. A block generated in 3 seconds should cause the difficulty to rise; a block generated in 30 seconds should cause it to fall; making the next block respectively harder or easier (slower or faster) to author.

4.3 Canonical Arbitration[8]

The "Total Difficulty" value for any chain segment is the sum of the H_d values of block headers in the segment. The canon-arbitration algorithm used today by Ethereum defines that preference is given with priority to any valid subtree having the greatest "Total Difficulty" value. In the case of equivalent TD values, segments are preferred having lesser latest block numbers. In the case of equivalent TD and block height (number) values, if a node acts on behalf of the block's registered author, that block is preferred. If there is no authorship beneficiary interest (as in a non-mining node, or that of a non-winning miner), a figurative coin is tossed, per the protocol described by Eyal and Sirer.[9]

4.4 Chain ID

As specified by EIP155[10] a feature called *Chain ID* was introduced on the Ethereum network (ETH) at block 2675000, and on the Ethereum Classic network (ETC) at block 3000000.

This feature, generally considered, allows transactions to be made exclusive to certain predefined segments, or entire chains, by validating a match between a transaction-specified *Chain ID* field value and that of a hardcoded (constant) network protocol configuration. The mechanism ascribes an arbitrary positive integer to a network configuration and transactions use this value when signing a transaction intended for that subset of chain state. Transactions are not required to specify a *Chain ID* value for legacy-compatibility reasons.

[4] https://ethresear.ch/t/deep-dive-into-current-pow-difficulty-adjustment-algorithm-and-a-possible-alternati 5267

[5] https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/

[6] https://en.wikipedia.org/wiki/Proof_of_work#Variants

[7] https://arxiv.org/pdf/1901.04620,SectionIII.A

[8] https://github.com/ethereum/go-ethereum/blob/f0328f241b7c3def217b0c2dce1a1b297f979a37/core/forkchoice.go#L77

[9] http://www.cs.cornell.edu/~ie53/publications/btcProcFC.pdf

[10] https://eips.ethereum.org/EIPS/eip-155

This feature was introduced after "The DAO Fork" (at block 1920000) caused Ethereum to become both Ethereum and Ethereum Classic by the network's partial rejection of an arbitrary chain state mutation. Until the introduction of Chain ID, transactions after this hardfork had no way of being exclusive to one of either of these two chains. Between blocks 1920000 and 2675000on ETH, transactions were ambiguously valid on either, or both. Today, Chain ID transactions specifying Chain ID `1` are only eligible on ETH, and those with Chain ID `61` are only eligible on ETC, per the respective configurations eventually introduced on both networks with their independent implementations of the EIP155 feature.

5 Specifications

In this section we describe the elements of the *TABS* modification to the *Modified-GHOST* algorithm.

5.1 Transactions with Ancestor Hash

Transactions are afforded an additional field T_{a} (*Ancestor Hash*) whose value either describes a block hash or is left undefined.[11]

If filled, the existence of a block with a header matching this hash anywhere in the foregoing canonical chain is a condition of the transaction's validity.

Transactions of this type are considered segment-specific. The degree to which they are specific depends on which block is referenced by the hash is provided. Transactions referencing relatively older headers are less specific than those referencing newer headers.

As proposed, transactions are not required to provide this value. However, discretionary use is not necessary. The field's annotation *could* be demanded by the protocol.

We will show that the rate of this feature's use (τ) will drive desirable security characteristics of the proposed canon-arbitration condition program.

5.2 TABS Validation

Total Active Balance Synthesis (TABS) is a positive scalar value, an aggregate of all transaction sender accounts (and the miner) for a blockchain of some length. Generally considered, it provides a representative measure of active capital on some chain state. It is denominated in Wei,[12] equivalent to 1000000000000000000[13] so-called Ether. Its value is used as a parameter in the proposed canon-arbitration condition.

Validation of the TABS value of a block of header H is asserted as $\mathrm{TABS}(H)$,

where:

$$\mathrm{TABS}(H) \equiv \begin{cases} K_0 & \text{if} \quad H_{\mathrm{i}} = 0 \,||\, P(H)_{H_k} \text{is undefined} \\ \max(K_0, P(H)_{\mathrm{H_k}} + x \times y) & \text{otherwise} \end{cases} \tag{1}$$

[11] Original and complete specification, rationale, and motivation for this feature is documented as a standalone proposal at https://github.com/ethereumclassic/ecips/tree/master/_specs/ecip-1105.md

[12] https://ethereum.stackexchange.com/questions/253

[13] 1e18

where:

$$K_0 \equiv 128 \times 10^{18} \tag{2}$$

$$x \equiv \left\lfloor \frac{P(H)_{\mathrm{H_k}}}{128} \right\rfloor \tag{3}$$

$$y \equiv \begin{cases} -1 & \text{if}\,\mathrm{TAB(B)} < \mathrm{P(H)}_{H_k} \\ 0 & \text{if}\,\mathrm{TAB(B)} \equiv \mathrm{P(H)}_{H_k} \\ 1 & \text{if}\,\mathrm{TAB(B)} > \mathrm{P(H)}_{H_k} \end{cases} \tag{4}$$

The TAB(B) value of a block is defined as the sum of the miner balance and all of the sender balances for transactions with an affirmative T_a ancestor hash field:

$$TAB(B) \equiv L(B, B_c) + \sum_{T \in B_{\mathrm{txes}}} L(B, T_{\mathrm{sender}}) \tag{5}$$

where:

$$L(B, s) \equiv P(B)_{\sigma[s]_{\mathrm{b}}} \quad \text{The account state balance at parent block } P(B) \text{ of account } s \tag{6}$$

The reader may note that the integer `128` appears shared between the x and K_0 components. This is not of necessity, but of convenience and for simplicity of design. Both values are tuneable constant parameters in the algorithm. K_0 expresses a minimum value for $TABS$, while x expresses an adjustment ratio of the $TABS$ value between two parent-child related blocks.

5.3 Canon Scoring

Canon-arbitration algorithms are assumed to compare two blocks and to indicate decisively which one of the two ought to receive canonical status. This section formally defines a value Canon Score *CS*, which is used as the *initial condition* for a canon-arbitration algorithm. This condition, in both the pre-existing and proposed algorithms, may result in a tie. If the condition is decisive (a winner is determined), subsequent conditions are not evaluated.

If indecisive (the result is a tie), subsequent conditions are evaluated, each potentially decisively, in program order.

For the sake of context, we write the assumed existing initial condition for canon-arbitration in our notation below.

The Canon Score value is defined as:

$$\text{CS} \equiv H_d \tag{7}$$

And derived from that a Total Canon Score (TCS) for any segment as:

$$\text{TCS} \equiv \sum_{H_{\text{i}=0}}^{\text{i}<=\text{n}} H_d \tag{8}$$

Respective to this assumed existing condition, we propose its modification as,

in the atomic case:

$$\text{CS} \equiv H_d \times H_k \tag{9}$$

and in the segment case:

$$\text{TCS} \equiv \sum_{H_{\text{i}=0}}^{\text{i}<=\text{n}} H_d \times H_k \tag{10}$$

In both the pre-existing algorithm and the proposed algorithm, a candidate block having a greater TCS value than the other should be selected for the exclusive canonical state.

6 Visual Glossary

Figure 1: A block

Figure 2: A transaction

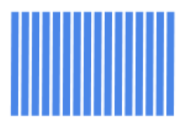


Figure 3: Several transactions

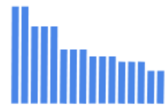


Figure 4: Several transactions representing relative gas consumption

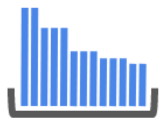


Figure 5: Transactions in a block

Figure 6: A block segment (chain)

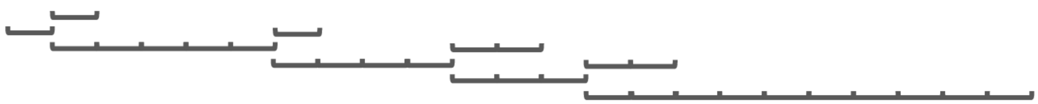


Figure 7: A block segment with forks

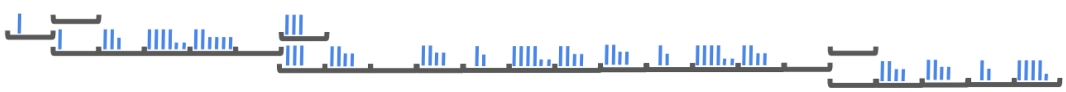


Figure 8: A block segment with forks and sparse transactions

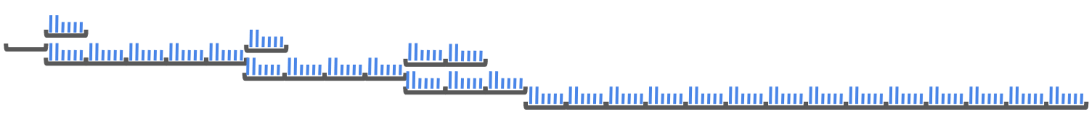


Figure 9: A longer block segment with forks and full of transactions

Figure 10: Block composition of a large reorg (eg. finality attack)

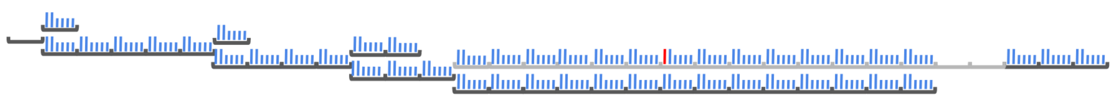


Figure 11: Block composition of a large reorg (eg. finality attack) including transactions. The ‘fraudulent‘ modified or omitted (censored) transaction is red. which the double-spend was made?

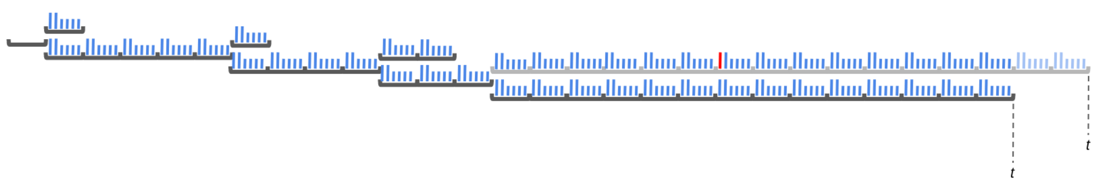


Figure 12: Block composition of a large reorg (eg. finality attack), noting a value t representing some point in time. Since the domain of this visualization is (implicity) block number, we deduce that the gray (top) chain has built more blocks per unit of time than its counterpart. The value of time t in this case can be understood as an arbitrary instant where the segments (via their head blocks) are evaluated for canonical status.

Figure 13: Two illustrations of transaction inclusion under the large reorg scenario and assuming successively aggressive τ rates (composite frequency and value of the proposed Transaction Ancestor Hash field and validation).

Figure 14: A component diagram of block and transaction occurrences under the assumed reorg scenario with transactional Ancestor Hash use.

Figure 15: Visual conceptualization of the incumbent canon-arbitration initial condition using Total Difficulty values.

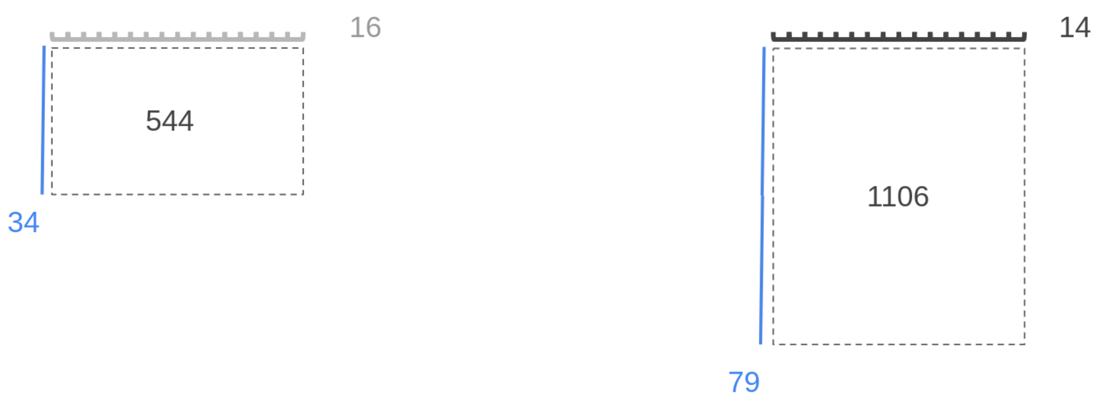


Figure 16: Visual conceptualization of the proposed canon-arbitration initial condition using Difficulty and TABS values.

7 Theoretical Models

7.1 Miner Expected Revenue

We naively define miner expected revenue over time t (in seconds) as:

$$\frac{Miner_{h/s}}{Network_{h/s}} \times P(E,t) \times P(C) \times (Block_{\text{reward}} + Block_{\text{tx}_\text{f}\text{ees}}) \tag{11}$$

(Note that the ratio $\frac{Miner_{h/s}}{Network_{h/s}}$ is intuitive but naive, and can be further developed.)

7.2 Probability of block production

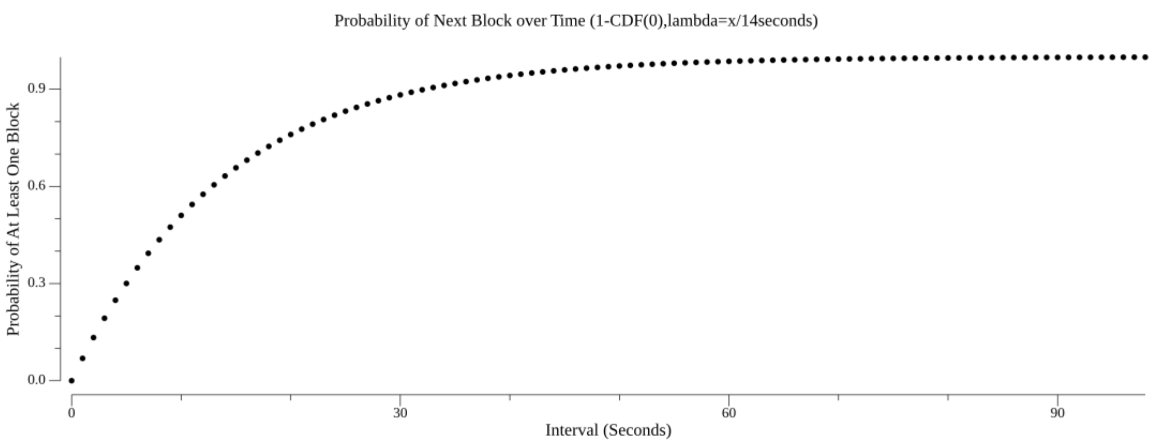


Figure 17: A Poisson CDF function is used to model the probability of a block ocurrence (Y) in some interval (X). This plot is produced by the included Go test `TestPoissonCDFNextBlock`.

We assume that the distribution of block intervals at the network level can be modeled as a Poisson Distribution.[14] From this, we can define the probability of any block being discovered in an interval of t seconds as a derivation of the Poisson CDF.

$$P(E,t) \equiv 1 - e^{-\lambda} \times \sum_{i=0}^{|k|} \frac{\lambda^i}{i!} \tag{12}$$

[14] One must note that while it may be granted that the model fits *recorded* network block interval data, it may not necessarily represent the *actual* production intervals. Neither must it represent accurately the production interval of blocks by any subset of miners.

In this model, k is held constant at 0 signifying the occurence of 0 blocks, and λ can be defined as the product of an average event rate r and some time interval t.[15]

7.3 Probability of a block's canonical state

The probability (at the network level) of a block being ultimately accepted as canonical by the network:

$$P(C) \equiv 1 - \epsilon \tag{13}$$

In this simple definition, ϵ represents the frequency in which a miner authors a valid (eligible,candidate,potentially-canonical) block which is not ultimately accepted into the public canonical chain.[16]

We claim and show that $\epsilon > 0$. We take $\epsilon < 1$ for granted as common sense; canonical blocks exist.

A practical value of ϵ can be approximated by an empirical measurement of a network's Uncle rate.[17] Alternatively, we can derive a reasonable definition of ϵ using purely theoretical models, as follows.

Given our assumption of the Poisson Distribution fit for network block intervals, we can set Poisson's k to 2, representing the occurence of 2 blocks. Using the Poisson Probability Mass Function, we see that given an average block interval of 1/14 seconds, the probability of seeing $k = 2$ blocks in some interval $t = 1$ seconds ($\lambda = \text{rate} * \text{interval} = 1 * \frac{1}{14}$) is:

$$\frac{\frac{1}{14}^2 e^{-\frac{1}{14}}}{2!} = 0.00237516... \tag{14}$$

[15] For example, given an average network block intervals of 14 seconds, the expectation that a next block occurs within 3 seconds becomes $\lambda = (1/14)*3$, yielding 19.2%. The plot in Figure 17 shows this probability for intervals of 0 through 99 seconds assuming an average block interval rate of 14 seconds.

[16]The solutions these blocks represent in a PoW chain have material cost; ie. wasted electricity and wasted money.

[17] There is a lot to say about this, and the nuances of this idea are important for understanding what kind of approximation this is, and what its limits are. A few brief statements are provided for context below, but should not be considered comprehensive or definitive:

The Uncle rate represents a record of the existence of blocks which are not canonical. Iff the revenue for those non-canonical blocks is less than their cost of production, the existence of any non-canonical blocks implies waste, which suggests a production inefficiency and an economically undesirable characteristic. From this, understanding that the uncle records provided on chain are potentially incomplete records of the existence of non-canonical blocks.

We might explore this further by look at the rewards more closely.

Is it actually reasonable for a miner to record an uncle, getting the other miner paid? Or are uncles only recorded when the author of the canonical block is also the author of the uncle?

But we must extend this to represent the probability of seeing 2 *or more* blocks. We approximate a generalization, using 99 as an arbitrary upper limit for the considered number of potential blocks[18] occurring in the interval:

$$\sum_{k=2}^{99} \frac{\frac{1}{14}^{k} e^{-\frac{1}{14}}}{k!} = 0.002432736... \tag{15}$$

Since by definition the canonical state is applied exclusively to 1 of ≥ 2 block occurrences, we have shown that even under an idealized game, the theoretical ϵ value is probably a positive, small, number.

While this model could be taken further, for the sake of argument it is not necessary and will not be pursued.

8 Models and Data

We use two computer program models to simulate competitive block emissions.

go-block-step models block emissions from the network level. It tests and compares simulated network emission outcomes against the expected Poisson distribution of block time intervals.

go-miner-sim models block emissions in an "actor-based" way, simulating block emissions given a set of independent miners following a common protocol.

As neighbors, these models can be considered as 'top-down' (`go-block-step`) and 'bottom-up' (`go-miner-sim`) approaches to network block emissions and consensus modeling.

We compare the data produced by these programs with empirical chain data, testing for representativeness. We want to understand the change the TABS protocol presents on block emissions rate, network consensus rate, and miner incentives and rewards, and transaction liveness[19].

8.1 Comparison of a Naive Model and Derived Data

We assume that the characteristics of a Poisson Distribution model accurately represent the characteristics of the PoW block emission model, and compare a theoretical model with empirical data under this condition.

[18] The number of potential blocks is the number of miners on the network.

[19] *Liveness* is a database-jargon term used in several related and referenced papers. In this context, it is the general chain state's processing of available transactions.

We assume that the empirical block interval data is accurately represented in Figure 18, and we assume the Ethereum network's mean block interval is reasonably considered as 14 seconds for the sake of the argument.

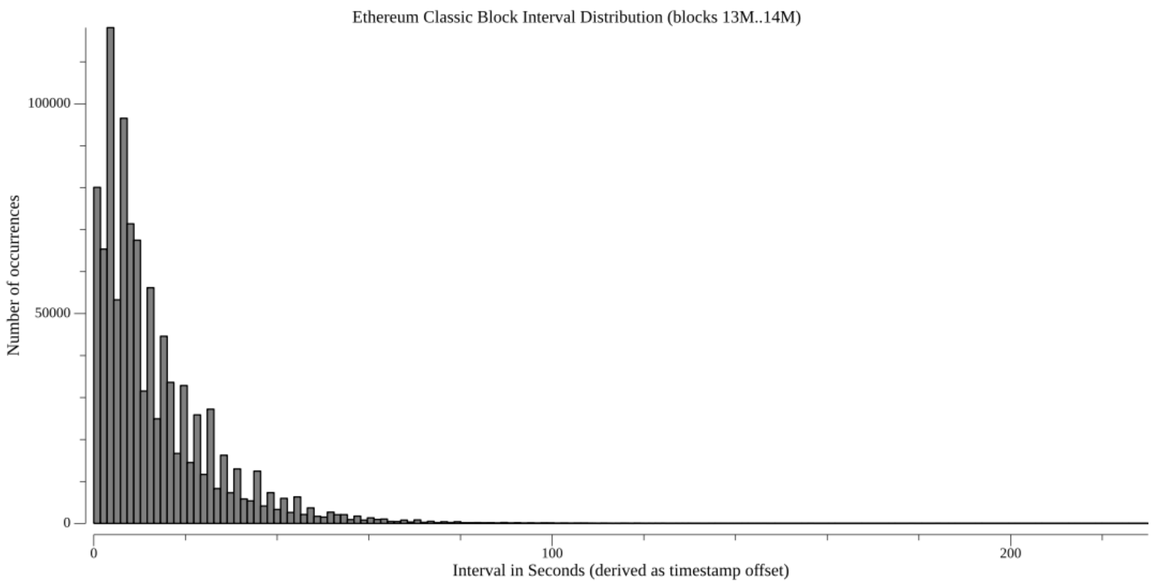


Figure 18: Exemplary block interval data for Ethereum.[20]

Next, we use a computer program to simulate block intervals under the Poisson model:[21]

[21] `TestPoissonIntervals`

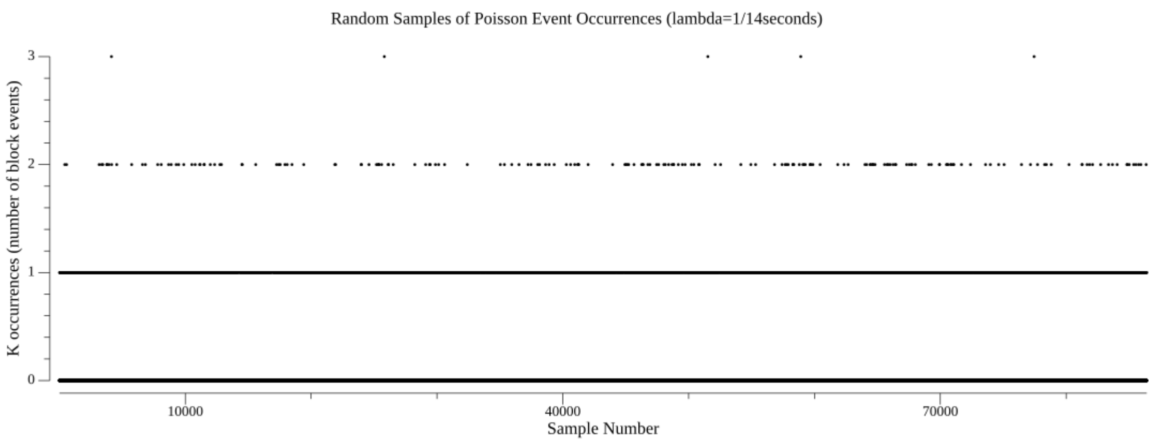


Figure 19: Procedurally generated event occurrence samples from a Poisson distribution with $\lambda = \frac{1}{14}$, sample size of 86400. We can interpret the data points at $y = 2$ in Figure 20 and Figure 21 as representative of occurrences of forks having 2 candidate blocks. For $y = 3$, the sample has 3 candidate blocks, etc.

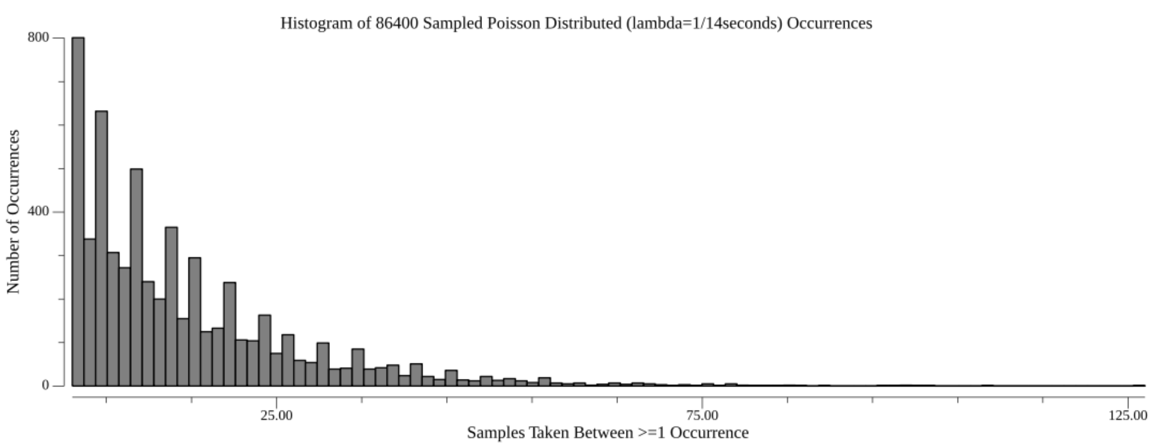


Figure 20: Procedurally generated intervals of samples where events ≥ 1 from a Poisson distribution with $\lambda = \frac{1}{14}$.

Where is the hump?

The empirical data clearly shows a 'hump' which is not visible in the procedurally sampled Poisson distribution values, which instead consistently slope downwards from lesser intervals to greater. From this simple napkin comparison, we must conclude that our model is as yet incomplete (or wrong!).

8.2 Network latency (η)

We propose that an explanation can be made for this mismatch presented in the previous section between map and territory by a consideration of network latency.

Proposition 8.1. *Latency exists.*

The material existence of network latency is irrefutable under current physical models of the world. Information cannot travel faster than the speed of light, which is finite. The networks in question assume the transmission of messages (information), and thus, this takes time. This is latency; the time required for message transmission.

The production of any block depends on the block it will be appended to; its Parent. A miner can only begin mining block $B_{i=n}$ after observing block $B_{i=n-1}$. The existence of network latency as a positive value has already been asserted.

Arbitrarily defining network latency as a constant $\eta = 3$ seconds allows us to produce a revised histogram of expected Poisson event intervals.

Proposition 8.2. *Latency is heterogenous.*

If we assume there exist m miners on the network, and that each miner controls $\frac{1}{m} \times Network_{h/s}$, we can estimate the rate of the same miner producing blocks $B_{i=n}$ and its child $B_{i=n+1}$ as $\frac{1}{m}$. When a miner produces both the Parent and Child blocks, the η value for that interval should approach 0; where time of program execution alone is considered relative to the aggregate time of program execution plus network transmission duration. We arbitrarily define $m = 8$, and, accounting for this probability, produce Figure 23.

Though still naive, we have found a hump.

Proposition 8.3. *Latency's heterogeneity depends on miner hashrate.*

We assume that miner hashrate for the Ethereum network is unevenly distributed. This assumption causes our estimation above of Parent/Child same-miner events as $\frac{1}{m}$ to be inaccurate. Under this theoretical model, we do not pursue further revisions for this approximation.

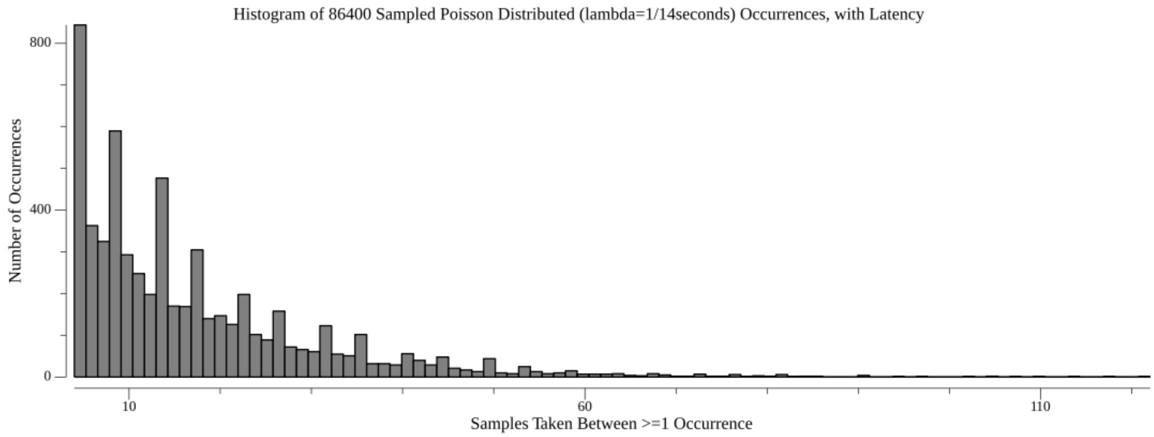


Figure 21: Procedurally generated events $k \geq 1$ interval samples from a Poisson distribution with $\lambda = \frac{1}{14}$, with a sample size of 86400. Every interval is increased by $\eta = 3$ seconds. Clearly, this general application of latency is insufficient. The shape is unchanged; only the domain is shifted by 3.

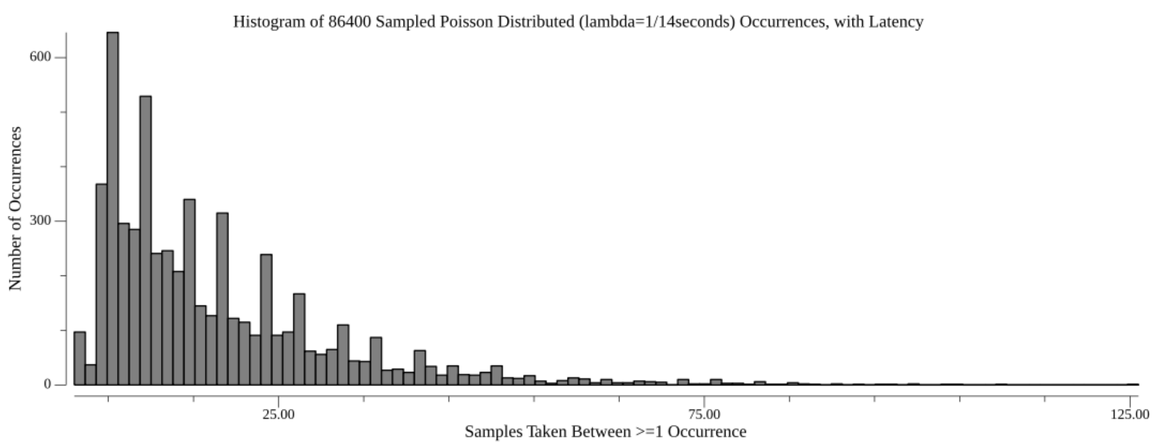


Figure 22: Procedurally generated events $k \geq 1$ interval samples from a Poisson distribution with $\lambda = \frac{1}{14}$, with a sample size of 86400. Every interval is increased by $\eta = 3$ for approximately $1 - \frac{1}{8}$ of the intervals.

Proposition 8.4. *Latency may be optional.*

If latency is defined as the time between transmission and reception of a message, it attributes should be mostly attributable to physical or otherwise 'external' limitations.[22]

However, if we were instead to define latency as the time between the availability of the message (to the transmitor) and the time of message reception, a new variable arises: delay.

By creating a delay on purpose, a miner gives themselves a head-start on the mining of a next block. They postpone the transmission of their message, stalling the game for the rest of the network. While this strategy can benefit the miner, it comes with the risk that another miner may produce (or have produced) a competitor block in this interval reducing their chances of success.

On this topic, the reader should reference Eyal and Sirer[23], who show that large-hashrate miners (ie. 25%, 33%, etc.) can be reasonably expected to demonstrate this behavior, and with it, a block emission rate superlinear to their hashrate ratio. They propose network bifurcation (via a simulated coin-toss) as a network protocol strategy for mitigating this incentive.

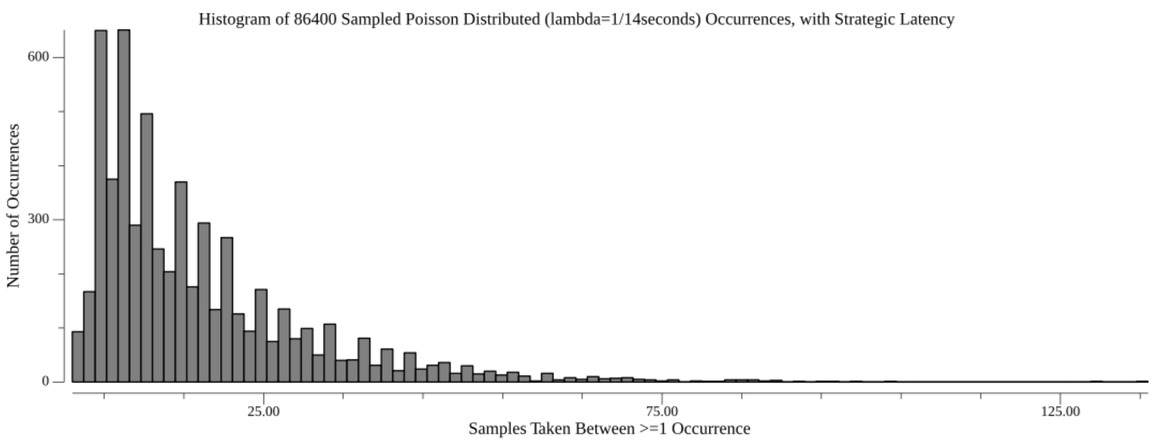


Figure 23: Procedurally generated events $k \geq 1$ interval samples from a Poisson distribution with $\lambda = \frac{1}{14}$, with a sample size of 86400. Every interval is increased by $\eta = 3 + \frac{1}{8} \times \text{interval}_{\Delta}$ seconds for approximately $1 - \frac{1}{8}$ of the intervals. This represents a loose event interval model including selfish miner delays.

[22] e.g Cable size, net neutrality (or lack thereof).

[23] TODO

9 A Different Model

The program `main` in *go-block-step/main.go* does not use a Poisson library or algorithm. Instead, mining is simulated. Search efficacy per miner is parameterized by network relative hashrate. The code relies on an arbitrary interval as the space from which a random needle is drawn. See code comments for more information.

This coded model can be used to simulate network block emission, constant latency[24], and block authorship data.

The model coded does not generate a blockchain. The model, instead, generates sample information for a single step (block increment) in the subtree, where the (assumed) parent block is held constant. Each sample is called a `round`.

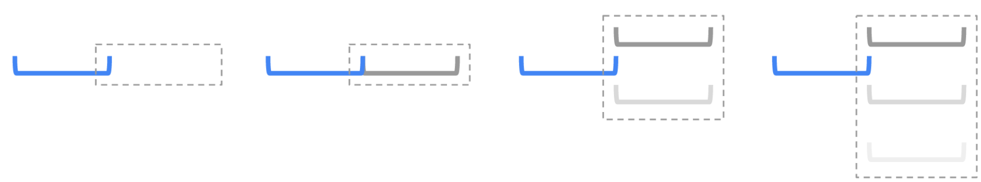


Figure 24: Visualizing theoretical outcomes for a block-space for some time interval t. The assumed existing parent block is blue.

For each round, the program is structured to assume a (any) sufficient t such that at least one child block is produced. This t value is pseudo-random, and is driven by the pseudo-randomness of the competing simulated miners. This t value is recorded as, and here referred to, as the resulting *interval* value.

Notably, the model does not use H_d (header difficulty) values. So long as we understand that the samples generated do not represent a continuous chain, we find that assuming a constant arbitrary 1 value for the parent TD value is reasonable.

Our aim is to model block emissions in a pseudo-statistical way, focusing on game theoretical decisions for miners rather than block-network propagation or network shape.

Latency is modeled only to the extent that it impacts block production intervals. A constant value is used; added to the recorded block interval for any miners who (probabilistically assigned) did not mine the parent block.

We could introduce a small bit of randomness to the latency value, but that would only produce noise.

[24] This is less of a concern than it may seem at first glance.

We can play with latency. The program wants it to be understood as a tuneable parameter. For now, the program knows that same-author rounds add a latency of zero, while everyone else adds a constant legacy value `RoundConfiguration.Latency`, which is, as you guessed, configurable per round.

We know that latency is an important thing. We assume, however, that the latency economy is efficient and nearly usally optimized, and that non-negligible-hashpower-share miners will be able to cost-effectively purchase sufficiently competitive latency values. With this, we assume that these same miners have approximately probably pretty much the same latencies.

The model could be extended or modified to do latency differently.

The model could also be changed to do hashrates and puzzle interval approximation differently.

Implementation of a stateful block difficulty characteristics would enable representation of random walk information concerning network relative hashrates for miners. This would be more realistic, but would also be more noisy.

The information generated is not deterministic.

Data

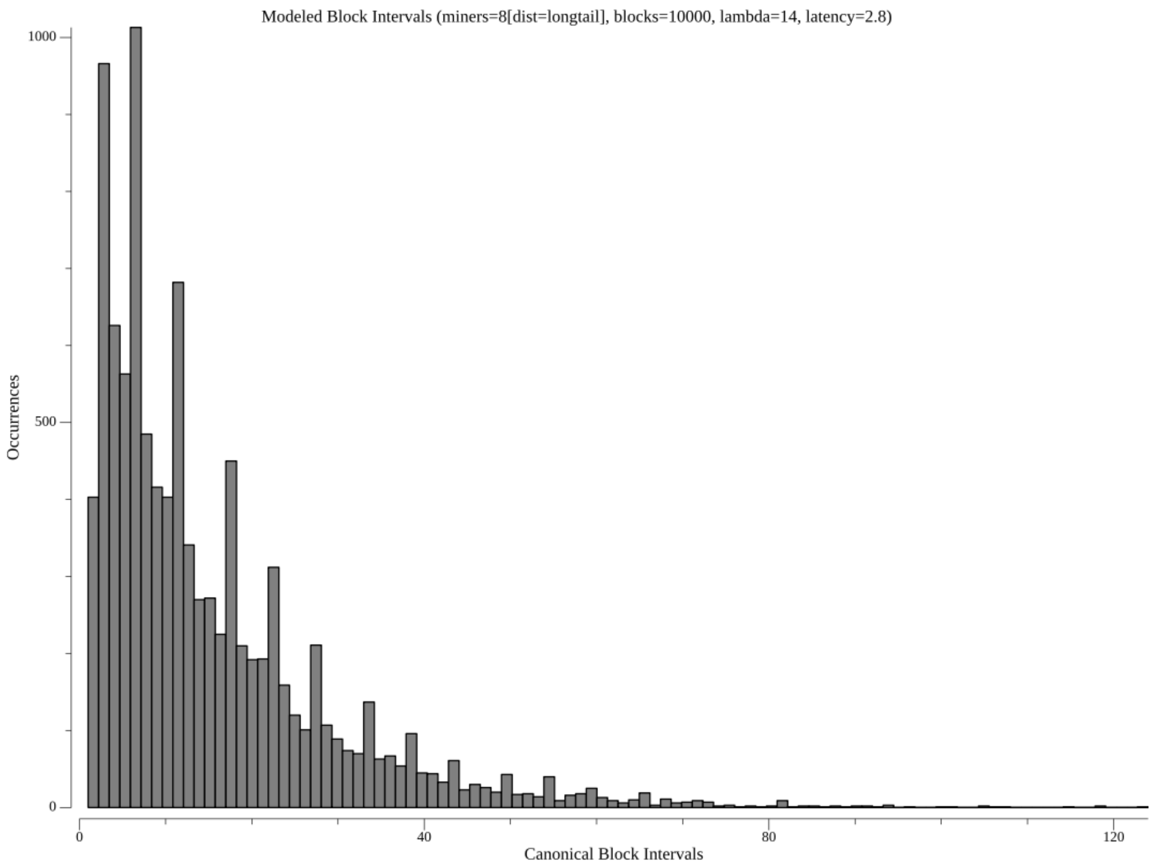


Figure 25: Simulated block intervals for $\lambda = \frac{1}{14}$, $\eta = 2.8$, with 8 miners, each with procedurally generated long-tail-ish shaping relative hashrate shares.

The first `Round` generates a print out:

```
CONFIG

Name:             A, ConsensusAlgorithm: TD,
NetworkLambda:    14, Latency:          2.80,
TickMultiple:     1, Rounds:           10000,
NumberOfMiners:   8, HashrateDistType: longtail,

GENERATED MINERS

Number: 8, Distribution: longtail, Hashrate Checksum OK:    true,
Hashrates: [0.333 0.222 0.148 0.099 0.066 0.059 0.044 0.02]

INTERVALS

Mean: 15.8650,
Med: 11.8000, Mode: [3.8000],
Min: 1.0000, Max: 139.8000,

ELIGIBLE AUTHORS PER BLOCK

Mean: 1.0819,
Med: 1.0000, Mode: [1.0000],
Min: 1.0000, Max: 3.0000,

MINER CANONICAL WINS

miner=0 hashrate=0.333 winrate=0.338 winrate/hashrate=1.013 (3375)
miner=1 hashrate=0.222 winrate=0.227 winrate/hashrate=1.020 (2266)
miner=2 hashrate=0.148 winrate=0.143 winrate/hashrate=0.963 (1427)
miner=3 hashrate=0.099 winrate=0.097 winrate/hashrate=0.979 (967)
miner=4 hashrate=0.066 winrate=0.066 winrate/hashrate=0.998 (657)
miner=5 hashrate=0.059 winrate=0.060 winrate/hashrate=1.022 (598)
miner=6 hashrate=0.044 winrate=0.043 winrate/hashrate=0.982 (431)
miner=7 hashrate=0.029 winrate=0.028 winrate/hashrate=0.953 (279)
```

```
ANALYSIS

Ticks: 136550, Rounds (Blocks): 10000, Ticks/Block: 13.655
AuthorSameParentChildTally/Block: 0.211
ArbitrationDecisiveRate: 0.028, ArbitrationDecisiveTally: 277
ArbitrationIndecisiveRate: 0.052, ArbitrationIndecisiveTally: 522
```

10 Interlude: In Consideration of the Tick

We have, until now, taken arbitrary rates, specifically *rate units* for granted.

Ethereum formally defines[25] block timestamps as:

H_{s} is the timestamp (in Unix's time()) of block H and must fulfil the relation:

$$H_{\mathrm{s}} > P(H)_{\mathrm{H_s}} \tag{16}$$

Ethereum Yellow Paper

This mechanism enforces a homeostasis in terms of the time between blocks; a smaller period between the last two blocks results in an increase in the difficulty level and thus additional computation required, lengthening the likely next period. Conversely, if the period is too large, the difficulty, and expected time to the next block, is reduced.

Timestamps are otherwise arbitrary.

Time in the real world moves with apparently infinite fluidity. An instant of time is as small as the instance of a Cartesian point.

Time in computers is as small as the time it takes for information to move. This is a bigger number than the theoretical instant.

Time in Ethereum gets as small as 1 second. The `go-ethereum` network client program `geth`[26], ignores (does not process) blocks having a timestamp $B_s > now() + 15$. This is a normally undocumented subjective behavior, parameterized by some computer's clock. Running `geth` on a computer with a clock set 'late' (reading actually past values) will fail to maintain synchronisation (consensus) with the network.

In coding a simulation, we can arbitrarily scale a *tick* interval, representing some arbitrary atomic unit of time corresponding to a program step or loop. The tick parameterizes the time domain of the simulation.

[25] Ethereum Yellow Paper
[26] The majority share client on the Ethereum network

Since much of our work here assumes the job of fitting a model to the empirical information available, we need to acknowledge the limits of this approach.

Our empirical knowledge is constrained (at this point in the research[27]) by using exlusively *objective* measurements. Block timestamps are objective measurements. If we were to take our own readings of network latency, or of block production intervals (for example, by actually mining and recording measurements), these would be what this paper considers *subjective* data.

We do not consider any empirical information with time units less than 1 second. Our programming can be configured to represent a simulation of arbitrary time units. Evaluation of the degree of fit between model and data should take this into consideration.

11 Model Comparison: Fork Rates

We have now two models which can be used to estimate fork rates during simulated chain growth. We compare the information provided by these models with empirical data.

We are concerned with fork rates because a fork represents, by definition, an instance of reduced finality. Only one block per block number can be ultimately considered canonical; the others will become obsolete (impermanent).

The ambiguity of *which* block shall become canonical is a matter of network, and miner, efficiency. Time and energy spent on impermanent blocks is wasted.

We note further, that the cost of forks are not limited to the two (or miner) beneficiary authors of those blocks. We can consider, as an example, that each of two miners broadcasts their candidates to the network nearly simultaneously. Attributed to network latencies and graph shapes, the network consensus bifurcates and global hashrate is halved (each miner having won-over half of the network). At this point 50% of network hashpower will be ultimately wasted; only one side of the fork will ultimately become canonical. The cost of the ambiguity is borne by all players (though not necessarily fairly distributed).

This assumed state of ambiguity of the candidate blocks is the result of the canonical arbitration algorithm. A canonical arbitration algorithm yielding a lower ambiguity rate will reduce the network's block emission waste and improve expectations of permanence for the overall chain state.

[27] For both practical and theoretical reasons.

11.1 Simulated Fork Rates vs. Empirical Uncle Rates

We know that uncle rates represent a theoretical minimum measurement of actual orphaned block rates; the orphans *may* be recorded but are *not necessarily*.

Can we use an economic model of the incentives of orphan records to suggest to what degree empirical uncle block rates suggest a complete record? If miners are not incentived to record orphans they do not mine themselves, then the recorded orphans will omit those. On the other hand, if it is profitable for the miner to record orphans regardless of their author, then we should expect the record to be more complete.

The revenues from orphan recording are inuitive; they are clearly defined and scaled to the block reward. They are positive. But what are the costs of recording an uncle?

Total network Wei supply is assumed to be finite for the sake of argument.[28]

Since an orphan record benefits the miner of the orphan *more* than it benefits the recording miner, we are lead to reason that it *may* be that the cost of distributing *any* capital to competitors dilutes the value of a miner's own capital.

We leave this line of reasoning open-ended.

For the sake of our proposed model, it is enough to assume an observable objective measurement (uncle rate), and to handle it as theoretical minimum proxy for actual fork rate.

We turn to empirical data.

At the time of writing, Etherscan.io[29] shows $1,207,850$ recorded uncles and a current canonical height of $13,753,436$.

$$1207850/13753436 = 0.0878217 \tag{17}$$

We interpret this as representing an orphan rate of about 8.7%.

Comparatively, using the `main` program we simulate a network with $\lambda = 14$, $\eta = 1.5$, 8 miners with hashrates distributed as an approximate long-tail, at a tick interval representing 100 milliseconds, over $10,000$ samples, and yielding a fork rate of 8.39%.

Raising latency to $\eta = 2$ can raise the rate to 11.7%. Alternatively, raising the number of miners to 16 seems to raise the rate to around 8.7%.

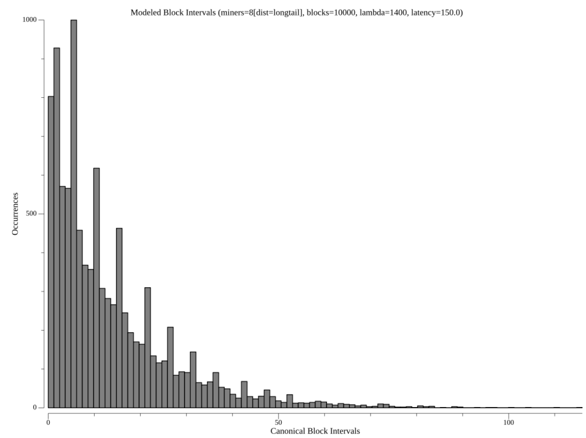
It is tempting to manipulate modeled latency values in order to meet the shape of empirical block interval data. But we must resist this temptation.

[28] This is not a guarantee for Ethereum, though it is for Ethereum Classic.

[29] https://etherscan.io/uncles

We do not know real network latency values, and have no objective way of measuring them. Further, we expect that even reliably measured subjective values would vary and are regularly subject to change. This point is especially pertinent when latency is considered as both mechanically-derived (tube limits) and arbitrarily decided values (eg. selfish head-start delays).

We note, however, that under this simulation, the generated block interval distribution (Figure right) doesn't seem to match the empirical data. The intervals are too small.



If we re-prioritize the fit to a back-of-napkin same-shape rubric on block intervals, a $\eta = 4.2$ approximates a better fit (Figure 25, below). At this latency rate, the model yields a simulated fork rate of approximately 23.5%. A Poisson distribution sampling for the same tick duration is overlaid in red.

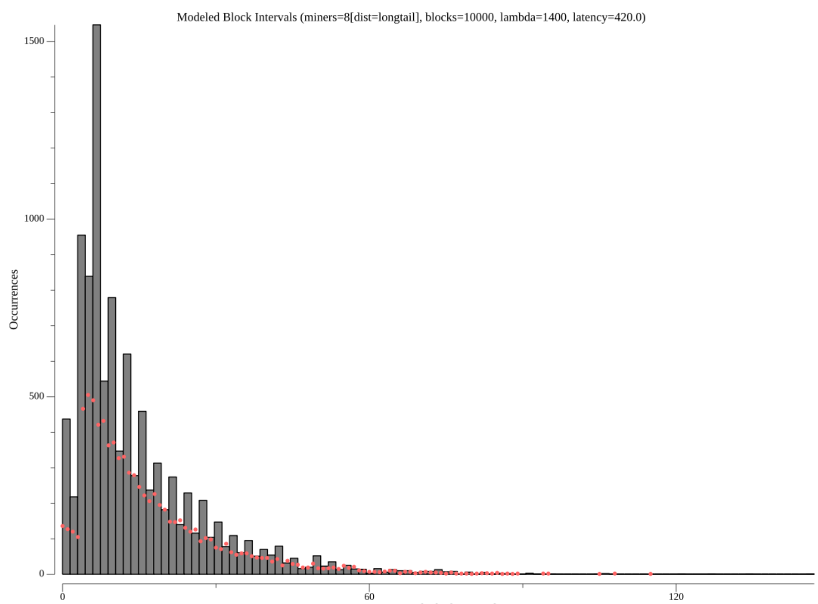


Figure 26: Poisson Distribution interval distribution samples overlaid on simulated intervals.

Our intention is not to build a best-fit model for empirical data.

The intention for the model is to adequately theoretically represent the game which drives the PoW network. With a model that *behaves like* the real world in its key measurements, we can explore the impact of the proposed algorithmic modification to the canonical selection logic.

11.2 Comparing Simulated Fork Rates: Proposed vs. Incumbent Canonical-Selection Algorithms

When an event $k \geq 2$ occurs, network protocol defines a set of rules for determining (deciding/arbitrating) canonical status for exclusively one event (where an 'event' is equivalant to an eligible block).[30]

We compare the incumbent Total Difficulty (TD) condition with the proposed Total Difficulty × TABS (TDTABS) condition by replacing the arbitration logic used by the model in scenarios (block-space rounds) with 2 or more candidate blocks, focusing on the respective rates of 'decidability', where the condition returns an unambiguous choice for the canonical block.

Under the same conditions of the scenario presented in the previous section, we find the following exemplary values:

```
GOROOT=/home/ia/go1.16.2.linux-amd64 #gosetup
GOPATH=/home/ia/go #gosetup
/home/ia/go1.16.2.linux-amd64/bin/go build -o /tmp/GoLand/___go_build_github_co
/tmp/GoLand/___go_build_github_com_whilei_go_poisson_collision
-----------------------------------------------------------------

CONFIG

        Name:              A, ConsensusAlgorithm: TD,
        NetworkLambda:     14, Latency:           1.40,
        TickMultiple:      100, Rounds:           10000,
        NumberOfMiners:    8, HashrateDistType: longtail,

GENERATED MINERS

        Number: 8, Distribution: longtail, Hashrate Checksum OK:
true,
        Hashrates: [0.333 0.222 0.148 0.099 0.066 0.059 0.044 0.02]
```

30 In fact, the protocol design for canonical-selection is applicable to *all* scenarios of canonical-status arbitration between *any* two blocks.

```
INTERVALS

        Mean: 1416.7087,
        Med: 973.0000, Mode: [163.0000],
        Min: 1.0000, Max: 13713.0000,


ELIGIBLE AUTHORS PER BLOCK

        Mean: 1.0793,
        Med: 1.0000, Mode: [1.0000],
        Min: 1.0000, Max: 4.0000,


MINER CANONICAL WINS

        miner=0 hashrate=0.333 winrate=0.344 winrate/hashrate=1.031 (3436)
        miner=1 hashrate=0.222 winrate=0.220 winrate/hashrate=0.990 (2201)
        miner=2 hashrate=0.148 winrate=0.149 winrate/hashrate=1.005 (1489)
        miner=3 hashrate=0.099 winrate=0.099 winrate/hashrate=1.001 (989)
        miner=4 hashrate=0.066 winrate=0.064 winrate/hashrate=0.969 (638)
        miner=5 hashrate=0.059 winrate=0.056 winrate/hashrate=0.948 (555)
        miner=6 hashrate=0.044 winrate=0.043 winrate/hashrate=0.968 (425)
        miner=7 hashrate=0.029 winrate=0.027 winrate/hashrate=0.912 (267)

ANALYSIS

        Ticks: 13056327, Rounds (Blocks): 10000, Ticks/Block: 1305.6327
        AuthorSameParentChildTally/Block: 0.207
        ArbitrationDecisiveRate: 0.039, ArbitrationDecisiveTally: 386
        ArbitrationIndecisiveRate: 0.036, ArbitrationIndecisiveTally: 362


        Elapsed: 2.892s

9305 POISSON INTERVALS

        Mean: 1515.6216,
        Med: 1095.0000, Mode: [231.0000],
        Min: 1.0000, Max: 13049.0000,
```

```
CONFIG

        Name:               A, ConsensusAlgorithm: TDTABS,
        NetworkLambda:      14, Latency:           1.40,
        TickMultiple:       100, Rounds:            10000,
        NumberOfMiners:     8, HashrateDistType: longtail,

GENERATED MINERS

        Number: 8, Distribution: longtail, Hashrate Checksum OK:
true,
        Hashrates: [0.333 0.222 0.148 0.099 0.066 0.059 0.044 0.02]

INTERVALS

        Mean: 1408.3236,
        Med: 973.0000, Mode: [248.0000],
        Min: 1.0000, Max: 18234.0000,


ELIGIBLE AUTHORS PER BLOCK

        Mean: 1.0803,
        Med: 1.0000, Mode: [1.0000],
        Min: 1.0000, Max: 4.0000,


MINER CANONICAL WINS

        miner=0 hashrate=0.333 winrate=0.348 winrate/hashrate=1.044 (3480)
        miner=1 hashrate=0.222 winrate=0.225 winrate/hashrate=1.012 (2248)
        miner=2 hashrate=0.148 winrate=0.148 winrate/hashrate=0.998 (1479)
        miner=3 hashrate=0.099 winrate=0.099 winrate/hashrate=1.005 (993)
        miner=4 hashrate=0.066 winrate=0.063 winrate/hashrate=0.960 (632)
        miner=5 hashrate=0.059 winrate=0.052 winrate/hashrate=0.882 (516)
        miner=6 hashrate=0.044 winrate=0.039 winrate/hashrate=0.882 (387)
        miner=7 hashrate=0.029 winrate=0.026 winrate/hashrate=0.906 (265)
```

ANALYSIS

```
Ticks: 12990256, Rounds (Blocks): 10000, Ticks/Block: 1299.0256
AuthorSameParentChildTally/Block: 0.219
ArbitrationDecisiveRate: 0.051, ArbitrationDecisiveTally: 515
ArbitrationIndecisiveRate: 0.025, ArbitrationIndecisiveTally: 250


Elapsed: 2.776s
```

9233 POISSON INTERVALS

```
Mean: 1515.0680,
Med: 1055.0000, Mode: [141.0000],
Min: 2.0000, Max: 15252.0000,
```

```
Process finished with the exit code 0
```

The TD model shows a decision rate of 3.9%, and an ambiguity rate of 3.6%, where both rates are measured against total rounds.

The TDTABS model shows a decision rate of 5.1%, and an ambiguity rate of 2.5%, where both rates are measured against total rounds.

12 THE REST OF THIS IS UNEDITED CRUFT

Here be dragons. Ye be warned.

12.1 Probability of canonical acceptance of an eligible block by the network

Intuitively (and ideally), this value is 1. Empirically, this value is less than 1.

We attribute this to network latencies and network graph shapes.[31]

Scenario:

Blocks are produced independently and simultaneously by at least two miners. These blocks have equivalent Total Difficulty values and numbers (for the sake of argument). We assume that the network is bifurcated (forked). For sake of argument, we assume 50% of the network (by hashrate, say) is on one side of the fork, and 50% on the other.

Assuming 'good' data availability is maintained for *the whole network*, we expect that the fork to resolve quickly. Probably one side of the fork will produce a block quicker than the other, yielding a subtree with a greater Total Difficulty value than either of the two forks previously. This 2-block chain segment is expected to be adopted as canonical as soon as it is made available to a node. For 50% of the network, this will result in a minus-1/plus-2 block reorg. For the other 50% who were originally on the eventually-winning segment, no reorg is required and the next block is appended. Consensus is reached again.

The general case for the expectation of this resolution is the probability of simultaneous block production in a series of arbitrary length. If we define the probability of simultaneous block production as P_s, then it follows that the probability of a bifurcation enduring n blocks (where n must be greater than 0) be $(P_s)^n$.

Next, we must examine more closely the criteria for what has been called simultaneity. In fact, we mean 'nearly simultaneously', or 'functionally simultaneous'. We can loosely establish the bounds for this window from assumed (or measured) network latency values. Blocks produced independently within some time interval shorter than the involved network latencies (ie. between the two competing miners) will cause each miner to deviate from the optimal protocol (mine the best block available), because while the next best block available will exist at the network level, it will not be available to *them* because of the mechanical limitations of the network.

We have variables defined for this scenario.

[31] Plural forms are used to emphasize that these values are variable. Both values are difficult to measure objectively with a high degree of confidence.

λ represents network mean block time in seconds.

η represents block message latency between two nodes in seconds.

If we assume a Poisson Point Process distribution for the independent miners, what is the probability expected for 2 independent new-block events to occur within an interval η of each other?

12.2 Impact of Proposed Algorithmic Condition on Network Consensus

Conceptually, TABS relies on a fundamental assumption: capital is not distributed evenly. If we hold transaction availability to miners constant, in order for TABS to be effective, we must assume that miner balances are not equal. If all miner balances are equal and transaction/block inclusion is held constant, TDTABS does not modify the expectations of an incumbent TD-only based algorithm.

Assume a waste rate for the network of 5%. The Ethereum network's current uncle rate is 5%. We consider this a minimum value (there could exist unrecorded/unobserved uncles), but all recorded uncles are observable and valid. We assume that uncles are economically undesirable (compared to full blocks), and as such, that their existence is optimized at the lowest level mechanically possible. The occurence of an uncle signals an instant of network consensus undecidability. An uncle represents a fork. An indecisive event signals a lower state finality expectation than a decisive event. It also signals, for the miners of each twin block, a reduced expected revenue for that block (since the probability of impermanence is greater than for an uncontentious block).

We use Digiconomist data from 20211203 claiming 92.42 TWh as Ethereum's annualized energy consumption.[32]

94.42 TWh $\rightarrow MWh = 94420000MWh$

We then cite Wikipedia's[33] data for projected LCOE by 2025 (as of 2020) for US simple average cost for advanced nuclear energy as \$81.65 / MWh.

94420000 MWh $\times\$81.65/MWh \equiv \$7,709,393,000$

If TDTABS reduces network block waste from 5% to 2.5%, that suggests an approximate annual global savings of \$192,734,825.

12.3 When Should Miners Mine on the Next Block?

[32] https://digiconomist.net/ethereum-energy-consumption/

[33] https://en.wikipedia.org/wiki/Cost_of_electricity_by_source

13 Rationale

14 Placeholder

14.1 Transaction Ancestor Hash

14.2 Another Thing

15 Theoretical Analysis

This section provides reasoning around the theoretical implications of the proposed algorithm.

15.1 Network Behavior Analysis

Demonstrate that the 'objectivity' characteristic of consensus-facing information is invariant. The additional observable data required by the proposed algorithm is chain data. It is in the same scope ('context') as data relied on for the existing canonical-preference scheme. All nodes can make the same decision given the same information at any point in time.

Need to show that consensus properties for the network are invariant.

Demonstrate that the incentive to mine the NEXT BEST block (ASAP) is preserved as an invariant.

Consider also: - theoretical additional block processing cost (*more* to do) - necessity of observable chain state (well, only balance) for block validation. Compare with requirement of Ethash for block validation. Compare with other validations, eg. Parent Hash, Number.

16 Game Theoretical Analysis

Show the equation of miner predicted (expected) revenue for some (next) block. Next blocks are uncertain, future events. We can show the probability of some miner winning the next block as the plot of the percentage chance of next-block being found over time, divided (scaled) by their relative hashrate. We show that for t=0..t=n that the expected revenue of the miner goes up.

We consider this plot for the scenario where a miner observes a next block.

16.1 Placeholder

17 Economic Analysis

18 Practical Analysis

This section offers analysis and interpretation of empirical and derived data.

19 General Proposition

We want to improve chain state finality characteristics without compromising other security characteristics of PoW network protocol.

We propose the introduction of segment-specific transactions and an accounting on a representative measure of capital saturation for some segment, yielding a "hybrid" canonical-score value.

We intend to extend the incumbent PoW canon-arbitration protocol to include reasoning about segments' relative "capitalization" rates. The introduction of such a variable can be shown to raise the expected cost of an alternative-history-based attack and to have minimal negative side effects.

We get to to keep, and still deeply rely on, an incumbent PoW emission and consensus pattern.

20 Open Questions

20.1 What if emissions revenues were separated from tx processing revenue?

20.2 Can we show that 3%/97% is a sensible simultaneous-production rate using just the Poisson emission model?

NOTE that we don't assume perfect simultanaeity; the Poisson model forbids this. We set a window we define as "competitive." We characterise this by a heterogenous acceptance of blocks at some number at the network-level; blocks produced near-enough-to-simultaneously to produce a network fork (of some scale; a point which could be additionally investigated: to what scale? 50% partisanship? 0.01%?), e.g. 300 milliseconds.

20.3 Can we show formally that it is game-optimized to mine on the next block immediately (no matter a miner's own anticipated TABS score)?

We assume under the standard Pow/Ethash game, that it is optimal to always mine on the greatest block available. However, we know that this assumption has limits; "selfish" miners with sufficient relative hashrate (greater than 25 percent) can/should be expected to optimize toward their own self interest.

Is the strategy around potential witholding of next-blocks maintained as an invariant? Is the strategy around mining the best-available block (vs. continuing work on a potential own fork) maintained as an invariant?

A miner will be able to calculate their anticipated TABS in advance (before the finalized production of their next block, depending/awaiting an Ethash solution). This leads one to wonder:
- If a miner foresees a "+" TABS block of their own (eg. they have receive in private a large transaction to process, or they are just rich),
- after an interval of 3 seconds from its parent, child block from another miner is broadcast having a "-" TABS value. This creates a block with CS=(2049*127)/(2048*128).
- Is a competitive, "rational" miner incentivized to continue mining for some period of time in the expectation of discovering a block with a comparatively greater CS?
- If they discover a block at the 8 second interval (5 seconds later), their block would have CS=(2049*129)/(2048*128).
- BUT, the competing miners will have had a 5 second "head start" on their *next* block (n+1).

I think this can be solved by finding the relative area under the PDF curve (CDF?) of the Poisson distribution for that time interval (the chance of another block appearing). This value should then be multiplied with the expected value of that next block (in CS units).

My intuition is that the CS value of a "whole" block is enough greater than the marginal differences caused by Difficulty and/or TABS adjustments that time spent toward that whole cookie is more valuable than chasing crumbs.

To demonstrate this, we can show that the CDF (?) of the Poisson distr. for lamda=13 gives a 2% likelihood to a next block at the 1-second interval. This number is totally made up. We show that 0.02 * NextBlockCS is greater than the marginal difference possible with the selfish miners competing block. So the expected value (after some very small amount of time) is higher for our potentially-selfish miner to pursue the next block instead of their own.

We could add realistic complexity by noting that our potentially-selfish miner controls some portion of the network hashrate. Their decision to pursue a selfish block/fork instead of mining the next-best block would remove hashrate from that block's potential child, thus increasing the expected interval to that child. For example, loosely: a miner with 25% network hashpower could (waves hands) extend the expected interval for the next block from a lambda = 9 seconds to 9*1.25 seconds = 12 seconds by pursuing their own next block. I have no idea if this math is reasonable. Would this mean they have a "free" 12-9=3 seconds to spend in pursuit of their selfish block? However, we should also note that the miner, pursuing independent solution, will have only 25% of the network's difficulty/hashrate calibration, and so should expect to see a block production rate averaging 25% of the network; this means expecting a slower block production rate (it will probably take them longer than 13 seconds, eg. 13*4 = 52 seconds).

To account for this last point, we can reuse the CDF logic from above; the selfish miner's expectation for block discovery in the next 1 second is small than that of the network.

$$P(\text{hps}, k) \equiv \text{probability of discovering Ethash solution in } k \text{ seconds given hps hashes per second} \tag{18}$$

- *hps* will be lower for the selfish miner than for the rest of the network (in aggregate)

21 Discussion

21.1 Ancestor Hash vs. Chain ID

Background information has been provided in this paper on Chain ID (see Section 1.4) for the purpose of contextualizing the proposed Ancestor Hash feature. There are clear similarities. In this section we show that the proposed Ancestor Hash feature is an generalized implementation of the existing Chain ID feature.

Chain ID is a single, static, arbitrary scalar value delimiting transactions per "chain". The meaning of *chain* in this context is congruent to the border drawn by ticker symbols. At least, this is the scope of which Chain ID *appears* to have been intended for.

Ancestor Hash proposes a protocol which generalizes the idea of delimiting transactions by chain. The single, static value is swapped for an arbitrary, dynamic value; and in turn, the idea of "chain" moves from a segment marked with a single, hardcoded block (genesisor a fork block), toward an idea that much more accurately represents and describes the dynamic and occasionally ambiguous growth of PoW chains.

The job that Chain ID does could also be done by Ancestor Hash. (Though this paper does not propose replacing it.) Where Chain ID values are controlled centrally, by developers (via client release defaults, usually). Ancestor Hash values are determined by transaction authors (read: chain users).

Chain IDs contain (or *can* contain) very little information compared with Ancestor Hash.

Ancestor Hashes do not force transactions authors to assume anything about the chain that they wouldn't otherwise. Or at least, to make no assumptions or compromises beyond their original, and intuitive, models.

The values that can be annotated are arbitrary. An Ancestor Hash specifying block 1.912.489 (TODO GET THE HASH) on ETH would be *more* ambiguous than Chain ID. On the other hand, an annotation citing their latest seen block represents the maximum expression of confidence (or dependence) on a specific chain chronology.

It is a way for transaction authors to declare a meaningful confidence interval (like exchanges do with confirmation delays). The delay enforced by the exchanges is an expression of doubt; the delay expresses by the transactor is an expression of confidence. (Exploring the thought experiment further: What HFC.Hash values would exchanges "like" to see on ETH for token deposits? Short ones; near the head. The nearer the transaction dependency toward the chain head, the more likely the deposit is to become *invalid* in the event of a reorganization. The denser the dependence on the chain head becomes... more gravity, more finality.)

By expressing a dependency on a specific block, transactions implicitly *also* express dependency on global chain state, and the transactions that drive it.

Another thought experiment:

I sell all my Bitcoin because Elon sold his last block. Then, in a twist of fate (is there such a thing as coincidence?), a two block reorg happens; removing Elon's sell, but keeping mine. With HFC.Hash, I could annotate the hash of the block with Elon's transaction in it. With that specific block missing from the reorg'd replacement block, my transaction won't be valid, and I won't have sold my Bitcoin like a sucker.

It can be argued that Ancestor Hashes give more control to the transaction authors in regard: transaction placement and assumptions about state.

We take it for granted that transactors make assumptions about chain state. When you send some ETH, it's not crazy to check your balance first. To check your transaction history; to make sure you're on the right nonce. These are assumptions about chain state.

Ancestor Hash transactions allow you to express these assumptions. To write them into code.

21.2 The Necessity of Ancestor Hash to the Proposed Canonical-Arbitration Algorithm

In the scenario of a malicious chain state-modifying 'attack', an attacker exploits an assumption by a counterparty on chain state finality.

The fraud strategy is based on the eventual censorship of one or more transactions on which a reciprocal exchange depends.

We assume that an attacker should only censor transactions on which the attack depends.

Spurious censorship beyond the interests of the attack results in risky collateral damage (ie. an increased risk of counter-attack).

21.3 Ancestor Hash: Database Availability Impact

Theoretically, the utilization of Ancestor Hash negatively impacts the 'availability' of the chain state database to pending transactions.

In other words, this reduces the chain's transactional 'throughput'.

We argue that the extent to which database availability is restricted is not necessarily undesirable. The marginal availability constraint proposed, we argue, aligns with reasonable assumptions by a transactor about current chain state, and that the reduction in state database availability reflects potentially undesirable/spurious transaction inclusion in alternative and unobserved chain state histories.

21.4 Will miners include as many transactions per block as possible? (Is network transaction throughput optimized/invariant?)

- Assume "full blocks" (more pending transactions than available block space).
- Assume transactions come from senders with positive balances.
- Don't consider: variable balances for transaction senders.

The more transactions in a block, the greater the $TABS$ value. The greater the $TABS$ value, the greater the chance the miner authors a canonical block. The more canonical blocks a miner authors, the more money they make.

QED There exists incentive for miners to include as many transactions as possible for each block they mine. NOTE as an *invariant* with original *Modified-GHOST* protocol.

21.5 Will miners prefer transactions from senders with relatively greater balances?

Yes. But transaction gas fees and costs need to be considered simultaneously (or at least not forgotten). That is, until $B_{iTAB} > B_{i-1k}$; then it will prefer transaction gas profit, an *invariant.*

21.6 Will miners be incentivized to raise the block gas limit indefinitely?

- Assume (or remember) that given sufficient transaction volume for equilibrium fee competition, miners are incentivized to include as many transactions in a given block as possible in order to maximize gas fee revenues by raising transaction volume.
- Assume that the existing network's GasLimit has not risen indefinitely.
- Assume there exists some reasonable reason for this, like block rationing processing time

and energy, network latency optimization, social norms, mechanical ignorance of or insufficient means of execution (don't know to code it, or don't know how to code), unexplained or unfamiliar (to your authors) distributed decision-making rationales etc.

Given these assumptions, we consider only the game theoretical impacts of additional transaction inclusion in regard to $TABS$ and the role of that value in the proposed segment preference algorithm.

Point 1. GasLimit adjustment bounds can be insufficient to achieve the case.

This is because the minimum cost of a transaction is 23000 Wei. The current GasLimit is 10.000.000 Wei. The GasLimit adjustment algorithm permits the change of the incumbent (parent) GasLimit header value by +/- $B_{iGasLimit}$/1024. 10.000.000/1024 = 9765.63. 9765.63 $\langle$ 23000. If we assume an initial upward march of the GasLimit (at B_i), then before arriving at block B_{i+3} we shall encounter a block with GasLimit y ¡ parent.GasLimit - (parent.GasLimit/1024), at which point a further increase to the GasLimit shall not be sufficient to permit the inclusion of another transaction. A raise at this instant does not bring any immediate benefit or future advantage for the block author. In some cases, a raise could benefit the competiton. The game equilibrium becomes stasis.

Point 1/Counter 1. GasUsed values are composites of diverse sets of transactions. It is theoretically possible for GasLimit (at any adjustment value/rate) to be sufficient to build a case where the GasLimit should be expected to be driven indefinitely higher.

For example:

Block.Parent.GasLimit = 10.000.000
Block.Transactions.GasUsed = 9.985.900

Extra pending TX GasCost = 23000

If Block.GasLimit is raised to 10.009.765, the "Extra" transaction will be eligible for inclusion (total GasUsed 10.008.900). Scenarios like this can be built with arbitrary transaction gas costs (eg. EVM use). However, practical fabrication of scenarios like this involve centralized dominance of the pending transaction market (ie. by paying exhorbitant gas prices, by controlling miners, by suppressing entry to the public transaction pool, etc.).

If we grant Point 1/Counter 1, are the remaining unconsidered conditions also sufficient?

(Before we dive into this, we should also remember that an additional, adjacent, protocol change could be introduced capping the valid GasLimit, or capping its max adjustment value. Either or both solutions would mitigate the risk here, and could do so without noticeable effects on the status quo.)

—

It is at this point we need to consider the block production and selection business of $TABS$. How does $TABS$ shape consensus? What's its impact on segment selection?

The TAB value for any given block is – from a block *production* perspective – "incidental." Blocks may (still) be produced with no transactions, and with the block miner having a 0 balance. This is an invariant. Also invariant is that block validity depends on a unique, verifiable Ethash solution; the design and implementation of which remains unchanged. Nakamoto assumes that an optimized game sees honest, minority miners mining immediately on any next best block available to them.

We expect about 97% of blocks produced to be distributed and adopted without any observable contention. TODO CITE my experimental design and evidence. Supplementarily, we expect about 3% of blocks to be contentious (to some degree at the network level). Uncle rates typically hover between 3-6%. Rates in excess of 3% we consider to be driven by what this paper considers externalities:

- selfish mining behavior,
- variable network latencies,
- program processing time and/or priorities (eg. between various client implementations),
- and potentially variable economic conditions (eg. uncle "rewards," which once-upon-a-time made uncles nearly (if not actually) more profitable than canonical blocks themselves on Ethereum).

NOTE that GHOST (and later, *Modified-GHOST*) specifies difficulty-accounting for recorded uncles.

THIS IS SUPER IMPORTANT ¿¿ NEED TO MAKE SURE ITS WRITTEN REEAALLL GOOOD ¿

We reason that in 97% of blocks, the consensus-driving factor will remain the discovery of an Ethash solution and the prompt composition and broadcast of that solution. In these cases, the impact of the $TABS$ value in the TCP product is ineffectual; a "non-uncling" block discovered in the network mode of 8 seconds will see its adoption as canonical regardless of its $TABS$ score; just by virtue of being first to the solve.

SIDENOTE. As such, we expect the impact of the $TABS \times \mathrm{H}_d$ scoring system on overall block production rates to be small.

In the 3% of cases where honest miners generate simultaneous solutions:

- Assume the usual case where the competition is bilateral (not tri-, or quad-, etc.).
- Assume blocks have equivalent objective consensus-algorithm scoring: total difficulty, block number.

- Assume these two miners each convince 50% (by hashrate) of the network to their fork per their network connection graphs.

In these cases, a figurative coin is tossed. Each miner wins 50% of the time. We see that miners see a 98.5% "efficiency" rate against their purely theoretical, independently-considered hashrate. Another way of saying this: the network operates at 98.5% efficiency measured as time and energy spent vs. the "ideal" market value of that work. 1.5% is lost to the confusion of a competitive, decentralized network; the ethereal (and theoretical) waste heat of the network.

$TABS$ will have an impact on the outcomes of these 3% of blocks. Experimentally, $TABS$ values will supercede what otherwise would become a coin toss between 50% and 80% of the time. Why?

- Assume 50% miner hashrate has less, and 50% more, relative to the instantaneous network $TABS$ value (at some public chain HEAD (or current state)).
- Assume 0% frequency of an producing a block with a $TABS$ score equivalent to its parent. (If/when we relax this assumption, we'll see that the undecideability rate returns upwards toward that of the proposed algorithm's predecessor.)

In these cases:

- Miners each have a 50% chance of a "high" $TABS$, and likewise a 50% chance of a "low" score.

What is the chance of H:L OR L:H (AND NOT H:H OR L:L)? 50%! 2 of these 4 possible cases are decisive; 2 are indecisive.

And so, in the "flattest" (least colorful, varied... most general) case, overall miner "efficiency" will increase by 0.5 * 0.5 * 3 = 0.75% for "rich" miners, decreasing by the same amount for "poor" miners. The theoretical network inefficiency (undecidability) rate will decrease by approximately half.

NOTE FOR ALL OF THIS: An uncle rate (and the waste-heat theory) is largely a function of block emission rate. Faster blocks yield more confusion, slower emission rates see less frequent simultaneous emission events by virtue of a wider range of discrete emission times (eg. 1-99 seconds on ETH vs. 480-900 seconds on BTC). (A Poisson model for emission intervals is assumed.)

21.7 Hashes vs. ETH: Unit Cancellation in Total Canon Scoring

This section shows in math how taking the product of the Difficulty and TABS values yields results that cancel the units (hashes and ETH, respectively). This is important because there's no good way interchange hashes and token values.

—

In reality, it is likely that many of the characteristics of this general scenario are less homogenous than assumed. For example:
- Mining hashpower is not evenly distributed.
- Capital (ie. tokens) are not evenly distributed among neither miners nor transaction senders.

—

Dad's comments/questions from Thanksgiving:

Difficulty does not have units. It is just an integer.

Rounding errors? Random walk re: TABS value algo, +/- 1/128 – does this tend to median? Is it a random walk? Why 128?

—