

# My Guess at Peter's Distributed Consensus Networks

---


**Jeffrey Morais<sup>a</sup>, Peter P. Rohde<sup>b,c</sup>**

<sup>a</sup>*Department of Physics, McGill University, Montréal, Québec, Canada*

<sup>b</sup>*Centre for Quantum Software & Information (UTS:QSI), University of Technology Sydney*

<sup>c</sup>*Hearne Institute for Theoretical Physics, Department of Physics & Astronomy, Louisiana State University, Baton Rouge LA, United States*

*E-mail:* [jeffrey.morais@mail.mcgill.ca](mailto:jeffrey.morais@mail.mcgill.ca), [dr.rohde@gmail.com](mailto:dr.rohde@gmail.com)

**ABSTRACT:** Following the papers on boson sampling, the rough draft of the distributed consensus network paper, and the slides on these networks, I present my understanding of these to be reviewed by Peter. Furthermore, I also include potential additions to further it with the use of topology and statistical mechanics. I begin by presenting the framework of the interaction of nodes and its application to blockchains. I then present proof-of-work as a consensus method for open networks and the inefficiencies associated with it. Next I look at classical decentralized consensus for closed networks, followed by its equivalent in the language of quantum theory. Finally, I present potential extensions of this analysis through the form of topological invariants and statistical mechanics. 

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Proof-of-work for open networks</b>	<b>2</b>
<b>3</b>	<b>Classical distributed consensus for closed networks</b>	<b>3</b>
3.1	Hash-based subset allocation	5
3.2	Quantum transformation allocation	6
<b>4</b>	<b>Quantum extensions to closed consensus networks</b>	<b>6</b>
<b>5</b>	<b>Potential extensions of analysis</b>	<b>8</b>
5.1	Topology	8
5.2	Statistical mechanics	10
<b>6</b>	<b>New Consensus Protocol</b>	<b>11</b>

---

## 1 Introduction

One could consider a system of parties which form a set known as a *network*, where each an element is referred to as a *node*. Much like an ensemble of particles, these nodes can interact with one another and also be subject to external constraints. Interactions between said nodes could include participation in *transactions*, and external interactions of the nodes could be characterized with centralized arbiters that oversee said transactions. Typical transactions that occur between nodes in nature are arbitrated by the central authority of a bank which can take a cut of the transaction between the nodes, as well as manipulate the information of said transaction. Naturally, it would be beneficial to the nodes to minimize the degrees of freedom between their interaction and remove the intermediate middle man taking form of a centralized authority; the bank. These independent transactions however are subject to parties colluding with one another to rig transactions between nodes without the oversight of a central authority that arbitrates the interaction to makes sure of its legitimacy. Thus, we would like a decentralized system to process transactions between nodes in a network of parties with ample security to avoid unlawful manipulation and rigging of interacting nodes. Herein lies the concept of blockchains.

A *blockchain* is a data structure whose information is stored non-locally across the network of nodes, where each element of the blockchain — known as a *block* — contains the data of a transaction event. For the purposes of security this data is encrypted via a function whose inverse is computationally infeasible to compute (known as a *hash*), and each block contains information of the previous block in the chain. The combination of the hash and information being stored in subsequent blocks prevent the transaction block from

being manipulated. Once an additional block is added to the chain, its information is fixed and cannot be altered. Now, one would wonder how to control the legitimacy of transactions when populating the blockchain if there are no central arbiters. This would have to be some form of *consensus* between the nodes of the network in which an agreement is made on the validity of the transaction and the order in which it is added to the blockchain. Such a consensus algorithm wouldn't only apply in the context of cryptography but could potentially lend a hand in game theory where low-risk bartering events between parties could be guaranteed to be secure without having to rely on a state of Nash equilibrium. We first consider consensus in open networks for which any external nodes could join in, followed by the case of closed networks.

## 2 Proof-of-work for open networks

An *open* network is one in which external nodes can join the network freely to participate in consensus events to agree on the validity of transactions. One could see this as more susceptible to colluding amongst nodes as an arbitrary amount of malicious/corrupt nodes can join to attempt to skew the consensus in their favour, which is why we consider closed networks in the following section for a fixed amount of corrupt nodes. For now we consider the more popular case of open nodes, specifically under the consensus algorithm known as *proof-of-work*. Given a transaction event which contains a hash for which it is infeasible computationally to solve for its inverse, à la style of brute force nodes compete to figure out the hash's input via sampling a random distribution of inputs. This in itself is extremely inefficient, and so does not fall into the complexity class **P** in theoretical computer science, a class of problems that can be quickly solved in polynomial orders of time. Once a node randomly stumbles upon the correct input and shares it with the rest of the network, the transaction can be quickly verified by the other nodes, and so this process of verification belongs to the **NP** complexity class which is the set of problems for which could be verified quickly in polynomial time. Once verified and thus consensus has been made, that node which initially found the input is rewarded with a portion of the transaction, and a its block is added to the blockchain. Problems present themselves when attempting to have a scalable proof-of-work consensus. The first is the egregious energy costs of a single transaction, which are many orders of magnitude higher than one arbitrated by a bank. The other is artificially fixing the transaction rate to prevent degeneracies in the pool of winners which compute the hash input first, and prevent inflation of the underlying currency.

What if we make use of quantum computers to compute these inputs? While classically the amount of inputs used to solve the hash problem is of the order  $\mathcal{O}(N)$ , where  $N$  is the size of the domain of the hash function, quantum algorithms can solve the problem more quickly given that they only need  $\mathcal{O}(\sqrt{N})$  inputs. Why not make use of quantum computers for consensus in networks one might ask, given a quadratic speedup. The problem with using quantum computers for solving unstructured problems with inverting injective functions is that the condition that the process is *progress-free* would no longer hold. This would mean that nodes that have solved previous block puzzles would have an advantage

over ones that have just joined to solve the current hash input. What we *can* do to still take advantage of the framework that quantum theory holds over classical computations is by *sampling*. Recall that the use of quantum mechanics is that one can store information non-locally through entanglement for which there is associated uncertainty. This presents quite the incentive to save on computational demand as one would not need to collapse the state until measurement and so working with a state in superposition which contains all possible information of the system is more efficient. This of course, comes with its own share of problems with scalability as the uncertainty increases further and the need for quantum error correction and fault-tolerant codes come into play, but that is beyond the scope of this write-up. Unlike a decision problem in which a precise solution is given to a problem (the hash input), with sampling you take measurements from a large superposition and converge to the solution given some estimated probability. One for instance could make use of *boson-sampling*, which is a consensus algorithm that estimates (via convergence) the expectation value of matrix permanents for boson scattering events. This could be implemented as a proof-of-work consensus algorithm for networks verifying the validity of a transaction when populating a blockchain. The rough idea of a boson sample is to prepare multi-particle state which has some distribution of vacuum states for measurement, and some distribution of first excited states for the bosons. The different states of the multi-particle state — known as *modes* — are then permuted and transformed under arbitrary unitary operations before being measured. The unitary transformations which are arbitrarily chosen come from the *Haar* measure which randomly assigns volume invariants (probabilities) to subsets of locally compact topological groups from which the unitary operators reside. If agreeing on the measurement of the output state, consensus could be achieved.

However, the fact still remains that open networks pose problems for minimizing the amount of malicious nodes, and consensus via proof-of-work is still heavily inefficient. For these reasons we move onto an alternate method of consensus for closed networks in the following section.

### 3 Classical distributed consensus for closed networks

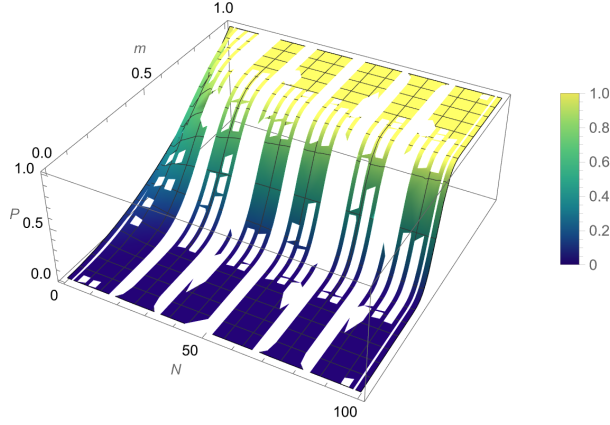
We consider instead a *closed* network in which the amount of nodes (and the fraction of which are malicious) is fixed. It is noted while these nodes can interact indirectly through means of consensus, they cannot interact *directly* with each other. The possibility of direct interaction is considered in the section where we present extensions to this analysis with the use of topology and statistical mechanics. For the purposes of consensus, we would like a method that is highly resistant to malicious nodes colluding to manipulate the outcome of consensus by forcing an unlawful majority. In this endeavour we could prevent this by splitting these malicious nodes into subsets and taking local consensus from these sets. That being said, however, we don't have the information of which nodes are malicious, only their fraction of the total network. Thus to the maximal degree, the allocation of these random subsets must be *random*, as to disperse these malicious parties among honest ones and suppress their strategy. If a subset of the network forms consensus, and it reflects the

will of the majority, the colluding minority will fail. Furthermore, we would like to also minimize the possibility of the malicious minority rigging the overall consensus by chance. Thus, we want these subsets — known as *consensus sets* — to be chosen to suppress this possibility.

First, how do we allocate these different subsets which will be used to consensus? First, let the set of all nodes  $\sigma$  be the network set  $\Sigma$ , where the fraction of malicious nodes is  $m$ . For some subset  $\chi \subset \Sigma$  of length  $N$ , the probability that this subset constrains a majority of dishonest nodes is give by:

$$P(N, m) = 1 - I_{1-m} \left( \left\lfloor \frac{N}{2} \right\rfloor + 1, \left\lfloor \frac{N}{2} \right\rfloor + 2 \right), \quad (3.1)$$

where  $I_a(b, c)$  is the regularized incomplete (analytically continued) beta function. Obviously for  $P = 0$ , then consensus achieved from the subset is trivially in favour of an honest majority, and so we would like an algorithm for which a non-trivial  $P$  still results in an honest majority for consensus. It is noted that the magnitude of  $P$  is independent of the distribution of dishonest nodes in the subset and so it suffices to work with  $P$  and  $m$  globally instead of trying to compute the fractions of  $m$  for the subsets, which we couldn't know being that these subsets are randomly partitioned. For a given subset  $\chi$ , we want to minimize the the likelihood that a localized corrupt minority gains majority vote, and so we desire an upper limit on that probability as  $P(N, m) \leq \varepsilon$  for  $\varepsilon \ll 1$ . For what size of the subset  $\chi$  could we achieve this? We derive this information by looking at the behaviour of  $P$  via the following figure:



**Figure 1.** Probability  $P$  as a function of the size  $N$  of the subset  $\chi$ , and the ratio of malicious nodes in the network given by  $m$ . Note the discontinuities come from the nature of the analytically continued beta function.

We see for a fixed corruption fraction  $m$ , the smaller the subset size, the lower the probability of having a dishonest majority in the subset decreases. However this is trivial for the minimizing case where  $N = 0$  and hence  $P = 0$ . Thus we impose the constraint

$P \leq \varepsilon$  which gives us the extremal size  $N_e$  of a non-trivial set which minimizes  $P$ , given by:

$$N_e(m, \varepsilon) = \arg \min_N \left( P(N, m) \leq \varepsilon \right) \quad (3.2)$$

With this constraint it follows that the size of all subsets  $\chi \subset \Sigma$  are of size  $N_e$ . Now that we have a condition on the size of the subsets, we should consider how exactly the partitioning of  $\Sigma$  into its constituent subsets is done. We first consider hash-based random subsets, and then move onto a more robust quantum formulation of it.

### 3.1 Hash-based subset allocation

The first method in which to securely and randomly allocate random subsets for consensus is via using *hash* functions. These are functions which are injective — and so have a unique input and output — and additionally are *one-way* functions meaning their inverse is computationally infeasible to calculate in finite polynomial time. Consider a hash function  $h \in C^\infty(\Sigma)$ . To participate in consensus and thus be partitioned into subsets, each node places a deposit which is returned if they show no signs of corruption, meaning they are *compliant*. Compliant nodes which follow through with consensus and agree with majority vote are kept in the network and will receive a small portion of the transaction.

Now, to participate each node puts up a deposit and announces in a shared channel  $\mathcal{B}$  a *salted* hash of the information  $\mathcal{A}$  of the previous block. The term *salt* refers to a random unique value  $S_\sigma$  for which is only known to the individual nodes  $\sigma$  for security. The salted hashes which are announced by each node on the channel are given by:

$$H_\sigma = h(\mathcal{A}; S_\sigma). \quad (3.3)$$

This gives us global information, known as a *global key*  $K$ , which is defined as the hash acting on the union of all individual hashes:

$$K = h \left( \bigcup_{\sigma} H_\sigma; 1 \right). \quad (3.4)$$

Individual keys which allocate different nodes to different subsets are then computed via yet another hash:

$$K_\sigma = h(H_\sigma; K). \quad (3.5)$$

We see that obtaining these keys  $K_\sigma$  — which are essentially the indices which assign different nodes to different subsets of size  $N_e$  — have the hash function applied three times. This means it is virtually *impossible* for a node to compute the salt of another node, given the order of the hash functions applied is  $\mathcal{O}(h^3)$ . As long as a single initial hash  $H_\sigma$  is honest, the assignment of consensus subsets will be random and cannot be manipulated by dishonest nodes. Now for a given subset  $\chi$  whose nodes have been selected randomly from applying the hash function, we proceed with consensus of a transaction for a new block to be added to the blockchain.

Each node within  $\chi$  gives an estimate of the time(stamp) at which the current transaction to be added to the block chain was done at. The time given by each node in the subset is denoted as  $t_\sigma$ , and the majority vote of the time of transaction is given by the median of the times:  $\text{median}(\{t_\sigma\})$ . In order for a node in the subset to be considered compliant and not malicious, the following constraint must hold:

$$|t_\sigma - \text{median}(\{t_\sigma\})| \leq \delta. \quad (3.6)$$

Here  $\delta \ll 1$  is some small parameter and the condition tells us that each node must have a time estimation sufficiently close to the majority vote (the median) to be considered compliant. Being that the way in which the random subsets cannot be tampered with by malicious nodes, it suffices that the majority vote — i.e. what the consensus of the time of the transaction was — of the subset reflects the will of the majority of the network. Once done, and consensus has been achieved, the compliant nodes receive their initial deposit and a portion of the transaction, and the transaction is appended to the blockchain as a new block.

### 3.2 Quantum transformation allocation

The classical algorithm used to randomly allocate subsets for consensus makes use of hash functions, which although cryptographically robust, are deterministic and not random. To deal with this we instead use of *quantum random* numbers which make use of measuring quantum states that have undergone arbitrary unitary transformations. One for example could consider the CNOT operator, denoted as  $\Lambda$ , which is used to entangle and disentangle quantum Bell states. One possible adjoint representation of this operator would be:

$$\Lambda = \frac{1}{2}\mathbb{1} \otimes \mathbb{1} + \frac{1}{2}\sigma_z \otimes \mathbb{1} + \frac{1}{2}\mathbb{1} \otimes \sigma_x - \frac{1}{2}\sigma_z \otimes \sigma_x, \quad (3.7)$$

where  $\sigma_i$  are the usual Pauli matrices. Acting with arbitrary linear combinations of this on quantum states (one quantum state for each node) and measuring the outcome apparently gives a sufficiently random output that is much more random than deterministic hash functions. These would give us the initial random numbers  $H_\sigma$ , for which we can define the global and individual keys respectively as:

$$K = \bigoplus_{\sigma} H_\sigma, \quad K_\sigma = H_\sigma \oplus K. \quad (3.8)$$

Here the symbol  $\oplus$  denotes the XOR operator which returns true if only one of the inputs is true. From this we could play the same game of allocating subsets of the network for consensus through compliance measures of some transaction time.

## 4 Quantum extensions to closed consensus networks

We now turn towards a fully quantum framework which makes use of *trapdoor claw-free* (TCF) functions and quantum conditions for proof consensus being achieved. As an alternative to classical consensus (denoted as *proof-of-consensus*), we instead turn our focus to

proofs of quantumness. In this case, we have two nodes in a closed network  $\Sigma$  consisting of a *prover* and a *verifier*. The goal is for the prover to convince the verifier that they have successfully done a quantum implementation of some function, and so is a compliant party.

Although the functions we considered before were injective functions whose inverse was computationally infeasible of calculating, finding the required input is done efficiently via quantum computers. These functions thus are no longer robust against evolving quantum technology. To adapt against this breach in security, we make use of 2-1 functions which have two unique inputs which match to the same output. Still falling under the condition of an extremely difficult inverse to compute, it is difficult even for quantum computers to find simultaneous pairs of inputs that satisfy the same output. Specifically, we make use of trapdoor claw-free functions  $f$  for which two inputs  $(x_0, x_1)$  give the same output  $\omega$  as:

$$f(x_0) = f(x_1) = \omega. \quad (4.1)$$

The term *claw* refers to two different input mapping to the same output. It is very difficult for the prover to find these inputs, while the verifier has extra information known as a *secret* for which they can very efficiently find the two inputs to determine the validity of the prover's work. This is known as using a backdoor or *trapdoor* and falls within the **NP** complexity class. We utilize this to characterize compliance of a node in some randomly assigned subset.

First, consider a TCF function  $f$  for which two inputs  $(x_0, x_1)$  satisfy the same output  $\omega$ . The verifier specifies a problem where they have an associated secret for efficiently verifying the work. The equivalent of brute force guessing by the prover is done by preparing a quantum state of all possible bit strings as:

$$|\psi\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |0\rangle, \quad (4.2)$$

where  $H$  is a Hadamard gate operator which causes a rotation of  $\pi$  about the  $(\hat{x} + \hat{y})/\sqrt{2}$  axis on the Bloch sphere of a state,  $H^{\otimes n}$  is shorthand notation for  $H$  being in a tensor product with itself  $n$ -times, and  $x \in \{0,1\}^n$  represents a string of  $n$ -different bit values. Applying the TCF function on this state changes the initialized vacuum kets  $|0\rangle$  to new kets  $|f(x)\rangle$ . The reason for first initializing with vacuum kets instead of acting it directly on the  $|x\rangle$  kets is to preserve unitarity and reversibility of these operations, which a crucial condition in quantum theory to preserve amplitudes with time. Next, given this ket which has been initialized with all possible bit strings, and has been acted on by the TCF, to prove a quantum implementation of the function, the prover collapses the state to:

$$|\psi\rangle = \frac{1}{\sqrt{2}} (|x_0\rangle + |x_1\rangle) \otimes |\omega\rangle. \quad (4.3)$$

Once the measurement has been made, it is committed on the block and cannot be modified (neither could it be un-collapsed anyway). With this information the prover in essence could provide the two inputs  $(|x_0\rangle, |x_1\rangle)$  which map to the output  $|\omega\rangle$  under the TCF function. This information is communicated to the verifier *classically*, and so the



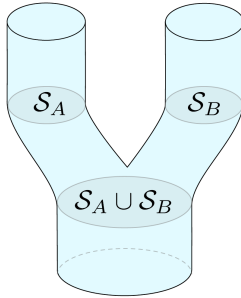
way to do this is by the verifier specifying a random basis. It would be easy enough to guess the inputs if always asked for the same basis, but upon the verifier alternating basis, this becomes very unlikely with each iteration for different  $\omega$ . The reason for such is that the measurements don't commute and so it is not possible to extract the information from both bases simultaneously. If measuring in the  $z$ -basis, the prover randomly measures either  $x = x_0$  or  $x = x_1$  and the verifier can easily validate them with the use of  $f$ , from which they can compare the prover's previous reported  $\omega$ . If measured instead in the  $x$ -basis, the prover gets a dispersion relation of the form  $x \cdot x_0 = x \cdot x_1$ . To pass verification in every round of iteration, the prover must measure in accordance to the verifier's requested basis. This makes it infeasible to rig as with every iteration the prover would have to spend computational resources to compute the inverse of a TCF for two starting inputs beforehand. For  $N$  rounds, the probability of the prover spoof guessing the inputs scales as  $1/2^N$  as so is asymptotically suppressed. Once finishing  $N$  rounds properly and the prover has successfully convinced the verifier of a quantum implementation of a function, the prover is seen as compliant and consensus passes.

## 5 Potential extensions of analysis

In this section we suggest potential extensions of what could be done with distributed consensus networks via the use of topology and statistical mechanics.

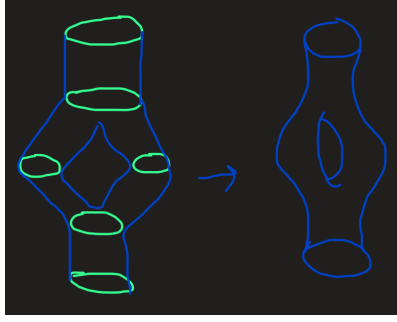
### 5.1 Topology

When nodes in a randomly chosen subset are not compliant, they not only lose their initial deposit and don't get a portion of the transaction to be put into a new block of the blockchain, but are also separated from the network as depicted in the following:



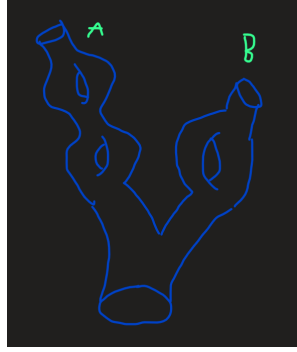
**Figure 2.** A network which comprises of a compliant subset  $S_A$  as well as a non-compliant subset  $S_B$ , splits into two separate networks.

These separate networks however can interact once more given that  $S_B$  (or at least a subset of it) turns out to be compliant in the next round of consensus. For the special case where the entire set becomes compliant again, we get the following figure:



**Figure 3.** A network which has forked into two non-interacting networks recombines again after a positive round of consensus. This forms a topological space of genus  $g = 1$ .

This seems easy enough. Although it is a good sign that the subset has been confirmed compliant after the second round, it would still be better had the network never split to begin with. In other words, if we think of the networks with time as a topological space, ideal networks are those with minimal genus. One could imagine a network that subdivides and recombines many time as:



**Figure 4.** A network which subdivides and recombines after many rounds of consensus. Branch  $A$  has a total genus  $g = 2$ , while branch  $B$  has a total genus of  $g = 1$ .

Multiple splittings of the network, and hence branches of splitting with higher genus, indicate instability and hint at underlying corruption of nodes. Thus we could characterize how non-compliant sub-networks are based on the genus of the branches it splits into, regardless of how the non-compliant nodes want to obscure that information. Thus an efficient algorithm to choose the most promising sub-networks for proof of consensus would be to follow along the branches with minimal genus.

The culprits could be even more clever and devise asymmetric patterns in the splitting of the network to have their strategy not immediately exposed. For this we would employ other topological indicators, such as link invariants which tell us information about different configurations of branches. In knot theory, these are literal knots that are isomorphic to the union of circles (more generally you can connect knots to get links), and tell us which knots are equivalent under certain moves. In our case, we would look at links which are isomorphic to unions of torii, to efficiently differentiate between different subdivided

networks. This would allow us to define unique levels of corruption amongst a large class of different branch splittings.

## 5.2 Statistical mechanics

Thus far we have only allowed nodes in a network to indirectly interact via some broadcast channel, but what if they were able to *directly* interact with one another? This would pose even more problems in terms of nodes colluding with each other, especially with the probability that one node *corrupts* another into colluding with it. This would be interesting to simulate with the Ising model, or higher dimensional forms of this such as the  $XY$ -model, in which at random points on nodes confined to a lattice, we can assign a probability of corruption. We could then simulate, taking advantage of multi-threading, how this would affect branch splitting and hence characterize how compliance evolves over time with each consensus event. We could further this analysis with the use of *quasi-nodes* which account for asymmetric colluding ratios within subsets of the network. In this framework, local subsets of the network can ask that their vote be done on their behalf, thus separating the network further (using the algorithm which randomly assigns subsets for consensus). In theory this would make the network more secure as which each iteration into clustering into quasi-nodes, the corrupt nodes would separate even further. What is more interesting however, is how direct interaction of nodes would affect the quasi structure, and we would see how the corruption spreads with time!

I hope you enjoyed reading this Peter, maybe we can put some of it in the paper draft?  
-Jeff

## 6 New Consensus Protocol

Abstractly we can consider a set of parties represented by some connected set — a *network* — whose elements which denote the different parties are called *nodes*. We can consider the interaction of such nodes whereby information can be shared directly or indirectly through some channel. However, there could be densities of information that require agreement to proceed which generically take on approval from majority vote: a *consensus*. In the game theoretic sense this could be used to describe sets of interacting parties wanting to barter information, or in the cryptographic sense this could be nodes wanting to update information on some ledger in an irreversible manner. A drawback of having a system of shared information which requires consensus (in a decentralized sense) is that subsets of nodes can collude to rig the system and put up a facade of majority vote. We want to create a protocol which is robust against manipulation given that the set of corrupt/colluding nodes strictly does not make up the entire network.

First, what we must consider is *incentive*. Why should a node spend time approving the information share — a general *transaction* — of another node? Instead of trying to come up with external incentives such as currency, we consider the incentive being *mutual benefit*. If a node wants one of their transactions approved through consensus, they too must participate in the approval of the transaction for another node, and so there is no debt and both parties benefit. Consider a set of nodes  $n_i$  which form a network  $\mathcal{N} = \{n_i\}$ , where  $i = 1, \dots, |\mathcal{N}|$  labels different nodes. Within this network we can consider a set of transactions  $\{t_j\} = \mathcal{T}$  requiring approval, and so one could look at the  $i$ -th node participating in the approval of the  $j$ -th transaction, to be an element of the product set:

$$n_i \otimes t_j \in \mathcal{N} \otimes \mathcal{T}. \quad (6.1)$$

For whatever reason, the node which proposed the  $j$ -th transaction could require a certain size of a consensus set  $c_{ij}$  which is parametrized by the  $\varepsilon$  parameter (see section 3). This could be interpreted as a node requesting for a certain level of robustness, which would require a stricter set (as long as it is not smaller than what is proposed in equation 3.2). The total information for a given node participating in the moderation of a given transaction — for a suggested consensus set size — is captured by the following:

$$\mathcal{R}_{ij} = \left( n_i \otimes t_j, c_{ij}(\varepsilon) \right). \quad (6.2)$$

For a given node participating in a set of transactions in the network, we simply denote this as the union of all transaction objects for a fixed node index:  $\mathcal{R}_i = \bigcup_j \mathcal{R}_{ij}$ . The nodes which want to participate in the set of transactions put forth salted-hashes (see section 3) from which the iterative action of a hash function generates public keys  $K_j^i$ . From this a global key  $K$  is formed and it is convolved with public keys via the same hash function to come up with unique numbers that allocate different nodes to different consensus sub-networks. From the *local* perspective of a node, the total set of participating nodes is the set of public keys, denoted as  $\mathcal{N}^{(i)} = \{K_j^i\}_j \subseteq \mathcal{N}$ . To participate in the set of transactions, each node commits the information tuple  $(\mathcal{R}_i, \mathcal{N}^{(i)})$  to some shared broadcast channel  $\mathcal{B}$ .

Committing refers to sharing the output of the salted-hash from which there is a unique salt. After all of the commits (generated by the salted hashes) are put forth and observed by all the nodes, each node then reveals the pre-image of the salted-hash from which all nodes can trivially verify if it outputs the same hash. All instances of sharing information on the broadcast channel is implicitly done via this commit-reveal scheme. Once all verified, we can begin the process consensus within different sub-networks.

Following verification, each node locally infers information of the global system given all the information from the broadcast channel. According to each node they *observe* a set of transactions  $\bar{\mathcal{R}} = \{\mathcal{R}_i\}$ , and a set of participating nodes  $\bar{\mathcal{N}} = \{\mathcal{N}^{(i)}\}$ . Before moving onto consensus, there is a network-agreed deterministic function  $f : \bar{\mathcal{R}} \times \bar{\mathcal{N}} \rightarrow \{0, 1\}$  which moderates if a transaction proposal is worth consideration for consensus to begin with. This could be for filtering out blatant malicious or impossible transactions<sup>†</sup>. Now, given the inferred information above, and the unique numbers which allocate the different nodes into sub-network consensus sets for a fixed transaction, all nodes participate in another round of committing the information of their assigned subsets, and then reveal the pre-image of their salted-hash sub-network index. The divided sub-networks, which are each allocated to a unique transaction, has each node within it vote (commit-reveal) on whether or not they approve the given transaction. For a given node  $i$  within the consensus set, we denote their vote as  $V_i(\bar{\mathcal{R}})$ .

Trivially we would tally up the votes and if the majority agreed with the transaction then it would go through and we would say we have achieved consensus. However, we must weed out corrupt nodes and the method of doing so will be by seeing which nodes are *compliant*. For a given consensus set — and hence transaction — each node reveals the time-of-receipt of all messages on the broadcast channel  $\mathcal{B}$ , and the median of the time for each message is considered to be the *consensus* time for a given message. If a node fails to report a time-of-receipt within a standard deviation (or stricter condition) of the consensus time, they are seen as *non-compliant*, and can potentially be trying to rig majority vote. Given this information, each node makes its own judgement on which nodes within the consensus set are compliant, and put forth this information on the channel. If a node is seen as non-compliant by the majority of the network, their vote  $V_i(\bar{\mathcal{N}})$  is not considered when approving or disapproving the associated transaction. Once the corrupt nodes have been separated from the network (and hence consensus sub-networks), the results of vote of each transactions given by each *compliant* node is tallied up, and if the majority approves a given transaction, then it follows through.

---

<sup>†</sup>For example, given a transaction that involves currency, it would be a waste of time to form consensus for a transaction that involves a diverging magnitude.