DECENTRALIZED REPUTATION MODEL AND GENERAL TRUST FRAMEWORK BASED ON BLOCKCHAIN AND SMARTCONTRACTS

MASTER PROGRAMME IN COMPUTER SCIENCE

UPPSALA
UNIVERSITET

Uppsala University
Department of Information Technology

Master's Thesis
SUJATA TAMANG

September 21, 2018

DECENTRALIZED REPUTATION MODEL AND GENERAL TRUST FRAMEWORK BASED ON BLOCKCHAIN AND SMARTCONTRACTS

MASTER PROGRAMME IN COMPUTER SCIENCE

Uppsala University
Department of Information Technology

Approved by

Supervisor, Jonatan Bergquist

Reviewer, Björn Victor

September 21, 2018

Abstract

Blockchain technology is being researched in diverse domains for its ability to provide secure and reliable information in a distributed and decentralized manner. Trust frameworks and reputation models of an online interaction system are one among several use cases that require these attributes for a secure system. The centralized nature of current online systems, however, leaves the valuable information as such prone to both external attack and internal modifications. This master's thesis project, therefore, studies the use of blockchain technology as a decentralized and distributed infrastructure for an online interaction system that can guarantee a reliable and immutable trust score. It proposes a system of smart contracts that specify the logic for interactions and models trust among pseudonymous identities of the system. The contract is deployed to a blockchain network where the trust score of entities are stored and updated in an immutable manner without any central entity. The proposed method and the trust metrics used is evaluated by simulating an interaction graph using an existing dataset to demonstrate the functionality, correctness, and usability. The obtained results then illustrate how the proposed model addresses the threat models of current reputation systems.

Acknowledgements

Table of Contents

List of Tables

List of Figures

List of Acronyms

nEG	Number of Endorsements Given
nER	Number of Endorsements Received
CP	Consumable Points
TRP	Total Received Points
TEI	Total Endorsement Impact
DHT	Distributed Hash Table
PoW	Proof-Of-Work
PoS	Proof-Of-Stake
PoR	Proof-Of-Reputation
DPoS	Delegated Proof-Of-Stake
EVM	Ethereum Virtual Machine
EOA	Externally Owned Account
PBFT	Practical Byzantine Fault Tolerant

1 Introduction

We rely heavily on the internet today, from using emails to communicate with one another, searching for an online source of news or media files to shopping for everyday things. One can say that the internet has become an inseparable part of our society. Statistics [1] suggest that there are approximately 4 billion internet users worldwide and this number is only increasing with each passing day. Internet live stats [2] is an online service that provides live statistics on various online activities such as blog posts, social media usage, internet traffic and emails sent. Given the global reach of the internet and diversity of users, one can reasonably assume that not all online interactions happen between known entities. The online identities that everyone uses to interact with one another provide no way to confirm the real-world identity or the attitude of the person behind. To determine the trustworthiness of an entity is a difficult task even in the real world. If Alice needs to interact with Bob, she has no way of measuring the trustworthiness of Bob with 100% certainty. She could get a reference from others in the society, i.e., by referring to Bob's reputation. If Bob has a good standing among members of the community due to good records or other objective measures, there is a high probability that Alice's interaction with Bob will result in a good outcome, i.e., Alice's perception of Bob being good will turn out to be true. However, this does not provide any guarantee of Bob's behavior. Bob could be a malicious actor who was waiting for the right moment to deal severe damage. There is no way of guaranteeing that the behavior of an entity will always be an honest one. Honesty can be seen as an attribute at a given time. Alice may trust Bob today but not tomorrow because he may have taken a damaging action. Depending on the damage caused by his action, the level of trustworthiness also gets affected. As such, anyone can be seen as honest until they deal damage and are known as a malicious actor. That is when they lose their reputation, and other members are less likely to trust them.

This notion of trust and reputation, when used in online interactions, can help to predict the outcome of online transactions. Any online platform where users communicate with each other for different purposes (e.g., digital transaction, news or file sharing) can be termed as online interaction system. Depending on the nature of the interaction, the failing outcome can have a significant impact on interacting entities. For instance, the failure of an interaction that involves buying a house is not the same as failing to receive a high-quality music file. To prevent severe damage that might result from a failed transaction, trust frameworks and reputation models of the interaction system play a crucial role. They attempt to avoid harm by giving enough information about the interacting entities to be able to predict the outcome. A reputation system collects information about the interacting entities by continually updating the state based on feedback. The risk of failure or probability of success when interacting with an entity relies

on the information provided by the underlying reputation system. This information is usually presented in the form of trust scores assigned to each online identity. Before engaging in an online transaction, the entities need to select the online identity of the user with whom they intend to interact. The interaction can then take place whose outcome is observed and stored by the reputation system to update the current trust value further. Examples of reputation systems in use by current online interaction platform are eBay [1], which is an e-commerce platform, StackExchange [2], which is a Q&A platform, and Reddit [3], which is a content rating and discussion platform. The trust score of users is based on their feedback, positive/negative ratings, upvotes/downvotes from the participants with an equal privilege to interact with the system. The final score aggregated via these measures can either raise or lower the reputation of the user and limit their interaction ability.

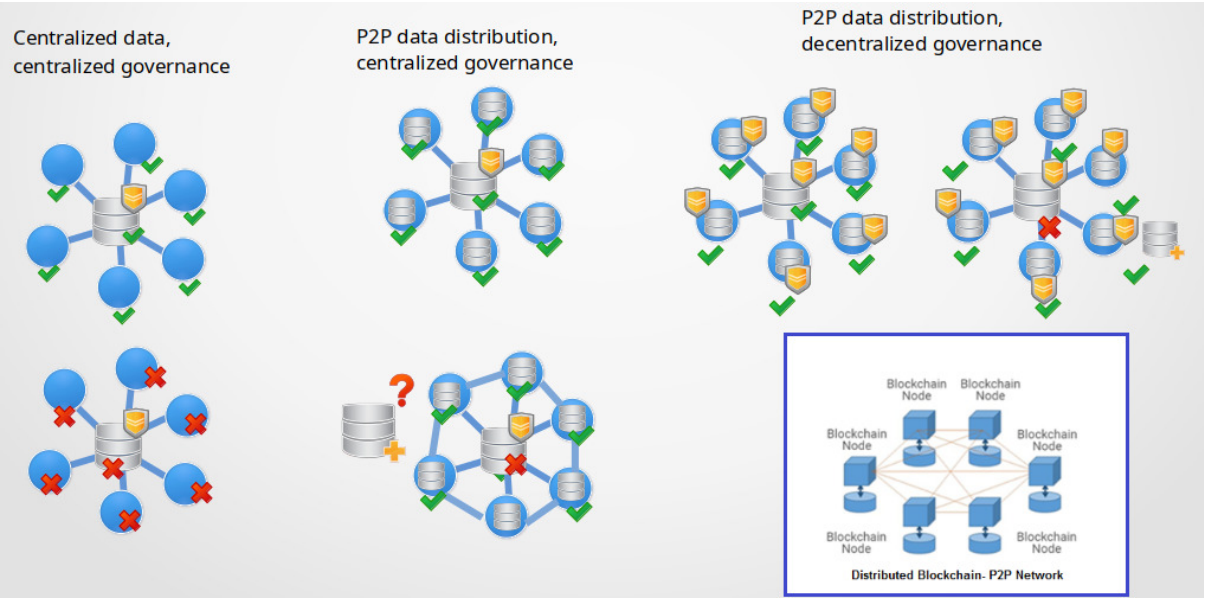

Figure 1.1: Centralized vs. decentralized network

A general trust framework and a good reputation model to specify the rules of interaction between online identities and the extraction of useful information from them is, therefore, essential to maintaining the security of an online interaction system. Equally significant is to protect the integrity of this information such that they are reliable, untampered (no deliberate alteration of data) and always available. Most of the online interaction systems make use of a client-server architecture to serve and govern the data usage. As such, the system is centralized and prone to both external attacks and internal modifications. If the server nodes fails, then the whole network fails since none of the clients can access the data anymore. An alternative architecture that is primarily utilized by file sharing systems is a Peer-to-Peer (P2P) network, and reputation-based trust management system for them exists as well [3]. In a P2P network, all nodes (peers) can act as both client and server, i.e., a peer can both request and serve data. Thus, if one node fails, clients can still request data from other nodes in the network. Earlier

[1] https://www.ebay.com/
[2] https://stackexchange.com/
[3] https://www.reddit.com/

P2P file-sharing services such as Napster [4] offered P2P data distribution such that clients could request data from any nodes connected to its network. However, the drawback of this system was that it kept a central server to index the location of files. This form of centralized data governance acted as a single point of failure, and the shutdown of its service was possible. Other P2P file-sharing services that followed employed a decentralized and distributed indexing method to overcome this vulnerability. Example include BitTorrent [5], [6], which allowed the participating nodes to track the location of a file (to discover peer) without the use of a central server. Blockchain technology [7] makes use of a P2P network although its main design goal was not that of a file-sharing service but instead started as a way to store and distribute data over the network in an immutable fashion. It allows the nodes to store blocks of data chained together in a cryptographically secure method such that the modification of stored data is computationally infeasible. These models can be seen in Figure 1.1 in Section 1.

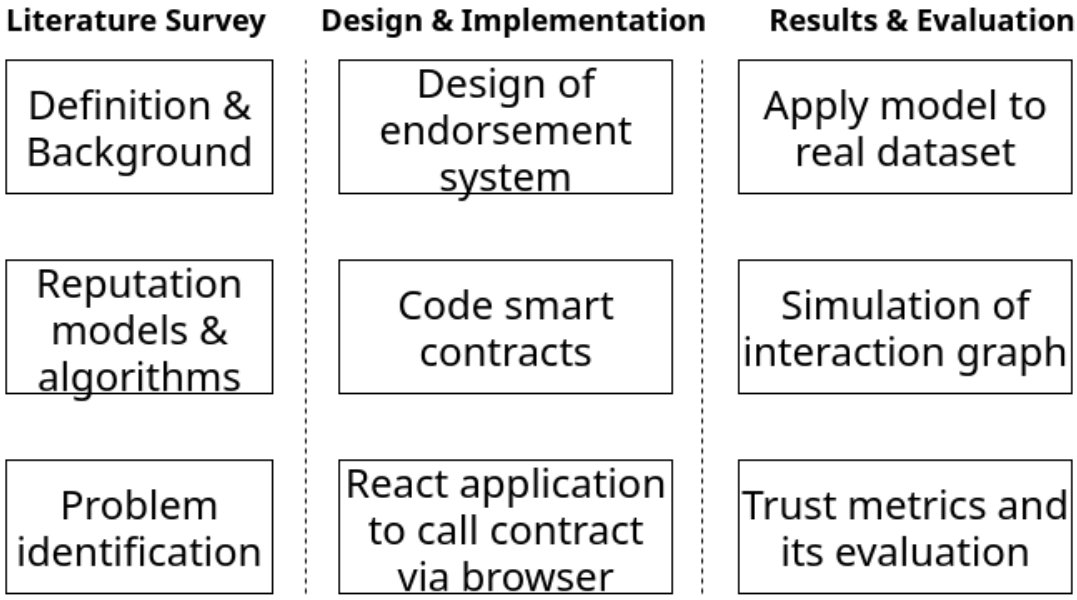

Figure 1.2: Phases of the project.

This master's thesis project, therefore, proposes the use of blockchain technology and smart contracts to model a trust framework and implement a reputation system. The initial step of the project was the identification of relevant concepts by performing a literature survey on existing reputation models, properties of graph-based algorithms and its relevance to the project. The survey motivated the design of a solution, wherein the proposed model, participating entities can endorse each other. Based on assumptions about different possibilities of endorsement behavior that may occur, trust metrics were defined in a way that honest participation would be encouraged. The definition of honest and malicious participation from the perspective of an endorsement network is discussed in Section 4. A method to collect and quantify this endorsement information to assign a trust score to each entity is presented. To assess the working of the proposed model, it is applied to an existing real dataset. Computation of trust scores for each participant based on the defined trust metrics is presented. The simulated graph and computed trust scores do support the defined metrics. A discussion of relevant threat

models on reputation systems and how the proposed model addresses them is presented. The Figure 1.2 in Section 1 gives the workflow of this master's thesis project.

The major contribution of this master's thesis project is:

- Design of an endorsement model where entities can endorse information about each other and a method to aggregate this information such that a global value can be assigned to individual entities to infer their trustworthiness.
- Deployment of the endorsement system on blockchain network that updates and computes the trust scores of entities by executing the smart contract code based on users' interaction ensuring reliable and verifiable information.
- Evaluation of the endorsement model by applying it to an existing data set and discussion of relevant threat models.

1.1 Motivation

Consider a simple scenario where Alice wants to buy a pair of headphones for which she browses a "buy/sell" platform. When she finds a relevant product on the platform published by Bob, an unknown entity to Alice, the success or failure of the transaction depends on two factors that may or may not be transparent.

Reputation of Bob: Reputation of Bob can be inferred from his history of transactions, ratings provided by previous buyers that have dealt with him, the reputation system of the platform in use and the integrity of all these relevant data.

Reputation of the platform: This factor can also be inferred similarly based on the history of services the platform has been able to provide or a general perception in the community. Here, the platform in use acts as the trusted third party that Alice must trust to present correctly computed, untampered data about Bob or the ad posted by him. The entity claiming to be Bob could be Eve, who found a way to bypass the security of the platform in use to inflate her reputation. Eve could delete the ad and associated account when the payment is complete, or she could gather details about Alice to misuse it later. Any malformed decision on the trustworthiness of an entity could be expensive and deal severe damage to the user.

Statistics suggest that online shopping is the most adapted online activity [8]. Reports by Experia [8] and Javelin [9] indicate that E-commerce fraud has increased by 30% in 2017 over 2016 while identity fraud victims have risen by 8% in 2017 (16.7 million U.S. victims). A recent report of the data breach [10] on an online shopping website, Macy [4], exposed details of its users such as their name, credit card number, expiry date. While there are several security reasons that have led to attack at such scale, one reason is the client-server architecture where everything is stored on centralized server and data flows in and out from the same source. On the other

[4] https://www.macys.com/

hand, distributing information over a decentralized network would require simultaneous attack to achieve the same effect, thereby increasing the difficulty level of attack. Similarly, Reputation models can help in measuring the reliability of interacting entities so that users can make an informed decision before participating in any transaction. Thus, a reputation system should be secure, robust, always available and aim for higher accuracy. The use of right reputation algorithms with blockchain technology could help to ensure the trustworthiness of online entities with the correctness of data and a high degree of accuracy.

1.2 Purpose and research questions

The primary goal of this master's thesis project is to use blockchain technology and smart contracts to model an interaction system where entities can endorse each other. It aims to specify the rules of interaction and provide a method to aggregate these interactions to generate a trust score for the interacting entities. The research questions that this project aims to address are:

- What are the requirements for storing trust values and linking them to associated identities stored on or off a blockchain network? How can a blockchain application be built to define a general trust framework and how would the overall system architecture look like?
- How can the proposed model help to infer the trustworthiness of interacting entities while preserving the anonymity of its users?

1.3 Scope

This master's thesis project attempts to answer all the research questions mentioned in section 1.2.

Research Question 1 An interpretation of connections between nodes and quantification of a score that can represent trustworthiness of an entity is presented. Overall system design and architecture to implement the application on blockchain network is presented along with a comparative analysis of an on-chain vs. off-chain storage.

Research Question 2 The study of threat models relevant to trust and reputation systems is presented along with a discussion on how the proposed system addresses them.

1.4 Structure of Report

This report is structured as follows. Chapter 2 provides a background overview of relevant concepts necessary to understand the following sections. Chapter 3 presents a literature survey on the existing algorithms and their implementations. In Chapter 4, system requirements and the approach taken for the model design are shown. It presents the overall design and architecture of the proposed system. Chapter 5 presents the results and evaluation of the system

requirements. In Chapter 6, the discussion, and analysis of the overall project along with the limitations of the proposed system are presented. Finally, conclusion and future works are presented in Chapter 7.

2 Background

Trust and reputation can be used across several domains, and their definitions vary depending on context. It is essential to recognize the context in which this term is being used. Similarly, blockchain can be seen as a relatively new technology although the underlying sets of cryptographic functions it uses have been around for a long time. Blockchain technology is being researched both by academia and industry to explore its possibilities and in diverse use cases, e.g., as a privacy-preserving smart contract [11] or as a way to protect personal data [12]. This chapter aims to provide the necessary background theory by discussing the definition of trust and reputation, blockchain technology and its components, cryptographic functions, and graph properties in detail.

2.1 Trust and Reputation

Trust encompasses a broad spectrum of domains and is context dependent. Therefore, its definition varies based on context and discipline and as such lacks collective consensus among researchers [13]. Using the classification from McKnight et al., 1996 [14], trust can be either Personal/Interpersonal, Dispositional or Impersonal/Structural. Personal trust is when one person trusts another specific person, persons, or things in a particular situation. Interpersonal trust involves more than one trusting entities, i.e., two or more people (or groups) trust each other. Dispositional trust refers to a more general trust that is based on the personality attribute of the trusting party, i.e., an entity is more likely to trust other entity based on their attitude and is cross-contextual. While the trust mentioned above are implicitly directed towards a person, impersonal/structural trust refers to the trust in institutional structure, i.e., it is based on belief in regulatory enforcement such as by contract law, judiciary systems rather than belief in involved parties.

Trust can be generally seen as an entity's reliance on another interacting entity to perform a specific set of the task given a specific situation. As pointed out by Gambetta et al. [15] "Trust is the subjective probability by which an agent assesses that other agent or group of agents will perform a particular action that is beneficial or at least not detrimental. For an entity A to trust another entity B or to evaluate the trustworthiness of B, the reputation of B plays a central role. Reputation is the perception of the character or standing of an individual. Like trust, reputation is context-dependent, e.g., Alice may be trusted to answer Linux questions efficiently but not Windows related questions [16]. A significant difference between trust and reputation is that the former takes the subjective measure as input whereas the latter takes an objective standard (e.g.,

history of transactional outcome) as an input to yield a resulting score that can aid in detecting reliability/trustworthiness of an entity [17], [18].

A study by Jøsang et al. [19] classifies trust and reputation measures as either subjective or objective. This classification is further divided into specific or general. A subjective measure is based on the perspective of an individual and has no formal (objective standard to measure the trust) metrics. Specific, subjective measures imply a subjective opinion of an entity for specific aspects (e.g., rating a merchant on a scale of 1-5 for response time) of trust. One way of measuring this is via survey questionnaires that ask specific questions. On the other hand, a general, subjective measure, aggregates all the individual scores and provides an average standing of the user on the network, e.g., the difference of accumulated positive and negative ratings used by eBay to give an aggregated score.

An objective measure is used for product tests that can have some formal criteria on which to rely on, e.g., hard disks can be measured based on performance metrics such as transfer rate, access time, CPU usage. A specific, objective measure takes an objective measure for a specific metric, i.e., how good is a transfer rate for a given hard disk whereas, a general, objective measure accounts for all the relevant aspects and averages the performance to give an average rating/score on a specific scale. Table 2.1 shows the classification of trust and reputation measures as discussed above.

	Specific, vector-based	General, Synthesized
Subjective	Survey questionnaires	eBay, voting
Objective	Product tests	Synthesised general score from product tests

Table 2.1: Classification of trust and reputation measures based on [19]

Individuals in online systems are identified by their online identities which can be anything and not necessarily linked to their real-world identities. Based on the online identity and the information associated with it, one has to decide (carry forward the transaction or not) on online interaction. Thus, online identities play a crucial role in digital interaction and require unknown entities to trust each other based on the reputation system of the platform in use. Trust and reputation can be seen as a soft security mechanism where it is up to the participants rather than the software/system to maintain security, i.e., to prevent harm by malicious interactions. By definition, a security mechanism [20] is a process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack. Unlike hard security mechanism such as access control, capabilities, authentication where a user can be allowed or rejected access to the resource, reputation systems do not provide a method to block or detect a security attack directly. However, they define a process to identify malicious users and avoid them from harming other users in the system. Rasmusson, and Jansson [21] first used the term "soft security" to describe the idea of identifying malicious users and preventing harm to other users in the context of secure open electronic commerce. Reputation systems need to continuously receive feedback about the user's behavior and maintain an updated record of

user reputation. It provides a way to calculate the probability of success or risk of failure of a transaction between interacting parties.

2.2 Threat scenarios in trust and reputation systems

A reputation system should provide correct and reliable information such that users can correctly infer the trustworthiness of the entity in question. Therefore, it is crucial to discuss several threat scenarios and methods to mitigate them in an online interaction system. Several classes of attack that can exist in trust and reputation system, as mentioned by [22], [23] are:

Self-promoting attack: In this case, an attacker tries to inflate his/her reputation score by falsely increasing it. This attack is more likely in systems that do not have a mechanism for data integrity verification or data authentication. An attacker could attempt to exploit a weakness in the calculation of reputation metric or during the dissemination of information. This attack can be performed by an individual or by forming a malicious collective where groups collude to increase the reputation score of an identity falsely. Even if the systems do provide cryptographic mechanisms for source data authentication, the self-promoting attack is possible by creating multiple Sybil identities. This form of attack is also known as Sybil attack. An attacker can create multiple user accounts to self-promote an identity, or colluding identities can mutually participate to generate real feedback. Defense techniques for such attack can be requiring the user to provide proof of successful transactions, and the ability to limit or prevent an attacker from obtaining multiple identities.

Whitewashing: Whitewashing is the process of exiting a system with an account with a bad reputation and entering afresh with a neutral or non-negative account. This attack exploits the system's formulation of reputation score. For instance, eBay's rating system uses a range of values {1, 0, -1} for representing positive, neutral and negative rating respectively. The formulation of the final score is done by calculating the difference between the number of positive and negative scores. In this case, it makes more sense for an identity with 100 negative and 5 positive scores to create a new account that gives a neutral score and start afresh. Another problem that can be observed in this formulation is that a user with 50 positive and 10 negative is the same as the user with 40 positive and no negative scores. A technique to defend against whitewashing attack is to have a better method for formulation of final score such that a new user would be distinguished from an old user.

Slandering: Slandering attack is the form of attack when an attacker (or groups) create false negative feedback about other identities with an aim to damage their reputation. Lack of mechanisms for data source authentication can lead to this attack, just like in self-promoting attack. Similarly, the high sensitivity of formulation to negative feedback facilitate slandering attacks. If the reputation system is sensitive to even lower value of the negative score, then an honest node will be more affected by this attack. On the other hand, if the reputation system is less sensitive to a negative score, then the amount of time an actual dishonest identity can live

in the system to deal damage raises. Therefore, an optimal balance needs to be found by the reputation system to address this trade-off. Techniques to defend against this attack include employing stricter feedback authentication mechanism, validating input to make sure that feedback is actually tied to some transaction, and incorporating methods to limit the number of malicious identities nodes can assume.

Orchestrated: In an orchestrated attack, attackers, collude with each other and combine multiple strategies to form a coordinated attack. They employ different attack vectors, change their behavior over time, and divide up identities to target. For instance, attackers can form teams with different roles where one team performs a slandering attack on benign users, and the other team makes the self-promoting attack to inflate their reputation. Another example is when one team acts honestly for a specific amount of time by serving good content or by giving negative feedback to the dishonest nodes of the network. The other team acts dishonest and gains the benefit from it until the reputation is too low to allow them to do so. At this point, the teams can switch roles, and the honest team starts acting dishonest and gain benefit. This form of attack is difficult to detect as there is no pattern to detect an anomaly in the network and they keep adapting to the situation. In such an attack, it is challenging to make a distinction between honest and malicious nodes.

Denial of Service: Denial of service is prevalent in systems that employ a centralized system and have no mechanism for load balancing. Attackers can send too many requests to the central entity and overload the system thereby, preventing the reputation system to operate correctly. These attacks target the calculation and dissemination of reputation information and therefore affects the data availability aspect of the network.

Free riders: Free-riding is a problem mostly associated with P2P systems. Usually, P2P systems rely on voluntary contributions. As such, individual rationality results in free riding among peers, at the expense of collective welfare [24]. The reputation system that aims to address this behavior differs from the classic use of reputation systems where the aim is to enhance the quality of transactions. In P2P systems that seeks to address free-riding behavior, the goal is to encourage contributors by giving them more benefits over consumers. The reputation system that addresses this behavior differs from classic reputation systems in the sense that it does not seek to increase the quality of transactions. Andrade, Nazareno, et al. [25] discusses the use of an autonomous reputation scheme by prioritizing resource allocation to peers with higher reputation.

2.3 Graph properties

A graph, as the name suggests can be used to represent objects and their relationships graphically.

Formally, a graph [26], G, is an ordered triple (V, E, ϕ, G) where:

- V is a non empty set of vertices v.
- E is a non empty set of edges e.
- e connects two vertices, where, $v \in V$ and $e \in E$.
- ϕ_G is an incidence function that assigns pair of vertices to each edge of graph G.
- $\phi_G(e) = uv$ represents that e is an edge that joins vertices u and v.

Based on these properties, any online interaction system can be modeled graphically including reputation system. Each node on the network can represent agents/users that interact with other users. This interaction can represent the relationship between nodes as the edges connecting vertices. The transfer of data between the nodes can be quantified to represent the weight of the edge. This weight value can be used to determine the strength or weakness of relationship between the nodes. Modeling the interaction as a graph can help to understand and analyze its complexity at different levels. By observing the local properties of a particular node such as its activity, connection degree, its neighbors, and interactions, one can derive useful information about a node. Similarly, the node's relative position in a given graph can help to determine its centrality and connectivity. An overall structure of the network (graph topology) can help to study the global properties of the graph.

2.4 Cryptography

Cryptography [27] offers algorithms to achieve confidentiality, integrity, authenticity, and non-repudiation. Confidentiality refers to keeping the information secret from unauthorized parties. Integrity relates to ensuring that the information being communicated is not altered by unauthorized or unknown means. Authenticity relates to corroborating the source of information (data origin authentication). Non-repudiation is associated with the property that any entity who has previously sent the message cannot deny their authorship.

A cryptosystem can be defined as a five-tuple (P, C, K, E, D) where:

- P is a finite set of plain texts.
- C is a finite set of ciphertexts.
- K, the keyspace is a finite set of keys.
- E is a set of encryption transformations such that $e_k : P \Rightarrow C$.
- D is a set of decryption transformations such that $d_k : C \Rightarrow P$.
- For each $k \in K$, there is $e_k \in E$ and $d_k \in D$ such that $d_k(e_k(m)) = m$ for every plaintext $m \in P$.

A cryptosystem can be either symmetric or asymmetric. Symmetric makes use of the same key for both encryption and decryption whereas asymmetric makes use of key pairs, public key, and private key. Consider any pair of associated encryption/decryption transformations (e_k, d_k) and suppose that each pair has the property that knowing e_k it is computationally infeasible, given a random ciphertext $c \in C$, to find the message $m \in P$ such that $e_k(m) = c$. This property implies that given e it is infeasible to determine the corresponding decryption key d_k. This is unlike symmetric-key ciphers where e_k and d_k are essentially the same. Consider two-party communication between Alice and Bob where Bob selects a key pair (e_k, d_k). Bob can send the encryption key e_k over any channel while keeping the decryption key d_k secure and secret. Alice can send the message m to Bob by applying the encryption transformation determined by Bob's public key to form ciphertext c. Upon receiving c, Bob can decrypt it using the corresponding private key d_k that is only known to Bob and corresponds to the public key that was used to form c. Besides public-key encryption, public-key cryptography also has use in the digital signature as discussed in section 2.4.2. RSA [28] is one example of a public-private cryptosystem.

2.4.1 Hash functions

Cryptographic hash functions are one-way functions, also known as mathematical trapdoor functions that transform an input message into a fixed length binary output. It is one-way because although converting a message input to a hash value or a message digest can be done in constant time, reversing the operation is practically impossible to achieve as it is computationally infeasible. An important characteristic of hash functions is its deterministic output, i.e., given an input, it will always produce the same output. This attribute contributes to data verifiability as anyone can always verify if the produced hash output for data matches by simply applying the data to the respective hash function.

Significant properties of hash functions that contributes to reliability in digital security are [27]:

One-way: Given a key k, and an output w, it should be computationally infeasible for an attacker to find x such that the hash of x applied with k, produces w, ie., $H_k(x) = w$.

Second pre-image resistant: Given a key k and a string x, it should be computationally infeasible for an attacker to find y such that $H_k(x) = H_k(y)$.

Collision-resistant: Given a key k, it should be computationally infeasible for an attacker to find x and y such that $H_k(x) = H_k(y)$.

MD5 is a hash function that produces an output of size 128 bits. A collision-resistant attack in MD5 is possible within seconds with a complexity of 2^{39} [29] MD5 operations. NIST published a secure hashing algorithm (SHA) in 1993 as the secure hash standard. Currently, SHA-3 is the latest in SHA family of standards that was released in 2015 with SHA-0, SHA-1, and SHA-2 as its predecessor algorithms. Collision attack for SHA-1 was shown to be practically possible [30] by creating two colliding pdf files (two distinct pdf files that produces same hash value) that

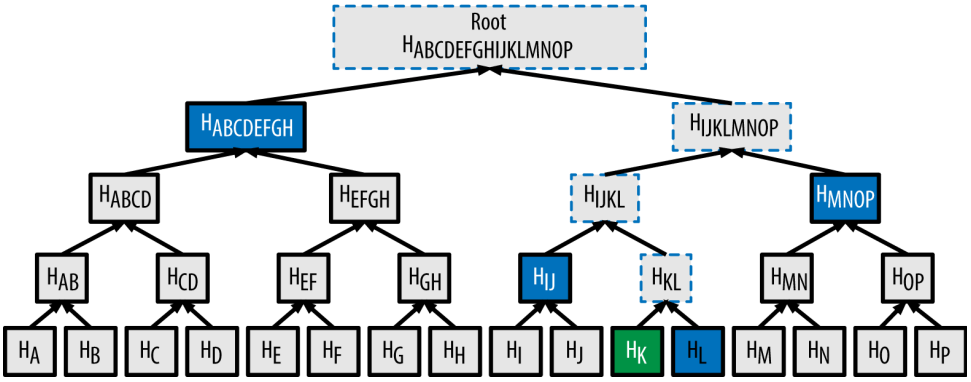

Figure 2.1: Merkle root and data inclusion verification

produced the same SHA-1 digest. It took equivalent processing power of approximately 6,500 years of single CPU computation time and 110 years of GPU computation time. In real time doing parallel computation on a high-performance machine such as as [31], [32] with 4,608 nodes each having 3 GPU V100, the collision can be found in approximately 12.29 hours. This calculation is based on the study by [30] which states that the time needed for the homogeneous cluster to produce collision is 95 k40-years. Given that a K40 [33] has a compute performance of 1.43 TFLOP and V100 [34] has 7 TFLOP, the time would be reduced to approximately 12.297 hours.

Applications of hashing algorithms include a digital signature, file checksums (i.e., generate a checksum value for a given file and check against the value that was originally distributed by the developer/creator of the file). If the checksum match then it ensures that the integrity of data is preserved.

Merkle tree [35] is a hash tree that is used by blockchain to verify the integrity and inclusion of data in a block. The Figure 2.1 shows a binary hash tree where the leaf nodes at height 0 ($H_A, H_B, ..., H_P$) represent the data block. These nodes are paired and hashed together. As we can see in the figure, H_A and H_B are paired and hashed together to form H_{AB}. This process of pairing and hashing continues until there is only one hash value left. This final hash value which is the root node of the tree is also known as a root hash. This root hash value is created by hashing the list of transactions in the blockchain and is included in a block header. This value is significant for two things as mentioned below:

1. Data integrity verification: An attempt to change any transaction data would completely change the root hash of the block.

2. Data inclusion verification: It is not necessary to include all the transactions in a block in order to verify the inclusion of a transaction in a block. As shown in Figure 2.1, the nodes labeled in blue are enough to verify the inclusion of transaction H_K. By looking at the root hash and the intermediate hashes, one can verify the inclusion of a specific data.

2.4.2 Digital Signature

A digital signature acts roughly like a physical signature in that the signature can represent the identity and authorship of the signer. A digital signature is a mathematical scheme that offers attribute such as authentication (ability to prove that sender created the message) and non-repudiation (sender cannot deny having sent the message).

The components of digital signature schemes are [28]:

- Security parameter, k, chosen by user when creating public and private keys.
- Message, M, set of messages to which the signature algorithm is applied.
- Signature Bound, B, an integer bounding the total number of signatures that can be produced with an instance of signature scheme.
- Key generation algorithm, G, which any user A can use to generate in polynomial time a pair (P_A^k, S_A^k) of matching public and private keys.
- Signature algorithm, σ, to produce a signature $\sigma(M, S_A)$ for a message M using the private key S_A.
- Verification algorithm, V, to check that S is a valid signature for a message M using the public key P_A, i.e., $V(S, M, P_A)$ is only true if and only if the signature is valid.

Two significant aspects of the digital signature scheme are signing algorithm and verification algorithm. If Alice wants to send a signed message to Bob, then she can create the signature using her private key and send the message along with the signature to Bob. Bob can verify that the message originated from Alice by using the verification algorithm on the signature using Alice's public key. If the verification function returns true, then Alice cannot deny having sent the message. Similarly, if the verification algorithm returns false, then that would imply that the signature is invalid.

In the context of the blockchain, if Alice wanted to send a value (transaction) to Bob, the following steps would have to take place. Alice has to create a transaction data structure with values for the required fields. The data is serialized using the underlying serialization algorithm. A hash function is then applied to this serialized message. The signature on this hash value is created by using the signature algorithm of the blockchain. In the case of Ethereum [1], Alice would compute the ECDSA signature. This signature is appended to the transaction and Alice can then submit this transaction over to the blockchain network. When the transaction gets broadcasted, there are some special nodes (miner/validator) that are supposed to pick this transaction and put it in a block. After this happens, the transaction can be seen and verified by anyone on the public blockchain network. If Bob or anyone wants to verify the signature, they need to provide the signature, serialized transaction, and the public key of

[1] https://www.ethereum.org/

Alice to the signature verification function. The public key of Alice can be derived from the ECDSA signature she created. Alice can always prove that she owns the public key because she owns the corresponding private key that generated the signature. The transaction message is unalterable since any modification to the message would refute the signature.

2.5 Distributed Hash Table

A Distributed Hash Table (DHT) [36], [37] provides a method to distribute data over a large P2P network along with an efficient mechanism to search and retrieve the given data item. The data items are assigned a random key from a large identifier space, such as 128-bit or 160-bit identifier depending on the hash function in use. Similarly, nodes are also assigned a random number from the same identifier space. The data structure follows a $(key, value)$ pairs such that a key (unique identifier) is mapped to a specific value. The key value controls which node(s) stores the value. A DHT based system is responsible for implementing an efficient and deterministic scheme that uniquely maps the key of the data item to the identifier of a node based on some distance metric. The main requirements of a DHT are that data be identified with a unique key and nodes be willing to store keys for each other. The operation $lookup(key)$ should return the network location of the node responsible for that data item which is accomplished by routing a request for a data item to the responsible node. This node responsible for the key k is referred to as the successor of k and denoted as $succ(k)$. Thus, the two major issues that a lookup algorithm should address are mapping keys to nodes in a load-balanced way and forwarding a lookup for a key to an appropriate node. Both of which is related to the distance function, i.e., the closeness of keys to nodes and nodes to each other.

A simple and non-scalable approach to resolve a key to the address of $succ(k)$ is by implementing a linear search such that each node p keeps track of the successor $succ(p+1)$ and its predecessor $pred(p)$. When a node p receives a request to resolve a key k, it will forward the request to one of its neighbors if it cannot find the data in its address. This approach does not scale because of the compute cycles consumed by many nodes that must handle these messages. Chord [38] maintains an efficient lookup approach where each node maintains a finger table containing the IP address of a node halfway around the ID space from it, a quarter-of-the-way, an eighth-of-the-way, and so forth, in powers of two, in a structure that resembles a skiplist data structure. This structure of chord ensures that the node can always forward the query at least half of the remaining ID-space distance to k, leading to O(log N) messages to resolve a query. The distance functions or the lookup schemes can vary in different DHT based systems. In all the cases, each forwarding step should reduce the closeness between the current node handling the query and the key.

2.6 Blockchain Technology

Blockchain technology can be defined as a distributed record of transactions that lets anyone on the network to audit state changes and prove with mathematical certainty that the transactions

transpired according to the blockchain protocol [39]. There are several ways to define blockchain, and every definition is relevant to its specific use cases. A formal standard definition of blockchain is under development as ISO/TC 307 [40]. As pointed out by Antonopoulos, Andreas M., and Wood, Gavin [41], the components that make up an open, public, blockchain are (usually):

- A P2P network connecting participants and propagating transactions and blocks of verified transactions, based on a standardized "gossip" protocol.
- Messages, in the form of transactions, representing state transitions.
- A set of consensus rules, governing what constitutes a transaction and what makes for a valid state transition.
- A state machine that processes transactions according to the consensus rules.
- A chain of cryptographically secured blocks that acts as a journal of all the verified and accepted state transitions.
- A consensus algorithm that decentralizes control over the blockchain, by forcing participants to cooperate in the enforcement of the consensus rules.
- A game-theoretically sound incentivization scheme (e.g., proof-of-work plus block rewards) to economically secure the state machine in an open environment.

The participating nodes in the blockchain network have an account address that is associated with their public/private key pair. The accounts are identified by their public address and have a state which can be changed by a transaction. The transactions are signed messages that originate from a user account, signed by the private key of the account owner. Every transaction gets broadcasted to the network. There are validator nodes (nodes with enough computational resource to solve the cryptographic puzzle) who picks up the list of unconfirmed transactions, verifies and orders them into a block. The rules for proposing and adding a new block to the blockchain by the validator nodes depends on the consensus mechanism in use. Different types of consensus mechanisms are discussed in detail by Section 2.6.1. If the network agrees on the proposed block based on the underlying consensus mechanism, then it gets added to existing blockchain with a hash pointer that links to the previous block. As such, blockchain offers a verifiable record of all the transactions that have occurred throughout the history of the blockchain all the way up to genesis block. Genesis block is the first block in blockchain and is the only block that has no parent hash associated with it. A block usually consists of transaction root hash (root hash of transactions included in the block), timestamp (to verify the time when the block got created), and parent hash (to refer to the parent of current block). Once the transactions are ordered and added to the blockchain, their immutability is guaranteed by the blockchain protocol. The illustration of a blockchain structure is given by Figure 2.2. As we can see that Block 10, Block 11 and Block 12 are chained together by a cryptographic hash and shows the chronological ordering of blocks.

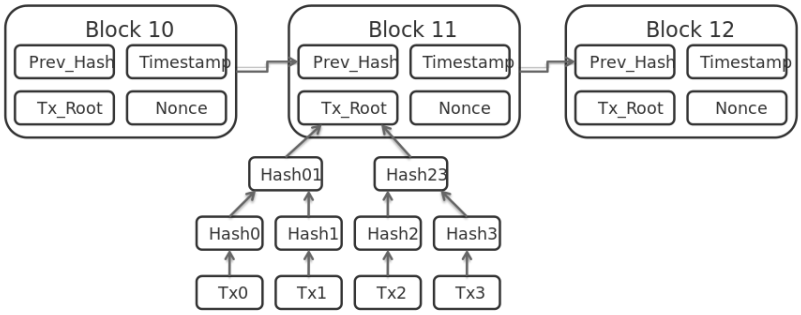

Figure 2.2: Blockchain Structure taken

2.6.1 Consensus Mechanisms

Blockchain, being a distributed database with multiple writers, should have a way for every node to reach a consensus on a shared global view of the network. Consensus mechanisms allow doing so. Based on consensus mechanisms, systems can be distinctly categorized into [42]:

Proof-Of-Work (PoW): This is the most widely used consensus mechanism in public permissionless setup. As the name suggests, a validator node (miner node) needs to provide the proof to the network that it has done a significant amount of work. This work requires them to invest a substantial amount of computational resources. The reason for this is that all the validator nodes compete to be the writer of the next block for which they need to solve a cryptographic puzzle. The puzzle requires miners to perform a hash operation on the data (hash function applied to the concatenation of message and nonce) until the output hash is less than a specific target value. As the hash function produces a random output each time, given an n-bit number, the probability that the first bit of the generated output hash is 0 is 50 %, the probability that the first two bits are 0 is 25 %, for the first three bits it becomes 12.5% and so on. In numeric terms (base 10 number), we can refer to it as the difficulty of finding a value less than 2^n, 2^{n-1}, 2^{n-2} and so on. As such, lower the target value (also known as difficulty target), lower is the probability of finding such a value and therefore higher is the difficulty of the puzzle. The process of repeatedly finding the nonce value until the output hash satisfies the requirement of being less than a specific target is given by Figure 2.3. An essential attribute of a hash function is that a change in even a single bit of input data will completely change the output hash value. Therefore, the only way to find a nonce value which when concatenated with the message will produce an output hash less than the specified target is via multiple SHA computations until the target is reached. The first validator node to solve the problem gets to add the proposed

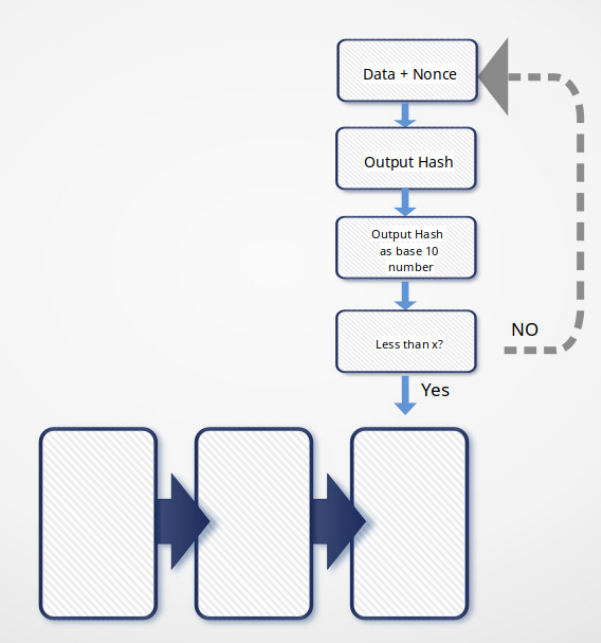

Figure 2.3: Cryptographic Puzzle

block to the existing blockchain. In such a system, the selection of the validator node is random in that anyone among the competing node could be the first to solve the cryptographic puzzle. There is no way to know a priori, which node is going to be the writer of the next block. As such, this consensus mechanism makes the system DDOS resistant. However, miners in a PoW setup can decide upon the order of transactions to include in the block although they cannot modify the transaction data. As such, one may have to wait for few blocks before having their transactions confirmed and placed into the blockchain. The transaction order is agreed upon by the network when the proposed block by validator node is broadcasted and accepted by the network. The randomness in the cryptographic puzzle makes it rare for two validator nodes to solve a block at the same time. However, it is sometimes possible to reach such a situation leading into several branches of the blockchain. In which case, the nodes build upon the block that they first received. This situation gets solved when the next block is solved, and one of the branches becomes longer over another. In which case, everyone switches to the longest branch. Given the rarity of such a situation, the blockchain is assumed to be eventually stabilized. Examples include Bitcoin [43], Ethereum [44].

Economy based systems: Consensus mechanisms such as Proof-Of-Stake (PoS) or Delegated Proof-Of-Stake (DPoS) can be seen as an economy based system. Unlike PoW, miners do not compete with each other to be the writer of the next block thus saving lots of computational resources. The general idea is that participants can put the respective platform based native token they own at stake to validate a block. Whoever has the higher value at stake gets to be the writer of the next block. Casper solves the problem of nothing at stake by penalizing the node that is detected to have validated two blocks at the same height. DPoS differs from PoS in that, the stakers do not directly involve in writing a block but rather vote for delegates/witnesses to

assign this responsibility. The elected delegates (node with majority of votes) gets to write its block to the blockchain. Compared to PoW, it is computationally efficient as it does not need to invest substantial resources. However, this also leads to the problem of nothing at stake [45], i.e., a node could vouch for two forks of the same blockchain. With this setup, a user could try to double spend the amount they have to two different accounts, and two different forks would be validating them both. Examples include Casper [46], Ouroboros [47]. Casper solves the problem of nothing at stake by penalizing the node that is detected to have validated two blocks at the same height.

Leader Based System: In this case, there is a leader that is responsible for collecting all the transactions and appending new records to the blockchain. Having a small group of consortiums, it has low computational requirements. As a blockchain protocol, it offers an immutable audit of the records. However, just like any other centralized system, this system is susceptible to DDOS attacks and third-party (leader) interference. Since the address of the leader is known to the nodes of the network, it is also known to the attackers. This form of consensus mechanism is generally used in a private or permissioned blockchain setup. Examples of blockchain platforms that use leader based consensus mechanism are Hyperledger Fabric [48], R3 Corda [49]. In Hyperledger Fabric, the entire transaction flow (proposal, endorsement, ordering, validation, and commitment of transactions) is considered to be part of the consensus algorithm [50], [51], unlike in other systems where consensus relates to the ordering of transactions. It makes use of leader election mechanism to elect a leader that is responsible for ordering transactions.

Examples of leader based algorithms are Practical Byzantine Fault Tolerant (PBFT) [52], RAFT [53]. PBFT comprises of several steps to reach consensus which is briefly explained here. The client first makes a request to the master server node. The master server node gives a timestamp to the request, records the request message and broadcasts a pre-prepare message to other server nodes. Other server nodes can either choose to accept or reject the request. The server nodes that accept the request will broadcast a prepare message to other server nodes and also receive a prepare message from other nodes. After having collected 2f+1 messages, if majority of the nodes choose to accept the request, then it will enter the commit state. If the server node receives 2f + 1 commit messages, then it believes that the nodes reached a consensus and executes the instructions in the request message. In the final stage, the server nodes only need to send a reply message to the client.

2.6.2 Categories

Along the dimension of validation and access control [54], Blockchain can be categorized as a public permissionless system, public permissioned, and private permissioned.

- Public Permissionless: Anyone can join the network and become a writer of the block as long as they can solve a problem or reach the consensus that satisfies the underlying protocol. The records are publicly available and thus publicly verifiable.

- Public Permissioned: Anyone can still join the network, but a writer of the block is known but not necessarily a trusted entity. The records are publicly verifiable.
- Private Permissioned: This is similar to the Public permissioned setting, but the records are not made public and therefore does not offer public verifiability. This kind of setup is more specific to business use-cases where one business does not need to know about other business policies or customer information etc.

2.6.3 Smart contracts

A contract in a classical sense is a set of rules with pre-defined obligations and permissions that participants are assumed to follow. It does not necessarily need to be legally binding or even associated with the outside world. The term Smart contact was first coined by Cryptographer Nick Szabo, in 1994 [55] and defined as a computerized transaction protocol that can execute the terms of a contract. Szabo points out that the contract design should fulfill four objectives [56]:

Observability, ability to observe the contract, performance of principal (agents who have agreed to the contract) and prove their performance.

Verifiability, the ability of principals to prove to the arbitrators that the contract has been performed or breached.

Privity, to ensure that the third party, other than the designated intermediaries should not have control or knowledge of the content or performance. It correlates to both privacy and confidentiality of principals of contract and the contract itself.

Enforceability, to make the contract self-enforcing which can be attributed to by verifiability, built-in incentives mechanism, self-enforcing protocols.

While privity relates to limiting knowledge and control to the third-party, on the other end, observability and verifiability demand invoking it to an extent. As such, a trade-off is required wherein an optimal balance between these objectives should meet. Thus, trusted intermediaries were introduced with minimal control/observability. However, privity was not guaranteed in case of dispute [57].

2.6.4 Ethereum

Ethereum being the first platform to offer programmable blockchains, introduced a virtual machine, Ethereum Virtual Machine (EVM), where the contract code can be executed that results in a deterministic output provided the same transaction context and blockchain state [41]. EVM often referred to as a single world computer, runs on every ethereum node and given the same initial state produces the same final state. Several high level languages can be used to write smartcontracts for different blockchain platforms. Examples include solidity [58],

LLL [59]. The contract code resides in the blockchain as an immutable form. They are not autonomous self-executing programs but rather needs to be called by a transaction or invoked by other contracts. Once the code is registered and deployed on the blockchain, its code cannot be altered by anyone, including the owner of the contract. However, a possibility to include a killable function by the owner exists which when called executes an EVM opcode called SELFDESTRUCT and logically deletes the contract from the blockchain, i.e., it removes the code and the internal state from the address of contract. After the deletion of this contract, sending any transaction to its address will not execute any code. Doing so does not delete the history of transactions as the blockchain itself is immutable. As in any Turing-complete language, it is affected by the halting problem, i.e., there is no way of knowing if the program will terminate given an input. In the case of a non-terminating program, a transaction that is transmitted to the network might run forever, and the whole network can be rendered useless if transactions cannot be processed. To avoid this, ethereum introduces the concept of gas, which is an expendable resource on the network that acts as a fundamental network cost unit. Gas [60] is used as a means to mitigate the possibility of abusing the network with excessive computational expenditures and is paid for exclusively in ether, which is the unit of currency in Ethereum. The smallest unit of currency in Ethereum is Wei ($1Wei = 10^{-18}$ ether). To store any state, or execute any operation, gas needs to be supplied. Thus, a program that has a bug or a non-terminating intention will eventually run out of gas and stop [61].

Accounts: Accounts in Ethereum can be either Externally Owned Account (EOA) (user account) which is controlled by the private key of the user or contract account, which is controlled by code. Every account is associated with a unique identifier (20-byte hexadecimal number) and has a state. The address of EOA is derived from the last 20 bytes of the keccak-256 hash of the public key. The contract account, however, is not associated with the public-private key pairs. There is no private key that owns the contract account. After the contract is written using a high-level language such as solidity, it can be compiled down to EVM bytecode, which can then be deployed to Ethereum. The deployment is done by executing a special transaction namely, CONTRACT CREATION which is sent to a special address, 0x0. The address of the contract is then derived from the contract creation transaction as a function of the originating account and nonce.

3 Related Works

The growth of online services offered possibilities to trade and interact with anyone on a global scale freely. It also offered an accessible way for bad actors to exploit the system, e.g., share malicious files on the network, steal user data, or participate as a seller but not deliver the product as promised. Thus, the advent of online interaction systems also brought along an interest in trust and reputation management systems to formulate a method to detect and limit possible malicious behavior in online systems. There is a significant amount of research literature in trust frameworks and reputation management systems. This section aims to provide an overview on some of them. First, it refers to the literature to get the definition and requirements of a reputation system. It then follows with a discussion on aggregation-based reputation systems employed by current online systems. As the focus of the project is a distributed and decentralized application, it concludes the section with a discussion on some of the reputation algorithms for distributed systems and the applications that use them.

3.1 Reputation systems and algorithms

As pointed out by Resnick et al. [62], a reputation system should be able to provide enough information to help infer the trustworthiness of participating users, encourage a user to be trustworthy and discourage a dishonest behavior. These attributes if enforced by a reputation system can spread honest interactions which can help to get a more accurate trust score based on the formulation method. The definition of honest and dishonest (or malicious) behavior depends on the context of the online interaction system in use. For instance, an honest behavior on a P2P file-sharing system can be based on the authenticity of the file being shared whereas an online shopping system can base it on a successful transaction outcome.

The earliest and most known internet reputation is that of eBay [63] which uses an aggregation-based reputation method that offers a feedback-based rating system. A user in eBay [64] can rate a transaction along with some textual feedback. The range of values used being {1, 0, -1}, positive, neutral and negative respectively. The final aggregated score is then computed by subtracting the total of positive and negative ratings. The system [64] could be judged as working based on the sales volume and the observation that more than half the buyers usually engage in providing feedback. However, this method fails to address issues such as Sybil attack, inactive participation (users fear retaliation from giving negative feedback), whitewashing attack. There are many non-eCommerce online systems such as StackExchange, yelp, Reddit that make use of similar reputation mechanism to filter the participating users and avoid serving

malicious participation. Most of the eCommerce systems employ a client-server architecture which lets a central entity in control of stored data. A single point of control is a single point of failure. In light of this, there have been studies and proposals on decentralized reputation methods for a distributed system. Especially significant for P2P systems such as file-sharing or content delivery applications for detecting the quality of file or content and the owner of those files.

TrustDavis [65] is a reputation system that addresses the concerns mentioned by Resnick et al. [63] by discouraging malicious participation and incentivizing honest participation and does so without a centralized service. It introduces the role of insurers between interacting entities such that a user can ask to be insured for their transaction or insure someone else transaction in exchange for a reward. If the transaction succeeds then the buyer receives the product, and the seller gets his/her payment as agreed upon. If the transaction fails, however, and the buyer does not receive the product (or receives an inferior quality product) then the buyer can contact the insurer and ask for the insured amount. The seller can do the same if the buyer acts maliciously. The system relies on the capability of an insurer to estimate failure probability.

Schaub, Alexander, et al. [66] proposes a Blockchain-based reputation model that recommends the use of blind signature [1] to disconnect the link between customer and ratings. Doing so lets a customer freely rate/review the transaction without fear of retaliation. The system addresses the fact that the information (shipping address, credit card number) about a customer will need to be disclosed to the service providers for processing the order. The system, therefore, allows a customer to create a unique identity for each transaction. It is more customer-centric in the sense that it allows only a customer to rate the transaction. A merchant has no say in the quality of transactions. Allowing only customers to rate and review a transaction leaves the possibility for a customer to initiate an unfair rating attack towards the merchant who has no say in it.

The most known and widely used [67] reputation algorithm in a P2P network is EigenTrust [68] which recommends a method to aggregate local trust values of all peers. It uses the notion of transitive trust, i.e., if a peer i trusts a peer j and peer j trusts peer k then i would also trust k. Peers can rate another peer as either positive or negative {-1, +1}. The users can decide if a peer can be considered trustworthy (e.g., as a download source) based on its total aggregated score. EigenTrust defines five issues that any P2P reputation system should consider. They are self-policing (i.e., enforced by peers and not some central authority), anonymity (i.e., peers' reputation should only be linked to an opaque identifier), no profit to newcomers (i.e., reputation should be obtained by continuous good behavior, and white-washing should not be profitable), minimal overhead for computation/storage and robust to malicious collectives of peers. This master's thesis project has followed these principles closely during the solution design of the proposed model. A significant disadvantage of EigenTrust is that it assigns a high rank to a static set of pre-trusted peers. Pre-trusted peer is a notion of trust, where few

[1] Blind signature schemes [27] is two-party protocols between a sender A and a signer B. A sends a piece of information to B which B signs and returns to A. From this signature, A can compute B's signature on an a priori message m of A's choice. At the completion of the protocol, B knows neither the message m nor the signature associated with it.

peers that join the network are assumed trustworthy by design. Given the dynamic nature of a distributed P2P networks, it is possible for a pre-trusted peer to make a poor decision and download inauthentic files. As the algorithm centers around pre-trusted peer, a group of nodes can form a malicious collective and share authentic files with the pre-trusted peers to get a good ranking while sharing malicious files with rest of the network. Thus, the system has limited reliability in the absence of trustworthy behavior from pre-trusted nodes [67].

As mentioned by Alkharji, Sarah, et al. [69], there are two primary methods of a reputation management systems, peer-based or file-based reputation system. The peer-based system allows peers in the network to be rated and assigned a reputation value. A file-based system is concerned more with the integrity of a file that is being delivered/served on the network regardless of who (peer) owns or serves it. AuthenticPeer++ [69] is a trust management system for P2P networks that combines both, i.e., it shares the notion of both trusted peers and trusted files. It only allows trusted peers to rank the file after they have downloaded it and uses a DHT-based structure to manage the integrity of file information.

Bartercast [70] is a distributed reputation mechanism designed for P2P file-sharing systems. It creates a graph based on data transfers between peers and uses the max flow algorithm to compute the reputation values for each node. Tribler [71] is a BitTorrent based torrent client that uses Bartercast to rank its peers.

PowerTrust [72] proposes a robust and scalable reputation system that makes use of Bayesian learning. Bayesian learning is a method that uses Bayes theorem to update the posterior probability of a hypothesis based on prior probability and probability of the event. It uses a Bayesian method to generate local trust scores and a distributed ranking mechanism to aggregate reputation scores.

3.2 Trust Model

Ries, S., Kangasharju, J. and Mühlhäuser, M., 2006, [73] analyzes the classification criteria of trust systems from the aspect of the domain, dimension, and semantics of trust values. Domains of trust values can be in binary, discrete or continuous representation. The dimension of trust values can be either one or multi-dimensional. In a one-dimensional approach, the value represents the degree of trust an agent assigns to another one whereas, in a multi-dimensional approach, the uncertainty of trust value is also allowed. The semantics of trust values can be in the set: rating, ranking, probability, belief, and fuzzy value. Ratings can be specified by a range of values on a scale, such as [1,4], where 1 can represent more trustworthy, and 4 can represent less trustworthy. Rankings are expressed in a relative way, such that a higher value means higher trustworthiness. The probability of a trust value represents the probability that an agent will behave as expected. Beliefs and fuzzy semantics are based on probability theory or belief theory. Abdul-Rahman, Alfarez, and Stephen Hailes [74] mentions that trust is dynamic and non-monotonic, i.e., additional evidence or experience with an entity can either increase or decrease the degree of trust in another agent. They propose a distributed trust

model [75] that addresses direct trust and recommender trust. Direct trust refers to the belief of an agent in the trustworthiness of another agent whereas recommender trust refers to the belief of an agent in the recommendation ability of another agent. It uses discrete labeled trust values as $\{vt, t, u, vu\}$ for representing very trustworthy, trustworthy, untrustworthy, and very untrustworthy respectively. Similarly, it uses $\{vg, g, b, vb\}$ for recommender trust to represent very good, good, bad and very bad respectively. An agent in this model maintains two separate sets for direct trust experience and recommender trust experience. PGP trust model [76] employs a web of trust approach instead of a centralized trusted authority for deriving certainty of the owner of a public key. Users sign each other's key that they know is authentic, and this process helps to build a web of trust. The two areas where trust is explicit in PGP are trustworthiness of public-key certificate and trustworthiness of introducer. Trustworthiness of introducer refers to the ability to trust the public key in being the signer of another public key. In the PGP web of trust, the discrete labeled values {undefined, marginal, complete} are used to represent the degree of confidence in the trustworthiness of a public-key. Undefined represents the statement "we cannot say if this key is valid or not", marginal represents, "maybe the key is valid", and complete represents the full confidence in that the key is valid. Similarly, to specify the trustworthiness of the introducer, it uses {full, marginal, untrustworthy}. Full refers to the full confidence in the key being able to introduce another key, marginal refers to it may be trusted to introduce another key but is uncertain if it can be fully trusted, and untrustworthy represents that the key cannot be trusted to introduce another public key. Jøsang, Audun [77] proposes a trust model that uses subjective logic to bind keys and their owners in public key infrastructure.

3.3 Blockchain applications

The significant attributes that constitute the blockchain are distributed, decentralized and time-stamped transactions that are stored in a cryptographically secure manner such that they are immutable and verifiable. There are several use cases and applications that require these attributes to implement a secure system. Trust and reputation system being one example. As mentioned earlier, the formulation and storage of reputation information need to be secure enough so that participants can rely on it. Verifiable information aids in predicting the outcome of a transaction correctly.

There are several blockchain platforms in operation today that allow for the creation of distributed and decentralized applications. Based on the use case, they vary in design goals and principles. Bitcoin [43] is said to be the first application that made use of blockchain technology. The purpose of bitcoin as the first application was to have a P2P electronic cash system in an open, public, distributed, decentralized and trustless (without the need to trust the participating nodes and without a trusted intermediary) environment. The idea of digital cash had been around since the 1990's with cryptographer David Chaum discussing untraceable electronic cash [78]. However, bitcoin was able to solve the flaw of a digital cash scheme, namely double spending problem, i.e., if A had 10 units of digital cash and sent it to B, then A should have this

amount deducted from his account such that he is not able to spend the same amount again. Bitcoin solved this problem without the use of trusted intermediaries and without requiring participating entities to trust each other. Similarly, it solved the Byzantine General's problem [79], [80] probabilistically by using Proof-Of-Work, which is a problem in distributed computing where several agents need to agree on a state over an unreliable network and without a trusted intermediary. Proof-of-work is discussed in more details by section 2.6.1. Examples of other platforms that extends bitcoin are Namecoin [81], which offers decentralized name resolution system, counterparty [2], which provides a platform for creating P2P financial applications. Hyperledger Fabric [82] is an open source permissioned distributed ledger technology platform that allows the applications to have blockchain components as a plug-and-play. Everledger [83] is a blockchain platform that provides a digital, global ledger to track and protect valuable assets. Ethereum [84] is a programmable blockchain platform that offers a Turing complete programming language and allows anyone to write smart contracts and execute code to change the blockchain state. Ethereum was considered to be suitable for this project for its open source ecosystem and the availability of several test networks and active community of developers with updated resources. Similarly, solidity [58] is a smart contract language employed by various blockchain platforms such as Ethereum [84], Tendermint [85], Counterparty. Therefore, this master's thesis project uses ethereum as a blockchain backend and solidity to write smart contracts and deploy the application.

[2]https://counterparty.io/docs/

4 Methodology and Implementation

The problem of measuring the trustworthiness of communicating entities is an essential aspect of any online system. This chapter states the problem of the current system and follows on a discussion of a proposed endorsement system where physically or digitally acquainted entities can endorse each other or their presented information. The user types and their roles in the endorsement system along with the design considerations are discussed. A system of smart contracts is set up to specify the rules of interaction and method to aggregate and compute the final global score for individual entities. Deployment of the contract to the blockchain network and analysis of data storage both on and off-chain is discussed.

4.1 Problem Statement

To be able to rely on the trustworthiness of an entity as presented by any online systems, the underlying reputation system needs to be robust and as transparent as possible. The assurance that available information has not been tampered with and correctness of claimed identity should be provided to sustain minimal risk of fraud. The centralized nature of current online systems leaves the propagation of reputation information vulnerable to external attacks as well as internal modifications. As such, it fails to provide the guarantee of reliable and immutable data. Additionally, the reputation systems do not take into account the anonymity of participants, which is an essential attribute for avoiding retaliation (e.g., a buyer may fear retaliation from a seller that provided bad service) from providing honest negative feedback. This master's thesis project proposes the use of blockchain technology for storage and governance of reputation data to ensure reliable and publicly verifiable information. By modeling trust among entities in a pseudonymous manner, the proposed system also considers the users' anonymity requirement.

4.2 User stories and System Requirements

This master's thesis project proposes a decentralized endorsement system to model trust among participating entities. As the name suggests, the proposed system allows participants to endorse each other to reflect their subjective opinion about other participants. Thus, the two distinct roles of users in this system are endorser (who sends an endorsement) and endorsee (who receives an endorsement). The relation between an endorser and an endorsee is an endorsement relationship. One might want to establish an endorsement relation with other entities in the network based on physical or digital acquaintance.

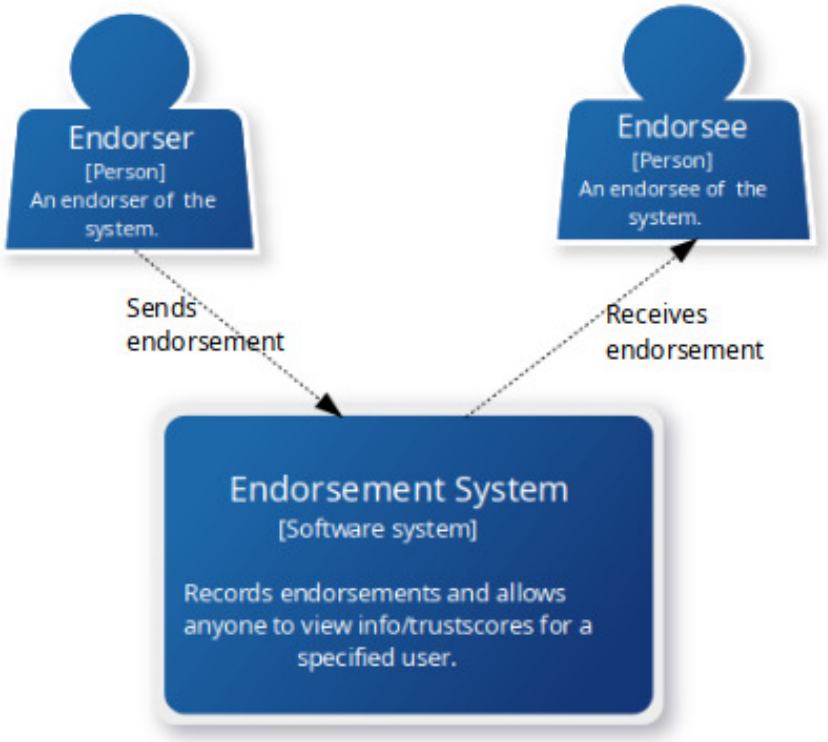

Figure 4.1: Context Layer

The acquaintance could be of the following form:

- Alice and Bob go to the same school/workplace, have worked on multiple projects together and therefore are confident of each other's reliability.
- Alice has dealt many times with Bob in an online shopping platform and always had a successful transaction outcome with him. In this interaction, Alice is sure that Bob is an honest seller and Bob is confident that Alice is a reliable buyer.
- Alice follows Bob on some social media and knows that Bob's article is good and sees lots of pre-research in his writing and is confident that Bob does not engage in spreading false news.

Based on Alice's previous experience with Bob, she is likely to endorse Bob on the endorsement system. Thus, the endorsement relationship aims to depict the direct, personal/interpersonal trust between entities. The endorsement system seeks to offer a simple computation model to aggregate the endorsement interactions and assign a reputation value to infer trustworthiness. If a participant A endorses a participant B, then it implies that A trusts B. As such, the domain of trust value in this system is binary. An entity A is either endorsed or not endorsed by B in the network. The Figure 4.1 shows the endorsement interaction between users (endorser and endorsee) and the endorsement system. In software development paradigm, user stories [86] provide an informal description of the feature that the system can have from an end user perspective. Sketching user stories for the endorsement system can help to define the roles and the associated specific features for a given user role. Table 4.2 presents the user stories for

As an	I need to be able to..	Requirement Id
Endorser	send an endorsement so that the endorsement is received by the endorsee.	R1
	remove endorsement so that the endorsement is removed from the endorsee.	R2
	view a list of endorsees so that i can see from whom i have received endorsements.	R3
	view/edit my personal information so that i can keep it up to date	R5
Endorsee	view a list of endorsers so that I can see from whom I have received endorsements.	R3
other users	compute the total endorsement impact (i.e., final computed score) of any registered members so that I can make an informed decision about the future transactions.	R4.1
	make a request to join the endorsement network so that I can start sending and receiving endorsements.	R4.2

Table 4.1: User Stories and Requirements

different user types of endorsement system and follows a role-feature-reason template [87]. The traceability column is used to trace back to that specific feature when checking the fulfillment of requirements in Section 5.3.

The users should be able to interact in the system based on their roles and the features allowed by the role. The endorsement acts like a transaction message that originates from a user account and is destined to another user account. As such, the originator account is an endorser and the destination account is that of endorsee. Every user maintains two separate lists of participants that they have interacted with. The first is the list of endorsers, that consists of the account addresses of all the participants that have endorsed the user. The second is the list of endorsees, that consists of the account addresses of all the participants that have been endorsed by the user. Account address acts like an identifier to the user. Based on this definition, we can lay out the system requirements.

The functional requirements can be listed as:

1. It must be impossible to make an endorsement if the endorser and endorsee belong to the same account.
   This requirement enforces a restriction that entities cannot endorse themselves.

2. It must be impossible to remove endorsement from a participant if the transaction initiator (account calling remove endorsement) is not in the list of endorsers for the participant.

3. All the endorsements must be stored such that, it is possible to see:

   - account address of endorser and endorsee for the given endorsement.

- degree of incoming and outgoing connections for all endorsers and endorsees.

4. There must be a way to link the account address (which is the public key address of an account) to their corresponding global score (the final score used to infer the trustworthiness).

5. It must be possible for a participant to edit their personal information such as their username. Since the endorsement system does not rely on the real-world identity of the participants, they should be allowed to edit or update their personal information at will. The trust score is linked to a unique identifier and cannot be actively updated by any participant. Computation of trust scores is explained by Section 4.3 which discusses the design of the endorsement system in detail.

The non-functional system requirements relate to the system architecture and can be listed as:

1. Security: The contract code for the endorsement system should take into account relevant security considerations as recommended by Solidity [88] to avoid known flaws [89] and minimize attack surface.

2. Reliability: The trust and reputation information should be stored in an immutable and publicly verifiable manner.

3. Trust metrics should describe the actual trust score of the nodes: The definition of honest or malicious interactions based on the endorsement model should be reflected by the final score assigned to the participant. As such, the nodes showing honest behavior should have a higher rank than the malicious ones in the endorsement system.

4.3 The Model - Endorsement Network

The figure 4.2 shows the steps required to have a complete "trust and reputation system" and is based on [23]. Among these steps, the endorsement system only concentrates on the first two steps, collection, and aggregation of information. The information in the endorsement system refers to the endorsement interactions between entities and values they represent. The last two steps of selecting a peer and getting punished or rewarded are based on a transactional system. As implementing the transaction-based system is not the main task of this project, feedback based on the success or failure of a transaction is purely based on an assumption about the transaction network.

4.3.1 Design of Endorsement System

The design of the endorsement system is based on the requirements mentioned in Section 4.2. This section explains the design considerations that were taken into account for the endorsement system. The Figure 4.3 shows different components of the endorsement system and how the users interact with them. Each of the components is described in Section 4.4. The endorsement

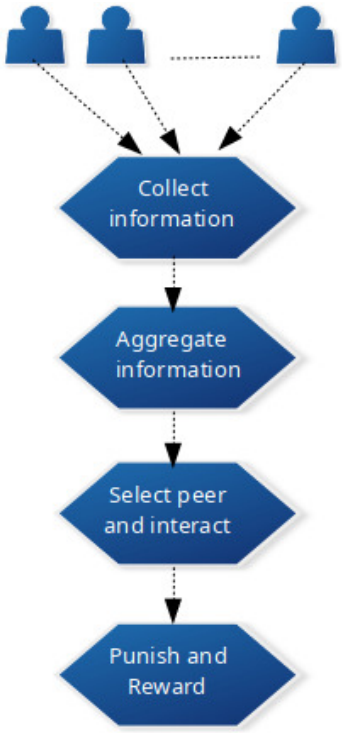

Figure 4.2: Trust and Reputation Model steps based on [23].

system allows participants to endorse each other based on their trust opinion. Before discussing the endorsement relationship, it is essential to understand the characteristics of trust that is taken into account by the system. Abdul-Rahman A, Hailes S. [75] have discussed these properties of trust in their study of a distributed trust model.

The endorsement system considers the following characteristics of trust:

Trust is Dynamic: The trust between two individuals can change over time. Thus, if an entity A endorsed B at a time t_1, then A can take back that endorsement from B at any time t_2 such that $t_2 > t_1$. The endorsement system then updates this information for both A and B to reflect the current state.

Trust is asymmetric: Trust relationship does not necessarily have to be bi-directional, i.e., A trusts B does not always imply that B trusts A too. Therefore, the endorsement system does not enforce any restriction on the endorsement relation between two entities to be asymmetric. This attribute is demonstrated in the endorsement system by not requiring a participant to endorse back the endorser, i.e., if A is endorsed by B, then it is entirely up to B to either endorse back A or not.

Trust is not transitive: The endorsement system does not consider the transitive trust path that can exist between endorser and endorsee. The only way an endorsement relation can be made between two entities is via direct endorsement. There exist trust metrics that are modeled

based on the transitive nature of trust. Depending on the context where the trust metrics is applied, this attribute can be meaningful, e.g., in eigentrust [68], a peer i is assumed to have a higher belief in the opinion of a peer from whom he/she has received authentic (non-malicious) files. Depending on the behavior of participants in the network, the transitive trust can have a positive or negative outcome. If the majority of participating nodes happen to be malicious actors (or collection of nodes that believes in false information) then the trust transitivity could result in the spread of incorrect information faster. Thus, to avoid the risk of spreading false beliefs, the endorsement system does not consider the transitive trust for the computation of a trust score. Christianson B, Harbison WS [90] have studied the non-transitive nature of trust in their study.

4.3.2 Computation of Total Endorsement Impact (TEI)

The endorsement system records all the endorsement interactions between endorsers and endorsees. This information is aggregated for individual entities in the system to assign a global trust score. We call this score a total endorsement impact as it is supposed to represent the total impact a node has made on the endorsement network. First, we define the terminologies related to the endorsement system and present the method to compute the value for total endorsement impact.

nEG$_A$: Number of Endorsements Given (nEG) by a participant A.

nER$_A$: Number of Endorsements Received (nER) by a participant A.

The participants must have a nEG and nER value equal to 1 to be considered a starting node and a value above 1 to be considered for making an impact in the endorsement system.

ratio$_A$: represents the ratio of nEG$_A$ to nER$_A$. This value can be used to ensure that the sent and received endorsement are not far off from each other. ratio$_A$ is always less than or equal to 1 and is given by:

$$ratio_A = \frac{min(nEG_A, nER_A)}{max(nEG_A, nER_A)} \tag{4.1}$$

CP$_A$: represents the Consumable Points (CP) of a participant A. Every node that joins the network receives an equal amount of consumable points from the endorsement network. This value keeps depleting with each outgoing endorsement connection. 1 being the initial consumable points received by participant A, CP_A is given by $\frac{1}{nEG_A}$.

TRP$_A$: This corresponds to Total Received Points (TRP), which is the accumulated sum of consumable points received by A from his/her endorsers.

If $E = \{e_1, e_2, e_3, ..., e_n\}$ represents the set of endorsers for a peer A and the size of E is n, then the TRP_A is given by:

$$TRP_A = \sum_{i=1}^{n} CP_{e_i} \tag{4.2}$$

Finally, the TEI made by A is given by:

$$TEI_A = ratio_A \cdot TRP_A \tag{4.3}$$

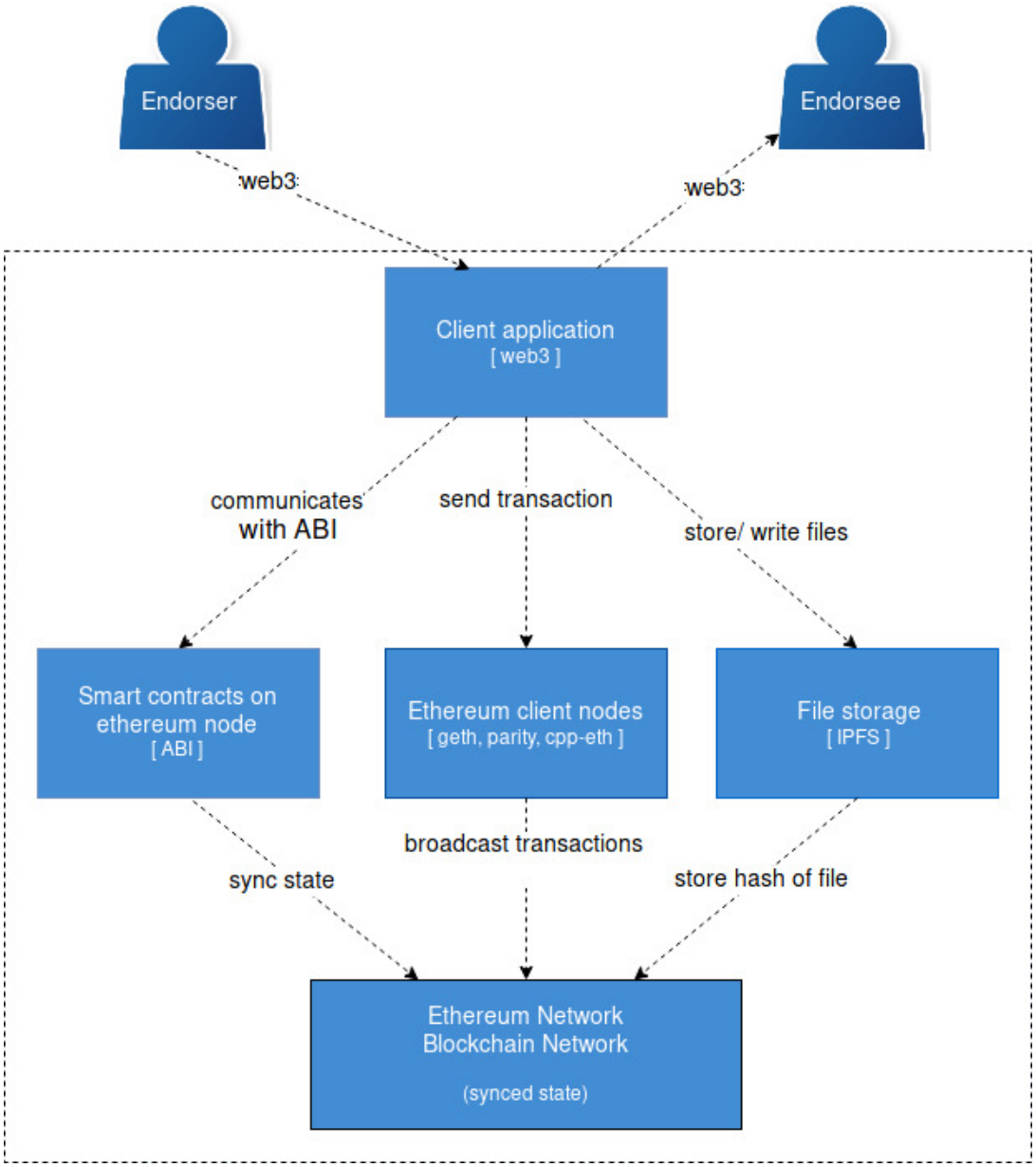

Figure 4.3: Components of Endorsement System

4.3.3 Design Considerations

The design of the endorsement system considers several possible behaviors that can result from the interaction between nodes, both honest and malicious. Any node that tries to manipulate (to inflate or damage the reputation) the trust score assigned by the endorsement system is a malicious node. For instance, a node can create multiple accounts to send multiple endorsements to a specific account. To limit dishonest behaviors while encouraging honest interactions, some assumptions and definitions were made from a game theoretic perspective of a behavioral outcome. Game theory [91] is a study of mathematical models of conflict and cooperation between intelligent rational decision-makers. The game refers to any social situation that involves two or more individuals, and players refer to the individuals involved in the game. Game theory makes assumptions that each player's objective is to maximize the expected value of his payoff, which is measured in some utility scale. Transferring this notion to the endorsement system, the players being the participants of the endorsement network. The assumption is made that the objective of each participant is to maximize the trust score in the endorsement system. Based on this assumption, some definitions were made to derive the network influencing factors. As such, these factors can encourage honest behavior while limiting malicious interactions in the endorsement system. The network influencing factors based on these assumptions are:

False endorsement with pseudonymous identities: Availability and public verifiability of reputation information is one of the primary concerns of endorsement system. As such, the system needs a public permissionless blockchain network which allows anyone to join the network and start sending endorsements immediately to whoever they wish to. This creates the possibility for an entity to create multiple pseudonymous identities with an aim to inflate their impact on the network by increasing the number of endorsements (given or received). There is no straightforward way to detect and stop such behavior. However, if doing so does not provide any significant advantage, then the assumption is that a rational decision would be not to do it. The endorsement network allocates an equal amount of consumable points to each user that joins the endorsement network. This value keeps depleting with each outgoing endorsement connection made along the way. As can be seen in Figure 4.4, the consumable points follow a convergent sequence that converges to the limit 0 as the number of connection 'n' increases. While there is no limit to the number of endorsements a participant can give, as this number increases, the value of consumable point decreases. This value accumulatively results to the total received points for the respective endorsee. Consider a scenario where a participant A receives endorsements from 3 endorsers, each having 2 outgoing connections. In this case, the value of TRP_A is 1.5. If the endorsers of A instead had 5 outgoing connections each, then the TRP_A would be only 0.6. TRP is one among other factors that contribute to making a better impact score on the endorsement network. As a rational endorser, one would be willing to make fewer meaningful endorsement connections that can contribute a larger value than to make many connections with minimal value. As such, the convergent behavior of consumable points is assumed to stop a participant from making too many endorsements.

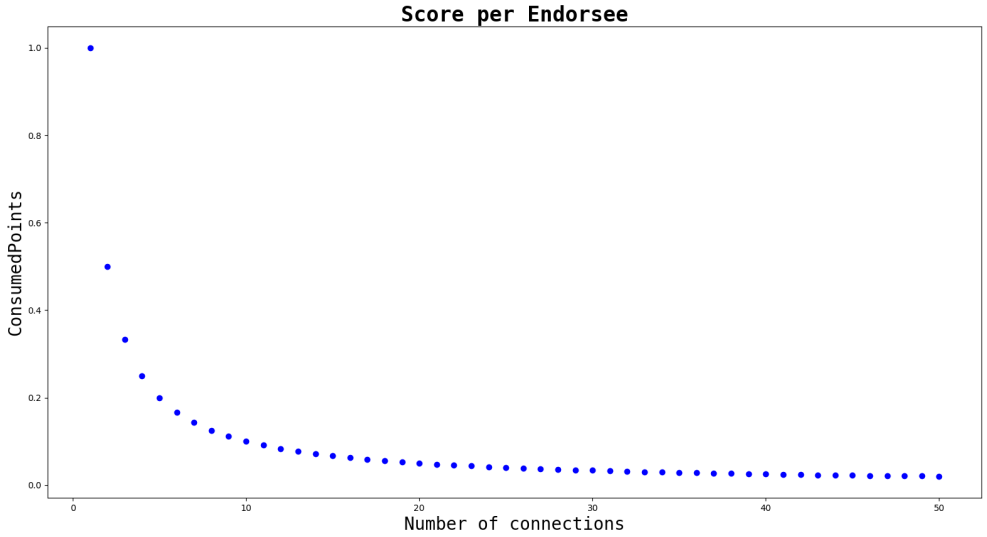

Figure 4.4: Convergent behaviour of consumable points as 'n' increases.

Transaction Cost: The endorsement system makes use of Ethereum as a blockchain infrastructure. As mentioned earlier, every operation executed on ethereum consumes a certain amount of gas, which is a scarce resource. The account that makes a call to endorse function is responsible for paying all the required gas costs. The endorse transaction if executed successfully updates the state variables nEG and nER for the source and destination account addresses. While the gas cost may not seem too high for making one transaction, a malicious node with multiple pseudonymous accounts needs to pay the gas cost of all transactions initiated from all the pseudonymous accounts. For instance, given the interaction graph in Figure 4.5, if Alice is an honest node, then she only needs to pay for the operation of two transaction. On the other hand, if both Bob and Charlie are the pseudonymous identities of Alice, she needs to pay for six transactions. As the number of pseudonymous accounts increases, the cost for maintaining the trust score on each (or one target) account also increases. Therefore, it is possible that the pseudonymous accounts exchange ether with each other for having the balance required to pay the transactions cost. This information can be publicly verified by anyone on the blockchain network to view the chain of ownership. If some interactions in the endorsement network look unusual (e.g., if an account has received too many calls for removing endorsements), then one could look up details as such. This kind of information acts as an additional factor that might be useful to look up before making a transaction decision. A successfully executed endorsement transaction modifies the nEG and CP for the source account and nER for destination account. Besides the source and destination account, the state of TRP for all past endorsers of source account also needs to be modified. Therefore, a node needs to keep track of all its neighboring (endorsers) nodes as well for correctly computing the TEI value. To get the total sum of CP, it requires iterating through the list of endorsers' accounts. The larger the size of this list, the larger would be the cost of the computation. While it is possible to iterate through the list of items in an array, Solidity does not generally recommend doing so. The reason being that an unbounded loop can grow too large and exceed the block gas limit, thereby causing the contract

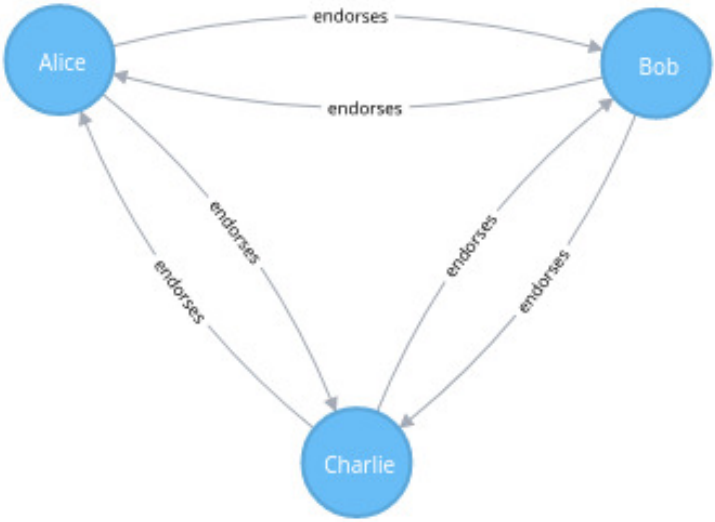

Figure 4.5: Interaction between participants that endorses each other

to be stalled. Regardless of the amount of gas spent, the function call will not succeed. We could assume that the list may not grow too big because a rational agent would try to limit their endorsement connections to contribute a larger value to other nodes in the network. Another reason we can assume so is that there is a limit to the number of entities that one can make trust decisions about. Dunbar's number [92] [1] says that there is a limit of 150-200 stable social relationships that humans can maintain. A study by Dunbar, R. I. (2016) [93] suggests that the growth of online media and interactions still do not overcome those limitations.

If we want to avoid relying on assumptions and expect the list to grow larger, there are a few ways to approach this issue. One way to address this is by setting an upper bound to the number of connections that a node can have. Another way to approach this problem without setting a threshold value is to move the computation to a non-blockchain platform. All the variables necessary to compute the TRP and TEI can be stored and updated on blockchain as a publicly verifiable information. The computation can be done on the client-side using language such as javascript and the client can compute the final score.

Free riders problem: The endorsement system is supposed to be a voluntary contribution network where entities can endorse each other. Therefore, the goal is to maintain a balanced ratio of incoming and outgoing endorsement connections. This is enforced in a way that if a node does not maintain the ratio then the TEI does not increase to make a significant impact on the network. This factor also discourages Sybil nodes because each identity needs to have an almost equal bi-directional connection. If one is only receiving from their pseudo identity that does not have too many connections, then the impact is ignorant and thus not worth the effort.

[1] Dunbar's number is a suggested cognitive limit to the number of people with whom one can maintain stable social relationships—relationships in which an individual knows who each person is and how each person relates to every other person.

4.3.4 Rewards and Punishment

The computation of a global score on the endorsement system is based on the subjective opinions of participants about each other. For the score to reflect accuracy in computing the probability of success for a real-world transaction, the score has to be updated based on some objective measure. As mentioned earlier, the endorsement system only considers the two steps of trust and reputation system. The complete steps can be seen in Figure 4.2 in Section 4.3. The reward and punishment relate to the fourth step which is based on a transactional outcome. Since the development and analysis of transactional network is not part of this thesis project, updating the trust score based on transactional feedback is not performed. A transactional network can retrieve the information on the endorsement system to provide additional conformity to its user about the reputation of an entity in question. Say, Alice is registered on Endorsement network and has made a decent score. If she wants to sell a product on a transaction network, she can claim the trust score she has on the endorsement system. Anyone can verify the claim by checking the score that corresponds to her public address. If both Alice and buyer are registered on the endorsement network, they can send a signed message to each other using their private key to prove the ownership of the address with a good score on the endorsement network. In case the buyer is not registered on the endorsement network, then Alice can prove the claim by signing a cryptographic challenge with her private key.

The notion of reward and punishment is an important one to reflect the current trust status of a peer. There are several ways one can reward or punish a node for its transactional behavior. For instance, a seller that failed to provide a good service for a certain amount of time (e.g., received five negative feedback continuously) could be punished. One simple method to punish the node would be by reducing the trust score made so far by 50%. Additionally, the nodes that endorsed an untrustworthy node could also be punished by reducing its trust score by 25%. Reducing the score made so far by a certain percentage will affect the users' reputation in the network. Anyone can see and verify this information. Similarly, punishing the endorser can encourage a user to be more careful about the trust decision they make.

4.4 Implementation

There are several components that make up the endorsement system. The process of sending the endorsement transaction from the client's browser to executing the contract code that can change the blockchain state is discussed in this section. It concludes with the discussion on storage of data and variables, blockchain network and consensus mechanism.

4.4.1 Smart contracts

The process of starting up the node and deploying the contract to the blockchain network is given by Figure 4.6. To compile the solidity code, solc compiler can be used. A successful compilation will give a binary representation of compiled EVM bytecodes and an ABI (Application Binary Interface). The binary output can be deployed to the blockchain network, after which

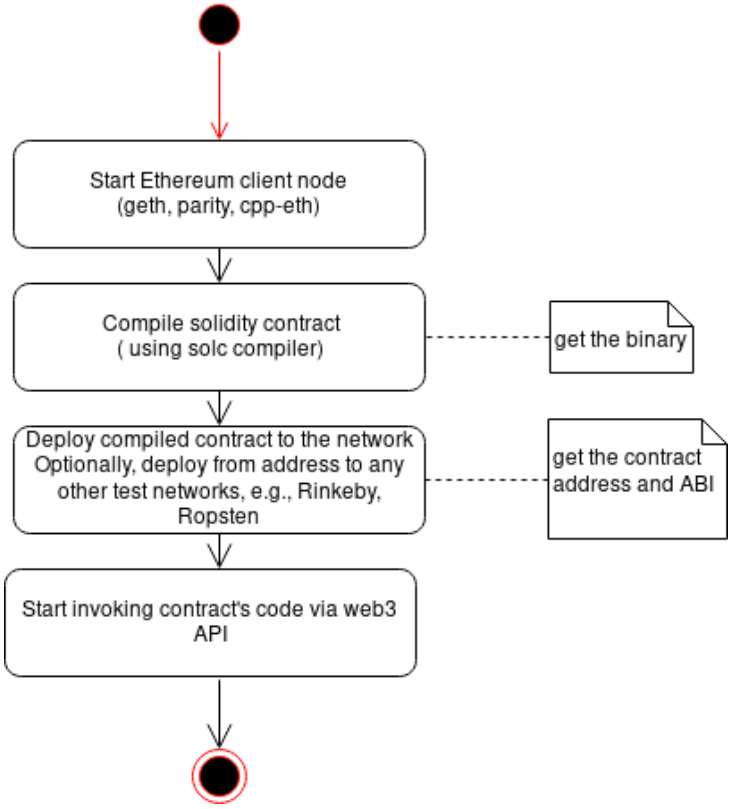

Figure 4.6: Deploy solidity contract on the blockchain network

the contract will get an address and the bytecode is stored on ethereum. ABI is a .json file that acts as a layer of translation for interacting with the deployed smart contracts and calling the functions. We can then start invoking contract's code using web3 API. For the compilation script and list of smart contracts, the reader is directed to Appendix A and B. A single contract that includes all the functionalities of the endorsement system is written as an endorsement contract. Other contracts to set the owner of the contract and allow the possibility to kill it is based on recommendations from zeppelin-solidity [94] and its reusable code.

The list of contracts written for the endorsement system and its functionalities are:

Ownable: tracks the owner of the contract by setting the address of owner.

Killable: inherits from Ownable and allows the owner to destroy the contract.

Endorsement: inherits from Ownable and killable. It specifies the core logic of the endorsement system. The method to join endorsement network, make endorsement interactions and request trust score of an entity are separate functionalities of this contract. Each of these methods is discussed below.

New participants: sets participant and stores their information. It maintains the records of all the participants and an index to access/query their information. When a user invokes the code to join the endorsement network, it stores the address of the user account that initiated the call

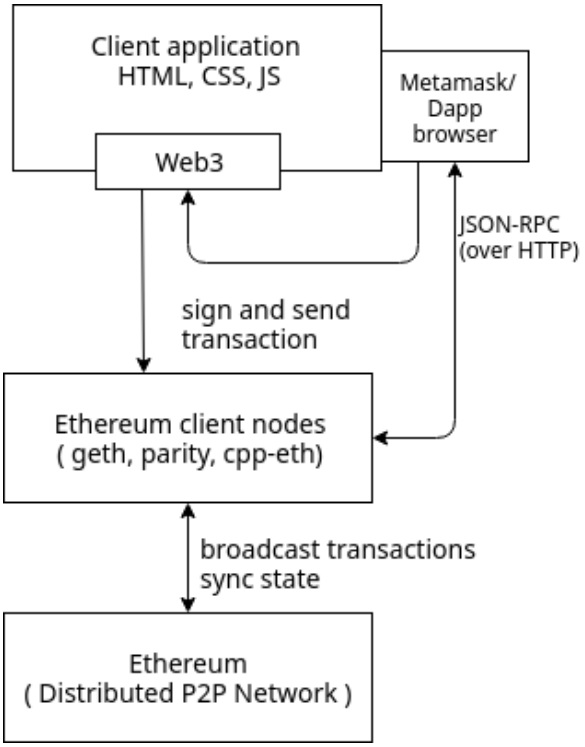

Figure 4.7: Client Application

and sets it as the participant. An index to access each participant is maintained that can be used to query the state.

Endorsement: allows participants to send endorsement transaction to the network. When the "endorse" method is invoked by the participant, the state variables nEG of the account that initiated the transaction and nER of the destination account gets incremented. The list of endorsers and endorsees is updated for both the accounts.

ComputeImpact: allows anyone to compute the total endorsement impact given the address of a participant. When this method is invoked, the TRP for the given participant is calculated by accumulatively summing the CP of the list of endorsers.

4.4.2 Client Application

The interaction between the client application and ethereum nodes is given by Figure 4.7 in Section 4.4.2. Reactjs [2] was used for front-end development of endorsement application. The endorsement contract that performs the endorsement logic was deployed to Ethereum test network, Rinkeby [3] which can be accessed publicly and the deployment steps can be seen in Appendix B. The client application makes use of web3 [95], which is a javascript API that allows clients to interact with a local or remote ethereum node. A user willing to interact with the endorsement system can submit the transaction via a client application. The client application

[2]https://reactjs.org/
[3]https://www.rinkeby.io/explorer

retrieves the required detail of the user from the key store, and signs the message with users private key and submit the transactions over to the client node. The client node executes the contract code that corresponds with the transaction message. A successful execution of the contract method would change the relevant states. One of the miner nodes would pick this transaction, broadcast it over to the blockchain network, and the transaction thus stays as an immutable and publicly verifiable information in the blockchain.

4.4.3 Data and variables on and off blockchain

For the endorsement system, the data required to identify the users are stored on the blockchain. However, it preserves the anonymity requirement mentioned in Section 4.2, as the publicly available information only links to the pseudonymous identity. The trust scores of an individual are just linked with the public key hash (account identifier). When sending a request to join the endorsement network, the user is asked to input a username. Unless a user explicitly wants to give out their real name, they are not required to be linked to real-world identity in any way. The current implementation of the endorsement system allows a user to only have a username as their profile information. Other variables related to trust scores are updated based on the endorsement interaction they make with others on the network. It is possible that storage requirements grow for reasons such as users willing to input more information about them (e.g., other online accounts, address) or the need to formulate complex trust metrics. As the data storage requirement increases, the amount of gas required for the transaction also increases. As such, we could use off-blockchain storage solution such as IPFS, Swarm [96]. The data can be stored off blockchain, and only the hash that points to the specific file in IPFS can be stored on the blockchain. Also, client-side assets (HTML, js) can be stored using the similar approach on the distributed off-chain file system, storing only the hash of the file location on the blockchain.

4.4.4 Blockchain and Consensus algorithms

The proposed blockchain platform for the endorsement system is Ethereum, a public, permissionless blockchain setup. As a public and permissionless network, it allows any nodes to collect transactions and act as a writer. The consensus mechanism that is generally used in a permissionless setting is PoW. As mentioned earlier, PoW is computationally expensive and wasteful. Using other consensus mechanisms such as DPoS requires finding enough trustworthy validators that can act as a leader or a master node which can be given authority to vote on behalf of the community. The endorsement system is meant to be a voluntary contribution network where anyone is free to join and endorse whoever they wish to. The participants do not necessarily know each other. Therefore, no "one" node can be trusted to collect transactions of everyone and make a final commit. Gochain [4] has mentioned the use of Proof-Of-Reputation (PoR) as a consensus mechanism in its ethereum based blockchain platform. The basic idea here is to allow participants on the network that have a high reputation to sign the block. It builds on the theory that it is not worthwhile for a reputed node to tarnish

[4] https://gochain.io/

the current reputation that took some time and effort to create. The endorsement system is designed to assign a global reputation score to each entity and could use PoR. Since reputation scores are significant to maintain, the nodes that have an impact score within some given range could be trusted to validate and sign the blocks of transactions. PoR on endorsement system can only be used if the endorsement that results in an impact value becomes a valuable asset over time through extensive use to be used on a transaction network. However, starting the endorsement system with only PoR is not recommended as the behavior of participants cannot be anticipated from the beginning. Only after several endorsement interactions and analysis of the nodes on the network and updating the trust score constantly based on transaction feedback, the trust score can become more reliable.

Therefore, the recommended consensus algorithm for the endorsement system is PoW despite its limitations. Various alternatives to PoW and consensus related researches focusing on current problems (e.g., transaction speed, transaction size, throughput) are under development. Hashgraph [97] proposes the recent advancement in consensus engine that claims to be fair (in the order of transactions), fast (transaction processing) and Byzantine fault tolerant. It is based on gossip protocol, where nodes gossip about transactions and gossips with each other, and the gossip eventually leads to all the nodes in the network having the same information. It offers mathematical proof of the total order of transactions with less communication overhead. However, the hashgraph conundrum is that their software is patented unlike other developments in the similar space. A developer must pay for making an API call using micropayment of the platform. Endorsement system can also be used on a permissioned setting, much like sovrin does [98]. Sovrin introduces a steward node who is trusted based on a signed agreement with sovrin [5] and has received approval as trusted institutions. The concept of steward nodes might be questionable regarding decentralization aspect of the network, but there are cases when this level of decentralization is enough for the security of an application. For instance, there can be a consortium of e-commerce platforms that could agree on a specific set of protocols. Validation of transactions could be based on a voting mechanism that requires the consent of 2/3 of trusted members. Doing so can significantly increase the transaction validation time compared to PoW.

[5] https://sovrin.org/library/steward-agreement/

5 Results and Evaluation

This chapter presents the evaluation of the endorsement model as proposed in Chapter 4 based on which the overall system design and functionality are assessed and final results are presented. It begins by discussing the relevant threat models and how the endorsement model addresses them.

5.1 Threat models

Relevant threat models and how the endorsement system addresses them is presented in this section.

Sybil attack: The endorsement system addresses the Sybil attack by requiring the endorsers of a peer to have a high impact value as well. A peer can create multiple identities and self-interact to send a large number of endorsements to direct to themselves. However, being endorsed by a new set of endorsers (or endorsers with no activity on the network) does not help to get a higher value as discussed earlier in Section 5.4. To overcome this, a malicious node may try to send endorsements among each other such that each identity has a significant impact value leading to a better trust score on the identity they intend to inflate the score of. However, doing so requires sending many endorsement transactions over to the etherum network and raises the cost of operation.

whitewashing: The idea of rewards and punishment discussed in Section 4.3.4 aids in lessening a whitewashing behavior. By punishing the misbehaving nodes in a way that decreases their score made so far significantly but still preventing the value to be lower than a new user, whitewashing is addressed. The punishment of a node requires communication with the transaction network to receive the feedback on a transactional outcome.

Free riders: Free riders issue is addressed by requiring the nodes to have a balanced ratio of outgoing and incoming connections.

Denial of service: Denial of service is addressed by deploying the endorsement system on a public, permissionless blockchain network. There is no way to know a priori the address of a validator node that will be signing the next block. Therefore, attackers do not know where to direct the attack to intrude the operation of endorsement transactions.

Self-promoting and Slandering attack: The cryptographic functions of the blockchain solve the possibility of this attack due to lack of data source authentication or data integrity verification, and once the transaction is added to the blockchain, the blockchain protocol provides the guarantee of the immutability of data and offers public verifiability of data. Another reason this attack is possible is by creating multiple Sybil identity which is addressed by the cost of operation to do so, i.e., the account that initiates the transaction (joining the network or sending endorsements) has to pay the cost of each operation.

Malicious collective: Malicious nodes can form a collective group and endorse each other until they all have a high TEI value to be considered trustworthy in the endorsement network. The endorsement system metrics allow the possibility for malicious collectives to be seen as more trustworthy. The current implementation of the endorsement system does not address this issue. However, the information available about an entity can be used, if needed to find out malicious nodes that explicitly interact within their group.

Consider a malicious collective of four nodes, $M = \{A, B, C, D\}$ that endorses each other. The endorsement system maintains two sets of data for each participant, one that contains the list of endorsers, and other that contains the list of endorsees.

If A represents the list of endorsers and A' represents the list of endorsees for the entity A, then, we can find the intersection of the sets A and A' to find out the list of common entities in these two sets. The elements of this intersection set should represent the entities with whom A has the symmetric trust relation with.

$$\begin{aligned} A &= \{B, C, D\} \\ A' &= \{B, C, D\} \\ A \cap A' &= \{B, C, D\} \end{aligned} \tag{5.1}$$

As we can see that the intersection set includes the same list as the list of endorsers and endorsees, it is most likely that these entities are forming a malicious collective to inflate each other's trust scores. However, this does not provide enough information to infer that a node is malicious with certainty. We cannot ignore the possibility that they know each other and the endorsement interaction is an honest one. The characteristic of trust as being asymmetric does not invalidate the existence of the symmetric trust, i.e., A trusts B does not imply B has to trust A but it is entirely up to B if he trusts A, and only B knows if the trust relationship is an honest or malicious one.

Hoffman, K., Zage, D. and Nita-Rotaru, C., 2009 [22] relates the process of finding the colluding nodes as such to finding a clique [1] of a certain size in a graph, which has been known to be NP-complete and only has heuristic based solutions. An important consideration to be taken if one wants to find the intersection of sets of endorsers and endorsees is that as the size of the sets grows, so does the computational complexity. Given, the amount of gas that each operation

[1] A clique [99] in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E.

costs, it does not seem to be a feasible solution for doing this form of computation on ethereum blockchain. As such, it is recommended to implement it on a client-side and only make the computation if invoked by the client.

5.2 Evaluation of Requirements

A descriptive approach based on the design evaluation method by Hevner A, Chatterjee S. [100] is used to assess the system requirements when relevant, e.g., security and reliability aspects of the system is assessed based on the information available regarding blockchain and smart contracts to demonstrate the utility of the system. The system is not only reliant on the smart contract code and its execution but also on the interaction theory discussed during the design of the endorsement system. For this, the endorsement interaction is simulated using the graph simulation tool neo4j [2]. The dataset used to simulate the interaction is taken from SNAP [101]. The details on the dataset are presented in Section 5.4. Given the time constraints, no form of structural/unit testing was performed. The purpose of this project was a PoC design to demonstrate the reputation model as a use case via the use of smart contracts and blockchain technology. An extensive experiment to simulate a real-world usage was not performed. However, manual testing was done during development using ganache [3] and deployment of contract on a local network. Additionally, front-end was developed to test contract function calls and communicate with contracts on blockchain network from the browser itself.

5.3 Fulfillment of User stories and Requirements

The method to examine fulfillment of user stories and requirement follows the method used by Hevner's descriptive design evaluation approach [100]. The table 5.3 presents the motivation for fulfillment of functional requirements. For the relevant smart contract code, refer to the appendix A. The fulfillment of non-functional system requirements is discussed below:

Smart contract security: Based on the recommendations by Solidity [88], contract function code was ordered as conditions, actions, and interactions where relevant. Doing so can avoid Re-entrancy bug, e.g., interrupting a function call in the middle of execution and re-entering before the previous execution is over. The function call can be made by both EOA or by a contract address. A maliciously crafted contract can make a function call repeatedly before the execution of the function ends or throws an exception which can cause the function to interact in unintended ways. As such, failing early by making the checks first in a function can avoid such a bug.

Reliability: This requirement relates to the immutability of data stored on the blockchain network which is ensured by PoW consensus algorithm. The data is stored on a public, permission less blockchain network which allows any node to commit a block of transactions to

[2] https://neo4j.com/
[3] https://github.com/trufflesuite/ganache

User	Traceability	Motivation for fulfillment
Endorser	R1	Any registered participant can make a call to endorse() function to send an endorsement to other registered participants on the network.
	R2	Any registered participant can call removeEndorsement() function just by providing the address of the endorsee they wish to remove.
	R3	Each call to endorse() function updates the state variable of the current endorser and endorsee, storing and updating the respective state variables. i.e., nEG, nER, index. This function call also invokes updateEndorsee() function to update the endorsee information accordingly.
Endorsee	R5.	The storage of personal information was not fully implemented by the endorsement PoC, therefore, editing personal information is irrelevant. However, change to pseudonym can be possible by just making a call to editProfile() by the participant.
other users	R4.1	Anyone can make a call to computeImpact() function to get the final computed score of a participant based on public key hash registered on the endorsement network.
	R4.2	Anyone can make a call to joinNetwork() function and become a registered participant of the network immediately.

Table 5.1: Fulfillment of User stories and Requirements for Endorsement PoC

the blockchain. A validator node collates the list of transactions into a block. A malicious node that intends to double spend a transaction can do so by solving the cryptographic puzzle in parallel with the rest of the network. If the malicious does not broadcast the solved blocks to the network and instead keeps solving the puzzle in isolation with the network. The transactions that the malicious node spent can be included in the blockchain that the network is in agreement with currently. After a certain length of blocks has been solved, the malicious node can then decide to broadcast his version of blockchain to the network. Blockchain protocol ensures that the network switches to the longest chain in the event of a fork. Since the malicious node is in the race with rest of the network, this form of attack is only possible if the malicious node owns more than 51% of the hashing power compared to the rest of the network. As long as honest nodes control half of the network, the PoW mechanism ensures an immutable record of transactions to be stored in the blockchain.

Trust metrics correctly describe the actual trust of the nodes: The fulfillment of this requirement is assessed by simulating an interaction graph and analyzing different scenarios in the network. The Section 5.4 presents the details regarding this requirement.

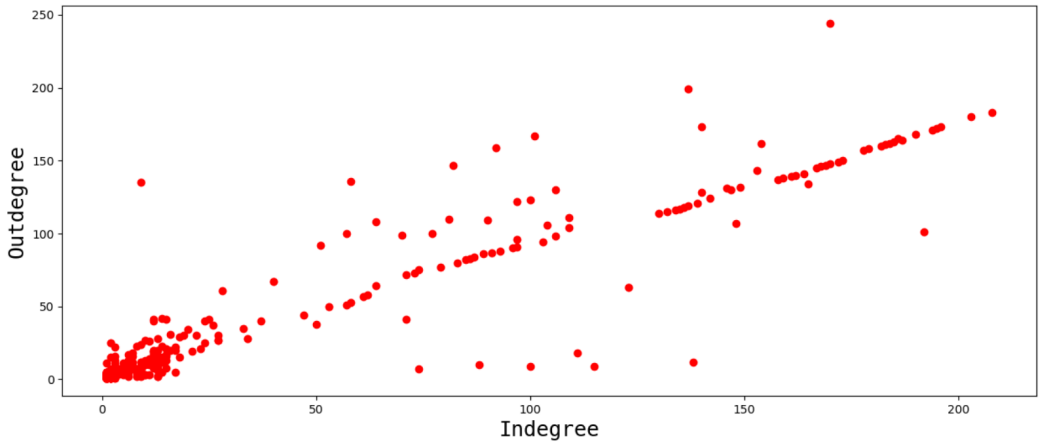

Figure 5.1: Distribution of degree of incoming and outgoing connections

5.4 Interaction graph

For the simulation of user interaction in endorsement network and the resulting impact value, a real-world data set was used. The dataset was extracted from Bitcoin Alpha trust [4] weighted signed network which is a who-trusts-whom network of people that trade on Bitcoin Alpha platform. Participants on this network rated each other on a scale of -10 to +10 where negative value represented total distrust whereas positive value represented total trust. It consisted of 3,783 nodes that made 24,186 edges out of which 93% of the edges were marked as positive edges[102]. The available information in the dataset for all the nodes was source, target, rating, and timestamp. All of which is essential information for endorsement network. The direction of endorsement is based on the source and target information. The timestamp information can help to decide on the order of transaction occurrence in the network. This information is particularly interesting for anomaly detection algorithm such as Net flow rate convergence as discussed in [103]. Unlike the Bitcoin Alpha network that let users rate on a scale of -10 to +10 to demonstrate the strength of their trust towards the users, the endorsement system only considers values from a binary domain, i.e., a user either endorses a specific claim made by the entity or does not endorse. There is no range of values to depict the strength or weakness. For making it more relevant to endorsement interaction, the existing dataset was filtered only to have edges with a rating above +2. No negative edges were considered for the endorsement simulation. As a result, the total number of nodes was reduced to 1677 with 4776 edges. Endorsement model was then applied to these nodes, and their total endorsement impact was computed based on their incoming and outgoing connections.

Total Endorsement Impact: This value is based on the degree of connections (nEG, nER)and TRP for each node. Out of 1677 nodes, there were 1175 nodes (70%) that turned out to have a TEI score of zero. On examining the nodes, they were found to have only one incoming or outgoing connections, i.e., the value of nEG or nEG was only one. As such, the TEI score

[4]https://alphabtc.com/blockchain/

of zero was expected because a node would only be considered for making an impact on the endorsement system if the number of connections is more than one. The score of zero, in this case, is not representative of a non-trustworthy node, but a starting node. Thus, we can say that 70% of the nodes in this network are new users. This computation leaves us with only 502 nodes to account for having a considerable TEI score. The distribution of nEG and nER among the participants of the network is given by Figure 5.1.

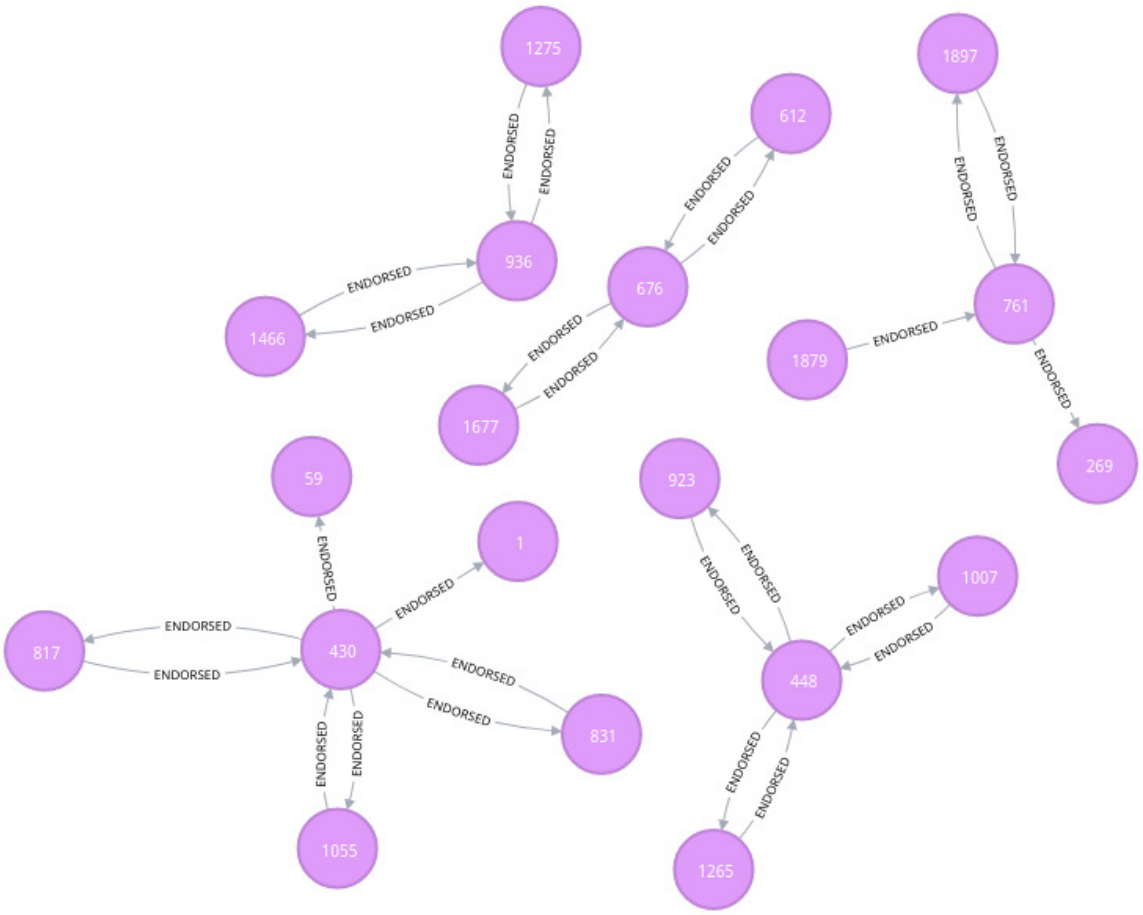

Figure 5.2: Interaction subgraph of nodes with zero impact

Total Received Points: Among the remaining 502 nodes, there were 5 nodes whose TEI score was zero despite having more than one incoming/outgoing connections. This value was because of TRP, which is another significant factor that the endorsement system takes into account. Though the nodes received endorsements to have a considerable amount of nER, their TRP was zero because their endorsers were not impactful (i.e., having a TEI score greater than zero or a nEG or nER score greater than one) in the network. The interaction subgraph for these five no-

Node label	nEG	nER	ratio	TRP	TEI
430	5	3	0.6	0	0
761	2	2	1	0	0
448	3	3	1	0	0
676	2	2	1	0	0

936	2	2	1	0	0

Table 5.2: Nodes with zero impact because of a non-impactful endorsers

des is given by Figure 5.2. As we can see from the figure, none of the endorsers for the five nodes have a significant number of connections for being an impactful node. This factor TRP corresponds to the prestige centrality metrics of a graph network where the significance of adjacent nodes determines the significance of a node. In this case, the significance of a node is not directly associated with TEI of the endorser but the value of CP of each endorser that accumulatively contributes to TRP for the respective node. Table 5.4 shows the TRP values across all the factors for these five nodes that explain why the TRP is zero which leads to them having a TEI score of zero too.

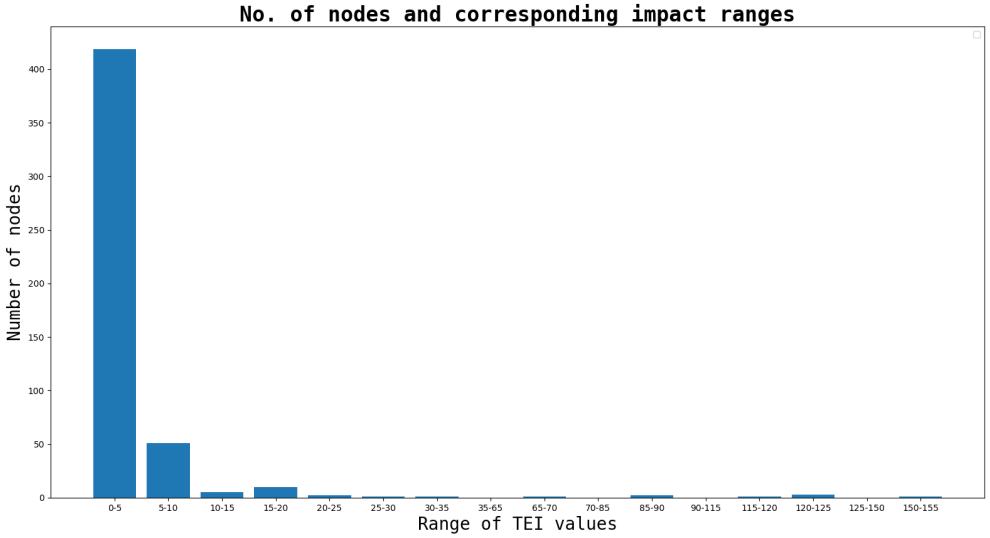

Figure 5.3: Distribution of TEI score over all nodes

Ratio: As we can see in Table 5.4 that some nodes have a maximum possible value of the ratio which is 1 and still has the lowest TEI score. It shows that maintaining the ratio between incoming and outgoing connections alone is not enough to have a significant impact on the network. As such, it is difficult to say if a node is impactful or not merely by looking at the ratio. A node that has a good ratio does not necessarily imply that it has a good TEI but a node with a good TEI value should have a higher ratio (value equal or approaching 1). This aspect is discussed in more detail by Section 5.5.

The distribution of TEI score over all nodes is given by Figure 5.3. As the endorsement system follows the ranking mechanism to rank the trustworthiness of nodes, the higher the value of TEI score higher is the trustworthiness of the given node. We can see from the Figure 5.3 that most of the nodes have a lower impact value within the range, i.e., about 400 of the nodes have the TEI in the range of 0-5. On the other hand, very few nodes have a high impact value, i.e.,

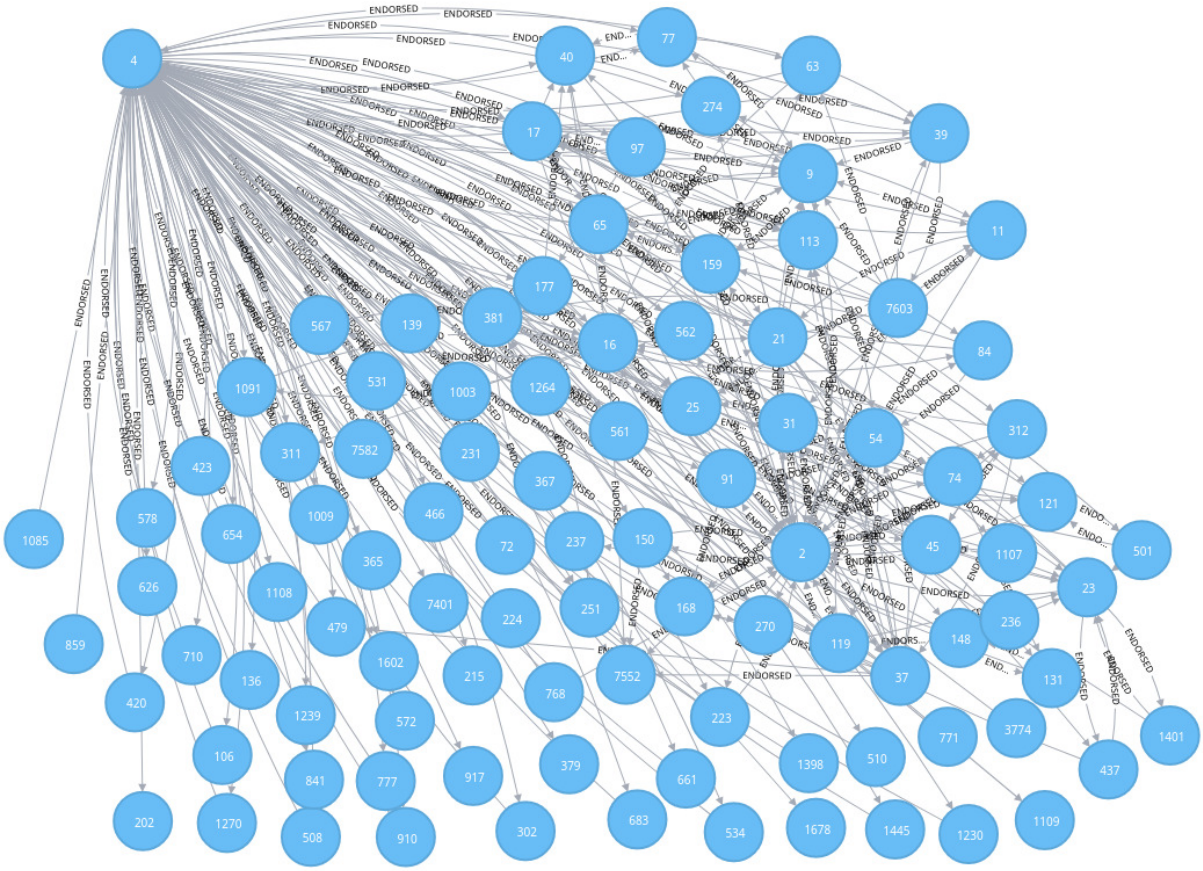

Figure 5.4: Interaction of nodes (labeled 2 and 4) with higher TEI score

about 1% of the total nodes have TEI score above the range of 100. Thus, these few nodes can be considered relatively much trustworthy than any other nodes in the network. The nodes labeled 4 and 2 in Figure 5.4 shows the interaction behavior of nodes with high TEI score.

5.5 Total impact across several factors with different scenarios

This section considers different scenarios to see how all the factors of the endorsement system (nEG, nER, ratio, TRP, TEI) are distributed for both high impact and low impact nodes. We have seen earlier that a high or low value of the ratio is not a useful measure to distinguish between a high impact or low impact node. Additionally, we also claimed that even though a high ratio value does not imply a high impact node, a high impact node should have a higher ratio (value equal to or approaching 1 and not 0 or approaching 0). Thus, we will consider four different scenarios here to test this behavior. The first two case takes a look at the node with a maximum impact value whereas the last two case (case 3 and case 4) looks at the node with a minimum impact value.

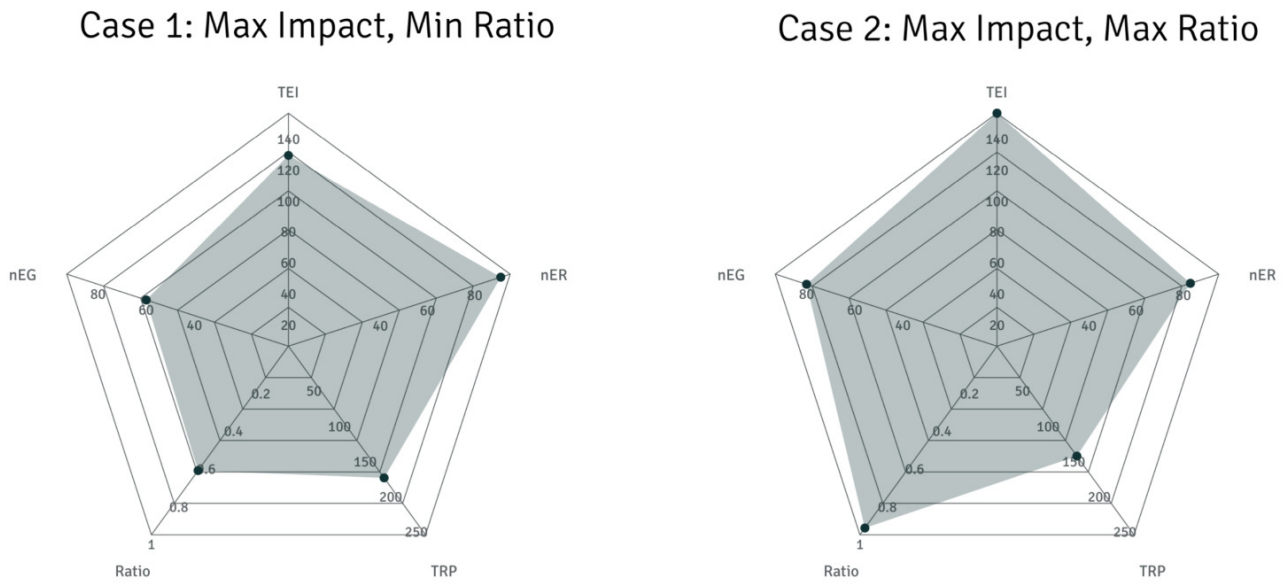

Figure 5.5: High impact node with maximum and minimum possible ratio

Case 1 & Case 2: Maximum impact nodes Here, we take the nodes that have a higher range of impact values and among them take the node that has the lowest ratio to depict case 1 and a node that has the highest ratio to depict case 2. We can see both these cases and their distribution across other factors in Figure 5.5. The lowest ratio for case 1 is 0.6 (value approaching 1) which is an expected value for a node with high impact. Similarly, for case 2, we see that the maximum ratio is 1 which is also expected as this node has the highest TEI score. We can see in the figure that for a high impact node, all other factors are almost evenly distributed and based on their value of nEG and nER (above 60 in both cases), we can say that they have a high degree centr-

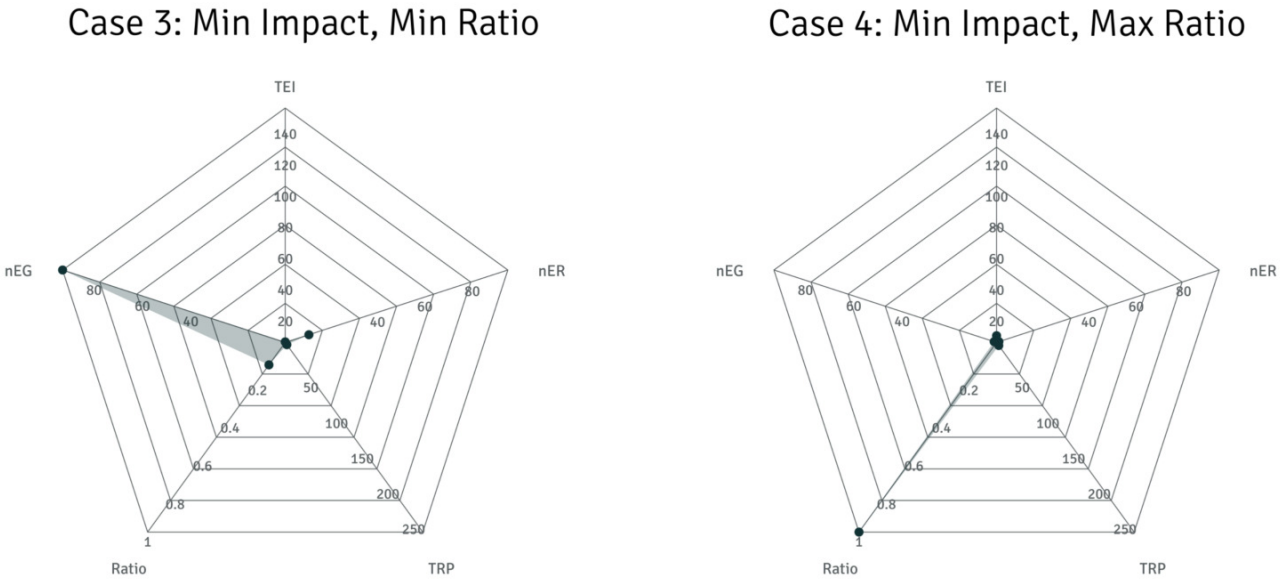

Figure 5.6: Low impact node with maximum and minimum possible ratio

ality as was shown by Figure 5.4 in Section 5.4. Therefore, they satisfy the requirement of being an impactful node in the network.

Case 3 & Case4: Minimum impact nodes This scenario is the same as the first two cases, except here we take the node with low impact values. We can see the distribution across all factors for these nodes in Figure 5.6. Based on our earlier assumption about the relation between ratio and TEI, these nodes can have a maximum ratio despite having a low TEI score. We can see from the figure that case 3 has too many outgoing connections but relatively very few incoming connections. As such the minimum possible ratio for this node is lower also that approaches zero. Similarly, case 4 shows the maximum possible ratio of a low impact node which is 1. Given that the node does not have enough incoming and outgoing connections yet, it has a very low impact value. This case shows the behavior of a starting node. The third case shows the behavior of a node that shows an extreme one-way connection whereas case 4 shows the behavior of the starting node. As such, the low impact made by these nodes is expected.

6 Discussion & Analysis

The endorsement system provides a method to assign a global trust score to the participants of the endorsement network. The TEI score represents the impact a node has made on the network by sending or receiving endorsements. Based on this score, one can infer the trustworthiness of an entity. Discussion of several possible threats that can exist in a trust and reputation system along with how the endorsement model addresses them is presented by Chapter 5. The issues that a reputation system should address based on principles defined by EigenTrust [68] and Resnick et. al [62] are discussed by the design considerations in Section 4.3.3 and taken into account by the endorsement model. To briefly summarize each of them:

Self-policing: By deploying a decentralized endorsement system on a public permission-less blockchain network and allowing anyone to execute the contract code and change the contract's state by performing endorsement interactions, the role of a trusted central entity is removed. The specifications of user interaction and the method to compute the trust score are defined by the smart contract. As such, the peers in the endorsement system are responsible for writing or modifying the reputation data that complies with the specified logic.

Anonymity: The reputation of a peer is associated with the public key hash of the account, i.e., the account identifier and not with the externally associated identity such as their IP address or a government-issued identity.

No profit to newcomers: As illustrated by Section 5.4 in Chapter 5, the starting node cannot have a significant impact value right away. They start with an ignorant value (i.e., 0) and it takes more than one endorsement interactions (incoming and outgoing) to be considered for the computation of TEI. After which, it takes several incoming and outgoing endorsement connections along with maintenance of several factors (i.e., ratio, TRP) to make a significant impact on the network. Thus, the model addresses the issue of "no profit to newcomers". Apart from these factors, the notion of punishment as discussed earlier ensures that there is no profit in joining as a newcomer by leaving the network as a non-reputable account.

Minimal Overhead: This relates to minimal overhead in terms of storage, computation, infrastructure and message complexity. Thus, this factor is directly related to Ethereum as infrastructure and the execution of the endorsement contract on the blockchain. There is no limit to the length of data a transaction message (e.g., computation of TEI) can hold but more data implies more gas fee since the amount of storage that every node (full nodes) have to store increases. The cost of transaction fee is paid by the account that initiates the transaction.

The transaction fee for the endorsement is associated with the update and storage operation. Sending an endorsement to another account updates the state variables nEG and nER and stores the new endorser and endorsee address for the respective accounts. The computation of total endorsement impact for a given entity A can be initiated by any node (not just the account owner). This computation step depends on the size of the endorsee array of A. As discussed in Section 4.4.3, it is recommended to do this computation on a non-blockchain platform while retrieving the variables stored on the blockchain network. The time it takes to have a transaction confirmed is dependent on the consensus mechanism and PoW being the recommended consensus mechanism, the network throughput should be less than 100 transactions per second.

Robust to malicious collectives of peers: This factor is rather complex, and the current implementation of the endorsement system does not enforce any behavior to avoid it. It is as likely for an honest group of nodes to endorse each other as it is for malicious nodes. The discussion on methods to address this issue is given by Section 5.1.

The attributes of a reputation system as pointed out by Resnick et al. [62] to be able to provide enough information to infer trustworthiness of users and encourage trustworthy behavior while discouraging dishonest behavior is addressed by the endorsement system. The TEI score associated with a participant can help to infer the trustworthiness of the participant in question. The contribution that a node can make on another node by sending out endorsements were limited by the consumable points (CP). Similarly, the ratio between incoming and outgoing endorsement connections had to be maintained for a significant impact value. As such, trustworthy behavior was encouraged within the network by requiring nodes to maintain these values. Details on these factors and assumptions regarding rational behavior outcome are given by Section 4.3.3.

The reputation score on the endorsement system is based on the number of endorsements one has received and the value associated with each connection. Unlike the formulation method used by online interaction systems such as eBay, where receiving 50 positive and 40 negative feedback is the same as receiving 10 positive feedbacks, in endorsement system, every endorsement received and given can play a role in raising the reputation score of an individual. A higher number of endorsement connections always imply a higher value for trustworthiness. As such, the formulation method employed by the endorsement system overcomes the limitation of simple aggregation-based methods used by online interaction systems such as eBay.

Similarly, it can compute the global trust score of an entity without using transitive trust path between entities or the notion of pre-trusted peers, unlike EigenTrust. Doing so allows making the computation step faster as it requires contacting only the respective node and the direct neighbor nodes (no need to ask friend or friends of friends) to infer the trustworthiness of the node. Aggregation of local trust scores can be done by looking at the list of endorsers (direct neighbors) for the node in question that can be used to compute the total received points. This

value along with the current state of the node is enough to calculate the global trust score in the endorsement system and infer the trustworthiness of the entity. There is no static set of pre-trusted peer defined in the beginning to act as a start vector for propagation of trust scores among nodes. However, if needed, a set of few trustworthy nodes can be picked after the aggregation of enough interactions that results in the significant impact for nodes. As shown by the results in Section 5.4, the distribution of higher reputation value among nodes is very few. Almost only 1 % of the nodes had a significant impact value (5 out of 496 nodes had an impact above 115) which is given by the Figure 5.3 in Section 5.4. These nodes can act as pre-trusted nodes (or trusted nodes) after several steps of aggregation so that their endorsement can be more impactful for other nodes they endorse. Given the dynamic nature of a P2P system, these nodes cannot remain static and will keep changing as more nodes join and leave, or they start behaving maliciously (receive calls for removing endorsements). Also, the notion of PoR as mentioned by Section 4.4.4 for assigning few reputable nodes to sign a block could use these few trustworthy nodes as inferred from the endorsement system. However, using these few selected trustworthy nodes will require additional steps on the distribution of scores and methods to dynamically set or change the list of trusted nodes at specific steps. This project does not perform the evaluation or analysis of using such a setup.

The research questions posed at the beginning of this projects and the sections that answer them are presented below:

Research Question 1: What are the requirements for storing trust values and linking them to associated identities stored off a blockchain network? How can a blockchain application be built to define a general trust framework for a transactional network? How could the overall system architecture look like?

The storage of data and variables along with the computation of trust scores is made on the blockchain network for the current implementation of the endorsement system. As the storage and computation requirement grows, a distributed P2P file system is recommended as an off-chain storage solution. The components of the endorsement system along with the overall flow from design to deployment is given by Section 4.3, 4.3.3, 4.3.2, 4.4.3.

Research Question 2: How can the discussed endorsement network ensure trustworthiness while also preserving user anonymity and how can it be generalized to other transactional network or added on top of it to serve other use cases such as content filtering, E-Commerce, etc.?

The endorsement system is designed as a general model where entities can endorse each other. The collection and aggregation of these endorsement interactions result in a final trust score to represent the trustworthiness of the respective entity. These aspects is discussed by Section 4.3.3, 5.3. Being a general model that collects the opinion of entities about each other, this system can be generalized and used by other online interaction systems for different use case which is discussed by Section 6.2.

6.1 Limitations

As mentioned earlier in Section 4.3, among the four steps required for a complete "trust and reputation system" the endorsement system only considers the first two steps. As such, the value of trustworthiness assigned by the system is based on the aggregation of endorsement information among entities only. For the reputation score to reflect higher accuracy and be useful in making decisions on real-world transactions, the score has to be continuously updated based on the transaction outcome. The update of score would require communication with the transaction network which can act as a feedback loop for the reputation score of the endorsement system. Based on the malicious behavior of a node on transaction network (e.g., failed to deliver the product, shared inauthentic files, low response time, etc.) they can be penalized, and the total endorsement impact can be reduced accordingly in the endorsement system. Given the limited timeframe of this project, the feedback system to update the reputation scores based on an objective measure from a transaction network was not implemented. The update of score would require implementation of a transaction network based on smart contracts (or existing transaction network with a method to communicate with them), devise of method for entities of transaction network to rate the transactions (e.g., 5 negative ratings on a transaction by an entity can be a measure of malicious node that can be sent to endorsement system) and a method of communication between transaction system and endorsement system to keep the feedback loop continuously run in an iterative fashion.

6.2 Generalization

The endorsement system is supposed to be a general model that allows entities to endorse each other (identity or the information associated with it) and aggregate these endorsement interactions to infer trustworthiness of associated identity. As such, any online interaction system (online commerce, file-sharing system, blog platforms) that requires inferring the trustworthiness of its participating entities could use it. Generally, most of the transaction system have their native reputation model (e.g., the difference between positive and negative ratings to assign a reputation score). In such case, the platform-specific reputation system can provide more information to endorsement system for the score of entities. If the transaction system does not have any reputation system, they can leverage the endorsement system to allow its participants to endorse the information as presented about each other in the transaction system, e.g., if an entity A had many successful transactions with other entity B, then A can endorse B. Other platforms/use cases can also similarly use the endorsement system. A centralized transaction system can use the endorsement system that can offer transactions on a centralized server (for transaction speed or scalability) but use a decentralized and distributed reputation system, such as the endorsement model to compute the reputation scores of entities. Doing so will increase the reliability of the reputation data and users are more likely to trust the scores.

7 Conclusion & Future works

Having a trust system for an online interaction system can aid in pre-evaluating the outcome of a transaction and help to increase the number of successful transactions. It is crucial to have a reliable and trustworthy reputation score that is resilient to both external attacks or internal modifications. This master's thesis project studied the existing trust and reputation models and proposed an endorsement model that allowed entities to endorse each other and provided a method to aggregate those endorsements to associate a reputation score to the participants. The proposed model does not seek to challenge the existing trust or reputation models. It only aims to offer a method for inferring the trustworthiness of entities based on simple computation and direct interactions, based on the perception of entities about one another. Additionally, the goal of the project was to study the advantages of using a distributed and decentralized network to store reputation information.

The limited time frame of this project did not allow to explore all the literature available about the trust and reputation models or to implement a complete reputation system that considers all the aspects. As such, many things were not covered by this project. The list of suggestions for future research or to further extend the current implementation are presented below:

- A method of communication between the endorsement system and transaction system so that the reputation scores assigned by endorsement system can be further updated to reflect the current state of entities.
- Use of existing trust and reputation models such as EigenTrust or PowerTrust on Blockchain network to evaluate its performance based on accuracy and usability.
- Use of anomaly detection algorithms to detect the malicious behavior of participating nodes and limit their interactions by penalizing them.
- Extend the ability of an endorsement system to include more information by entities (other online accounts, email address, etc.) and to endorse each information separately.
- Extensive research and implementation on the use of distributed file systems such as IPFS for a blockchain based reputation system. This can help to store more information and perform large computation (if needed).

Literature

[1] (2018). 'Internet world stats'. Accessed Aug 22, 2018, [Online]. Available: https://www.internetworldstats.com/stats.html/.

[2] (2018). 'Internet live stats'. Accessed Aug 22, 2018, [Online]. Available: http://www.internetlivestats.com/.

[3] A. A. Selcuk, E. Uzun, and M. R. Pariente, "A reputation-based trust management system for p2p networks", in *ccgrid*, IEEE, 2004, pp. 251–258.

[4] R. Stern, "Napster: A walking copyright infringement?", *IEEE micro*, vol. 20, no. 6, pp. 4–5, 2000.

[5] B. Cohen, "Incentives build robustness in bittorrent", in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[6] ——, *The bittorrent protocol specification*, 2008.

[7] M. Atzori, "Blockchain technology and decentralized governance: Is the state still necessary?", 2015.

[8] Experian, *The 2018 global fraud and identity report*, Accessed Aug 17, 2018, [Online]. Available: https://www.experian.com/assets/decision-analytics/reports/global-fraud-report-2018.pdf/, 2018.

[9] S. M. Al Pascual Kyle Marchini, Accessed Aug 17, 2018, [Online]. Available: https://www.javelinstrategy.com/coverage-area/2018-identity-fraud-fraud-enters-new-era-complexity#/, 2018.

[10] (2018). 'Macy's & bloomingdale's data breach: What you need to know'. Accessed Sep 05, 2018, [Online]. Available: https://www.experian.com/blogs/ask-experian/macys-bloomingdales-data-breach-what-you-need-to-know/.

[11] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts", in *2016 IEEE symposium on security and privacy (SP)*, IEEE, 2016, pp. 839–858.

[12] G. Zyskind, O. Nathan, *et al.*, "Decentralizing privacy: Using blockchain to protect personal data", in *Security and Privacy Workshops (SPW), 2015 IEEE*, IEEE, 2015, pp. 180–184.

[13] D. H. McKnight and N. L. Chervany, "The meanings of trust", 1996.

[14] ——, "Trust and distrust definitions: One bite at a time", in *Trust in Cyber-societies*, Springer, 2001, pp. 27–54.

[15] D. Gambetta *et al.*, "Can we trust trust", *Trust: Making and breaking cooperative relations*, vol. 13, pp. 213–237, 2000.

[16] G. Zacharia, A. Moukas, and P. Maes, "Collaborative reputation mechanisms for electronic marketplaces", *Decision support systems*, vol. 29, no. 4, pp. 371–388, 2000.

[17] J. Sabater and C. Sierra, "Review on computational trust and reputation models", *Artificial Intelligence Review*, vol. 24, no. 1, pp. 33–60, 2005.

[18] C. Castelfranchi and R. Falcone, "Trust and control: A dialectic link", *Applied Artificial Intelligence*, vol. 14, no. 8, pp. 799–823, 2000.

[19] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision", *Decision support systems*, vol. 43, no. 2, pp. 618–644, 2007.

[20] W. Stallings, *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, NJ, 2017.

[21] L. Rasmusson and S. Jansson, "Simulated social control for secure internet commerce", in *Proceedings of the 1996 workshop on New security paradigms*, ACM, 1996, pp. 18–25.

[22] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems", *ACM Computing Surveys (CSUR)*, vol. 42, no. 1, p. 1, 2009.

[23] F. G. Mármol and G. M. Pérez, "Security threats scenarios in trust and reputation models for distributed systems", *computers & security*, vol. 28, no. 7, pp. 545–556, 2009.

[24] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and whitewashing in peer-to-peer systems", *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 5, pp. 1010–1019, 2006.

[25] N. Andrade, M. Mowbray, W. Cirne, and F. Brasileiro, "When can an autonomous reputation scheme discourage free-riding in a peer-to-peer system?", in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, IEEE, 2004, pp. 440–448.

[26] J. A. Bondy, U. S. R. Murty, *et al.*, *Graph theory with applications*. Citeseer, 1976, vol. 290.

[27] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[28] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[29] X. Wang and H. Yu, "How to break md5 and other hash functions", in *Annual international conference on the theory and applications of cryptographic techniques*, Springer, 2005, pp. 19–35.

[30] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full sha-1", in *Annual International Cryptology Conference*, Springer, 2017, pp. 570–596.

[31] (). 'Oak ridge national laboratory's next high performance supercomputer'. Accessed Sep 05, 2018, [Online]. Available: https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.

[32] (). 'Top500 list - june 2018'. Accessed Sep 05, 2018, [Online]. Available: https://www.top500.org/list/2018/06/.

[33] (). 'Nvidia tesla gpu accelerators'. Accessed Sep 05, 2018, [Online]. Available:https://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf.

[34] (). 'Nvidia tesla v100 gpu architecture'. Accessed Sep 05, 2018, [Online]. Available:http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[35] G. Becker, "Merkle signature schemes, merkle trees and their cryptanalysis", *Ruhr-University Bochum, Tech. Rep*, 2008.

[36] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in p2p systems", *Communications of the ACM*, vol. 46, no. 2, pp. 43–48, 2003.

[37] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[38] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[39] G. Mike, "Just enough bitcoin for ethereum", Accessed 23 Dec 2017, [Online]. Available: https://media.consensys.net/time-sure-does-fly-ed4518792679/, Consensys, Oct 12, 2015.

[40] "Blockchain and distributed ledger technologies", ISO/TC 307, Standards Australia, 2016.

[41] A. M. ANTONOPOULOS, *MASTERING ETHEREUM, Building Smart Contracts and Dapps*. O'REILLY MEDIA, 2017.

[42] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain", in *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, IEEE, 2017, pp. 2567–2572.

[43] N. Satoshi, "Bitcoin: A peer-to-peer electronic cash system", Accessed 23 Dec 2017, [Online]. Available: https://bitcoin.org/bitcoin.pdf/, Bitcoin, Oct 31, 2008.

[44] V. Buterin *et al.*, "Ethereum white paper", *GitHub repository*, 2013.

[45] N. Houy, "It will cost you nothing to'kill'a proof-of-stake crypto-currency", 2014.

[46] V. Buterin and V. Griffith, "Casper the friendly finality gadget", *arXiv preprint arXiv:1710.09437*, 2017.

[47] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol", in *Annual International Cryptology Conference*, Springer, 2017, pp. 357–388.

[48] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains", in *Proceedings of the Thirteenth EuroSys Conference*, ACM, 2018, p. 30.

[49] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An introduction", *R3 CEV, August*, 2016.

[50] H. Fabric. (2017). 'Gossip data dissemination protocol'. Accessed Aug 31, 2018, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/gossip.html.

[51] ——, (2017). 'Hyperledger fabric model'. Accessed Aug 31, 2018, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/fabric_model.html.

[52] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance", in *OSDI*, vol. 99, 1999, pp. 173–186.

[53] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm.", in *USENIX Annual Technical Conference*, 2014, pp. 305–319.

[54] K. Voronchenko, "Do you need a blockchain?", 2017.

[55] N. Szabo, *Smart contracts http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html/*, 1994.

[56] ——, "Smart contracts: Building blocks for digital markets", *EXTROPY: The Journal of Transhumanist Thought,(16)*, 1996.

[57] ——, "Formalizing and securing relationships on public networks", *First Monday*, vol. 2, no. 9, 1997.

[58] (2016-2018). 'Introduction to smart contracts'. Accessed Aug 31, 2018, [Online]. Available: https://solidity.readthedocs.io/en/v0.4.24/introduction-to-smart-contracts.html.

[59] (2017). 'Lll introduction'. Accessed Aug 31, 2018, [Online]. Available: https://lll-docs.readthedocs.io/en/latest/lll_introduction.html.

[60] (2018). 'Beigepaper: An ethereum technical specification'. Accessed Sep 07, 2018, [Online]. Available: https://github.com/chronaeon/beigepaper/blob/master/beigepaper.pdf.

[61] *What are smart contracts, http://www.chainfrog.com/wp-content/uploads/2017/08/smart-contracts-1.pdf/*, ChainFrog, 2017.

[62] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, "Reputation systems", *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, 2000.

[63] P. Resnick and R. Zeckhauser, "Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system", in *The Economics of the Internet and E-commerce*, Emerald Group Publishing Limited, 2002, pp. 127–157.

[64] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood, "The value of reputation on ebay: A controlled experiment", *Experimental economics*, vol. 9, no. 2, pp. 79–101, 2006.

[65] D. B. DeFigueiredo and E. T. Barr, "Trustdavis: A non-exploitable online reputation system", in *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference on*, IEEE, 2005, pp. 274–283.

[66] A. Schaub, R. Bazin, O. Hasan, and L. Brunie, "A trustless privacy-preserving reputation system", in *IFIP International Information Security and Privacy Conference*, Springer, 2016, pp. 398–411.

[67] N. Chiluka, N. Andrade, D. Gkorou, and J. Pouwelse, "Personalizing eigentrust in the face of communities and centrality attack", in *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, Accessed Sep 05, 2018, [Online] Available:https://www.researchgate.net/profile/Dimitra_Gkorou/publication/229059481_Personalizing_EigenTrust_in_the_Face_of_Communities_and_Centrality_Attack/links/551afadd0cf2bb75407877f8.pdf, IEEE, 2012, pp. 503–510.

[68] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks", in *Proceedings of the 12th international conference on World Wide Web*, ACM, 2003, pp. 640–651.

[69] S. Alkharji, H. Kurdi, R. Altamimi, and E. Aloboud, "Authenticpeer++: A trust management system for p2p networks", in *European Modelling Symposium (EMS), 2017*, IEEE, 2017, pp. 191–196.

[70] M. Meulpolder, J. A. Pouwelse, D. H. Epema, and H. J. Sips, "Bartercast: A practical approach to prevent lazy freeriding in p2p networks", in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, IEEE, 2009, pp. 1–8.

[71] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. Epema, M. Reinders, M. R. Van Steen, and H. J. Sips, "Tribler: A social-based peer-to-peer system", *Concurrency and computation: Practice and experience*, vol. 20, no. 2, pp. 127–138, 2008.

[72] R. Zhou and K. Hwang, "Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing", *IEEE Transactions on parallel and distributed systems*, vol. 18, no. 4, pp. 460–473, 2007.

[73] S. Ries, J. Kangasharju, and M. Mühlhäuser, "A classification of trust systems", in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2006, pp. 894–903.

[74] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities", in *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, IEEE, 2000, 9–pp.

[75] ——, "A distributed trust model", in *Proceedings of the 1997 workshop on New security paradigms*, ACM, 1998, pp. 48–60.

[76] A. Abdul-Rahman, "The pgp trust model", in *EDI-Forum: the Journal of Electronic Commerce*, vol. 10, 1997, pp. 27–31.

[77] A. Jøsang, "An algebra for assessing trust in certification chains.", in *NDSS*, vol. 99, 1999, p. 80.

[78] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash", in *Conference on the Theory and Application of Cryptography*, Springer, 1988, pp. 319–327.

[79] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[80] A. Miller and J. J. LaViola Jr, "Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin", *Available on line: http://nakamotoinstitute. org/research/anonymous-byzantine-consensus*, 2014.

[81] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design.", in *WEIS*, Citeseer, 2015.

[82] C. Cachin, "Architecture of the hyperledger blockchain fabric", in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, vol. 310, 2016.

[83] N. Lomas, "Everledger is using blockchain to combat fraud, starting with diamonds", *URL: https://techcrunch. com/2015/06/29/everledger*, 2015.

[84] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[85] (2018). 'What is tendermint?' Accessed Aug 31, 2018, [Online]. Available: https://github.com/tendermint/tendermint/blob/master/docs/introduction/introduction.md.

[86] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[87] M. BERTEIG. (2014). 'User stories and story splitting'. Accessed Sep 03, 2018, [Online]. Available: http://www.agileadvice.com/2014/03/06/referenceinformation/user-stories-and-story-splitting/.

[88] Ethereum, "Security considerations", Accessed Sep 03, 2018, [Online]. Available: https://solidity.readthedocs.io/en/v0.4.24/security-considerations.html.

[89] ——, "List of known bugs", Accessed Sep 03, 2018, [Online]. Available: https://solidity.readthedocs.io/en/v0.4.24/bugs.html#known-bugs.

[90] B. Christianson and W. S. Harbison, "Why isn't trust transitive?", in *International workshop on security protocols*, Springer, 1996, pp. 171–176.

[91] R. B. Myerson, *Game theory*. Harvard university press, 2013.

[92] R. A. Hill and R. I. Dunbar, "Social network size in humans", *Human nature*, vol. 14, no. 1, pp. 53–72, 2003.

[93] R. I. Dunbar, "Do online social media cut through the constraints that limit the size of offline social networks?", *Royal Society Open Science*, vol. 3, no. 1, p. 150 292, 2016.

[94] S. C. Solutions. (2016). 'Zeppelin-solidity, docs'. Accessed Sep 05, 2018, [Online]. Available: https://openzeppelin.org/api/docs/open-zeppelin.html/.

[95] Ethereum. (2016). 'Web3.js - ethereum javascript api'. Accessed Sep 05, 2018, [Online]. Available: https://web3js.readthedocs.io/en/1.0/.

[96] J. Benet, "Ipfs-content addressed, versioned, p2p file system", *arXiv preprint arXiv:1407.3561*, 2014.

[97] L. Baird, "Hashgraph consensus: Fair, fast, byzantine fault tolerance", Swirlds Tech Report, Tech. Rep., 2016.

[98] A. Tobin and D. Reed, "The inevitable rise of self-sovereign identity", *The Sovrin Foundation*, 2016.

[99] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009, Accessed Sep 04, 2018, [Online]. Available: http://thuvien.thanglong.edu.vn:8081/dspace/bitstream/DHTL_123456789/3760/2/introduction-to-algorithms-3rd-edition.pdf.

[100] A. Hevner and S. Chatterjee, "Design science research in information systems", in *Design research in information systems*, Springer, 2010, pp. 9–22.

[101] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*, http://snap.stanford.edu/data/, Jun. 2014.

[102] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos, "Edge weight prediction in weighted signed networks", in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, IEEE, 2016, pp. 221–230.

[103] M. Buechler, M. Eerabathini, C. Hockenbrocht, and D. Wan, "Decentralized reputation system for transaction networks", Technical report, University of Pennsylvania, Tech. Rep., 2015.

Appendices

A [Appendix: SmartContracts]

```
pragma solidity ^0.4.18;

import "./Ownable.sol";
import "./Killable.sol";
import "./MarketPlace.sol";

contract Endorsement is Ownable, Killable, MarketPlace {

  address owner;

    struct Participant {
        address identifier;
        string name;
    }

    struct Endorser {
    uint index;
        address sender;
        uint nEG;
        uint usedPower;
        address[] givenTo;
        mapping(address => bool) hasGivenTo;
    }

  struct Endorsee {
    uint index;
    address receiver;
    uint nER;
  //  uint receivedPoints;
    address[] receivedFrom;
    mapping(address => bool) hasReceivedFrom;
  }
  Participant [] public participants;

  uint numberOfParticipants;
  mapping(address => bool) joined;
  mapping (address => Endorser) endorsers;
  address[] public endorserAccts;

```

```
40  mapping (address => Endorsee ) endorsees;
41  address[] public endorseeAccts;
42
43
44  mapping (address => uint) public prevLength;
45  mapping (address => uint) public receivedPoints;
46
47  // modifiers
48  // set owner of contract - replace eventually with Ownable contract
49  modifier onlyOwner() {
50    require(msg.sender == owner );
51    _;
52  }
53
54  //reject any ether transfer
55  modifier HasNoEther( ){
56    require(msg.value == 0);
57    _;
58  }
59
60  //constructor
61  function Endorsement() public {
62    //EDSToken( );
63    owner = msg.sender;
64  }
65
66  //event logs
67  event LogJoinNetwork(
68    address _participant,
69    string _name
70  );
71
72  event LogEndorse(
73    address _endorser,
74    address _endorsee
75  );
76
77  address [] public allParticipants;
78
79  mapping (address => uint ) participantIndex;
80
81
82  //Join Network as any user
83  function joinNetwork(string _userName) public HasNoEther {
84
85    //only allow unregistered participant
86      require(!joined[msg.sender]);
87
88      joined[msg.sender] = true;
89
90    // store senders id and name
91      Participant memory newParticipant = Participant({
```

```
92            identifier: msg.sender,
93            name: _userName
94        });
95
96      //add new participant to the existing list of joined members
97      participants.push(newParticipant);
98      numberOfParticipants++;
99      participantIndex[msg.sender] = numberOfParticipants-1;
100
101     LogJoinNetwork(msg.sender, _userName);
102
103     allParticipants.push(msg.sender);
104   }
105
106   //get list of all participants
107   function getAllParticipants() public view returns(address[]) {
108
109     return allParticipants;
110   }
111
112   //get index of participant by address, helper function, view modifier
113   function getParticipantIndex(address _participant) public view returns (uint ) {
114
115     uint userIndex = participantIndex[_participant];
116     return userIndex;
117   }
118
119   //Profile-related changes of participants
120   function getName(uint _index ) public view returns (string ) {
121
122     string name = participants[_index].name;
123     return name;
124   }
125
126   function editProfile(address _participant,
127               string _name
128             ) public HasNoEther {
129
130     //verify editor is same as profile owner
131     require(msg.sender == _participant);
132
133     //change state
134     uint id = getParticipantIndex(_participant);
135     participants[id].name = _name;
136   }
137
138   //send Endorsement - from  endorser to endorsee
139   function endorse(uint _index) public HasNoEther {
140
141     // get address of endorsee
142       address receiver = participants[_index].identifier;
143
```

```
144
145     //verify endorser and endorsee are different and registered
146     require(joined[msg.sender]);
147     require(receiver != 0x0);
148     require(receiver != msg.sender);
149
150     //store and update new endorser information
151     Endorser storage endorser = endorsers[msg.sender];
152
153     endorser.index++;
154     endorser.sender = msg.sender;
155     endorser.nEG++;
156
157 //    if (endorser.nEG >1 ){
158 //      endorser.usedPower = Division(1, endorser.nEG,9);
159 //    } else {
160 //      endorser.usedPower = 0;
161 //    }
162
163       endorser.usedPower =Division(1,endorser.nEG, 9);
164       endorser.givenTo.push(receiver);
165       endorser.hasGivenTo[receiver] = true;
166
167       endorserAccts.push(msg.sender) - 1;
168
169     //trigger call for updating endorsee information
170     updateEndorsee(receiver, msg.sender);
171
172     //Log endorsement event
173     LogEndorse(msg.sender, receiver);
174   }
175
176   //store and update new endorsee information after transaction call
177   function updateEndorsee(address _receiver,
178               address _sender) internal {
179
180     Endorsee storage endorsee = endorsees[_receiver];
181     endorsee.receiver = _receiver;
182     endorsee.index++;
183     endorsee.nER++;
184   //  endorsee.receivedPoints = computeReceivedPoints(_receiver);
185     endorsee.receivedFrom.push(_sender);
186     endorsee.hasReceivedFrom[_sender] = true;
187
188     endorseeAccts.push(_receiver) - 1;
189   }
190
191   //remove endorsement as an endorser of an endorsee
192   function removeEndorsement(address _endorsee) public returns(uint) {
193
194     require (joined[_endorsee]);
195
```

```
196     Endorser storage endorser = endorsers[msg.sender];
197     Endorsee storage endorsee = endorsees[_endorsee];
198
199     //proceed only if endorsee is in the endorser's list of endorsees
200     if (endorser.hasGivenTo[_endorsee]) {
201       endorser.hasGivenTo[_endorsee] = false;
202       endorser.nEG--;
203
204       //remove endorsee from endorser.givenTo array
205       endorsee.hasReceivedFrom[msg.sender] = false;
206       endorsee.nER--;
207
208       //remove endorser address from endorsee.receivedFrom array
209     }
210     return endorsers[msg.sender].index;
211   }
212
213   //computation of total received points of an endorsee
214   function computeReceivedPoints(address _endorsee) public view returns(uint) {
215
216     require(joined[_endorsee]);
217
218     //get list of endorsers addresses from whom _endorsee has received eds from
219     address [] memory receivedFrom = getReceivedFrom(_endorsee);
220
221     uint TRP = receivedPoints[_endorsee];
222     uint len = prevLength[_endorsee];
223
224     for (uint i=len; i< receivedFrom.length; i++){
225       TRP = TRP + endorsers[receivedFrom[i]].usedPower;
226     }
227     receivedPoints[_endorsee] = TRP;
228     prevLength[_endorsee] = receivedFrom.length;
229     return TRP;
230
231
232     //aggregate total received points from  the accumulated receivedFrom list
233 //    uint receivedPoints;
234 //
235 //    for (uint i=0; i<receivedFrom.length; i++) {
236 //      receivedPoints = receivedPoints + endorsers[receivedFrom[i]].usedPower;
237 //    }
238 //
239 //    return receivedPoints;
240   }
241
242   //computation of total endorsement impact of a participant
243   //the degree of connection should be strictly greater than 1 to be considered for
244   //impact computation, else, the impact by default should be ignorant, i.e., 0
245   function computeImpact(address _participant) public view returns (uint) {
246
247     require (joined[_participant]);
```

```
248
249     uint nEG = endorsers[_participant].nEG;
250     uint nER = endorsees[_participant].nER;
251
252     uint _RE = computeReceivedPoints(_participant);
253
254     uint impact;
255     uint totalImpact;
256
257     if (nEG <=1 && nER <=1 ) {
258       impact = 0;
259       return impact;
260       //return impact and exit here
261     } else {
262
263       uint minval = min(nEG,nER);
264       uint maxval = max(nEG,nER);
265
266       uint ratio = Division(minval, maxval,9);
267       uint usedUpByParticipant = endorsers[_participant].usedPower;
268       uint RE = _RE;
269
270       impact = ratio * RE;
271     }
272
273
274
275     // call feedback function here
276     totalImpact = transactionFeedBack(_participant, impact);
277     return totalImpact;
278   }
279
280   //Receive feedback from Transaction Network and penalize the nodes
281   function transactionFeedBack(address _participant,
282                 uint _impact )
283                 public returns (uint) {
284
285     if (flagCount[_participant] >= 1) {
286       //Decrease the current impact by 50 %
287       uint res = Division(_impact,2,9);
288       uint penalty = _impact - res ;
289
290     } else {
291       penalty = _impact;
292
293     }
294
295     return penalty;
296   }
297
298   //Single function to get all the details of a registered participant
299   function getProfile(address _participant) public view returns (
```

```
    uint,
    uint,
    address[],
    uint,
    uint,
    address[] )
    {

    uint outDegree = endorsers[_participant].nEG;
    uint usedPower = endorsers[_participant].usedPower;
    address[] outConns  = endorsers[_participant].givenTo;

    uint inDegree =  endorsees[_participant].nER;
    uint receivedPoints = computeReceivedPoints(_participant);
    address[] inConns   = endorsees[_participant].receivedFrom;

    return (
        outDegree,
        usedPower,
        outConns,

        inDegree,
        receivedPoints,
        inConns
    );
  }

  //get connections and degree of connections - helper function
  function getConnections(address _participant) public view returns (
    address [],
    address []
  ){

    require (joined[_participant]);

    address [] inConns = endorsees[_participant].receivedFrom;
    address [] outConns = endorsers[_participant].givenTo;

    return (inConns, outConns);
  }

  //count total number of registered participants
  function getCount( ) public view returns (uint) {

    return numberOfParticipants;

  }

  //return array of all endorser accounts
  function getEndorsers() view public returns (address []) {

      return endorserAccts;
```

```
352
353   }
354
355   //return the total consumable power used by an endorser
356   function getUsedPower(address _endorser) view public returns(uint) {
357
358       return (endorsers[_endorser].usedPower);
359
360   }
361
362   //return list of addresses that an endorser has sent endorsement to
363   function getGivenTo(address _endorser) view public returns(address []) {
364       return (endorsers[_endorser].givenTo);
365   }
366
367   //return number of endorsees an endorser has sent endorsement to
368   function getGivenToCount(address _endorser ) view public returns (uint) {
369     return (endorsers[_endorser].givenTo).length;
370   }
371
372   //return a boolean value from the matrix of hasGivenTo, quick access for checking
373   //if an endorsee's address is in the list of endorsee addresses of the particular
        endorser.
374   function gethasGivenTo(address _endorser,
375               address _endorsee) view public returns(bool) {
376       return (endorsers[_endorser].hasGivenTo[_endorsee]);
377   }
378
379   //return an array of all endorsee accounts - front end
380   function getEndorsees() view public returns (address []) {
381       return endorseeAccts;
382   }
383
384   //return address of all the endorsers for an endorsee, helper function to
385   //compute total received point
386   function getReceivedFrom(address _endorsee) view public returns(address []) {
387       return (endorsees[_endorsee].receivedFrom);
388   }
389
390   //count total number of endorser for an address of endorsee
391   function getReceivedFromCount(address _endorsee) view public returns (uint ) {
392     return (endorsees[_endorsee].receivedFrom).length;
393   }
394
395   //return a boolean value from the matrix of hasReceivedFrom, to check if
396   // an endorser's address is in the list of endorser address of the particular
        endorsee
397   function gethasReceivedFrom(address _endorser,
398                address _endorsee) view public returns(bool) {
399       return (endorsees[_endorsee].hasReceivedFrom[_endorser]);
400   }
401
```

```
402
403   //some helper  functions for floating point calculation
404   function Division( uint _numerator,
405             uint _denominator,
406             uint _precision) internal pure returns (uint _quotient) {
407
408     uint numerator = _numerator * 10 ** (_precision + 1);
409     uint quotient = ((numerator / _denominator) + 5  ) / 10;
410
411     return (quotient);
412   }
413
414   //some helper maths function to compute max, min value.
415   //used for computing ratio and ensuring that the ratio is always less than 1.
416   function max (uint x, uint y ) internal pure returns (uint) {
417     if (x < y) {
418       return y;
419     } else  {
420       return x;
421     }
422   }
423
424   function min (uint x, uint y ) internal pure returns (uint) {
425     if (x < y) {
426       return x;
427     } else {
428       return y;
429     }
430   }
431 }
```

Listing A.1: Endorsement Contract

```
1  pragma solidity ^0.4.18;
2
3  contract Ownable {
4    address public owner;
5    address newOwner;
6
7    event ownershipTransfer (
8      address indexed oldOwner,
9      address indexed newOwner
10   );
11
12   //constructor function to set the owner of contract
13   function Ownable( ) public {
14     owner = msg.sender;
15   }
16
17   modifier onlyOwner(){
18     require(msg.sender == owner);
19     _;
20   }
```

```
21
22    function transferOwnership(address _newOwner) public onlyOwner{
23      require(_newOwner != 0x0);
24      newOwner = _newOwner;
25      owner = newOwner;
26    }
27  }
```

Listing A.2: Ownable

```
1  pragma solidity ^0.4.18;
2
3  import "./Ownable.sol";
4
5  contract Killable is Ownable {
6
7    function kill() onlyOwner {
8      selfdestruct(owner);
9    }
10 }
```

Listing A.3: Killable Contract

B [Appendix: Ethereum Application]

```
1  import Web3 from 'web3';
2
3  //Assuming that metamask has already injected a web3 instance onto the page.
4  //window is a global variable "only" available in the browser.
5
6  let web3;
7
8  if (typeof window !== 'undefined' && typeof window.web3 !== 'undefined') {
9    //In the browser, metamask has already injected web3
10   web3 = new Web3(window.web3.currentProvider);
11 } else {
12   //on server OR user is not running metamask
13   const provider = new Web3.providers.HttpProvider(
14     'http://localhost:7545'
15   );
16   web3 = new Web3(provider );
17 }
18
19
20 export default web3;
```

Listing B.1: web3 connector

```
import web3 from './web3';
import Endorsement from './build/Endorsement.json';

//many different addresses as user visits different addresses
export default (address) => {
  return new web3.eth.Contract(
    JSON.parse(Endorsement.interface ),address);
};
```

Listing B.2: Participants addresses

```
const path = require('path');
const solc = require('solc');
const fs = require('fs-extra');

const buildPath = path.resolve(__dirname, 'build');
//1.Delete entire build folder
fs.removeSync(buildPath);


const edsPath = path.resolve(__dirname, 'contracts', 'Endorsement.sol');
//2. Read Endorsement.sol from contracts folder
const source = fs.readFileSync(edsPath,'utf8');

//3. Use solidity compiler to compile the contract
const output = solc.compile(source, 1 ).contracts;

//check if dir exists, if not create it
fs.ensureDirSync(buildPath);

for (let contract in output ) {
  fs.outputJsonSync(
    path.resolve(buildPath,contract.replace(':','') + '.json'),
    output[contract ]
  );
}
```

Listing B.3: Compile script

```
const HDWalletProvider = require('truffle-hdwallet-provider');
const Web3 = require('web3');
const compiledEds = require('./build/Endorsement.json');

const provider = new HDWalletProvider(
  'http://localhost:7545'
);
const web3 = new Web3(provider);

const deploy = async (accountNumber = 0) => {
  const accounts = await web3.eth.getAccounts();
  const deployAccount = accounts[accountNumber];
  const data = compiledEds.bytecode;
  const gas = 4000000;
```

```
15   const gasPrice = web3.utils.toWei('2','gwei');
16
17   console.log('Attempting to deploy from account', deployAccount);
18
19   const result = await new web3.eth.Contract(JSON.parse(compiledEds.interface) )
20   .deploy({
21     data
22   })
23   .send({
24     gas,
25     gasPrice,
26     from: deployAccount
27   });
28
29   console.log('Contract deployed to', result.options.address);
30 };
31 deploy();
```

Listing B.4: Script to deploy locally or to Rinkeby

C [Appendix: Application]

```
1  import React, { Component } from 'react';
2  import { Card, Button } from 'semantic-ui-react';
3  import eds from '../ethereum/eds';
4  import Layout from '../components/Layout';
5  import { Link } from '../routes';
6
7  class ParticipantIndex extends Component {
8    static async getInitialProps() {
9
10     const participants = await eds.methods.getAllParticipants().call();
11     return {participants: participants};
12
13   }
14
15
16   renderParticipants(){
17     const items = this.props.participants.map(address => {
18       return {
19         header: address,
20         description: (
21           <Link route={`/participants/${address}/` }>
22             <a>View Details </a>
23           </Link>
24         ),
```

```
        fluid: true
      };
    });

    return <Card.Group items={items } />;

  }

  render( ) {
    //return <div> {this.props.participants[0]} </div>
    return (
      <Layout>
      <div>
        <h3>Get All Participants </h3>

        <Link route="/participants/new">
        <a>
          <Button
            floated="right"
            content = "Join Network"
            icon  = "add circle"
            primary = {true}
          />
        </a>
        </Link>

        {this.renderParticipants()}
      </div>
      </Layout>
    );
  }
}

export default ParticipantIndex;
```

Listing C.1: Application Index

```
import React, { Component } from 'react';
import { Form, Button, Input, Message } from 'semantic-ui-react';
import Layout from '../../components/Layout';
import eds from '../../ethereum/eds';
import web3 from '../../ethereum/web3';
import { Router } from '../../routes';

class ParticipantNew extends Component {
  state = {
    pseudonym: '',
    errorMessage: '',
    loading: false
  };

  onSubmit = async (event ) => {
    event.preventDefault( );
```

```
17
18      this.setState({ loading: true, errorMessage:'' });
19
20      try {
21
22      const accounts = await web3.eth.getAccounts();
23
24      await eds.methods
25        .joinNetwork(this.state.pseudonym)
26        .send({
27          from: accounts[0]
28        });
29
30        Router.pushRoute('/');
31
32
33      } catch (err) {
34        this.setState({ errorMessage: err.message });
35      }
36      this.setState({ loading: false });
37    };
38
39
40    render( ) {
41      return (
42        <Layout>
43          <h3> New Participant </h3>
44
45          <Form onSubmit = {this.onSubmit} error={!!this.state.errorMessage} >
46            <Form.Field>
47              <label>Pseudonym</label>
48              <Input
49                label="User Name"
50                labelPosition="right"
51                value={this.state.pseudonym}
52                onChange={event => this.setState({ pseudonym: event.target.value})}
53              />
54            </Form.Field>
55
56            <Message error header="Oops!" content={this.state.errorMessage } />
57            <Button loading={this.state.loading } primary>
58              Join!!
59            </Button>
60          </Form>
61
62        </Layout>
63      );
64    }
65
66  }
67  export default ParticipantNew;
```

Listing C.2: New participants

```
import React, {Component } from 'react';
import { Card, Button, Form, Input, Message, Group, Grid, Table } from 'semantic-ui
    -react';
import Layout from '../../components/Layout';
import Endorsement from '../../ethereum/participants';
import ConnectionRow from '../../components/ConnectionRow';
import OutRow from '../../components/OutRow';
import eds from '../../ethereum/eds';
import web3 from '../../ethereum/web3';
//import Endorse from '../../components/Endorse';

class ParticipantShow extends Component {
  static async getInitialProps(props) {

    const accounts = await web3.eth.getAccounts();
    //props.query.address=address of the actual participant that we show
    //console.log(props.query.address);
    const address = props.query.address;
    const user = Endorsement(props.query.address);

    const summary = await eds.methods.getProfile(props.query.address).call();
    const impact = await eds.methods.computeImpact(props.query.address).call();
    const index = await eds.methods.getParticipantIndex(props.query.address).call()
        ;
    const name = await eds.methods.participants(index).call();

    const givenTo = await eds.methods.getGivenTo(props.query.address).call();
    const receivedFrom = await eds.methods.getReceivedFrom(props.query.address).
        call();

    const givenToCount = givenTo.length;
    const receivedFromCount = receivedFrom.length;

    const outConns = await Promise.all(
      Array(givenToCount)
        .fill()
        .map((element,index) => {
          return givenTo[index];
        })
    );

    //console.log(outConns );

    const inConns = await Promise.all(
      Array(receivedFromCount)
        .fill()
        .map((element,index) =>{
          return receivedFrom[index];
        })
    );

    //console.log(inConns);
```

```
50
51      return {
52        index: index,
53        name: name[1],
54        address: props.query.address,
55        impact: impact/1000000000000000000000000000,
56
57        outDegree: summary[0],
58        usedPower: summary[1]/1000000000,
59      //  outConns: summary[2],
60
61        inDegree: summary[3],
62        receivedPoints: summary[4]/1000000000,
63      //  inConns: summary[5]
64        inConns,
65        outConns
66      };
67
68
69      }
70
71    onHandleClick = async() => {
72      const accounts = await web3.eth.getAccounts();
73      await eds.methods.endorse(this.props.index).send({
74        from: accounts[0]
75      });
76
77      //console.log(this.props.address);
78
79    }
80
81    onRemove = async () => {
82
83      const accounts = await web3.eth.getAccounts();
84
85      await eds.methods.removeEndorsement(this.props.address).send({
86        from: accounts[0]
87      });
88    }
89
90    renderOutRows() {
91      return this.props.outConns.map((outConns,index)=>{
92        return (
93          <OutRow
94            key={index}
95            outConns={outConns}
96            id={index}
97            address ={this.props.address}
98          />
99        );
100     });
101   }
```

```
102
103   renderRows(){
104     return this.props.inConns.map((inConns,index)=>{
105       return (
106         <ConnectionRow
107           key = {index}
108           inConns = {inConns}
109           id = {index}
110           address ={this.props.address }
111         />
112       );
113
114     });
115   }
116
117
118   renderCards( ) {
119     const {
120       outDegree,
121       usedPower,
122       outConns,
123       inDegree,
124       receivedPoints,
125       inConns,
126       impact,
127       name
128     } = this.props;
129
130     const items = [
131       {
132         header:this.props.address,
133         meta:'Public key used when joining the network',
134         description: 'Public key of the participant',
135         style: {overflowWrap: 'break-word'}
136
137       },
138       {
139         header: name,
140         meta: 'User Name',
141         description: 'Pseudonym used when joining the network',
142         style: {overflowWrap: 'break-word'}
143
144       },
145       {
146         header: outDegree,
147         meta: 'nEG',
148         description:'Degree of Outgoing connections',
149         style: {overflowWrap: 'break-word'}
150       },
151       {
152         header: usedPower,
153         meta: 'consumed Points',
```

```
154         description:'Amount of points already consumed',
155         style: {overflowWrap: 'break-word'}
156       },
157       {
158         header: inDegree,
159         meta: 'nER',
160         description:'Degree of Incoming Connections',
161         style: {overflowWrap:'break-word'}
162       },
163       {
164         header: receivedPoints,
165         meta: 'Received Endorsement Points',
166         description: 'Sum of all the endorsement points received',
167         style: {overflowWrap:'break-word'}
168       },
169       {
170         header: impact,
171         meta: 'Endorsement Impact',
172         description: 'Total Endorsement Impact made by the participant',
173         style: {overflowWrap:'break-word'}
174       }
175 //      {
176 //        header: outConns,
177 //        meta: 'Outgoing Connections',
178 //        description: 'Array of addresses to whom the participant has endorsed',
179 //        style: {overflowWrap: 'break-word'}
180 //      },
181 //      {
182 //        header: inConns,
183 //        meta:'Incoming Connections',
184 //        description:'Array of addresses from whom the participant has received
    endorsement',
185 //        style: {overflowWrap:'break-word'}
186 //      }
187     ];
188     return <Card.Group items={items} />;
189   }
190
191   render( ) {
192     const {Header, Row, HeaderCell, Body } = Table;
193
194     return (
195       <Layout>
196         <h3> Participant Details </h3>
197         <Grid>
198           <Grid.Column width={15}>
199             {this.renderCards()}
200           </Grid.Column>
201         </Grid>
202         <Grid>
203           <Grid.Column width={10}>
204           <Button color="green" basic onClick={this.onHandleClick}>
```

```
            Endorse this Participant!
          </Button>
          <Button color="teal" basic onClick={this.onRemove} >
            Remove Endorsement
          </Button>
          </Grid.Column>
        </Grid>
      <Table>
        <Header>
          <Row>
            <HeaderCell>ID</HeaderCell>
            <HeaderCell>IncomingConnections</HeaderCell>
          </Row>
        </Header>
        <Body>
          {this.renderRows()}
        </Body>
      </Table>
      <Table>
      <Header>
        <Row>
          <HeaderCell>ID</HeaderCell>
          <HeaderCell>Outgoing Connections</HeaderCell>
        </Row>
      </Header>
      <Body>
        {this.renderOutRows()}
      </Body>
      </Table>
      </Layout>
      );
  }
}

export default ParticipantShow;
```

Listing C.3: Show all details of Participants

Application Components:

```
import React from 'react';
import { Menu } from 'semantic-ui-react';
import { Link } from '../routes';

export default ( ) => {
  return (
    <Menu style={{ marginTop: '15px' }}>

      <Link route="/" >
        <a className="item" >
          Endorsement
        </a>

```

```
      </Link>

      <Menu.Menu position="right">

      <Link route="/" >
        <a className="item" >
          Participants
        </a>

      </Link>

      <Link route="/participants/new" >
        <a className="item" >
          +
        </a>
      </Link>

      </Menu.Menu>
    </Menu>

  );

};
```

Listing C.4: Header

```
import React from 'react';
import { Container } from 'semantic-ui-react';
import Head from 'next/head';
import Header from './Header';

export default (props ) => {
  return (
    <Container>
      <Head>
      <link
        rel="stylesheet"
        href="//cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.12/semantic.min.css"
            >
      </link>
      </Head>

      <Header />
      {props.children}
    </Container>

  )

}
```

Listing C.5: Layout

```
import React, {Component} from 'react';
```

```
import { Table } from 'semantic-ui-react';

class OutRow extends Component {
  render() {
    const { Row, Cell } = Table;

    return (
      <Row>
        <Cell>{this.props.id} </Cell>
        <Cell>{this.props.outConns}</Cell>
      </Row>
    );
  }
}

export default OutRow;
```

Listing C.6: Out Rows

```
import React, {Component} from 'react';
import { Table } from 'semantic-ui-react';

class ConnectionRow extends Component {
  render() {
    const { Row, Cell } = Table;

    return (
      <Row>
        <Cell>{this.props.id} </Cell>
        <Cell>{this.props.inConns}</Cell>
      </Row>
    );
  }
}

export default ConnectionRow;
```

Listing C.7: Connection Rows

```
const { createServer } = require('http');
const next = require( 'next' );

const app = next ( {
  dev: process.env.NODE_ENV != 'production'

});

const routes = require( './routes' );
const handler = routes.getRequestHandler( app );

app.prepare().then(() => {
  createServer(handler).listen(3000, (err ) => {
    if (err) throw err;
```

```
15      console.log('Ready on localhost:3000');
16    });
17  });
```

Listing C.8: Server

```
1  const routes = require('next-routes')();
2
3  routes
4    .add('/participants/new','/participants/new')
5    .add('/participants/:address','/participants/show' );
6
7  module.exports = routes;
```

Listing C.9: Routes