TRUECHAIN: HIGHLY PERFORMANT DECENTRALIZED PUBLIC LEDGER
FIRST DRAFT
WORK IN PROGRESS

ARCHIT SHARMA[1], JASPER L[2], HAN ZHANG[3], ERIC ZHANG[3],

1 ARCHIT@PM.ME
2 JASPER@TRUECHAIN.PRO
3 HAN.ZHANG2@GMAIL.COM
4 ERIC@TRUECHAIN.PRO

ABSTRACT. In this paper we present the initial design of truechain consensus protocol and other technical details. Briefly, our consensus design enjoys the same consistency, liveness, transaction finality and security guarantee, a de-facto with the Hybrid Consensus. We discuss optimizations like frequency of rotating committee members and physical timestamp restrictions. We go on to propose the idea of a new virtual machine on top of Ethereum which adds permissioned-chain based transaction processing capabilities in a permissionless setting. We also use the idea of data sharding and speculative transactions, evaluation of running of smart contracts in a hybrid cloud infrastructure and usage of existing volunteer computing protocols for something we introduce as a compensation infrastructure.

In the next version of this Yellow Paper, we will address some of these properties formally along with few of the future directions listed at the end of the paper.

1. INTRODUCTION

With the surging popularity of cryptocurrencies, blockchain technology has caught attention from both industry and academia. One can think blockchain as a shared computing environment involving peers to join and quit freely, with the premis for a commonly agreed consensus protocol. The decentralized nature of blockchain, together with transaction transparency, autonomy, immutability, are critical to crypocurrencies, drawing the baseline for such systems. However top earlier-designed cryptocurrencies, such as Bitcoin[21] and Ethereum[11], have been widely recognised unscalable in terms of transaction rate and are not economically viable as they require severe energy consumptions and computation power.

With the demand of apps and platforms using public blockchain growing in real world, a viable protocol that enables higher transaction rates is a main focus on new systems. For example, consider a generic public chain that could host computationally intensive peer to peer gaming applications with a very large user base. In such a chain, if it also hosts smart contracts for Initial Coin Offerings in addition to a Digital Advertisement applications, we could easily expect a huge delay in transaction confirmation times.

There are other models like delegated mechanism of Proof of Stake and Permissioned Byzantine Fault Tolerant (BFT) protocols. The BFT protocol ensures safety as long as only one third of the actors in the system are intentionally/unintentionally malacious adversaries, at a time. This is a really good mechanism, however a BFT chain alone has a problem with scalability and pseudo-decentralization. The Proof of Stake protocol with a small number of validators could although facilitate high throughput, but the system in itself is highly dependent on a few stakeholders to make the decisions on inclusion and exclusion of delegates. Moreover, there is no transparency without Merkel trees and this type of system could always suffer from nothing-at-stake paradox.

In this Paper, we propose TrueChain, a Hybrid Protocol[24] which incorporates a modified form of PBFT (Practical Byzantine Fault Tolerance)[13] and POW (Proof of Work) consensus. The POW consensus ensures incentivization and committee selection while the PBFT layer acts as a highly performant consensus with capabilities like instant finality with high throughput, transaction validation, rotating committee for fair trade economy and a compensation infrastructure to deal with non-uniform infrastructure. The nature of hybrid protocol allows it to tolerate corruptions at a maximum of about one third of peer nodes.

2. BACKGROUND

The core strength of this proposal lies in the recognition of the theorems proposed in the Hybrid Protocol[24] by Pass and Shi. We benefit from the fact that there is a lot of design space for further optimizations in that paper. The use of DailyBFT as committee members allows for the rotating committee feature which provides for better fairness for the consensus validating peers.

The POW nodes could benefit from incentivization infrastructure while they are also a part of the slower snailchain, helping deploy smart contracts.

Date: April 28th, 2018.

2.1. **Related Works.** Hybrid Consensus follows a design paradigm where BFT and PoW are combined together so that it enjoys best of both worlds. In general, Hybrid Consensus will utilize BFT protocols, which by default works in a permissioned setting where all the identities are known a priori, as a fast path dealing with large amount of incoming transactions. While PoW protocols choose the BFT committee based on a combination of csize (number of blocks mined) and of stake_in / stake_out. This provides the barebone necessary to deal with dynamic membership and committee switching in the permissionless setting.

3. Consensus

Our consensus design is largely based on Hybrid Consensus by Pass and Shi [24], with several modifications and improvements in order to tailor for the application scenarios that we focus on. In this section we will assume the readers are familiar with every detail of Hybrid Consensus protocol.

3.1. **Design Overview.** In this subsection we will present an overview of our consensus protocol. In this protocol, we use the same abstract symbols and definitions in [24]. In the following part of this section, we will explain our modifications and further constructions on top of Hybrid Consensus.

Our adversary model follows the assumptions in [24] where adversaries are allowed to mildly adaptively corrupt any node, while corruptions do not take effect immediately. In the next version of Yellow Paper we will formally explain our modifications in Universal Composabililty model [12].

Note that all the pseudocodes in this Yellow Paper are simplified for the sake of easy explanations. They are not optimized for engineering.

3.2. **Recap of Hybrid Consensus Protocol.** In this subsection, we recall major components and definitions from the Hybrid Consensus protocol.

3.2.1. *Fruitchains as opposed to Nakamoto chain.* We follow the adoption of Fruitchain instead of Nakamoto (traditional chain) for the slowchain (snailchain), so as to defend against $1/3 - \varepsilon$ corruption (in hashpower) for a random small constant ε, to obtain an optimal resilience.

3.2.2. *Daily offchain consensus protocol.* In DailyBFT, committee members run an offchain BFT instance to decide a daily log, whereas non-members count signatures from committee members. It extends security to committee non-members and late-spawning nodes. It carries with it, a termination agreement which requires that all honest nodes agree on the same final log upon termination. In DailyBFT, committee members output signed daily log hashes, which are then consumed by the Hybrid Consensus protocol. These signed daily log hashes satisfy completeness and unforgeability.

On keygen, add public key to list of keys. On receiving a comm signal, a conditional election of the node as committee member happens. The environment opens up the committee selectively.

Here is how the subprotocol works for when the **node is a BFT member**:- A BFT virtual node is then forked. BFT virtual node here, denoted by BFT_pk, then starts receiving the TXs (transactions). The log completion is checked and stopped iff the stop signal has been signed off by atleast a third of the initial comm distinct public keys. During this, a continuous “Until done” check happens and once completion of gossip happens at each step, all the stop log entries are removed

Here is how the subprotocol works for when the **node is not a BFT member**:- On receival of a transaction, the message is added to history and signed by a third of the initial comm distinct public keys

The signing algorithm tags each message for the inner BFT instance with the prefix 0, and each message for the outer DailyBFT with the prefix 1 to avoid namespace collision.

3.2.3. *The mempool subprotocol.* Initializes TXs with 0 and keeps a track of incoming transactions with a Union set. On receiving a propose call, it adds the transaction to log and communciates with gossip protocol. It also supports query method to return confirmed transactions. By keeping track of transactions in a set, it purges the ones already confirmed.

3.2.4. *Main Hybrid Consensus protocol.* A newly spawned node with an implicit message routing that carries with it history of the transcripts sent and received. This interacts with the following components - Mempools, Snailchain, Preprocess, Daily Offchain Consensus, and on chain validation.

3.3. **Variant Day Length and Hybrid Committee Election.** In [24], BFT committee instances are switched after a fixed period of time (with the snailchain as a logical clock). And new committee is formed simply by the miners of the latest $csize$ number of blocks inside snailchain. In our consensus design, we want to exploit the intuition that, if the committee behaves well, we don’t have to force them to switch, and therefore the overhead of switching committee could be prevented in some situtations. On the other hand, this will raise difficulty for new nodes to get elected as a committee member if the previous committee keeps good records. Therefore we still keep the design of forcibly switching the committee every fixed amount of time, but with a much lower frequency, (for example, the committee will be switched every K days). On the other hand, we incorporate the idea of authenticated complaints from Thunderella [26] where the slowchain can be used as the evidence of misbehaviour by BFT committee members. That is, whenever a committee misbehavior is detected from the snailchain, the next day starting point (not necessarily the K-th day) will trigger a forced switch.

More over, we will replace the committee election criterion. In Hybrid Consensus, committee members are chosen from the miners of the most recent snailchain blocks. We decide to select committee members based on a hybrid criterion by stakes and randomness. To be more specific, we allow full nodes to propose special $stake_in$ and $stake_out$ transactions to temporarily freeze their token assets. And whenever a committee switch happens, we will elect $\theta \cdot \mathsf{csize}$ accounts based on their frozen stakes where $\theta \in [0, 1]$ is a manual parameter. And following the design of [16], we choose the rest $(1 - \theta) \cdot \mathsf{csize}$ nodes based on the result of VRF [20] where the seed is determined by the seed used in previous committee selection, as well as the proposed randomness from recent csize blocks. Different from Algorand [16], here we don't count stake weights for this part of selection.

Notice that the nodes that are chosen by random functions would have a high probability of not being online. For this reason, given the estimated online rate r_{on}, since we don't want the offline nodes to take the Byzantine quota, we need to make sure $r_{on} \cdot \theta < \frac{f}{\mathsf{csize}}$. Usually we take $\theta = \frac{f}{2r_{on}\mathsf{csize}}$. Another potential design alternative is that we can allow offline nodes by design [25].

Note that a party in charge of overwhelming computation resources will be likely to manipulate the input of VRF, but that already goes beyond our security assumption.

3.4. **Application Specific Design.** Our consensus design is aware of application specific requirements and tailors for them, under the conditions that the consistency, liveness and security properties are not compromised.

3.4.1. *Physical Timing Restriction.* Conventional consensus design by default allows miners / committee members / leaders to re-order transactions within a small timing window. This raises a problem for some decentralized applications such as commercial exchanges where the trading fairness requires the timing order between transactions to be carefully preserved, or otherwise malicious (or, even normal rational) participants will have the incentive to re-order transactions, or even insert its own transactions, to gain extra profits. And this incentive will be magnified under high throughputs.

And what is even worse, is that such malicious re-ordering is impossible to distinguish because naturally network latency will cause re-ordering and such latencies can only be observed by the receiver itself and therefore it has the final evidence of numbers regarding network latency.

To support decentralized advertisement exchanges, we try to reduce such problems by incoporating one more restriction called sticky timestamp. More specifically, with a heuristic parameter T_Δ, when proposing transactions, we require the client to put a physical timestamp T_p inside the metadata of the transaction, and this physical timestamp is signed together with the other parts of the transaction. Later when validators inside BFT verify the transaction, it will do the following extra checks as shown in Algorithm 1.

At the stage of materializing logs inside BFT, the leader will sort the transaction batch according to its physical timestamps and break ties (though very unlikely) with the sequence number. Actually this step is not necessary because we can enforce the order later in the evaluation and verification. But for simplicity, we put it here.

This set of modications give us several extra properties:

(1) The order of transactions from any node N_i is internally preserved according to their physical timestamps. Thus the sequence order of these transactions is strictly enforced. This will get rid of the possibility of some malicious re-ordering that involves two transcations from the same node.
(2) The order within a batch of transactions output by the BFT committee is strictly ordered by timestamps.
(3) Nodes cannot manipulate fake physical timestamps because of the timing window restriction.

One obvious disadvantage of this modificaion will be the reduction in terms of throughput due to aborting transactions when the parameter T_Δ is inappropriate for the varying network latency. Another disadvantage is that, the BFT committee members are still allowed to lie about their local time and reject certain transactions. However, committee members can reject certain transactions anyway. But honest nodes could potentially reject ignorant transactions because of their unsynchronzied clocks. This issue can be reduced by adding restrictions on the eligibilities of the BFT committee. Later we will see that to get into the committee, the nodes should present evidence of synchronized clocks.

3.5. **Computation and Data Sharding, and Speculative Transaction Execution.** In this subsection we introduce our sharding scheme.

An important modification over the original Hybrid Consensus is that we add computation and data sharding support for it. And even more, first of its kind, we design a speculative transaction processing system over shards. The idea is clear. In Hybrid Consensus, the DailyBFT instances are indexed into a determininstic sequence $\mathsf{DailyBFT}[1 \ldots \mathsf{R}]$. We allow multiple sequences of DailyBFT instances to exist at the same time. To be precise, we denote the t-th DailyBFT sequence by shard S_t. For simplicity, we fix the number of shards as C. Each DailyBFT is a normal shard. Besides C normal shards, we have a primary shard S_p composed of csize nodes. The job of the primary shard is to finalize the ordering of the output of normal shards as well as implementing the coordinator in distributed transaction processing systems. And the normal shards, instead of directly connecting with Hybrid Consensus component, submit logs to the primary shard, which in turn talks to Hybrid Consensus.

We don't allow any two shards (either normal or primary) to share common nodes, which can be enforced in the committee selection procedure. The election of multiple shards is similar to the election procedure described in Section 3.3.

We partition the state data (in terms of account range) uniformly into C shards. This will make sure that every query to the corresponding shard will return a consistent state. Since we are going to include meta data for each data unit, we split data into units of data sectors and assign each data sector with an address. We have a mapping from data position to data sector address. For simplicity, from now on, we only discuss at the level of data sectors. Each data sector $\mathsf{DS}[\mathsf{addr}]$ has metadata of rts, wts, readers, writers.

```
Algorithm 1: Extra Verification Regarding Physcal Timestamp
    Data: Input Transaction TX
    Result: A Boolean value that indicates whether the verification is passed
    current_time ← Time.Now();
    if $|current\_time - \mathrm{TX.T_p}| > T_\triangle$ then
        return false;
        // if the time skew is too large, reject TX.
    var txn_history = new static dictionary of lists;
    if txn_history[TX.from] == NULL then
        txn_history[TX.from] == [TX];
    else
        if $\mathrm{txn\_history[TX.from][-1].T_p - TX.T_p} > 0$ then
            return false;
            // To make sure the transactions from the same node preserve timing order.
        else
            txn_history[TX.from].append(TX);
            return true;
```

FIGURE 1. Pseudo-Code for Extra Verification

We assume the partition principle is public and given the address addr we can get its host shard by calling the function host(addr).

Notice that if we treat every normal shard (when the number of adversaries is not large) as a distributed processing unit, we can incorporate the design of logical timestamps [31] in distributed transaction processing systems [19], which will empower the processing of transactions. Here we utilized a simplified version of MaaT where we don't do auto-adjustment of other transaction's timestamps.

For normal shards, it acts exactly as described in DailyBFT except the following changes to make it compatible for parallel speculative execution.

For the primary shard, it collects output from all the normal shards. Notice that, the data dependency of transactions can be easily inferred by their metadata. And a fact is that, if a transaction visits multiple remote shards, it will leave traces in all the shards involved. When a normal shard submit logs to the primary shard, it will also write to the snailchain.

When the primary shard receives (or fetchs from the snailchain) a batch of txns from a shard, it will check if it has received from all the shards transactions within this batch. If after certain timeout it has not received transactions from a particular batch, it means that batch has failed. In this case, a whole committee switch will be triggered at the next day starting point. After receiving all the shards' logs, the primary shard sorts the transactions based on their commit timestamps (if some transaction has earlier batch number, it will be considered as the first key in the sorting, however, if its physical timestamp violates the timestamps from many shards, we decide that batch as invalid and all the transactions inside that batch are aborted). After sorting, the primary shard filters all the transactions and keeps a longest non-decreasing sequence in terms of physical timestamps. Out the log to the Hybrid Consensus component as that day's log.

There are still many optimisation spaces. One certain con is that the confirmation time in this design is not instant.

4. Smart Contracts in Virtual Machines

4.1. **Design Rationale.** Of all the reasons to have an Ethereum Virtual Machine (EVM) [30], one of aims is to meter the usage with a transaction fee in a Proof of Work model. Since ours is a hybrid model, we'll take the liberty of exploring this design space a little bit further. Let us consider the possibility of a hybrid cloud ecosystem.

A basic problem people have faced is the kind of crude mathematical notations followed in Ethereum's Yellow Paper [30]. We therefore hope to follow something like KEVM jellopaper [17] to list out the EVM and TVM (described in 4.2) specifications. And in future, we hope to maintain our own specifications through Truechain's github account (https://github.com/truechain).

4.1.1. *What about containers instead of VMs?* One of the blockchain frameworks out there that come as close to this idea as possible, is Hyperledger's Fabric framework [9]. If one sets out to convert Fabric's permissioned nature into permissionless, one of the foremost challenges would be to solve the chaincode issue. What this means is while it's possible to keep a chaincode/smart contract in a single container, that is not a scalable model for a public chain. Having such a model for public chain means having to run several thousand containers, per se, several thousand smart contracts on a single node (because each node maintains a copy).

There have been attempts from the community on being able to run a certain maximum containers per node. The limit

```
Algorithm 2: Sharding and Speculative Transaction Processing
On BecomeShard:
  Initialize all the state data sectors: lastReaderTS = −1, lastWriterTS = −1, readers = [], writers = []
With transaction TX on shard $S_i$:
On Initialization:
   TX.lowerBound = 0;
   TX.upperBound = $+\infty$;
   TX.state = RUNNING;
   TX.before = [];
   TX.after = [];
   TX.ID = rand;
On Read Address(addr):
if host(addr) == $S_i$ then
    Send readRemote(addr) to itself;
else
    Broadcast readRemote(addr, TX.id) to host(addr);
    Async wait for $2f + 1$ valid signed replies within timeout $T_o$;
    Abort TX when the timeout ticks;
Let val, wts, IDs be the majority reply;
TX.before.append(IDs);
TX.lowerBound = max(TX.lowerBound, wts);
return val;
On Write Address(addr):
if host(addr) == $S_i$ then
    Send writeRemote(addr) to itself;
else
    Broadcast writeRemote(addr, TX.id) to host(addr);
    Async wait for $2f + 1$ valid signed replies within timeout $T_o$;
    Abort TX when the timeout ticks.
Let rts, IDs be the majority reply;
TX.after.append(IDs) TX.lowerBound = max(TX.lowerBound, rts);
return;
On Finish Execution: for every TX′ inTX.before do
    TX.lowerBound = max(TX.lowerBound, TX'.upperBound);
for every TX′ inTX.after do
    TX.upperBound = min(TX.upperBound, TX'.lowerBound);
if TX.lowerBound ¿ TX.upperBound then
    Abort TX;
Broadcast $Precommit(TX.ID, \lfloor \frac{TX.lowerBound+TX.upperBound}{2} \rfloor)$ to all the previous remote shards which TX has accessed;
// If TX.upperBound = ∞, we can set an arbitrary number larger than TX.lowerBound.
On receive readRemote(addr, ID):
if host(addr) == $S_i$ then
    DS[addr].readers.append(ID);
    return DS[addr].value, DS[addr].wts, DS[addr].writers;
else
    Ignore
On receive writeRemote(addr, ID):
if host(addr) == $S_i$ then
    DS[addr].writers.append(ID);
    Write to a local copy;
    return DS[addr].rts, DS[addr].readers;
else
    Ignore
```

FIGURE 2. Pseudo-Code for Sharding and Speculative Transaction Processing

Algorithm 3: Sharding and Speculative Transaction Processing (cont.)

```
On receive Precommit(ID, cts):
Look up TX by ID;
if Found and cts not in [TX.lowerBound, TX.upperBound] then
    Broadcast $Abort(ID)$ to the sender's shard.;
TX.lowerBound = TX.upperBound = cts;
For every data sector $DS[addr]$ TX reads, set $DS[addr].rts = max(DS[addr].rts, cts)$;
For every data sector $DS[addr]$ TX writes, set $DS[addr].wts = max(DS[addr].wts, cts)$;
Broadcast $Commit(ID, batchCounter)$ to the sender's shard.;
// batchCounter is a number which increases by 1 whenever the shard submit a batch of log to the primary
   shard.
On receive $2f + 1$ Commit(ID, batchCounter) from each remote shards which $TX$ has accessed:
TX.lowerBound = TX.upperBound = cts;
For every data sector $DS[addr]$ TX reads, set $DS[addr].rts = max(DS[addr].rts, cts)$;
For every data sector $DS[addr]$ TX writes, set $DS[addr].wts = max(DS[addr].wts, cts)$;
Mark $TX$ committed;
Let $TX.metadata = [ShardID, batchCounter]$;
On output log
Sort $TX$'s based on their $cts$. Break ties by physical timestamp.
```

currently is 100 pods per node, per se, approximately 250 containers per node, as illustrated in Kubernetes container orchestration platform [5] and Red Hat's Openshift Container Platform 3.9's Cluster Limits [7]. Even with latest storage techniques like brick multiplexing [1], the max possible value (say $\mathsf{MAX\_CONTR}$) of containers could not possibly reach (at least right now) 1000. This issue could further be looked up in the discussions on kubernetes issues github page [4] around workload-specific limits that usually determine the maximum pods per node. People who wish to scale containers usually prefer horizontal scaling rather than a vertical scaleup [2, 6], as the latter significantly increases complexity of design decisions. And there's no one-size-fits-them-all rule for a cluster scale configuration as that entirely depends on the workload, which being more in our case due to its decentralized nature, isn't very convincing for taking a step towards scaling this. At this point, it becomes more of an innovation problem than a simple technical specification search. Ethereum currently has > 1000 smart contracts deployed. Therefore this would be nothing but a crude attempt at optimizing the container ecosystem's design space.

Now let us expand a bit on the container scenario. Given the above crisis, a possible solution is to use container in a serverless architecture. But consider a scenario where > 2000 contracts are online and the concurrent requests, i.e., invocation calls to chaincode (a moving window) at a time exceed $\mathsf{MAX\_CONTR}$ value, we then face the same problem all over again. Therefore, it is only advisable to add a throttling rate limit on the max concurrent requests. This severly limits the Transactions Per Second from the consensus, by design. Engineering should not be a bottleneck to what could be achievable alternatively. Therefore, we choose to stick to EVM design, although a bit modified for our purpose.

4.2. **Truechain Virtual Machine (TVM).** A typical example in this space would be that of the Ethereum Virtual Machine (EVM) [30], which tries to follow total determinism, is completely optimized and is as simple as it gets, to make incentivization a simple step to calculate. It also supports various features like off-stack storage of memory, contract delegation and inter-invocation value storage.

We would reuse the EVM specifications for the snailchain, but add a new specification for TVM in the next version of this Yellow Paper, after careful consideration of the design rationale similar to EVM, deriving the stack based architecture utilizing the Keccak-256 hashing technique and the Elliptic-curve cryptography (ECC) approach.

The Truechain infrastructure will utilize a combination of EVM and another EVM-like bytecode execution platform for launching smart contracts. We choose to use one VM for POW and another for PBFT, embedded within each full node, so they could manage invocation calls on per-need basis.

The TVM backs the DailyBFT powered chains, which interact with the following components:

- re-using some of the concepts from tendermint, like the ABCI (Application BlockChain Interface) which offers an abstraction level as means to enable a consensus engine running in one process to manage an application state running in another.
- A different consensus engine pertaining to dailyBFT chain,
- A permissioned Ethereum Virtual Machine
- An RPC gateway, which guarantees (in a partially asynchronous network) transaction finality

#TODO - formally define transition states of TVM, smart contracts deployment strategy and a way to deploy permissioned VM onto a permissionless chain.

#TODO - define parameter to switch between the POW and the full node (POW and PBFT).

5. Blocks, State and Transactions

# TODO - Talk about changes to blocks, world state flow, transactions and execution model

6. Incentive Design

The Proof of work protocol have a proven track record of attracting computational resources at an unprecedented rate. While existing PoW networks such as bitcoin and ethereum have been successful in their own right, the computational resources they attracted have been nothing more than very powerful hash calculators. They cost a lot of electricity to run, and produce nothing useful.

In this section we will present a concept of compensation infrastructure in order to balance the workload of BFT committee members and non-member full nodes. We invented a new incentive design for PoW, where participating resources can be redirected to do useful things, such as scaling transactions per second (referred to as “TPS“ from hereon), and providing on-chain data storage.

Ethereum gas price is determined by a spot market with no possibility of arbitrage, similar to that of electricity spot market studied in [28]. We consider this market to be incomplete, and therefore fundamental theorem of asset pricing does not apply[14]. Thus, the underlying gas price will follow a shot-noise process, that is known for its high volatility. We introduce a “gas market place” where gas will be traded as futures, and this market is complete in the infinitesimal limit. This is expected to significantly reduce gas price volatility compared to Ethereum.

The following subsections will talking about each component of the incentive design in detail.

6.1. **Fruitchain as snailchain.** The greatest challenge that PoW based consensus faces today are efficiency and scalability. Slow execution and confirmation time make it unfit to develop complex applications, and excessive energy consumption make it environmentally unfriendly. The protocol we propose use a hybrid of PBFT and PoW as core consensus. Unlike Ethereum, where transactions and smart contracts are executed by every node on the network, a rotating BFT committee will handle the bulk of heavy lifting while PoW (the snailchain) will only be used to select the committee members. Observe that in the limiting case, where we take committee rotating frequency to one block and $\mathsf{csize} = 1$, we recover the traditional PoW consensus.

A functioning BFT committee require 2/3 of its members to be honest[13]. Hence, for some $\epsilon > 0$, we require the chain quality $Q > 2/3 + \epsilon$. A naive implementation using a Nakamoto chain as the snailchain will be vulnerable to obvious selfish mining attack strategies. If a selfish miner controlled more than 25% of the blockchain's hash power, she could control more than 33% of block production[22][15]. The probability of being elected to the BFT committee, according to the procedure described in [24], is equal to one's block production fraction. Hence, the selfish miner is likely to control over 1/3 of the BFT committee, and thus the BFT protocol is compromised.

The worst case scenario is possible through a strategy illustrated in [27]. If a selfish miner controls 40% of the hash power in a Ethereum-type blockchain, she can control 70% of block production through optimized selfish mining. According to the committee election procedure in [24], she will have control over 70% of the BFT committee. The BFT committee is not only compromised, the selfish miner will have the dictatorial dominance to declare any honest committee member as 'dishonest' at her will.

We choose the fruitchain[23] as our underlying snailchain for hybrid consensus. The fruitchain is more resistant to selfish mining, as stated by the fairness theorem in [23]. However, the BFT committee is still vulnerable should an attacker directly control over 33% of the blockchain's hash power. Hence, we will take further deviations from [24] and [23] to alleviate the issues. We also remark that given the current market share of mining pools, it's not at all inconceivable for a party to command significant proportion of hash power in a blockchain.

There are two undesirable extremes that we need to find a balance in,

- Randomly select BFT members via VRF[20]. This is vulnerable against a Sybil attack.
- Any selection procedure where probability of being selected is proportional to hash power. The BFT committee is vulnerable against mining pools who are rich in hash power.

Our proposed solution are as follows. When an honest BFT node's chain reaches λ in length, it will publish the unique miner IDs of every fruit in the chain as candidates (or, every miner ID who mined more than ν fruits). The new BFT committee is randomly selected from the candidates though application of VRF.

The obvious Sybil attack is not possible under this scheme, as you require a minimal level of PoW to become a candidate. Neither it will be easy for a large mining pool to achieve compromise the BFT committee. A fruit will be orders of magnitude easier to mine than a block.

6.2. **Mining process.** Inheriting the variable definitions from [23] (p.14 - 16), the fruitchain consist of a blockchain with each block containing its own set of fruits. Transactions executed by the BFT will be initially packaged as a record m to be mined as a fruit. Fruits more recent than a recency parameter R will be packaged into a block when the next block is mined.

The miner will run only one mining algorithm that produces hash values h from a random oracle. A fruit is mined when $[h]_{-\kappa} < D_{p_f}$, and a block is mined when $[h]_\kappa < D_p$, where D_{p_f} and D_p are the mining difficulty parameter of the fruit and block respectively. The tuple (R, D_p, D_{p_f}) determines mining process.

In order to discourage the deployment of ASICs, we will make the recency parameter $\kappa = \kappa(t)$ time-dependent. VRF will generate and broadcast a new $\kappa(t)$ (to fall within the valid range) using VRF once every 3 months. Details of this process will be included in a future version of the yellow paper.

More specifically, the mining algorithm goes as follows. A fruit is a tuple $f = (h_{-1}; h'; \eta; \mathsf{digest}; m; h)$, while a block is a tuple $\mathsf{b} = ((h_{-1}; h'; \eta; \mathsf{digest}; \mathsf{m}; h), F)$ where each entry means

- h_{-1} points to the previous block's reference, only useful for fruit verification.
- h' points to a block that contains the fruit, only useful for block verification.
- η is the random nonce.
- digest is a collision resistant hash function, value used to check the fruit's validity.
- m is the record contained in the fruit.
- $h = H(h_{-1}; h'; \eta, d(F); \mathsf{m})$ is the hash value of the block / fruit.
- F is a valid fruitset as defined in [23].

The blockchain $chain = \{chain[0], chain[1], ..., chain[l]\}$ is a collection of individual blocks ordered by index i, where $chain[i].h_{-1} = chain[i-1].h$, and l is the length of $chain$. We call a fruit f 'recent' w.r.t. $chain$ if $f \in \{chain[l-R+1].F \cup ... \cup chain[l].F\}$. In our implementation, we choose $R = 17$.

We tentatively choose D_p and D_{p_f} such that expected time to mine a fruit and block are respectively 1 second and 10 minutes.

We make the following remark with our mining process,

- Fruits are easier to mine than blocks, and therefore miners are less incentivized to join or form mining pools. This make PoW a fairer process.
- Since fruit mining difficulty is low, its quite likely that two fruits are mined simultaneously. One way of determining which is the valid fruit is by choosing the one with a lower hash value.
- Fruits are not stable until they are written in a block. Therefore the mining reward will be paid to the block miner, who will then distribute her reward to miners of fruits that are included in the block.
- One advantage of the fruitchain protocol is that fruits can be mined in any order, and this can make mining highly parallel. This will be particularly useful when combined with sharding.

6.3. **Storage mining.** Truechain aim to achieve 10,000 TPS on each shard, and number of shards is designed to grow linearly with respect to the volume transaction demand (roughly correlates to number of nodes). The bitcoin network has generated roughly 170 gb of transaction history data over the course of 10 years at 10 TPS. Going from 10 to 10,000, the same volume of data is expected to be generated every 3 days. At 100 shards, or 1 million TPS, we can expect it to generate every 45 minutes. Hence, having every node to store the entire transaction history like bitcoin, is no longer feasible in a high TPS public chain.

A number of solutions have been proposed such as having a check point once every few hours/days, where every node is only required to store transaction history from previous n check points. But where do we store rest of the transaction history, as nobody is incentivized to store that.

Our solution is to seamlessly merge transaction processing with the storage capability of an IPFS in a unified incentive infrastructure. This will provide a solution to store transaction history, and allow a plethora of sophisticated applications to be running completely decentralized on the Truechain architecture. Data storage on Truechain will be possible in three levels,

- Level 1: Stored on every PoW node like bitcoin and ethereum. It is the most permanent way of storage, alas also the least efficient. It's predominately designed to store the block's Merkel root, and other information of high value density. Users will pay a gas fee to PoW miners.
- Level 2: There will be an IPFS-like file system where a limited copy of the data is distributed to storage nodes throughout the chain. This is designed for storing the bulk of main net's historical transactions and data-heavy decentralized applications. Users will pay miners a fee for both storage and retrieval.
- Level 3: Local storage. The file will be only stored your own local node. This is useful for peer to peer communication such a chat program, where it's in the interest of users to be certain that nobody is eavesdropping. This is free of charge.

One advantage of the fruitchain protocol is interoperability due to the flexible structure of fruit mining. Two central pillars of an IPFS are the proof of spacetime (PoSt) and proof of replication (PoRep) consensus. The fruitchain protocol can be modified so that these can be mined by storage miners as fruits. Non-PoW miners will have their own compensation structure, independent to that of PoW fruit miners.

6.4. **Gas fee and sharding.** Gas price is traded in a futures market, where the futures contract is manifested by a smart contract. Specifically, the contract will be executed as follows.

- Party A agree to pay party B xxx TRUE, while party B promises to execute party A's smart contract, between time T_0 and T_1, that cost exactly 1 gas to run.
- Party B will contribute xxx TRUE to a pool corresponding to the committee C that executed party A's smart contract. This is called the gas pool.
- Members of C will receive an equal share of the pool, and return an average cost per gas μ for the pool.
- If B contributed less than μ, she must make up for the difference by paying another party who contributed more than μ. If B contributed more than μ, she will receive the difference from another party.

Algorithm 4: Blockchain growth process

```
**Initialize**
$chain = chain[0]$
$chain[0] = (0; 0; 0; 0; \perp; H(0; 0; 0; 0; \perp), \emptyset)$
$F = \emptyset$
**if** *heard fruit* $f'$ **then**
    **if** $f'$ *is the unique fruit corresponding to message* $\mathbf{m}'$ **then**
        $F = F \cup f'$
    **if** $f'.h_{-1} < f.h_{-1}$ *for all other fruits* $f$ *such that* $f.\mathbf{m} = \mathbf{m}'$. **then**
        $F = F \cup f'$
**if** *heard blockchain* $chain'$ *and* $|chain'.F| > |chain.F|$ **then**
    $chain = chain'$
    where $|chain.F|$ is the total number of fruit contained in $chain$.
**foreach** *time step (1 sec)* **do**
    Heard signed message **m**, broadcasted by PBFT. Let
    $l = |chain| - 1$, so $chain = (chain[0], ..., chain[l])$.
    $F' = \{f \in F : f$ recent w.r.t. $chain, f \notin chain\}$
    $h' = chain[pos].h_{-1}$ where $pos = \max(1, l - \kappa)$.
    $h_{-1} = chain[l - 1].h$.
    **while** *mined = FALSE* **do**
        Randomly pick $\eta \in \{0, 1\}^{\kappa}$
        Compute $h = H(h_{-1}; h'; \eta; \mathbf{d}(F'); \mathbf{m})$
        **if** $[h]_{-\kappa:} < D_{p_f}$ **then**
            $f = (h_{-1}; h'; \eta; \mathbf{d}(F'); \mathbf{m}, h)$
            $F = F \cup f$
            boardcast fruit $f$
            mined = FRUIT
        **if** $[h]_{:\kappa} < D_p$ **then**
            $chain[l] = ((h_{-1}; h'; \eta; \mathbf{d}(F'); \mathbf{m}, h), F')$
            broadcast blockchain $chain$
            mined = BLOCK
```

Under this scheme, liquidity providers are rewarded when they correctly anticipate network stress, and hence creating a complete market in the infinitesimal limit. Price volatility are absorbed by the averaging mechanism in the gas pool making the price itself a good indicator of network stress.

Our intention to ensure gas price is traded roughly within a predetermined interval. Hence, if the moving average price sustain above a certain threshold, a new PBFT committee is spawned through a quantum appearance process. On the other hand, if the moving average price sustain below a certain threshold, an existing PBFT committee will not be given a successor after it finished serving its term.

The proportion of mining reward will be distributed as follows. Let n be the number of PBFT committee running at a certain instance, and $\alpha > 1$. Proportion of mining reward going to PBFT nodes is equal to $\frac{n}{\alpha+n}$, and PoW nodes $\frac{\alpha}{\alpha+n}$. This is to reflect that in later stages of the chain, new nodes are incentivized to contribute to the blockchain's overall TPS, hence ensuring continued scalability of the chain. The parameter α represent the number of PBFT committees when mining reward is divided 50-50.

Treating all shards as equivalent of each other in terms of network and CPU bandwidth could produce skewed results, with inconsistent TPS, or worse, sometimes cross timeout limits, while ordering of transaction takes place from the Primary shard. To tackle this, we propose a compensation infrastructure, that works along the lines of Berkeley Open Infrastructure for Network Computing. There has been a previous attempt in this area from Gridcoin [3] and Golem network [29].

Gridcoin's distributed processing model relies pre-approved frameworks to the like of Berkeley Open Infrastructure for Network Computing (BOINC) [8], an opensource distributed volunteer computing infrastructure, heavily utilized within cernVM[10] in turn, harnessed by the LHC@Home project [18] A framework like this has to tackle non-uniform wealth distribution over time. On the other hand, Golem is another great ongoing project with concrete incentivization scheme, which would be used as an inspiration for compensation infrastructure's incentivization methodology. However keeping in mind, a widely known problem is that a blockchain powered volunteer computing based rewarding model could easily fall prey to interest inflation if the design lacks a decent incentive distribution scheme over time. So to speak, an increasing gap between initial stake holders minting interest due to beginner's luck (algorithmic luck) and the contributors joining late, could thence be found fighting for rewards from smaller compensation pools that further condense.

Depending on the kinds of transactions and whether we'd need decentralized storage for some of the smart contracts, we propose the use of a hybrid infrastructure that utilizes BOINC and IPFS/Swarm, along side of EVM and TVM. This would make use of Linux Containers to deal with isolation of resources and we hope to expand on this section in the next version of this yellowpaper.

7. Future Direction

Even after optimizations to the original Hybrid Consensus, we acknowledge various optimizations possible on top of what was proposed in this paper. There are following possibilities:

- Improving timestamp synchronization for all nodes, with no dependency on centralized NTP servers.
- Detailed incentivization techniques for compensation infrastructure, so heavy infrastructure investors don't suffer from 'left-out', 'at a loss' problem
- Sharding techniques with replica creation to minimize the transaction set rejection from the BFT committee.
- Addition of zero knowledge proof techniques for privacy.
- Hybrid infrastructure of EVM, TVM and Linux container ecosystem.
- Sections for Virtual Machine Specification, Binary Data Encoding Method, Signing Transactions, Fee schedule and Ethash alternative.

8. Conclusions

We have formally defined Hybrid Consensus protocol and its implementation along with plausible speculations in the original proposal. In this draft, we have introduced various new concepts some of which we will detail in the next version very soon. We recommend people to choose ASIC resistant hardware for deployment of the POW only versus full nodes, although more details on hardware shall follow soon.

- A permissioned BFT chain that runs on a few nodes in the permissionless POW based network.
- The BFT committee is a rotating one, preventing corruption in a timely manner
- The BFT committee is responsible for transaction validation, and the POW nodes are only responsible for choosing/electing the committee members according to some rules we've derived and re-defined.
- The new permissioned VM, we've surmised, could be inspired from the EVM, but with different block states and transaction execution flows
- The contemporary permissionless EVM in the POW chain co-exists with this new permissioned VM (which we call Truechain Virtual Machine - TVM)
- The TVM would be the one to validate any transactions related to consensus, while the traditional EVM would need to be re-worked to not really mine for consensus, but for election of BFT using Variable Day length puzzle.
- The incentivation model needs to be re-worked such that it is based off of TVM, and we still reward the miners in POW chain.
- We invented a new market mechanism, satisfying the assumptions of fundamental theorem of asset pricing, for how gas should be traded.
- We would eventually support sharding for the BFT committee nodes, for scalability.
- We address the storage issue for high TPS public chains, and introduced a method that seamlessly merge transaction process with decentralized data storage.
- A compensation infrastructure, which accounts for node configuration non-uniformity (different CPU/memory/network bandwidth in the node pool), would eventually be a part of the consensus, thus speeding up transactions.
- The smart contracts execution would thus only happen in TVM (BFT node).

9. Acknowledgements

We owe a great deal of appreciation and are thankful, to the following folks for their untiring work towards pushing the protocols for a decentralized sovereignty, for their design rationale and implementations which served as a solid reference architecture in our proposal above. These folks and their legacies are as mentioned below:

- Rafael Pass, Miguel Castro, Satoshi Nakamoto, Vitalik Buterin, Gavin Wood, Ethan Buchman, Andrew Miller et al for their untiring work, contributions and continuous improvisations while spearheading the glamorous Improvement Proposals forums in addition to the active participation through Reddit, Mailing lists, chat forums, white and Yellow Papers, and rest of the mediums alike.
- CNCF and Kubernetes communities for their inspiring ventures into hybrid cloud computing.

References

[1] Container-native storage for the openshift masses. *URL https://redhatstorage.redhat.com/2017/10/05/container-native-storage-for-the-openshift-masses/.*

[2] Deploying 2048 openshift nodes on the cncf cluster. *URL https://blog.openshift.com/deploying-2048-openshift-nodes-cncf-cluster/*.

[3] Gridcoin whitepaper: The computation power of a blockchain driving science and data analysis. *URL https://www.gridcoin.us/assets/img/whitepaper.pdf*.

[4] Increase maximum pods per node: github/kubernetes/kubernetes#23349. *URL https://github.com/kubernetes/kubernetes/issues/23349*.

[5] Kubernetes: Building large clusters. *URL https://kubernetes.io/docs/admin/cluster-large/*.

[6] Kubernetes scaling and performance goals. *URL https://github.com/kubernetes/community/blob/master/sig-scalability/goals.md*.

[7] Red hat openshift container platform's cluster limits. *URL https://access.redhat.com/documentation/en-us/openshift_container_platform/3.9/html/scaling_and_performance_guide/*.

[8] D. P. Anderson. Boinc: A system for public-resource computing and storage. *URL https://boinc.berkeley.edu/grid_paper_04.pdf*.

[9] E. Androulaki, A. Barger, and V. e. a. Bortnikov. Hyperledger fabric: A distributed operating system for permissioned blockchains. *URL https://arxiv.org/pdf/1801.10228v1.pdf*, 2018.

[10] J. Blomer, L. Franco, A. Harutyunian, P. Mato, Y. Yao, C. Aguado Sanchez, and P. Buncic. Cernvm a virtual software appliance for lhc applications. *URL http://iopscience.iop.org/article/10.1088/1742-6596/219/4/042003/pdf*, 2017.

[11] V. Buterin. Ethereum white paper, 2014. *URL https://github.com/ethereum/wiki/wiki/White-Paper*.

[12] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.

[13] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[14] W. Delbaen, Freddy; Schachermayer. A general version of the fundamental theorem of asset pricing. *Mathematische Annalen. 300 (1): 463520.*, 1994.

[15] E. G. Eyal, Ittay; Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *eprint arXiv:1311.0243*.

[16] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[17] E. Hildenbrandt, M. Saxena, and X. e. a. Zhu. Kevm: A complete semantics of the ethereum virtual machine. *URL https://www.ideals.illinois.edu/handle/2142/97207*, 2017.

[18] D. e. a. Lombraa Gonzlez. Lhchome: a volunteer computing system for massive numerical simulations of beam dynamics and high energy physics events. *URL http://inspirehep.net/record/1125350/*.

[19] H. A. Mahmoud, V. Arora, F. Nawab, D. Agrawal, and A. El Abbadi. Maat: Effective and scalable coordination of distributed transactions in the cloud. *Proceedings of the VLDB Endowment*, 7(5):329–340, 2014.

[20] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 120–130. IEEE, 1999.

[21] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *URL http://bitcoin.org/bitcoin.pdf*, 2008.

[22] S. M. A. S. E. Nayak, Kartik; Kumar. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IACR Cryptology ePrint Archive*.

[23] R. Pass and E. Shi. Fruitchains: A fair blockchain. In *IACR Cryptology ePrint Archive*, volume 2016:916, 2016.

[24] R. Pass and E. Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[25] R. Pass and E. Shi. The sleepy model of consensus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 380–409. Springer, 2017.

[26] R. Pass and E. Shi. Thunderella: blockchains with optimistic instant confirmation, 2017.

[27] F. Ritz and A. Zugenmaier. The impact of uncle rewards on selfish mining in ethereum. In *http://ieee-sb2018.cs.ucl.ac.uk/presentations/5-Ritz.pdf*, 2018.

[28] T. Schmidt. Modelling energy markets with extreme spikes. *In: Sarychev A., Shiryaev A., Guerra M., Grossinho M..R. (eds) Mathematical Control Theory and Finance, pp 359-375. Springer, Berlin, Heidelberg*, 2008.

[29] T. G. team. The golem project: The golem project. *URL https://golem.network/doc/Golemwhitepaper.pdf*, 2016.

[30] G. Wood. Ethereum: A secure decentralized generalized transaction ledger. *URL https://ethereum.github.io/yellowpaper/paper.pdf*, 2018.

[31] X. Yu, A. Pavlo, D. Sanchez, and S. Devadas. Tictoc: Time traveling optimistic concurrency control. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1629–1642. ACM, 2016.

Appendix A. Terminology

TrueChain Virtual Machine (TVM): In contrast to EVM which handles incentivization and rotating committee selection, a TVM is based on similar design principles but carries out actual consensus and voting based off of PBFT based Hybrid Consensus.