# ABEYCHAIN YELLOW PAPER

ABEYCHAIN: MULTI-LAYERED BLOCKCHAIN
FOR HIGH-VOLUME TRANSACTIONS

Abstract. In this paper we present the initial design of ABEY 3.0 Blockchain (ABEYCHAIN) and other technical details. Briefly, our consensus design enjoys the same consistency, liveness, transaction finality and security guarantee. We discuss optimizations like the frequency of rotating committee members and physical timestamp restrictions. The primary focus of ABEYCHAIN is to advance these concepts and to build a blockchain that is uniquely designed for the ABEY community. We also utilize (i) data sharding and speculative transactions, (ii) evaluation of running of smart contracts in a hybrid cloud infrastructure and (iii) usage of existing volunteer computing protocols for something we introduce as a compensation infrastructure.

## 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 premise for a commonly agreed consensus protocol. The decentralized nature of blockchain, together with transaction transparency, autonomy, immutability, are critical to cryptocurrencies, drawing the baseline for such systems. However The top earlier-designed cryptocurrencies such as Bitcoin[1] and Ethereum[2] , however, have been widely recognized as 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 blockchain that could host computationally intensive peer to peer gaming applications with a very large user base. If such a chainalso hosts smart contracts for Initial Coin Offerings (ICO) in addition toother applications, we could readily expect a huge delay in transaction confirmation times.

There are other models like delegated mechanism of Proof of Stake (PoS) and Hybrid Consensus. ABEY 2.0 adopts hybrid consensus which incorporates a modified form of PBFT (Practical Byzantine Fault Tolerance)[3] and Proof of Work (PoW) consensus. ABEY 3.0 is phasing out the energy-intensive portion of its validation mechanism known as proof-of-work (PoW), to a much more energy-efficient process known as “staking,” or more specifically, Proof-of-Stake (PoS). The PoS protocol with validators could although facilitate high throughput, and is much more energy-efficient than PoW. The PoS protocol ensures safety as long as only one third of the validators in the system are intentionally or unintentionally malicious adversaries, at a time.

In this Paper, we propose ABEYCHAIN 3.0, a PoS Protocol which validators archive consensus using a modified form of PBFT (Practical Byzantine Fault Tolerance)[4]The PoS consensus ensures incentivization and committee selection while the validators act as a highly performance 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 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 of DPoS. The use of DailyBFT as committee members allows for the rotating committee feature which provides for better fairness on the consensus validating peers.

**2.1. Related Works.** In PoS systems, the native token stores value and voting power rather than just value as in PoW systems. PoS protocol achieves Sybil resistance in a BFT way while consuming a fraction of the energy.Rather than relying on computers racing to generate the appropriate hash, the act of locking up tokens, or staking, determines participation in a PoS protocol. This mechanism attempts to reduce the computational cost of PoW schemes by selecting validators in proportion to their quantity of staked holdingst. PoS-based blockchains come with notable benefits and considerations that differ from PoW.

## 3. CONSENSUS

Our consensus design is a PoS consensus, 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 the details of the PoS 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 Hybrid Consensus[5]. In the following part of this section, we will explain our modifications and further constructs .

Our adversary model follows the assumptions in[6] where adversaries are allowed to mildly adaptively corrupt any node, while corruptions do not take effect immediately. In the next version of the Yellow Paper we will formally explain our modifications in Universal Composability model[7].

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 DPoS Consensus Protocol**. In this subsection, we articulate major components and definitions from the ABEY 3.0 PoS Consensus protocol.

**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 PoS Consensus protocol. These signed daily log hashes satisfy completeness and unforgeability.

On keygen, it adds 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. The BFT virtual node, denoted by $BFT_pk$, then starts receiving the transactions (TXs). The log completion is checked and stopped if the stop signal has been signed off by at least a third of the initial comm distinct public keys. During this process, 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 track of incoming transactions with a Union set. On receiving a propose call, it adds the transactions to block and communicates 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 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, Preprocess, Daily Offchain Consensus, and on chain validation.

**3.3. Variant Day Length and Committee Election**. BFT committee instances are switched after a fixed period of time (with the chain as a logical clock)[8]. A 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 situations. 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[9]where the SlowChain can be used as the evidence of misbehavior by BFT committee members. That is, whenever committee misbehavior is detected from the SnailChain, the next day starting point (not necessarily the *K*-th day) will trigger a forced switch.

**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 allow 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 throughput.

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 incorporating one more restriction called sticky timestamp. More specifically, with a heuristic parameter $T_{\Delta}$, 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 modifications give us several extra properties:

1) The order of transactions from any node Ni 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 transactions 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 modification will be the reduction in terms of throughput due to aborting transactions when the parameter $T_{\Delta}$ 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 unsynchronized clocks. This issue can be reduced by adding restrictions on the eligibility of the BFT committee. Later we will see that to get into the committee, the nodes should present evidence of synchronized clocks.

**Algorithm 1**: Extra Verification Regarding Physical Timestamp

Data: Input Transaction TX

Result: A Boolean value that indicates whether the verification is passed

**1** *current_time* ←Time.Now();

**2** **if** | *current_time* - TX.$T_p$| > $T_{\Delta}$ then

**3** | return false;

| // **if** the time skew is too large, reject TX.

**4** var txn_history = new static dictionary of lists;

**5** **if** txn_history[TX.from] == NULL then

**6** | txn_history[TX.from] == [*TX*];

**7** **else**

**8** | **if** txn_history[TX.from][-1]. $T_p$ - TX.$T_p$ > 0 then

**9** | | return false;

| | // To make sure the transactions from the same node preserve timing order.

```
10|  else
11|  |  txn_history[TX.from].append(TX);
12|  |  return true;
```

FIGURE 1. Pseudo-Code for Extra Verification

**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 deterministic sequence DailyBFT [1 . . . 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 St. 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 do not 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 DS[addr] has metadata of rts, wts, readers, writers.

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[10]in distributed transaction processing systems[11], 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 the primary shard receives 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 DPoS Consensus component as that day's log.

There are still many optimization 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.** 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[12]. We therefore hope to follow something like KEVM Jello Paper[13] to list out the EVM and AVM (described in 4.2) specifications.

**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[14]. 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 is possible to keep a chaincode/smart contract in a single container, it 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 being able to run a certain maximum number of containers per node. The limit currently is 100 pods per node, per se, approximately 250 containers per node, as illustrated in Kubernetes container orchestration platform[15]and Red Hat's Openshift Container Platform 3.9's Cluster Limits[16]. Even with latest storage techniques like brick multiplexing[17], the max possible value (say MAX CONTR) of containers could not possibly reach (at least right now) 1000.

**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 = +∞;

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 To;
|  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 2*f* + 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'*in*TX.*before* **do**
|  TX.lowerBound = max(TX.lowerBound, TX'.upperBound);
**for** every TX'inT X.*after* **do**

**35** | TX.upperBound = min(TX.upperBound, TX'.lowerBound);

**36 if** TX.lowerBound ¿ TX.*upperBound* **then**

**37** | Abort TX;

**38** Broadcast *Precommit*(T X.ID, |TX.*lowerBound*+TX.upperBound/2|) to all the previous remote

shards which T X has accessed;

// If TX.upperBound = ∞, we can set an arbitrary number larger than

TX.lowerBound.

**39 On receive** readRemote(addr, ID):

**40 if** host(addr) == $S_i$ **then**

**41** | DS[addr].readers.append(ID);

**42** | return DS[addr].value, DS[addr].wts, DS[addr].writers;

**43 else**

**44** | Ignore

**45 On receive** writeRemote(addr, ID):

**46 if** host(addr) == $S_i$ then

**47** | DS[addr].writers.append(ID);

**48** | Write to a local copy;

**49** | return DS[addr].rts, DS[addr].readers;

**50 else**

**51** | Ignore

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

**1 On receive** Precommit(ID, cts):

**2** Look up TX by ID;

**3 if** *Found and* cts *not in* [TX.lowerBound, TX.upperBound] **then**

**4** | Broadcast *Abort*(ID) to the sender's shard.;

**5** TX.lowerBound = TX.upperBound = cts;

**6** For every data sector DS[*addr*] TX reads, set DS[*addr*].rts = *max*(DS[*addr*].rts, cts);

**7** For every data sector DS[*addr*] TX writes, set DS[*addr*].wts = *max*(DS[*addr*].wts, cts);

**8** 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.

**9 On receive** 2*f* + 1 Commit(ID, batchCounter) **from each remote shards which** T X **has**

**accessed:**

**10** TX.lowerBound = TX.upperBound = cts;

11 For every data sector DS[addr] TX reads, set DS[addr].rts = max(DS[addr].rts, cts);

12 For every data sector DS[addr] TX writes, set DS[addr].wts = max(DS[addr].wts, cts);

13 Mark *TX* committed;

14 Let *TX.metadata* = [*ShardID, batchCounter*];

15 **On output log**

16 Sort *TX's* based on their *cts*. Break ties by physical timestamp.

This issue could further be looked up in the discussions on Kubernetes issues GitHub page[18]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[19], 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 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 severely 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. ABEY Chain Virtual Machine (AVM**). A typical example in this space would be that of the Ethereum Virtual Machine (EVM)[20], 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 invocation value storage.

We would reuse the EVM specifications for the SnailChain, but add a new specification for AVM 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 ABEYCHAIN 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 PBFT, embedded within each full node, so they could manage invocation calls on per-need basis.

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

- re-using some of the concepts from tendermint, like the ABCI (Application Block Chain 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 AVM, smart contracts deployment strategy and a way to deploy permissioned VM onto a permissionless chain.

# 5. BLOCKS, STATE, AND TRANSACTIONS

## 5.1. Block and committee signatures

The block of ABEY 3.0 Chain , which mainly contains the transactions and smart contracts, is generated by the PoS committee when they reach a consensus. Similar to Ethereum block, fast block provides TxHash, Root, ReceiptHash for other non-members to verify transactions included at fast block body. Different from Ethereum block, the block of ABEY Chain includes the committee information and signs from validators.

The commitInfo field includes all the committee members, it presents at the first block of every epoch.

The signs filed includes the parent block signatures and this block signatures from validators, but only the parent block signatures will be calculated as SignHash in block header.

**5.2. Parallel transaction execution** As of today, the multi-core processor architecture has become a major trend in the industry, as parallelization technology can fully utilize the potential of CPUs. ABEYCHAIN's Parallelizing Transaction Execution mechanism uses the advantages of multi-core processors to the fullest by enabling transactions in blocks to be executed in parallel as much as possible.

The traditional transaction execution mechanism works mostly as such: transactions are read one by one from the block. After each transaction is executed, the state machine will move to the next state until all transactions are executed sequentially.

If two transactions have no actual dependency between each other but are deemed so, this will cause unnecessary loss of performance efficiency. On the contrary, if the two transactions rewrite the state of the same account but are executed in parallel, the final state of the account may be uncertain. Therefore, the determination of dependency is an important issue that affects performance and even indicates whether the blockchain can work normally.

We can easily tell whether two transactions are dependent by observing the address of the sender and receiver, such as the following example transactions: A→B, C→D, D→E. Here, you can see that transaction D→E depends on the result of transaction C→D, but transaction A→B has little relationship with the other two transactions, therefore can be executed in parallel.

This analysis is correct in a blockchain that only supports simple transactions, but it may not be as accurate once it is placed in a Turing-complete blockchain that runs smart contracts – because it is impossible to know exactly what is in the transaction contract written by the user. Transaction A->B seems to have nothing to do with the account

status of C and D. However, in reality party A is a special account. When each transaction is transferred through A's account, transaction fees must be deducted from party C's account first. In this scenario, all three transactions are actually related, so they cannot be executed in parallel.

In order to resolve this type of situation, we adopt a trial and error method. Specific steps are as such:

1. Preparing: convert all transactions to "message" type;
2. Grouping: group transactions according to associated address;
3. Executing: execute each transaction group in parallel;
4. Check for conflicts: check whether the transaction groups are in conflict according to returned results. If there is a conflict, roll back the conflicting transactions, then regroup and re-execute.
5. Collect results: collect the execution results of each group, update the tree, and generate the state root

With the above method, transaction-related issues in parallel transactions can be resolved. Of course, inaccurate initial grouping, transaction rollback may still cause performance inefficiency at times. But in most cases, transactions are irrelevant. It is feasible to accurately group and execute transactions in parallel to improve transaction execution efficiency.

## 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 PoS committee members and non-member full nodes, where participating resources can be redirected to do useful things, such as scaling transactions-per-second (referred to as "TPS" from here on) 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[21]. We consider this market to be incomplete, and therefore, fundamental theorem of asset pricing does not apply[22]. Thus,

the underlying gas price will follow a shot-noise process, that is known for its high volatility. We introduce a “gas marketplace” 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 be talking about each component of the incentive design in detail.

**6.1. 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* ABEY, 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* ABEY 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.

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 $n/\alpha + n$, and PoW nodes $\alpha/\alpha + n$. 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 ↲ 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[23]and Golem network[24].

Gridcoin's distributed processing model relies pre-approved frameworks to the like of Berkeley Open Infrastructure for Network Computing (BOINC)[25], an opensource distributed volunteer computing infrastructure, heavily utilized within cernVM[26] in turn, harnessed by the LHC@Home project[27] 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, alongside of EVM and AVM. 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 Yellow Paper.

## 7. FUTURE DIRECTION

Even after optimizations to the original ABEY 3.0 PoS 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 pos committee.
- Addition of zero knowledge proof techniques for privacy.
- Hybrid infrastructure of AVM 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 ABEY 3.0 PoS 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.

- The PoS committee is a rotating one, preventing corruption in a timely manner
- The PoS committee is responsible for transaction validation, and staking is responsible for choosing/electing the committee members according to some rules we've derived and re-defined.
- The new VM (which we call ABEY Chain Virtual Machine - AVM), we've surmised, could be inspired from the EVM, but with different block states and transaction execution flows, transaction parallel processing will be adopted.
- The incentivization model needs to be re-worked such that it is based on of AVM.
- We would eventually support sharding for the PoS 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 AVM (BFT node).

[1] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. URL http://bitcoin.org/bitcoin.pdf, 2008.
[2] V. Buterin. Ethereum white paper, 2014. URL https://github.com/ethereum/wiki/wiki/White-Paper.

[3] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In OSDI, volume 99, pages 173–186, 1999.
[4] Id at 173 – 186.

[5] E. Androulaki, A. Barger, and V. e. a. Bortnikov. Hyperledger fabric: A distributed operating system for permissioned blockchains. URLhttps://arxiv.org/pdf/1801.10228v1.pdf, 2018.
[6] 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.
[7] 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.
[8] 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
[9] R. Pass and E. Shi. Thunderella: blockchains with optimistic instant confirmation, 2017.
[10] 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.
[11] 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.
[12] G. Wood. Ethereum: A secure decentralized generalized transaction ledger. URL https://ethereum.github.io/yellowpaper/paper.pdf, 2018.
[13] 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.
[14] E. Androulaki, A. Barger, and V. e. a. Bortnikov. Hyperledger fabric: A distributed operating system for permissioned blockchains. URLhttps://arxiv.org/pdf/1801.10228v1.pdf, 2018.
[15] Kubernetes: Building large clusters. URL https://kubernetes.io/docs/admin/cluster-large/.
[16] Red hat openshift container platform's cluster limits. URL https://access.redhat.com/documentation/enus/openshift container platform/3.9/html/scaling and performance guide/.
[17] Container-native storage for the openshift masses. URL https://redhatstorage.redhat.com/2017/10/05/containernative-storage-for-the-openshift-masses/.
[18] Increase maximum pods per node: GitHub/kubernetes/kubernetes#23349. URL https://github.com/kubernetes/kubernetes/issues/23349.
[19] Deploying 2048 openshift nodes on the cncf cluster. URL https://blog.openshift.com/deploying-2048-openshift-nodes-cncf-cluster/.. See also Kubernetes scaling and performance goals. URL https://github.com/kubernetes/community/blob/master/sigscalability/goals.md.

[20] G. Wood. Ethereum: A secure decentralized generalized transaction ledger. URL https://ethereum.github.io/yellowpaper/paper.pdf, 2018.
[21] 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.
[22] W. Delbaen, Freddy; Schachermayer. A general version of the fundamental theorem of asset pricing. Mathematische Annalen. 300 (1): 463–520., 1994.
[23] Gridcoin whitepaper: The computation power of a blockchain driving science and data analysis. URL https://www.gridcoin.us/assets/img/whitepaper.pdf.
[24] T. G. team. The golem project:URL 2016.https://golem.network/doc/Golemwhitepaper.pdf, 2016.
[25] D. P. Anderson. Boinc: A system for public-resource computing and storage. URL https://boinc.berkeley.edu/grid paper 04.pdf.
[26] 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.
[27] D. e. a. Lombra˜na Gonz´alez. Lhchome: a volunteer computing system for massive numerical simulations of beam dynamics and high energy physics events. URL http://inspirehep.net/record/1125350/.