IMPORTANT NOTICE

NOTHING IN THIS WHITEPAPER CONSTITUTES LEGAL, FINANCIAL, BUSINESS OR TAX ADVICE AND YOU SHOULD CONSULT YOUR OWN LEGAL, FINANCIAL, TAX OR OTHER PROFESSIONAL ADVISOR(S) BEFORE ENGAGING IN ANY ACTIVITY IN CONNECTION HEREWITH. NEITHER METAPHYSICS OF FUN PTE. LTD. (THE FOUNDATION), ANY OF THE PROJECT TEAM MEMBERS (THE JURA TEAM) WHO HAVE WORKED ON THE JURA NETWORK (AS DEFINED HEREIN) OR PROJECT TO DEVELOP THE JURA NETWORK IN ANY WAY WHATSOEVER, ANY DISTRIBUTOR/VENDOR OF JURA TOKENS (THE DISTRIBUTOR), NOR ANY SERVICE PROVIDER SHALL BE LIABLE FOR ANY KIND OF DIRECT OR INDIRECT DAMAGE OR LOSS WHATSOEVER WHICH YOU MAY SUFFER IN CONNECTION WITH ACCESSING THIS WHITEPAPER, THE WEBSITE AT HTTPS://JURA.NETWORK/ (THE WEBSITE) OR ANY OTHER WEBSITES OR MATERIALS PUBLISHED BY THE FOUNDATION.

All contributions will be applied towards the advancing, promoting the research, design and development of, and advocacy for a new blockchain-based peer-to-peer network consisting of a set of solutions and protocols for running smart contracts as well as building decentralised applications in a decentralised network, designed to meet the large scalability, fast finality and strong reliability requirements that a robust network requires, as well as similar technologies. The Foundation, the Distributor and their various affiliates would develop, manage and operate the JURA Network.

This Whitepaper is intended for general informational purposes only and does not constitute a prospectus, an offer document, an offer of securities, a solicitation for investment, or any offer to sell any product, item or asset (whether digital or otherwise). The information herein below may not be exhaustive and does not imply any elements of a contractual relationship. There is no assurance as to the accuracy or completeness of such information and no representation, warranty or undertaking is or purported to be provided as to the accuracy or completeness of such information. Where this Whitepaper includes information that has been obtained from third party sources, the Foundation and/or the JURA team have not independently verified the accuracy or completion of such information. Further, you acknowledge that circumstances may change and that this Whitepaper may become outdated as a result; and the Foundation is under no obligation to update or correct this document in connection therewith.

This Whitepaper does not constitute any offer by the Foundation, the Distributor or the JURA team to sell any JURA (as defined herein) nor shall it or any part of it nor the fact of its presentation form the basis of, or be relied upon in connection with, any contract or investment decision. Nothing contained in this Whitepaper is or may be relied upon as a promise, representation or undertaking as to the future performance of the JURA Network. The agreement between the Distributor and you, in relation to any sale and purchase of JURA is to be governed by only the separate terms and conditions of such agreement.

By accessing this Whitepaper or any part thereof, you represent and warrant to the Foundation, its affiliates, and the JURA team as follows:
in any decision to purchase any JURA, you have not relied on any statement set out in this Whitepaper;
you will and shall at your own expense ensure compliance with all laws, regulatory requirements and restrictions applicable to you (as the case may be);
you acknowledge, understand and agree that JURA may have no value, there is no guarantee or representation of value or liquidity for JURA, and JURA is not for speculative investment;
none of the Foundation, its affiliates, and/or the JURA team members shall be responsible for or liable for the value of JURA, the transferability and/or liquidity of JURA and/or the availability of any market for JURA through third parties or otherwise; and
you acknowledge, understand and agree that you are not eligible to purchase any JURA if you are a citizen, national, resident (tax or otherwise), domiciliary and/or green card holder of a geographic area or country (i) where it is likely that the sale of JURA would be construed as the sale of a security (howsoever named) or investment product and/or (ii) in which access to or participation in the JURA token sale or the JURA

Network is prohibited by applicable law, decree, regulation, treaty, or administrative act, and/or (including without limitation the United States of America, Canada, New Zealand, People's Republic of China and the Republic of Korea).

The Foundation, the Distributor and the JURA team do not and do not purport to make, and hereby disclaims, all representations, warranties or undertaking to any entity or person (including without limitation warranties as to the accuracy, completeness, timeliness or reliability of the contents of this Whitepaper or any other materials published by the Foundation). To the maximum extent permitted by law, the Foundation, the Distributor, their related entities and service providers shall not be liable for any indirect, special, incidental, consequential or other losses of any kind, in tort, contract or otherwise (including, without limitation, any liability arising from default or negligence on the part of any of them, or any loss of revenue, income or profits, and loss of use or data) arising from the use of this Whitepaper or any other materials published, or its contents (including without limitation any errors or omissions) or otherwise arising in connection with the same. Prospective purchasers of JURA should carefully consider and evaluate all risks and uncertainties (including financial and legal risks and uncertainties) associated with the JURA token sale, the Foundation, the Distributor and the JURA team.

The information set out in this Whitepaper is for community discussion only and is not legally binding. No person is bound to enter into any contract or binding legal commitment in relation to the acquisition of JURA, and no virtual currency or other form of payment is to be accepted on the basis of this Whitepaper. The agreement for sale and purchase of JURA and/or continued holding of JURA shall be governed by a separate set of Terms and Conditions or Token Purchase Agreement (as the case may be) setting out the terms of such purchase and/or continued holding of JURA (the Terms and Conditions), which shall be separately provided to you or made available on the Website. In the event of any inconsistencies between the Terms and Conditions and this Whitepaper, the Terms and Conditions shall prevail.

This is only a conceptual whitepaper describing the future development goals for the JURA Network to be developed. This Whitepaper may be amended or replaced from time to time. There are no obligations to update this Whitepaper or to provide recipients with access to any information beyond what is provided in this Whitepaper.

All statements contained in this Whitepaper, statements made in press releases or in any place accessible by the public and oral statements that may be made by the Foundation, the Distributor and/or the JURA team may constitute forward-looking statements (including statements regarding intent, belief or current expectations with respect to market conditions, business strategy and plans, financial condition, specific provisions and risk management practices). You are cautioned not to place undue reliance on these forward-looking statements given that these statements involve known and unknown risks, uncertainties and other factors that may cause the actual future results to be materially different from that described by such forward-looking statements, and no independent third party has reviewed the reasonableness of any such statements or assumptions. These forward-looking statements are applicable only as of the date of this Whitepaper and the Foundation and the JURA team expressly disclaims any responsibility (whether express or implied) to release any revisions to these forward-looking statements to reflect events after such date.

The use of any company and/or platform names or trademarks herein (save for those which relate to the Foundation or its affiliates) does not imply any affiliation with, or endorsement by, any third party. References in this Whitepaper to specific companies and platforms are for illustrative purposes only. This Whitepaper may be translated into a language other than English and in the event of conflict or ambiguity between the English language version and translated versions of this Whitepaper, the English language version shall prevail. You acknowledge that you have read and understood the English language version of this Whitepaper.

No part of this Whitepaper is to be copied, reproduced, distributed or disseminated in any way without the prior written consent of the Foundation.

JURA
An ultrafast, feeless self-regulated decentralized network

contact@jura.network

Version 2.5

November 10, 2018

Abstract

Having witnessed the rapid development of decentralized applications over the past several years, JURA NETWORK has the vision of building the next-generation blockchain technologies. The requirements for a robust decentralized network are high in terms of scalability, finality and reliability in the face of malicious network activities. Although a plethora of networks have been proposed and adapted in recent years, current levels of transactions per second (TPS), network security, and lack of scalability have left ample room for improvement. With these in mind, the JURA team introduces JURA, a novel technology designed by JURA NETWORK, in details in this article. It encompasses the Fusus, a DAG-lattice data structure as the basis for a distributed ledger, a *proof of utility (PoU)* consensus mechanism to self-regulate in a decentralized network, an anti-spamming *proof of verifiable random time (PoVRT)* module, a *sharding* technology for scalability design, and an AI filter as the security layer. The JURA team believes that JURA will pave the way for widespread deployment of decentralized applications and open up new avenues of improvement to the current market landscape.

1 Background

In 2008, Satoshi Nakamoto created the first cryptocurrency, Bitcoin[1], by introducing what is currently known as *blockchain technology*. Since then, many blockchain applications have been born, primarily in the finance industry. Ethereum[2], a decentralized global computing platform that enables anyone to create, store, and run "smart contract"-based decentralized applications, greatly enhanced what is possible to be built on blockchain technology. IOTA[3] effectually extended this application to the IoT industry, designing the tangle, a directed acyclic graph (DAG) with vertices representing transactions and cryptographically connected edges. Filecoin, a blockchain-based storage network and cryptocurrency, made a few innovations on what can be utilized to achieve consensus. JURA therefore aims to design the next generation blockchain technology with significant improvements to these key components.

Although a large number of blockchain technologies are under development, the current publicly available blockchain solutions are still at the exploratory stage, primarily aiming to meet the large scalability, fast finality and strong reliability requirements that a robust network requires. Unfortunately, no single blockchain technology is currently able to meet all three requirements. Bitcoin currently can handle 7 transactions per second (TPS) on average, and Ethereum can handle about 15 TPS. Many new technologies claim they can achieve unlimited TPS but have only realized small numbers. For examples, Raiblocks (Nano)[4] can achieve 100 TPS on average and 300 TPS at peak performance[6]. Ripple can achieve around 1500 TPS, however, it is a permissioned blockchain technology, not a permissionless one. This is critical

in a truly decentralized network where participants are anonymous. The fastest permissionless blockchain technology by far is Steem, which currently clocks in at about 3500-7000 TPS, or approximately 500 to 1000 times faster than BTC. This remarkable gain is largely due to the *delegated proof of stake (DPoS)* consensus protocol, which can be vulnerable to malicious attacks. It is also worth noting that none of these technologies are currently able to satisfy the projected future scalability requirement of having millions of TPS.

With regards to finality, Bitcoin takes about 6 blocks or 1 hour to confirm a transaction. Ethereum on the other hand takes about 12 confirmations in approximately 3 minutes. Many other blockchain applications claim to have instant finality but in practice have not achieved that goal yet. A close examination of their respective designs can show that there is ample room for improvement. Reliability also comprises a key component of a robust network; this has gained particular light due to reports on the safety of using cryptocurrencies and P2P networks over the past few years. In a decentralized network environment, malicious parties have strong motivations to attempt financial gains and system demise. To design a sound decentralized network, it is paramount to take considerations from multiple angles.

In light of the limitations of current blockchain technologies and to better satisfy foreseeing the future needs and requirements, the JURA team proposed JURA, a set of solutions in a decentralized network, by designing a novel fundamental data structure the Fusus as the backbone for a distributed ledger, a proof of utility consensus mechanisms to secure a well-secured network, applications of verifiable random functions (VRF)[7] that prove to bring benefits in security, an AI security layer to filter out invalid transactions and raise alerts on potentially malicious nodes, and a sharding technique to greatly increase the scalability by reducing the amount of information and communication each node has to deal with in a P2P network.

The paper is organized as follows: Section 2 will be discussing the general design of the JURA protocol. Section 3 then discusses the implementation of the protocol. Section 4 provides details on the sharding technique in the network. Section 5 introduces two functionalities that AI can provide and illustrates its importance to the technology of JURA. Section 6 expands on this and covers the multiple attacks scenarios, along with how these scenarios would be resolved under the JURA protocol. Section 7 then concludes the paper.

2 JURA Protocol

The JURA protocol distinguishes itself from the others by providing a set of well-integrated solutions as a new entity. At the core of JURA is the Fusus data structure. PoVRT, which replaces PoW, is the anti-spamming tool. Each account has its own Fusus chain. Proof of Utility (PoU) then functions as the overseer for the entire system.

2.1 The Fusus

One insight about blockchain technology is the revolutionary design and use of blockchain as the fundamental data structure to cryptographically attach a later segment of transactions to the previous one. The transactions are sorted and organized in a Merkle tree whose root is put into the header of a block. This design provides the convenience of searching for, validating the existence of and checking the integrity of a transaction. This design is also convenient for Simplified Payment Verification (SPV) with Bloom Filter (BF) and HyperLogLog (HLL) for fast transaction operations. This exposition shows that the data structure plays a foundational role of shaping all the other parts in the system. Theoretical and practical endeavors have been put into finding new possible data structures. In particular, the use of a directed acyclic graph (DAG) stands out as a strong alternative to traditional blockchain.

2.1.1 DAG

The use of a directed acyclic graph (DAG) as a generic graphical structure has found applications in many fields, such as scheduling, data processing networks, causal structures, data compression, etc. As the basis, it has also been applied in the cryptocurrency field. For example, DagCoin, was the first attempt to build a blockchain-free cryptocurrency based on DAG by Sergio Lerner in 2012, but only stayed on the conceptual level. IOTA, Byteball, and NANO later all came up with their own variants of DAG to leverage in their protocols. In particular, Tangle, the DAG designed by IOTA, introduces additional concepts to DAG, such as depths, weights, cumulative weights, and tips, which together serve to a tip selection algorithm based on a Markov chain Monte Carlo (MCMC) random sampling algorithm. Unfortunately, the algorithms are rather complicated and the use of a single DAG does not scale with the system well, which thus leaves vulnerable spots open to malicious attacks. Byteball introduced the concept of witnesses to build the main chain. This essentially introduces a mechanism akin to real-world reputation and a centralized authority to run their decentralized system. Any form of centralization in a decentralized system leaves the possibility of being controlled and attacked. In an ideal environment, the system may be functional, however, in the long run, the vulnerable spot will make the collapse of the system highly probable. NANO stands out as a unique and different framework by using a block-lattice DAG data structure. In this structure, each account or user has his or her own blockchain, and a distinction is made between sending and receiving transactions such that accounting is easily maintained and finality speed is increased. The balance of each account is tracked on each node. This saves considerable amounts of time from searching all UTXOs (unspent transaction outputs) to calculate the current balance before every transaction, as in current Bitcoin blockchain systems. This is especially useful when the number of transactions is large, the transaction history is long and the requirement for fast retrieval is high. Furthermore, the linear structure of a blockchain in NANO treating sending and receiving transactions equally and using PoW as the primary anti-spamming strategy causes a bottleneck of speed and shortage of security.

These observations give rise to the need of a new data structure to overcome current limitations inherent in current blockchain technologies. In the next section, the JURA team introduces the Fusus, the backbone of the JURA blockchain technology.

2.1.2 Fusus

The Fusus is a multiple inheritance of blockchain, block-lattice and DAG as a flexible yet practical and effective data structure to achieve scalability, fast transaction, light-weight, instant confirmation all at once.

Account

An account is the public-key portion of a cryptographical signature key-pair. The address, derived from the public key, is shared with the entire P2P network while the private key is only known to the account holder. Traditionally, an account can create a transaction and send the transaction to the target address. It does not store its own transaction history cryptographically. In the Fusus, each account is allowed to hold, store and keep track of its own transactions. Whether it is Bitcoin's blockchain or IOTA's Tangle, every time a transaction is made, the system needs to search through the entire history to calculate the balance for an account. Scenarios can happen when the history is pruned by a snapshot, a technique to remove the earlier part of history, the balances of many accounts would become 0 due to the random key generation and balance searching mechanisms. Therefore, keeping track of the balance in each account can not only achieve fast offline retrieval, but also make the balance less error-prone.

Each account is initiated with a genesis block with a non-zero balance. This is an important requirement to protect oneself from Sybil attacks that will be covered in a later section. Any changes to the Fusus on

each account will require the private key of the account, thus adding a layer of security on top of the speed and convenience benefits.

Transaction

A transaction is any transfer of value from one address to another. What distinguishes JURA from the traditional blockchain technology is the separation between sending and receiving transactions. In the traditional blockchain technology, a transaction is viewed from a system-wide level and considered simply as one single action from A to B. However, from the perspective of an individual account, each account has either sending transactions (STs) or receiving transactions (RTs). They should be treated separately, given the different processing time requirements of the two: STs do have an inherent time urgency in them, and the central prerequisite is there must be enough balance in the account to make a ST possible. On the other hand, RTs do not have the time urgency in them and are thus relatively passive. It is possible to have many RTs incoming at once, whereas many incoming STs may cause strain on the system. There is no requirement on the balance of the account and unless specially required, accounts can tolerate slow updates. This can be backed up by data analysis of usage information of existing cryptocurrencies.

```
send {
    previous: 29DF39JD...F9dfd,
    balance: 010a8033s..1dfd9d,
    destination: mft_df90dfafd,
    elapsedTime: 0.23214289304,
    timestamp: 1521161583,
    type: send
    signature: 83BE...EFND89F
}
```

Listing 1: Fields in a send transaction

Fusus data structure

The Fusus data structure is essentially a DAG-lattice. That is, the transaction histories of accounts form the lattice, and the history of each account is constructed by a DAG. When the traffic volume is low, the DAG is effectively reduced to a blockchain structure. See Figure 1 for an example. When the traffic volume is high, the receiving transactions form a DAG and every send transaction plays the role of a new genesis block. See Figure 2 for an illustration. The Fusus starts with the genesis block and the receiving transactions form the DAG. New transactions reference to $k \geq 1$ previous receiving transactions depending on traffic conditions. If the traffic volume is high, it can fully utilize the potential of a DAG to absorb as many transactions and as quickly as possible. Whenever there is a sending transaction, it serves as a validator to all RTs between the genesis block or previous sending transaction, if they have not been confirmed, to the current sending transaction.

1. Genesis block recording the initial balance.
2. Receiving blocks come in either low or high modes, creating narrow/wide fusus.
3. Every sending block, shrinks the fusus to a single node, referring to all confirmed receiving blocks and calculate the net balance after accounting for the balances in the receiving blocks.
4. The send block will be inserted into the receiving DAG as a new genesis block and the numbers of confirmations of existing receiving blocks remain unchanged.

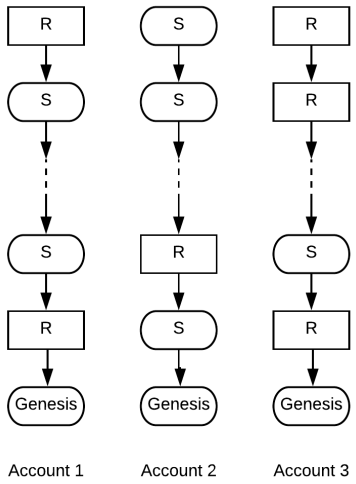

Figure 1: The Fusus data structure in a low-volume traffic scenario.

5. The unconfirmed receiving blocks will keep growing DAG.
6. Pruning is allowed to shrink the size of history.

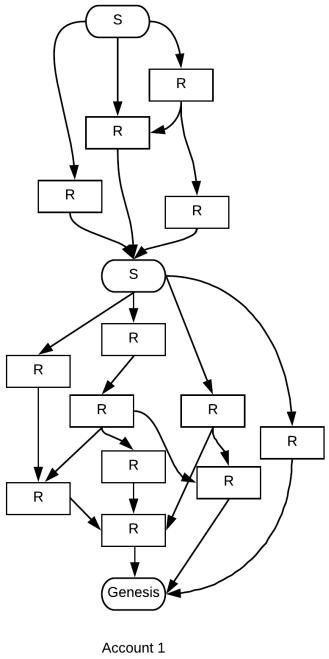

Figure 2: The Fusus data structure in a high-volume traffic scenario.

Transaction time

An analysis of the transaction process can expose the time needed to finish a transaction.

- Traffic time of the sending transaction from sending address to receiving address: t_1

- Traffic time of the receiving transaction broadcasted from receiving address to sending address: t_1
- Processing time of issuing a sending transaction, PoVRT and hashing time, etc.: t_2
- Processing time of confirming a receiving transaction, attaching to DAG, broadcast to the network, etc.: t_3
- The total time is approximately: $t_0 = 2 * t_1 + t_2 + t_3$

Here, t_1 is generally fixed and t_2 is random depending on the PoVRT consensus mechanism (explained in the next section). The Fusus can reduce t_2 and t_3 significantly, which are places where it is believed that the Fusus will outperform single DAG or block-lattice networks by orders of magnitude in terms of processing speed. IOTA and NANO both were criticized for being unable to handle spam attacks given their zero transaction fee and even the anti-spamming PoW. Super-computing power can easily initiate millions of transactions, penetrate through the PoW step and initiate an attack. JURA also charges a zero transaction fee, but the Fusus introduces the following mechanism to avoid the above attacks from the design.

PoVRT

A key observation of the time it takes to generate a Bitcoin block is that the proof of work is essentially a mechanism of assigning a random mandatory waiting time. The net effect does not have to be achieved by solving sophisticated mathematical puzzles which leaves one vulnerable of being attacked due to the processing speeds of today's supercomputers. Instead, the JURA team proposed a proof of verifiable random time (PoVRT) mechanism to generate verifiable random time through a Verifiable Random Function (VRF) module[7]. This module can create an attestation that can be used by any node to verify that the sending node has correctly waited for the proper random time.

It should be noted therefore that the application of PoVRT is not simply a technical alternative to PoW, but can actually act as a safeguard against the aforementioned malicious attacks. Regardless of how strong the underlying computing power is, the system will always impose a random waiting time. The following design would make the transactions faster for normal accounts and increasingly difficult for malicious attacks.

1. The first sending transaction in a time window of length t_{cap} has to wait only milliseconds of time.
2. The succeeding transactions will have to wait gradually prolonged time until the cap t_{cap} is reached.
3. The exact plan is 1st Tx: $0 \sim t_{lim1}$s; 2nd Tx: $t_{lim1} \sim t_{lim2}$s; 3rd Tx: $t_{lim2} \sim t_{lim3}$s; 4th Tx: $t_{lim4} \sim t_{lim5}$s; 5th or further Tx: $t_{lim5} \sim t_{lim6}$s. t_{cap} seconds is the capped waiting time.

Therefore, normal users would not sense an imposed waiting time, but malicious users would experience increased difficulty in making transactions over time.

Broadcasting and validating algorithms

Every account holder has his own Fusus transaction history. As mentioned before, sending transactions serve as new genesis blocks. Whenever a new ST is created, the previous ST, the new ST and all the RTs in-between will be broadcast to the entire network to be confirmed. For the engineering purpose, the RTs are attached in a DAG structure for large scalability. However, for the purpose of validation and calculation, the exact diagrammatic structure is not important. The broadcast data is simplified a structure given below. For a fixed account A, let D^k be the k-th broadcast data:

$$D^k \equiv S_1^k \rightarrow \{R_1^k, R_2^k, \ldots, R_{N_k}^k\} \rightarrow S_2^k,$$

where S_1 is the previous ST, S_2 is the current ST, R_i^k is the i-th RT in the k-th broadcast and N_k is the number of RTs. On the high-level, there is a sequential order of the three components, i.e., S_1 is the first, the chunk $\{R_1^k, R_2^k, \ldots, R_{N_k}^k\}$ originally organized in a DAG diagrammatically is the second, and S_2 is the third. D^k is broadcast to the entire network and received by all full nodes for validation. Here it is assumed that S_1^k has come to a consensus on the network. More specifically, S_i^k is a ST from Account A sending the amount of m_i^k to Account B_i^k having balance n_i^k left, whereas R_i^k is a RT from Account C_i^k receiving the amount of u_i^k.

Denote L as the ledger a given full node keeps, which is a mapping from public keys of a full node to the corresponding Fusus histories. For the j-th full node FN_j, denote its current ledger as L_j and let $L_j(A)$ be the Fusus of A on L_j.

Definition: $F(\text{ledger}, \text{receiving transaction}) \mapsto \{0, 1\}$ is as follows:

$$F(L, RT) = \begin{cases} 1 & \text{if we can find RT in L,} \\ 0 & \text{otherwise.} \end{cases}$$

The function is used to check if a RT is in a specific ledger.

For A, the waiting time between 2 STs is at least δt_A. Denote t_j^k as the time FN_j heard D^k from the network, then $t_j^k - t_j^{k-1}$ would be the difference between two RTs on A on FN_j. A simple condition can be used to check if the two sending transactions are too close in time. If $t_j^k - t_j^{k-1} < \delta t_A$, the new ST is rejected. If $t_j^k - t_j^{k-1} >= \delta t_A$, the new ST is considered passing the time test.

Now, set $B = n_1^k$ as the running balance, and D_{New}^k be a NULL Fusus, then the algorithm of checking the validity of the new ST on one full node goes as follows:

for *i in range(1, N_K)* **do**
  if $F(L_j, R_i^k) = 1$ **then** $B = B + u_i^k$; Append R_i^k in D_{New}^k;
  else
    Pass
  end
end
if $B < m_2^k$ **then** Reject the new ST S_2;
else
  Update $S_2^k.balance = B - m_2^k$;
  Append S_2^k in D_{New}^k;
  $L(A) += D_{New}^k$
end

Algorithm 1: The algorithm of checking the validity of a new sending transaction on a full node.

With this algorithm, every sending transaction will be validated by each single individual full node.

2.2 Proof of Utility

2.2.1 Background

Currently, the most widely implemented consensus mechanism is proof of work (PoW) used in both Bitcoin and Ethereum networks. It is known to be energy-consuming, non-scalable, and inefficient at transforming

computing powers to meaningful outputs. Alternatively, proof of stake (PoS) models have been proposed with the goal of overcoming these limitations.

In a PoS system, validators randomly take turns proposing and voting on the next block with weights depending on the size of the stakes, an amount of cryptocurrencies put up on hold for the purpose of gaining voting power. The weights are then used in calculating the probabilities of being chosen to be the single validator of generating the next block in each time slot (e.g., every 10 seconds) in a chain-based PoS system. This model can overcome the aforementioned limitations, but still suffers from malicious attacks due to the single stake-based model and non-penalizing of the dishonest behavior.

Delegated proof of stake (DPoS) introduces the concept of delegates to the PoS system to overcome the shortcomings a PoS system has. The use of delegates in this model serves to essentially put a layer of centralization with trusted entities on top of the nodes to realize a form of technological democracy. BitShares, NANO and Steem are all applications currently using DPoS, however, there have been numerous concerns about the centralization that DPoS brings to the systems is widely complained about. By design the delegates should be chosen randomly to ensure proper security, in practice, however, finding trustworthy non-official representatives in a decentralized network is challenging. The consequence is that the designers of the systems have to establish trustworthy delegates themselves who gradually become the dominating forces in the system. This basically makes the system centralized. For example, in the NANO network, as of the time of writing, 52% of the stakes are in the hands of official representatives who are the core developers of NANO. Furthermore, it was discovered all of the default settings in wallets of users pointed to one of these official representatives. This centralization gives the power to the developing team and also leaves the vulnerability of these representatives being taken over and controlled for malicious activities.

The JURA team believes that the introduction of centralization and real-world trust system into a decentralized system is only an intermediate step to a stage of full decentralization, i.e., a design that is encouraging for everyone to participate is a proper foundation for eventually achieving a true decentralized consensus.

2.2.2 Consensus mechanism

In a P2P network, a consensus mechanism ideally would incentivize honest nodes for behaving properly according to the protocols and prevents dishonest nodes from malicious attacks. In economic terms, the incentives for sustaining the system are good enough and the costs for malicious attacks are high.

With this principle in mind, partially inspired by the coin-age concept proposed by Peercoin[8], JURA proposed a *proof of utility (PoU)* mechanism with two basic design paradigms: the first is to regulate the voting via utilities, and the second is to penalize dishonest behaviors. The utility is obtained by a function of a set of metrics measuring the contribution to the normal operation of the system, whereas the penalty mechanism is currently under research for the next PoS protocol, Casper[13] by Ethereum.

Time is an important piece of information for PoS systems. Henceforth, time refers to any information that can be directly derived from the timestamps recorded in transactions and stakes. These time-based metrics can be related to metrics in the physical world such as credit scores which have a large impact on regulating the behaviors of people. However, metrics like credit scores do not exist in a distributed system where participants are anonymous and privacy is actually given a high priority. JURA aims to utilize the time information to gain insights about the "creditworthiness" of nodes and create rules into protocols to have the system operate in a self-regulated way.

Abstractly, the size of stakes can be considered as the utility of votes. Utility is a concept largely used in studying economic behaviors. Here it is used to refer to the degree of influence, contribution and goodwill in maintaining the well-being of the system. Stakes with high utilities should have stronger motivation to contribute to the healthy operation of the system and vice versa.

Mathematically, let stake size be a random variable $S \in (0, \infty)$ and the utility be a function of S denoted as $U(S) : R^+ \to R^+$, then in the current PoS systems, the utility of a stake is simply its size, i.e., $U(S) = S$. The linear relationship between the stake size and the voting power can easily lead to the Monopoly Problem that will be covered in Section 6.

Upon close examination to what is available in a distributed system, the JURA team believes that believe by taking the time information into considerations, the utility function could be able to prevent the problems the PoS systems have from happening and make the stakes smart to sustain the well-being of the system. The *proof of utility (PoU)* consensus mechanism is built upon this revelation and the details are given in the next section.

2.2.3 PoU

The concept of using age was known to Nakamoto at least as early as 2010 and used in Bitcoin to help prioritize transactions for example, although it did not play much of an critical role in the security model of Bitcoin. The age is defined as the product of the cryptocurrency amount and the holding period. In a simple example, if Bob received 10 coins from Alice and held them for 90 days, then Bob has accumulated 900 days of age. A similar concept can be found in Peercoin, Vericoin[9] and Reddcoin, respectively. The multiplication of age to stake size can cause a consequence that some deeply-buried coins may come out suddenly with a big weight in voting. Vericoin came up with a non-linear weighting function to impose bounds on age, however, it is an arbitrary *ad hoc* remedy on the stake-level rather than a high-level design feature.

Put this concept into the native utility framework, the utility of coin-age is simply $U(S, T) = S{*}T$, where T is the random variable of holding period. However, age is not the only way of using time information.

In JURA, time information is converted into three metrics: one is the cumulative stake time (CST) or the age of stakes which can be denoted as the random variable $T_c \in (0, \infty)$, the next is the average inter-staking time (AIST) $T_a \in (0, \infty)$ and the third is the last inter-staking time (LIST) $T_l \in (0, \infty)$. The three components play three different roles: CST indicates seniority, AIST implies the active rate of participation and LIST reveals the most recent active rate of participation. These three can easily be derived from transactions and system information and are the additional input variables to the utility function. The new utility function will take the form of $U(S, T_c, T_a, T_i)$. Notice that T_c is the same with the holding period mentioned above, however, the interaction between T_c and S will be different and two additional metrics are introduced into the utility function. The exact form will be given upon further discussions.

The exact form of utility by considered the *relativity*. Any numerical quantity describing a characteristic of a cohort is only meaningful when put in its distribution. For example, in a fairly active system, one day might be a long time, whereas in an inactive system, one day might be very short. This implies a system-level view of the underlying random variables needs to be put into the utility function.

The JURA protocol implements the utility function with a system-level view of the four metrics. First each metric will be examined and put them together to obtain the form of the utility function.

CST

Suppose the creation time of a single stake is T_0 and the current time is T_{now}; in this case, the CST is $T_c = T_{now} - T_0$. For a specific stake i, the CST is observed as $t_{ci} = t_{now} - t_{0i}$, for $i = 1, 2, \ldots, n$, where n is the total number of stakes in the voting system.

The CST, be it a day, a month or a year, reveals nothing about a stake unless it is compared with others. Quantile, calculated from the cumulative density function (CDF), is a metric of ranking values to

the cumulative density between 0 and 1. Let $\Phi : R^+ \to (0,1)$ denote the CDF of T_c. Φ therefore evaluates the probability that T_c will take a value less than or equal to t_c, i.e., $\Phi(t_c) = P(T_c < t_c)$. Figure 6 shows the histogram of simulated CSTs and Figure 4 is the corresponding CDF plot. Clearly, this is a non-linear standardization process, from a positive real number to its percentile $\Phi(t_c)$, which at the moment can be used as the marginal weight assigned to the stake by the metric T_c.

On the one hand, the standardization of CST ensures that no single stake can dominate the system by simply being old. On the other hand, stakes that just jump into the system will not have an impact since their CST will be close to zero and the weight will be negligible, thus preventing rushed malicious attacks. As stakes become older, the weights would gradually increase, thus achieving higher levels of influence. The time it takes for a stake to rise from zero to any significant weight can be considered as the opportunity cost of initiating an attack.

The idea of interacting stakes with age also resonates well with the democratic voting system, where although every person has one vote: senior members of greater expertise and a larger network tend to have more influence.

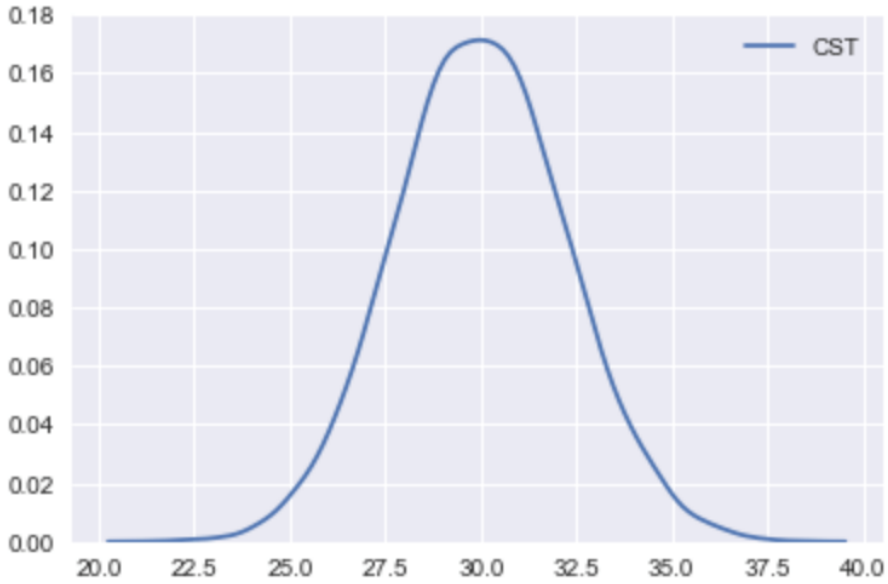

Figure 3: Simulated probability density function of CST.

AIST

Suppose since its creation, the i-th stake has participated in staking n_i times, at the sequence of entrance and exit time points $t_{i1}^{En} < t_{i1}^{Ex} < \ldots, < t_{in_i}^{En} < t_{in_i}^{Ex}$, then the average inter-staking time is $t_{ai} = \frac{1}{n_i - 1} \sum_{k=2}^{n_i} \left[t_{ik}^{En} - t_{i(k-1)}^{Ex} \right]$. This quantity can be considered as measuring the rate of participation in staking. Since AIST is an average, by the Central Limit Theorem (CLT) and assuming that both the mean and variance are finite, i.e., $0 < \mu_{ai} < \infty$ and $\sigma_{ai}^2 < \infty$, then AIST can be expected to be normally distributed as the number of participation gets higher and higher, i.e., as time goes by, the habitual behavior of every account will emerge and converge to a distribution with certain parameters. Formally, as $n_i \to \infty$,

$$\sqrt{n_i}\big(t_{ai} - \mu_{ai}\big) \xrightarrow{d} N(0, \sigma_{ai}^2).$$

On the population level, the percentile of t_{ai} is then $\Psi(t_{ai})$ where $\Psi(T_a) : R^+ \to (0,1)$ is the marginal CDF

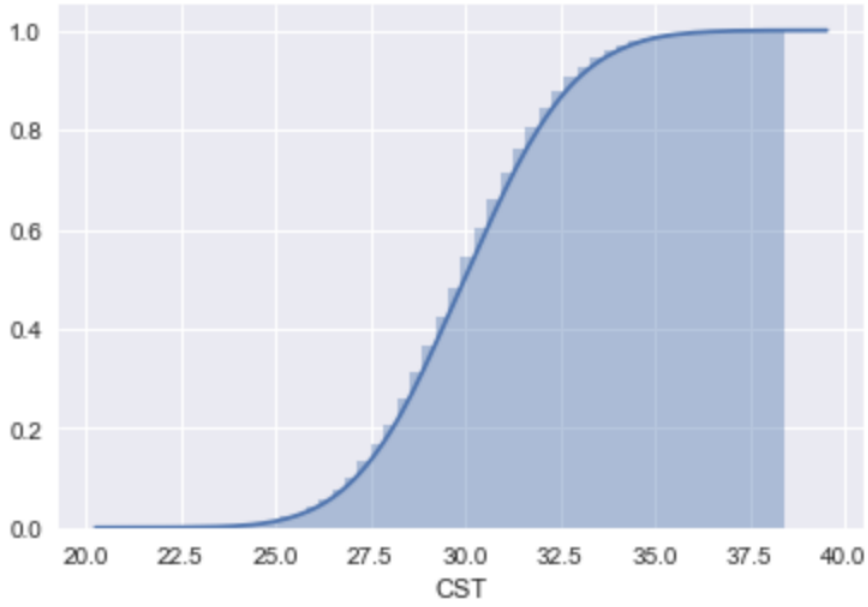

Figure 4: Simulated (empirical) cumulative density function of CST.

for T_a. In this case, smaller values are encouraged over larger ones, the marginal weight is $1 - \Psi(T_a)$. If AIST is small, the rate of participation will then be high, this participant is hardly dishonest and should be given a higher weight. On the contrary, an inactive participant should not be given a high weight unless he makes it up by behaving actively over a period of time. This can also prevent any stakes that do not have any history to have a large influence in the consensus mechanism. Furthermore, the lack of participation can also be considered as an opportunity cost. If a stake does not participate often, their utility becomes lowered. This naturally gives stakeholders incentive to participate more.

LIST

Use the same notations above, the last inter-staking time is calculated as $t_{li} = t^{En}_{in_i} - t^{Ex}_{i(n_i-1)}$. This quantity measures the last time a participant stakes from now and is thus more concerned about recent short-term behavior rather than long-term habitual behavior, which is described above by AIST. It is also tied to market conditions and other macro factors.

The CDF is $\Gamma(T_l) : R^+ \to (0, 1)$. A smaller value of T_l is encouraged and the marginal weight is $1 - \Gamma(t_l)$. This can serve as a buffer for large stakes with active participation histories but large most recent staking gaps.

In implementation, the stack time is tracked in a discrete manner. There is no need to ensure the accuracy of time-stamps for the network. The solution is to keep track of which round of vote the stack is used then the round number shows the approximate time and consistency is easily maintained for the network

Numerical calculations

Although the exact CDFs for systems are not readily available, given the large amount of data the system is anticipated to generate, the empirical cumulative density functions (ECDF)s can be obtained. Even in such an early stage of the system, the data from other *PoS* systems can be used to provide rough estimates of the *priori* ECDFs for the network.

Note that here, the distributions are not parameterized. Hacking or using AI to infer for the parameters in this situation are not feasible given today's computing power. Furthermore, since the timestamps are immutable, knowing the distributions would not help in inflating the weights of stakes. Further, the utility function will make any attempt to game the system even more difficult.

Bivariate joint CDF

The discussions about CST and AIST above reveal one insight that these two metrics describe the long-term characteristics of the stake holders. They should not be independent and are likely to covary in a way that can be best depicted by a joint CDF. Using the joint CDF rather than the marginal CDFs independently can improve the consistency of an estimator of the true utility of the stake, i.e., the estimator $f[S, \Lambda(T_c, T_a), T_i]$ of $U(S, T_c, T_a, T_i)$ has a better consistency than another estimator $g[S, \Phi(T_c), \Psi(T_a), T_i]$, where $\Lambda : R^+ \times R^+ \to (0, 1)$ is the joint CDF of (T_c, T_a). An illustration of the joint CDF can be seen in Figure 5, where for each marginal selection in one dimension, the other dimension is still varying. This can further illustrate the limitation of the current PoS systems by being univariate.

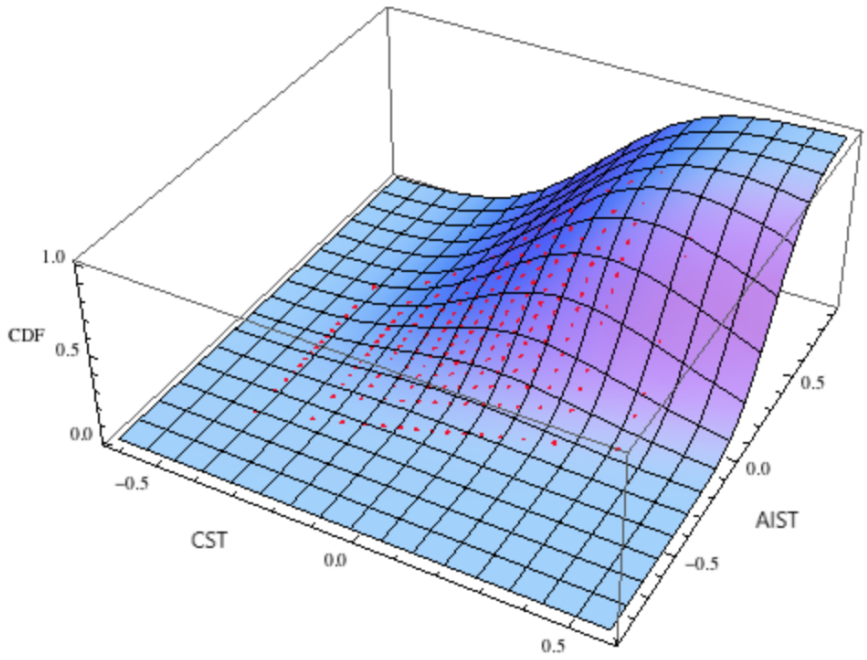

Figure 5: Simulated bivariate joint CDF of CST and AIST.

Utility of four metrics

The consistency can be further improved if all four metrics are considered in the utility function. Let $V = (S, T_c, T_a, T_l)$ denote the four dimensional random vector of the three time-related metrics and the stake size, and $v_i = (s_i, t_{ci}, t_{ai}, t_{li})$ be the observed i-th stake, then the utility of it is

$$
\begin{aligned}
U(v_i) &= P(S < s_i, T_c < t_{ci}, T_a > t_{ai}, T_l > t_{li}) \\
&= \int_0^{s_i} \int_0^{t_{ci}} \int_{t_{ai}}^{\infty} \int_{t_{li}}^{\infty} \theta(x_1, x_2, x_3, x_4) dx_1 dx_2 dx_3 dx_4 \\
&\approx \frac{1}{n} \sum_{k=1}^{n} \delta_k(s_i, t_{ci}, t_{ai}, t_{li})
\end{aligned}
$$

where $\theta(x_1, x_2, x_3, x_4) = \dfrac{\partial^4 \Theta(x_1, x_2, x_3, x_4)}{\partial x_1 \partial x_2 \partial x_3 \partial x_4}$ is the *probability density function (pdf)* of V and $\Theta(x_1, x_2, x_3, x_4)$ is the corresponding CDF, and $\delta_k(s_i, t_{ci}, t_{ai}, t_{li})$ is Dirac delta function indicating whether an arbitrary k-th stake satisfies the conditions, i.e.,

$$
\delta_k(s_i, t_{ci}, t_{ai}, t_{li}) = \begin{cases} 1 & \text{if } s_k < s_i, t_{ck} < t_{ci}, t_{ak} > t_{ai} \text{ and } t_{lk} > t_{li}, \\ 0 & \text{otherwise.} \end{cases}
$$

Then the weight for the i-th stake is $p_i = \dfrac{U(v_i)}{\sum_{i=1}^{n} U(v_i)}$, where n is the total number of stakes participating in the voting.

Practical issues

Fast calculation of the utility score In practice, calculating empirical joint CDF is typically challenging and time consuming, which takes $O(nlog(n))$ sorting complexity. When the stake size is large, the time that the task of calculating the utility score grows fast.

Sklar's theorem states that any multivariate joint distribution can be written in terms of univariate marginal distribution functions and a copula which describes the dependence structure between the variables.

In particular, the marginal distributions of v as $f_1(t_c)$, $f_2(t_a)$, $f_3(t_i)$ and $f_4(s)$ are obtained by fitting the parametric distribution by maximum likelihood estimation (MLE) method on the sample of stakes. The resulting parametric marginal distributions can then be used to calculate the marginal CDFs (or the survival function, i.e., $1 - CDF$) like $\hat{F}_1(t_c)$, $\hat{S}_2(t_a)$, $\hat{S}_3(t_i)$ and $\hat{F}_4(t_s)$. An estimated correlation matrix Σ is calculated on a regular basis, then the utility score is calculated by the Gaussian copula $C_{\Sigma}^{\text{Gauss}}(v) = \Phi_{\Sigma}\left(\hat{F}_1^{-1}(t_c), \hat{S}_2^{-1}(t_a), \hat{S}_3^{-1}(t_i), \hat{F}_4^{-1}(t_s)\right)$.

The estimation of the parameters and distributions can be done on a daily or weekly basis. The benefit of using this methodology is to reduce the conversion of information in stakes to utility scores in $O(1)$ time, which is a highly desired feature in a distributed network.

The potential issues include some degree of loss of precisions, i.e., the errors in fitting the marginal distributions, correlation matrix will make the utility of a specific stake inaccurate. The errors should be controlled in a minimal range. The reason is in a typical CDF as depicted by 4, if the value is in the normal range, i.e., the linear section of the CDF, the error from the MLE estimation should be very small, since the data point is in the high density range. If the data points are in the far ends of the spectrum, since both the tail parts are nonlinear, even though the errors are big, the resulting deviation of the quantiles is still small. Moreover, since the utility serves as the input of a probabilistic sampling procedure, the impact will be further reduced, i.e., an error of 0.2 in utility score will not change the probability of being chosen dramatically, i.e., $0.2/\sum_{i=1}^{n} U(v_i) \to 0$ as $n \to \infty$, however, it would be a huge error for a utility score to deviate 0.2 units from its true value.

Sampling issues The time of calculating the weights is of the same order of magnitude with that for PoS, however, a huge benefit of using the PoU system is having a standard and consistent way of filtering out unqualified stakes by setting up a threshold of utility scores, for example, only utilities higher than 0.6 qualify for the next-round of sampling. This provides a huge reduction in the sampling space. A sample of 1 million stakes can be reduced to a mere 50 thousand for sampling. A significant boost in sampling speed. This convenience is not possible in PoS system given the stake sizes are not bounded and consistent from cycles of sampling. The process of calculating the utility and probability of being chosen takes ignorable time. The random sampling step is straightforward once the probability of being chosen for each stake is obtained.

Updating frequency Updating the ECDFs with traffic in the JURA system can be done on a daily, weekly or monthly basis depending on the stability and necessity, in either a cumulative approach or a running window approach. A system-level multivariate Kolmogorov-Smirnov goodness of fit test[11] can be constantly applied to monitor the stability of the ECDFs. The updates can be made in the low traffic time, which would not have an impact on the system, very similar to how the Bitcoin network adjusts the puzzle difficulties on a regular basis.

Scenario analyses

For the basic scenario, 0-10 is used to represent the possible values of each metric, and a qualitative analysis of scenarios of staking can be expressed by combinations of these numerical values in the utility function $U(S, T_c, T_a, T_l)$. By default, the AIST and LIST are set at 10 for new stakes.

1. $U(10, 0, 10, 10)$: A large amount of new purchase of stake in the system. Although the size of stake is the highest possible, the other three metrics are all at the unfavorable values. The utility of this stake would be effectively 0, hence, the stake does not have any voting power.

2. $U(5, 5, 5, 5)$: A normal stake with a normal length of history, a normal AIST and a normal LIST. Nothing suspicious about this stake, and the utility ought to be a high number, although marginally the values are all just average.

3. $U(5, 10, 10, 10)$: A typical amount of oldest stake in the system, yet with close-to-none participation history. This stake can be dangerous if given a high weight immediately. Hence, the utility ought to have a very low value as well.

4. $U(5, 10, 2, 10)$: A typical amount of an older, historically active stake, yet inactive for a long time. Just to be safe, a buffer effect is set up for this stake. That is, the weight is moderately small, and once the stake has some base level of participation, the utility would go up quickly to a high value.

5. $U(5, 10, 8, 2)$: A typical amount of older stake that has been historically inactive yet active recently. This stake has the potential to be influential, however, it takes time to build the history. The utility would be in the lower half initially and grow to the upper half quickly.

6. $U(1, 5, 1, 1)$: A small amount of stakes of average age with very high historical and recent participation rate. Although the size of the stake if very small, the high rates of participation would be able to compensate for the small size. Hence, although the utility would not be high, it would not be as low as the marginal position by stake size.

2.3 Consensus Model

This section gives the details of achieving the consensus on the network by combining the Fusus data structure and proof of utility (PoU). The steps and key items are given in a sequential order.

1. n full nodes are chosen randomly by PoU. The probabilities of making right decisions for these nodes should be much higher than those chosen by the PoS system. Denote the change of ledger from full node i as ΔL_i.

2. For each candidate ΔL_i, the network needs to check the consistency of ΔL_i and reject ΔL_i if it is not consistent. The exact checking process is as follows:

   (a) Check if ΔL_i is consistent with the broadcast data, i.e., if there is any conflict of transactions.

   (b) Check if each RT can trace back to a ST in each ΔL_i.

   (c) Check if each ST in ΔL_i has enough balance to be valid.

3. For any ΔL_i, a typical sending data is in the form

$$D_A^k(i) \equiv S_1^k(i) \to \{R_1^k(i), R_2^k(i), \ldots, R_{N_k}^k(i)\} \to S_2^k(i),$$

   where $D_A^k(i)$ is the k-th data broadcast by account A, recorded from ledger i.

   Remarks:

   - $D_A^k(i)$ recorded in ledger i may not be recorded in ledger j.
   - For ledger L_i and L_j, ΔL_i and ΔL_j may choose different sets of RTs between S_1 and S_2, so $D_A^k(i)$ and $D_A^k(j)$ may not be the same.

4. Denote the union of R's from $D_A^k(i), i = 1, \ldots, n$ as $R_{set} = \{R_t | R_t \in D_A^k(i) \text{ for any } i\}$, then define $\widetilde{D_A^k} = S_1^k \prod_{R \in R_{set}} R^k S_2^k$ as the "super" broadcast data. Note by 2(a) above, $\widetilde{D_A^k}$ is still a subset of broadcast data. Also, denote $\widetilde{\Delta L}$ as the change of ledger formed by all $\widetilde{D_A^k}$. Since 2(b), it is known that $\widetilde{\Delta L}$ is also consistent. Moreover, $\widetilde{\Delta L}$ contains all potential transactions the network may approve. $\widetilde{\Delta L}$ forms a candidate pool for transactions to be validated.

5. It can be claimed that $\widetilde{\Delta L}$ forms an inter-account DAG with each node being a transaction. For any pair of sending and receiving transactions, both Node(S) and Node(R) are denoted as the node representing the transactions. For any ST S_2 from $D \equiv S_1 \to \{R_1, R_2, \ldots, R_N\} \to S_2$, A direct edge is put from $Node(S_2)$ to $Node(S_1)$ and every $Node(R_i)$. For any $Node(R_i)$, it can be identified as the $Node(S)$ for its sending transaction and put further direct edge.

   Note this inter-account DAG may not be connected, it can be called a DAG forest, and each connected component is a DAG tree.

6. For each DAG tree, the network will validate it from leaves to root. The validation is in a greedy approach. For any validating node(transaction), the network checks two things: one is if the current balance can support this transaction, which is done by checking the output direct edge (RTs), and the other is if this node gets 70% vote from all ΔL_i. If one node is not considered valid, the DAG tree is pruned by deleting this node and all connecting edge.

7. At this point, the model must have a pruned DAG forest, and all transactions will form $\widetilde{\Delta L^*}$ to all full nodes and accounts. All full nodes will append $\widetilde{\Delta L^*}$ to its current ledger and all accounts can check $\widetilde{\Delta L^*}$ to see if its transaction has been approved. At this point, the network has come to a new consensus.

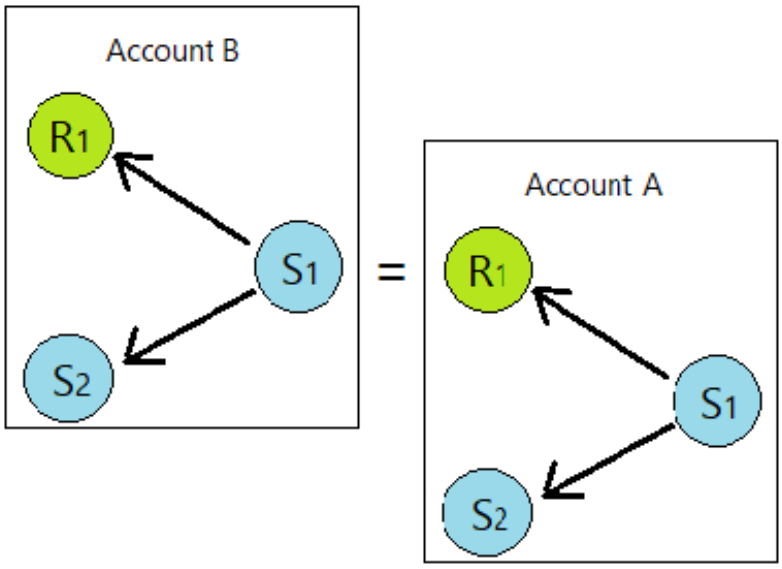

Figure 6: inter-account DAG example.

3 Implementations

3.1 Verifiable random function

To replace the anti-spam module PoW in the nodes, a verifiable random function is used to generate the random waiting time for the PoVRT mechanism. The proof of waiting for the specified random time can be later verified by other nodes in the distributed system. This decentralized random, unique-deterministic, non-interactive, Distributed Key Generation (DKG)-friendly signature scheme are derived from BLS[10].

An efficient and practical verifiable random function was proposed by Yevgeniy Dodis and Aleksandr Yampolskiy[5]. The mechanism of VRF is as follows:

On input x and a secret key SK, the VRF computes $(F_{SK}(x), \pi(x))$, where $F_{SK}(x) = e(g,g)^{1/(x+SK)}$ is the VRF value and $\pi(x) = g^{1/(x+SK)}$ is the proof of correctness. The random waiting time with no restriction on the precision can be first converted to a string and by a collision-resistant hash function.

$$F_{SK}(x) = e(g,g)^{1/(x+SK)} \text{ and } p_{SK}(x) = g^{1/(x+SK)},$$

where $e(\cdot,\cdot)$ is a bilinear map. To verify whether $F_{SK}(x)$ was computed correctly or not, one can check if

$$e[g^x PK, p_{SK}(x)] = e(g,g) \text{ and } e[g, P_{SK}(x)] = F_{SK}(x).$$

The proof of security relies on a new decisional bilinear Diffie-Hellman inversion assumption, which asks given $(g, g^x, \ldots, g^{(x^q)}, R)$ as input to distinguish $R = e(g,g)^{1/x}$ from random.

The time interval between inter sending actions is random but verifiable across the network. This provides a controllable, verifiable and most importantly effective signature scheme such that the sending actions are all self-regulated by the design of the systems.

3.2 Account opening fees

One more protection against swarm attacks is by requiring a certain amount of account opening fees, i.e., an address is active and finds it possible to make a transaction only when the balance is above a certain threshold. For normal users, this fee is negligible. The idea of creating millions of accounts and sending out transactions to flood the network however would be expensive.

Here is the account opening fee model: assume x is the number of current existing account number in the network, the opening fee f would be

$$f(x) = 10^3 \times \log_{10}\left(\frac{x + 2 \times 10^6}{10^6}\right)$$

By this model, the first 1 million account in JURA network will have the opening fee from around 480 JURA to 600 JURA, 2 million to 10 million accounts will need 600 to 1000 JURA. And by the time the network has around 100 million accounts, opening fee will be around 4000 JURA. This model will largely incentive the people to be early users of JURA network.

3.3 Time in a distributed system

The time in a distributed system is an important concept which lays out the foundation for the possibility of a successful evaluation of it across the network with accuracy and consistency. Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks. In operation since before 1985, NTP is one of the oldest Internet protocols in current use. NTP was designed by David L. Mills of the University of Delaware. This provides a framework of evaluating the time-related components in the system.

3.4 Smart contracts

A smart contract refers to a specific type of protocol for regulating contracts. A smart contract is a special protocol intended to contribute, verify or implement the negotiation or performance of the contract. Smart contracts allow to perform credible transactions without third parties. These transactions are trackable and irreversible. Smart contracts contain all the information about the contract terms and execute all envisaged actions automatically. The idea was originally described by computer scientist and cryptographer Nick Szabo in 1994 [12].

The main principle can be compared to the work of vending machines. They execute only the instructions given to them automatically. At first, assets and contract terms are coded and put into the block of a Blockchain. This contract is distributed and copied multiple times between the nodes of the platform. After the trigger happens, the contract is performed in accordance with the contract terms. The program checks the implementation of the commitments automatically.

A smart contract is usually comprised of the following components:

- Subject of the contract: The program must have access to goods or services under contract to lock and unlock them automatically.
- Digital signatures: All the participants initiate an agreement by signing the contract with their private keys.
- Contract terms: Terms of a smart contract take the form of an exact sequence of operations. All participants must sign these terms.

- Decentralized platform: The smart contract is deployed to the Blockchain of this platform and distributed among the nodes of the platform.

In JURA, the smart contract implementation is easy and natural. Unlike other DAG structure, the order for related transactions can be easily sorted out in Fusus data. Every time an account creates a smart contract, the protocol will create an autonomous account which acts as the smart contract.the contract call or message call from external account is identified as a sending transaction from those accounts. Smart contracts identify these transactions as receiving transactions. Likewise, the contract response can be regarded as sending transaction for smart contract and formed the sending chain. If the order of receiving transactions for smart contract is not required, they will be trivially attached to its sending chain. Otherwise, for the receiving transactions with order, every time a smart contract processes a receiving transaction, it will send to itself right away. Thus, the order property is kept by the sending chain, which has total order.

WebAssembly (WASM) Virtual Machine

JURA will launch using the WebAssembly (WASM) virtual machine. Ethereum currently uses a proprietary VM called the Ethereum Virtual Machine (EVM). WASM is widely acknowledged to be a faster and all-around better solution than the EVM. Even Ethereum is working on a WASM implementation. Other Ethereum competitors such as EOS and Dfinity will launch with WASM. WASM has the following advantages:

- Improvements in terms of speed and performance
- Support for C, C++, and Rust, with compilers for other languages in progress

This means that developers who already have experience with those languages can quickly begin building on JURA, instead of having to learn a new language like Solidity in order to create dApps and smart contracts. Further, this means that developers can leverage all sorts of tooling and software libraries that have already been built for those languages when building on JURA. Finally, the use of WASM will provide superior optimization and debugging tools. All of these features will help to accelerate and simplify the development process.

4 Dynamic Monitored and Distributed Sharding (DMDS)

DMDS is the general type of database partition that separates large database into smaller, faster, and easier managed data shards. The concept of Fusus sharding is similar. Due to space storage issues, there is a need to divide the single Fusus to multiple subsections which are called Fusus shards. Each shard contains different sub-Fusus which will work as usual in parallel. There are two motivations to do this:

1. More and more clients will be involved as the system grows. The amount of information might become overwhelming, and each node in Fusus graph will find it difficult to store all the transaction histories.
2. As traffic increases, the frequency of compression and new generation will increase dramatically which will lead to high timing and maintenance costs.

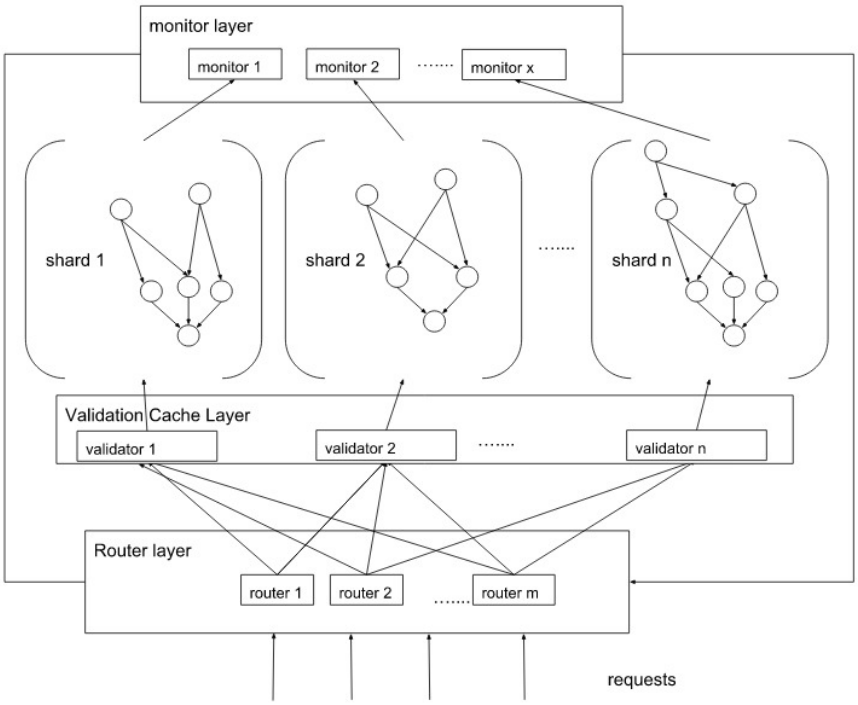

Figure 7: The architecture of DMDS

4.1 Router layer

A router layer which contains m routers is built. When requests come in, a round robin process will be conducted to select one router that redirects the request to the corresponding shard. Each router contains two same hashmaps as the following image:

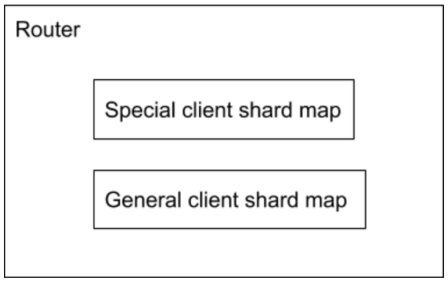

Figure 8: The illustration of router layer

The key of special client shard map is special client sharding key, the value is real shard address. When a request comes in, the validator checks whether the client sharding key of this request is in this map or not. If it is in this map, the request will be routed to the shard directly. Otherwise, request will be passed to general client shard map. The key of general client shard map is shard index, value is the real shard address.

*shardIndex = hash(sharding key) * mod(the num of shards)*

Sharding key is the means to identify which shard this transaction request belongs to. There are multiple choices for the sharding key, for example: "transaction id" or "address".

When transaction ID is used to do sharding, the hash space of transaction ID is divided to multiple

ranges, and map each range to a certain shard. In this way, nodes in each shard will only contains n_{tx}/n_{shd}, where n_{tx} is the number of transactions and n_{shd} is the number of shards. However, the big issue for this case is double spending. As traffic grow, if the conflict happens in different shard, node in this shard have to contact all shards to find the transaction which is too expensive.

To make sure double spend checking works as proposed, the addresses can be also used as sharding keys. In this way, all the transaction from the same address hash range will fall in the same shard. Thus, there is no need to contact other shards. All double spend checking will be done inside the shard as per regular operation.

4.2 Validation Cache Layer and Transaction Flooding Defender

After a request passes the router layer, it will go into validation cache layer. Each shard has a validator which will validate whether request contains correct info or not to avoid bad request.

However, transaction flooding attacks will occasionally happen, i.e., where attackers may send as many valid transactions as possible in order to saturate the network. Usually an attacker will send transactions to other accounts they control so it can be continued indefinitely. A normal validator cannot identify the intent of valid transactions. Thus, a cache module is added to each validator. When the address hash in a request is in the cache, the validator will reject the request, otherwise it will be directed it to the shard.

Each cache is designed using a LRU (least recently used) algorithm. In this process, a queue is maintained which will contains all the transaction requests in time window T (most recent T seconds). Each node in the queue is a pair (address hash, the last timestamp that requests with the same address hash that access this validator server). the head pointer of the queue is checked every second, and remove the least recently used node if they are expired (i.e., the last timestamp request with the same address that has accessed this validator is before the current timestamp - time window T).

When a new request come in, the validator will check whether the address hash has been in the queue or not. If not, a new node which only contains the pair will be created and added to the tail of the queue. If the address hash has already been in the queue, it will be moved to the tail, and the timestamp in the pair is updated. For example, if the time window T is 3 seconds:

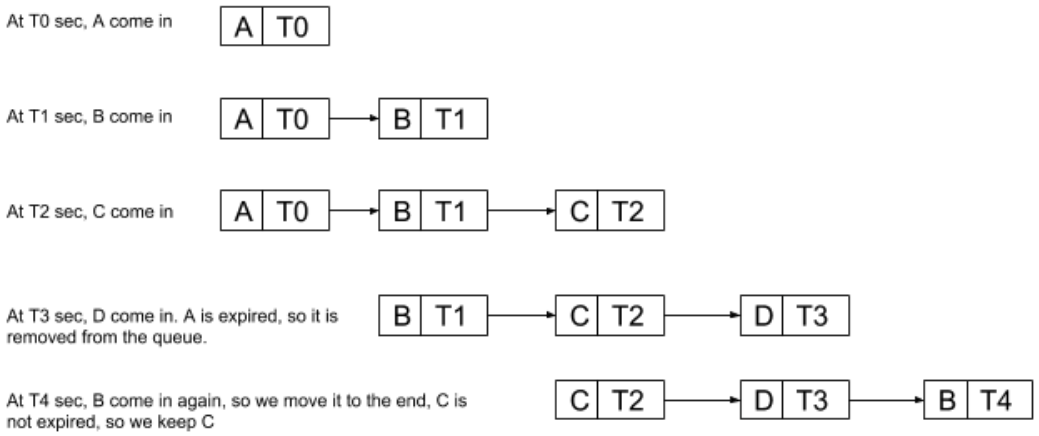

Figure 9: LRU (least recently used) algorithm

It is possible that a lot of requests comes to the validator at same time, but since the validator only keeps the very small address hash and timestamp, the queue can be stored in either cache or system memory.

This LRU cache will be able to handle transaction flooding attack-type problems. When this type of

attack happens, only the first request will be accepted, and others will be rejected and wait for T seconds.

4.3 Monitor Layer

This layer contains multiple monitors, each of which is defined as a group of coordinator nodes in the same shard. Each monitor will oversee the general info of the shard. Sometimes, even though sharding transaction request are done using address as sharding key, the storage space and cpu that a single node uses are still hard to be afford because those requests can be spread evenly. Some addresses will inherently be much more active than others. To those, monitor layers will be added to fix this issue. Regular node need staking to be selected as the coordinator, who gets incentives is based on bid rate of the overall shard groups. Each monitor server has three models:

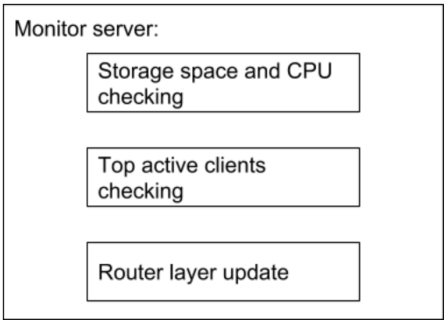

Figure 10: The illustration of monitor server

Storage space and a CPU checking model will check currently shard storage space and CPU resources used for nodes. If it is the above threshold, compression will be triggered and a new genesis will be built, and the top active clients' addresses will be identified. The system would then add their addresses to a special client shard map in the router layer for router updating. After updating the router layer, those special clients would be routed to some new backup shards with less traffic and fewer account addresses. Since traffic and users in this shard are far less as compared to be a normal shard, Penny-spend attacks can be afforded in those backup shards by doing regular compression each time to save storage space.

4.4 Practical issues

A few key issues can arise in practice using sharding in the network: cross-shards communication, full node selection and shard number imbalance. Each one will be addressed in the following subsections.

Cross-shards communication

Even though the transactions from a sending address will always be directed to the same shard, the destination of the receiving end is not necessarily in the same shard. This would require finding the shard of receiver account by the router layer and sending a receiving request to the receiver account and updating the receiver account in another shard. If the updating is successful, the sender account is then updated. Otherwise, return an error message to user and ask user to try again later.

Full-node selection

Fixed at beginning. Controlled by monitor layer. When the monitor layer finds out that some full nodes are down, it shall automatically add new full nodes to this shard.

The number of shards

The number of shards will be fixed at beginning, and this aspect will be controlled by the monitor layer. When more than half of full nodes find their traffic exceeds the limitation they can afford, a new shard can be created and half of the requests redirected to the new shard by dividing the hash range of original shard into 2 equal ranges.

5 Artificial Intelligence

Artificial intelligence (AI) is not a new concept. It was widely discussed in the 1990s and has recently become a hot topic again thanks to the surplus of data generated by newer internet-capable devices, such as mobile phones and specific websites. There are two major applications of AI in JURA:

1. Transaction filtering: the characteristics of transactions will be studied and learned in an offline learning fashion and be posted as the first layer after the receiving the transactions. Pattern recognition techniques based on features in the transaction data, such as timestamp, balance, etc, can be used to infer the extremeness of the transaction. If it is deemed suspicious, a soft lock to the address can be issued, and a notification to the sender address will be sent out. If the notification is confirmed, the soft lock is released, otherwise, it will become a hard lock, and elicit stricter investigative actions.
2. Malicious node detection. System-wise node-level information can be recorded and again studied in an offline fashion. Nodes that appear to be outliers or extremes can be studied further to prevent malicious attacks.

There are 3 module of AI framework:

1. Online Logging Module

   In this module, we will select features of transaction behaviors which processed by each fullnode, and the fullnode behaviors, and log them in each fullnode to collect training data. At beginning, we will fake abnormals transaction behaviors and malicious nodes with labels. In the future, we will have human reviewed data for training.

2. Offline Training Module

   We will gather all the data from each fullnode, doing preprocessing and training model. This is continuous process, we will update our model once a month to make sure our model getting better and better during serving time.

3. Online Serving Module

   We will serve our model in monitor layer which has two places. One is located in each fullnode which is used to detect abnormals transaction, the other one is located in the groups of monitors, a special role in the architectural which is used to detect malicious node.

6 Attack vectors

JURA, like all decentralized cryptocurrencies, may be attacked by malicious parities for attempted financial gain or system demise. In this section, a few possible attack scenarios are outlined, the consequences of such attacks, and how JURA takes preventative measures.

6.1 Monopoly problem

If a single entity (hereafter a monopolist) takes control of the majority of staking resources such as in the traditional PoS systems, it could use these resources to impose conditions on the rest of the network. Potentially, the monopolist could choose to do this in malicious ways, such as double spending or denying services. If the monopolist chooses a malicious strategy and maintained this control for a long period, confidence in the cryptocurrency would be undermined and the currency purchasing power would collapse. Even though the monopolist could choose to act benevolently, it is centralized, and the system is monitored and controlled by a single party.

PoU systems make establishing a monopoly extremely difficult: in addition to the arguments the PoS systems make about this problem, such as the high cost of acquiring more than 50% of the stakes, no economic benefits if behaving maliciously, etc., PoU simply makes the definition of monopoly hidden and no one, even the core developers would not have the knowledge of how to obtain more than 50% of the voting power. The voting power of every single entity is a result of many factors in a game-theoretic approach in the population, and the non-linear transformation via CDF from the stake size to the marginal utility eliminates the possibility of dominating in any sort. This analogy can be found in the difference between mean and median. An outlier can change the mean by a large amount but it would not change the median no matter how far off it is.

6.2 Block gap synchronization

A transaction may not be properly broadcasted, causing the network to ignore subsequent blocks. If a node observes a block that does not have references to a previous block, it has two options:

1. Ignore the block as it might be a malicious garbage block.
2. Request a resync with another node.

In the case of a resync, a TCP connection must be formed with a bootstrapping node to facilitate the increased amount of traffic a resync requires. However, if the block was actually a bad block, then the resync was unnecessary and needlessly increased traffic on the network. This is a Network Amplification Attack and results in a denial-of-service.

To avoid unnecessary resyncing, nodes will wait until a certain threshold of votes have been observed for a potentially malicious block before initiating a connection to a bootstrap node to synchronize. If a block does not receive enough votes it can be assumed to be junk data.

6.3 Double spending

Double spending can happen in multiple ways: one way is by users who deliberately send the same balance to two recipients. In this case, a double spending attack is basically not possible given the design of Fusus. The second sending action. Once the first send action is done, the balance would be put on hold, and the new send action would not be able to spend the same balance twice. Different from blockchain's validation-at-confirmation model for the prevention of double spending. Whenever a sending block is activated, it has to be issued based on the ending balance of either the previous sending block or the total of all pending previous receiving blocks. Given the above time constraint in making sending blocks, the second sending block must have an updated balance of the account which must be able to indicate whether the balance is enough for making the payment. Hence, the Fusus prevents double spending from happening, i.e., once a user sends a certain amount of balance to another address, the exact same piece of balance cannot be

sent to another address at the same time unless the first balance did not go through, or the balance was increased with receiving blocks such that there is now enough balance.

Double spending can also happen due to wrong programming and poor network environment. If this were not to be prevented, validators would vote for consolidating the conflicts. Since the voting power is hidden to everyone and is dynamically changing with respect to the population, no single party would have the confidence and motivation to vote otherwise. Besides, penalty will be imposed for voting dishonestly.

6.4 Sybil attacks

In this situation, an entity can create many accounts with the intention of gaining disproportional voting power. However, the requirement of having a non-zero account opening fee makes this attack expensive at the first place. Besides, the PoU consensus mechanism discounts the new stakes with the CST, AIST and LIST metrics as weights. The economic cost is high and the systematic rules discourage this type of behavior. The chance of a successful Sybil attack is therefore remote.

6.5 Penny-spend attacks

The Nano and IOTA networks were all at a point suffered from large amount of zero-value spam attacks at various points. This type of attacks falls under the umbrella of penny-spend attacks as well. In the network of JURA, the spam attacks could not happen in the first place given the VRF mechanism approach (instead of PoW anti-spam). Even if the attacks were to happen, on the receiving-side, the Fusus has the capacity to absorb and consume the large amount of sending blocks fast and hence, no congestion would happen.

To be specific, the PoVRT required for issuing a sending block simply makes issuing an unreasonable amount of transactions exponentially harder and harder, i.e., the first sending action since the last sending action or a thresh-hold has zero time elapsing requirement, however, the immediate sending actions will be forced to be separated by the VRFs. Again, the basic mechanism of sending blocks and PoVRT prevent this type of attacks.

7 Token economy

In a feeless system like JURA, for the common users, the friction in transactions is reduced to a minimal amount and the volume should be reversely increased significantly. It also allows players of small capitals to take a share of the benefits the network brings, by reducing the $3\% - 5\%$ merchant fees such that the applications can be adopted by the majority of people.

For the nodes that are providing the resources to sustain the operation of the system, there are two layers of incentives: one is implicit and hidden, and the other is explicit by requiring the nodes to provide certain functionalities to the system.

Implicitly, business operations relying on the technology will run full nodes to support the network to satisfy the commercial needs. The cost of running full nodes are cheap, approximately three dollars per month, yet the small businesses that use traditional payment vendors (Visa, Mastercard, Amex) would pay anywhere from $3\% - 5\%$ of all sales, which are much higher than running a JURA full node.

Explicitly, a direct source of incentives is available from the penalties in the staking system, i.e., if a participant in the staking process does not do the voting job right, the stake will be collected and distributed as incentives to the full nodes. Besides, a much larger pool of incentives is only open to full-node runners who can take at least one of the three primary roles: judger, storager, and coordinator. The details of each

role are introduced in the following subsections.

7.1 Judger

A judger is a role that is chosen by the PoU system to validate transactions and to resolve conflicts. A group of few stake holders selected in a periodic cycle to form a committee to resolve issues like conflicts of transactions, changes of protocols. They are incentived with JURA when making right decisions and penalized for the incorrect and obvious mistakes they make for private benefits or gains.

Those judgers who put stakes with high utilities will be chosen in every hour, and the size of the committee should be a relatively big odd number, like 101. The rationale is to protect the committee from Byzantine faults and the voting results would be more stable.

Given the nature of PoU mechanism, the 101 chosen judgers will be expected to be of good intentions in the first place. A number like 101 should be more than sufficient to maintain the consistency of the system.

The incentives go to these judgers in every 1 hour cycle. The PoU mechanism ensures that it is not a system of "Rich gets richer", but rather a system of "High utility gets richer". This creates a strong incentive for all users to participate in staking.

7.2 Storager

Storager are defined as storage entities that can store large, unstructured data (image, audio, video). Storager help reduce the cost and complexity of storing content for web and IoT applications. Unlike FileCoin, JURA enables fast store and retrieval of the stored data with no induced latency, something that Filecoin does to enable replication integrity. Additionally, JURA does not cater to Retrieval markets, and expect to perform a CDN (content delivery network) like function. Also, storagers on JURA would typically compose of a select group of full nodes with high network bandwidth connectivity as opposed to home storage devices. The storagers on the network of JURA obtain reward of JURA tokens based on amount of data added per cycle. Data is distributed evenly among storagers.

7.3 Coordinator

In the monitor layer of sharding architect introduced previously, the full nodes will be constantly monitoring the information of the shards by doing periodic computation, communication and occasionally alerting. The nodes that will provide such functionalities are called coordinators. They will be incentived with JURA tokens also periodically based on again a shard-wide evaluation of the contribution made by the coordinators.

7.4 Reward pool

The JURA reward pool will be reduced over time, as inflation protocol regulate the number of tokens given out over a period. As the number of applications increase, the reward pool will be used more often for the judgers, storagers, coordinators. This may increase the inflation rate of the rewards. The inflation protocol sets the reward size based on the bid rate of the incentivized entities, and this rate may change daily as the market price of the token changes, for the mutual benefit of the network and the miners.

7.5 Usage of token

The native digital cryptographically-secured utility token of the JURA Network (JURA) is a major component of the ecosystem on the JURA Network, and is designed to be used solely on the network. JURA will initially be issued as ERC-20 standard compliant digital tokens on the Ethereum blockchain, and these will be migrated to tokens on the blockchain of the JURA Network when the same is eventually launched.

In addition, JURA is required as virtual crypto fuel for using certain designed functions on the JURA Network, thus providing the economic incentives which will be consumed to encourage participants to contribute and maintain the ecosystem on the JURA Network. Computational resources are required for validation of transactions, execution of sharding techniques, as well as for providing storage space for other nodes, thus providers of these services / resources would require payment for the consumption of these resources (i.e. "mining" on the JURA Network) to maintain network integrity. JURA will be used as the unit of exchange to quantify and pay the costs of these consumed computational resources.

JURA is an integral and indispensable part of the JURA Network, because without JURA, there would be no incentive for users to expend resources to participate in activities or provide services for the benefit of the entire ecosystem on the JURA Network. Users of the JURA Network and/or holders of JURA which did not actively participate will not receive any JURA incentives.

JURA does not in any way represent any shareholding, participation, right, title, or interest in the Foundation, its affiliates, or any other company, enterprise or undertaking, nor will JURA entitle token holders to any promise of fees, dividends, revenue, profits or investment returns, and are not intended to constitute securities in Singapore or any relevant jurisdiction. JURA may only be utilised on the JURA Network, and ownership of JURA carries no rights, express or implied, other than the right to use JURA as a means to enable usage of and interaction with the JURA Network.

In particular, you understand and accept that JURA:

1. is non-refundable and cannot be exchanged for cash (or its equivalent value in any other virtual currency) or any payment obligation by the Foundation or any affiliate;

2. does not represent or confer on the token holder any right of any form with respect to the Foundation (or any of its affiliates) or its revenues or assets, including without limitation any right to receive future dividends, revenue, shares, ownership right or stake, share or security, any voting, distribution, redemption, liquidation, proprietary (including all forms of intellectual property), or other financial or legal rights or equivalent rights, or intellectual property rights or any other form of participation in or relating to the JURA Network, the Foundation, the Distributor and/or their service providers;

3. is not intended to represent any rights under a contract for differences or under any other contract the purpose or pretended purpose of which is to secure a profit or avoid a loss;

4. is not intended to be a representation of money (including electronic money), security, commodity, bond, debt instrument or any other kind of financial instrument or investment;

5. is not a loan to the Foundation or any of its affiliates, is not intended to represent a debt owed by the Foundation or any of its affiliates, and there is no expectation of profit; and

6. does not provide the token holder with any ownership or other interest in the Foundation or any of its affiliates.

The Distributor of JURA shall be an affiliate of the Foundation. Approximately 50% of the total token supply issued by the Distributor will be reserved for mining rewards. The contributions in the token sale will be held by the Distributor (or its affiliate) after the token sale, and contributors will have no economic

or legal right over or beneficial interest in these contributions or the assets of that entity after the token sale. To the extent a secondary market or exchange for trading JURA does develop, it would be run and operated wholly independently of the Foundation, the Distributor, the sale of JURA and the JURA Network. Neither the Foundation nor the Distributor will create such secondary markets nor will either entity act as an exchange for JURA.

8 Risks

You acknowledge and agree that there are numerous risks associated with purchasing JURA, holding JURA, and using JURA for participation in the JURA Network. In the worst scenario, this could lead to the loss of all or part of the JURA which had been purchased.

8.1 Uncertain Regulations and Enforcement Actions

The regulatory status of JURA and distributed ledger technology is unclear or unsettled in many jurisdictions. The regulation of virtual currencies has become a primary target of regulation in all major countries in the world. It is impossible to predict how, when or whether regulatory agencies may apply existing regulations or create new regulations with respect to such technology and its applications, including JURA and/or the JURA Network. Regulatory actions could negatively impact JURA and/or the JURA Network in various ways. The Foundation (or its affiliates) may cease operations in a jurisdiction in the event that regulatory actions, or changes to law or regulation, make it illegal to operate in such jurisdiction, or commercially undesirable to obtain the necessary regulatory approval(s) to operate in such jurisdiction. After consulting with a wide range of legal advisors and continuous analysis of the development and legal structure of virtual currencies, the Foundation will apply a cautious approach towards the sale of JURA. Therefore, for the token sale, the Foundation may constantly adjust the sale strategy in order to avoid relevant legal risks as much as possible. For the token sale the Foundation is working with Tzedek Law LLC, a boutique corporate law firm in Singapore with a good reputation in the blockchain space.

8.2 Inadequate disclosure of information

As at the date hereof, the JURA Network is still under development and its design concepts, consensus mechanisms, algorithms, codes, and other technical details and parameters may be constantly and frequently updated and changed. Although this white paper contains the most current information relating to the JURA Network, it is not absolutely complete and may still be adjusted and updated by JURA from time to time. JURA has no ability and obligation to keep holders of JURA informed of every detail (including development progress and expected milestones) regarding the project to develop the JURA Network, hence insufficient information disclosure is inevitable and reasonable.

8.3 Competitors

Various types of decentralised applications are emerging at a rapid rate, and the industry is increasingly competitive. It is possible that alternative networks could be established that utilise the same or similar code and protocol underlying JURA and/or the JURA Network and attempt to re-create similar facilities. The JURA Network may be required to compete with these alternative networks, which could negatively impact JURA and/or the JURA Network.

8.4 Failure to develop

There is the risk that the development of the JURA Network will not be executed or implemented as planned, for a variety of reasons, including without limitation the event of a decline in the prices of any digital asset, virtual currency or JURA, unforeseen technical difficulties, and shortage of development funds for activities.

8.5 Security weaknesses

Hackers or other malicious groups or organisations may attempt to interfere with JURA and/or the JURA Network in a variety of ways, including, but not limited to, malware attacks, denial of service attacks, consensus-based attacks, Sybil attacks, smurfing and spoofing. Furthermore, there is a risk that a third party or a member of the Foundation or its affiliates may intentionally or unintentionally introduce weaknesses into the core infrastructure of JURA and/or the JURA Network, which could negatively affect JURA and/or the JURA Network. Further, the future of cryptography and security innovations are highly unpredictable and advances in cryptography, or technical advances (including without limitation development of quantum computing), could present unknown risks to JURA and/or the JURA Network by rendering ineffective the cryptographic consensus mechanism that underpins that blockchain protocol.

8.6 Other risks

In addition, the potential risks briefly mentioned above are not exhaustive and there are other risks (as more particularly set out in the Terms and Conditions) associated with your purchase, holding and use of JURA, including those that the Foundation cannot anticipate. Such risks may further materialise as unanticipated variations or combinations of the aforementioned risks. You should conduct full due diligence on the Foundation, its affiliates and JURA, as well as understand the overall framework, mission and vision for the JURA Network prior to purchasing JURA.

9 Conclusions

This paper proposed a system for a decentralized peer-to-peer network with ultrahigh TPS and minimal to zero cost. The JURA protocol revolves around a novel data structure Fusus which lays down a solid processing infrastructure, and is furthermore enhanced with a PoU system that makes a reliable self-regulated consensus possible. Furthermore, AI adds an additional layer of safeguarding, and sharding provides for vastly increased scalability. This proposed network therefore is extremely robust in its structural clarity and statistical governance model. The JURA team is confident therefore that the technical advancements of the JURA protocol will lead to widespread adoption in various industries.

References

[1] S. Nakamoto. (2008) “Bitcoin: A peer-to-peer electronic cash system”.

[2] V. Buterin. (2013) “Facilitating Online Contractual Agreements With Vitalik Buterin”. *Eyerys.com.*

[3] S. Popov (2016). “The tangle”

[4] C. LeMahieu. (2017). “Nano: A feeless distributed cryptocurrency network”.

[5] Y. Dodis; A. Yampolskiy. (2005). "A Verifiable Random Function With Short Proofs and Keys". *8th International Workshop on Theory and Practice in Public Key Cryptography.* pp. 416–431.

[6] B. Pugh. (2018). "Stress Testing The RaiBlocks Network: Part II". *https://medium.com/@bnp117/stress-testing-the-raiblocks-network-part-ii-def83653b21f.*

[7] S. Micali; M. Rabin; S. Vadhan. (1999). "Verifiable random functions". *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science.* pp. 120?130.

[8] S. King; S. Nadal. (2012). "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake".

[9] D. Pike; P. Nosker; D. Boehm; D. Grisham; S. Woods; J. Marston. (2015). "A time-accepted periodic proof factor in a nonlinear distributed consensus".

[10] D. Boneh; B. Lynn; H. Shacham. (2001). "Short Signatures from the Weil Pairing". *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Crytology*, ASIACRYPT'01, pages 514-532, London, UK, UK. Springer-Verlag.

[11] A. Justel; D. Pea; R. Zamar. (1997). "A multivariate Kolmogorov-Smirnov test of goodness of fit". *Statistics & Probability Letters.* **35** (3): 251-259.

[12] N. Szabo. "Smart Contracts: Building Blocks for Digital Markets". `www.fon.hum.uva.nl`. Retrieved 2017-07-29.

[13] `https://github.com/ethereum/casper`