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

MASTER PROGRAMME IN COMPUTER SCIENCE

UPPSALA
UNIVERSITET

Uppsala University
Department of Information Technology

Master's Thesis
SUJATA TAMANG

September 7, 2018

BLOCKCHAIN BASED DECENTRALIZED REPUTATION MODEL AND GENERAL TRUST FRAMEWORK

MASTER PROGRAMME IN COMPUTER SCIENCE

Uppsala University
Department of Information Technology

Approved by

Supervisor, Jonatan Bergquist

Reviewer, Björn Victor

September 7, 2018

Abstract

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

1 Introduction

From using emails to communicate with one another, searching for an online source to get the news or media files to shopping for everyday things, we rely heavily on the internet today. 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 actually measuring the trustworthiness of Bob with 100% certainty. One way to do that is to 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 will turn out to be true. However, this doesn't provide any guarantee of Bob's behavior. Bob could be a malicious actor in a society 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 of certain actions. Depending on the action and damage caused, the level of trustworthiness also gets affected. As such, we can say that anyone is honest until they deal damage to be known as a malicious actor in the society. 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 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 plays 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 continuously by continually updating the state based on feedback. The risk of failure or probability of success when transacting with

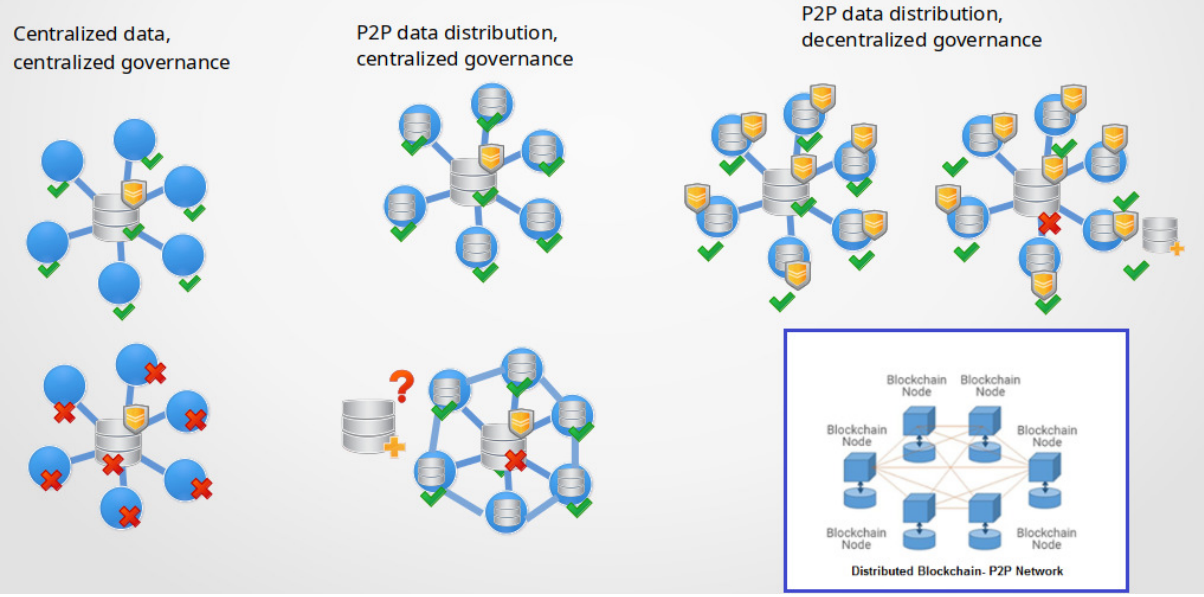

Figure 1.1: Centralized Vs. decentralized network

an entity is reliant 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 objective measures can either increase or lower the reputation of the user and limit their interaction ability.

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.

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

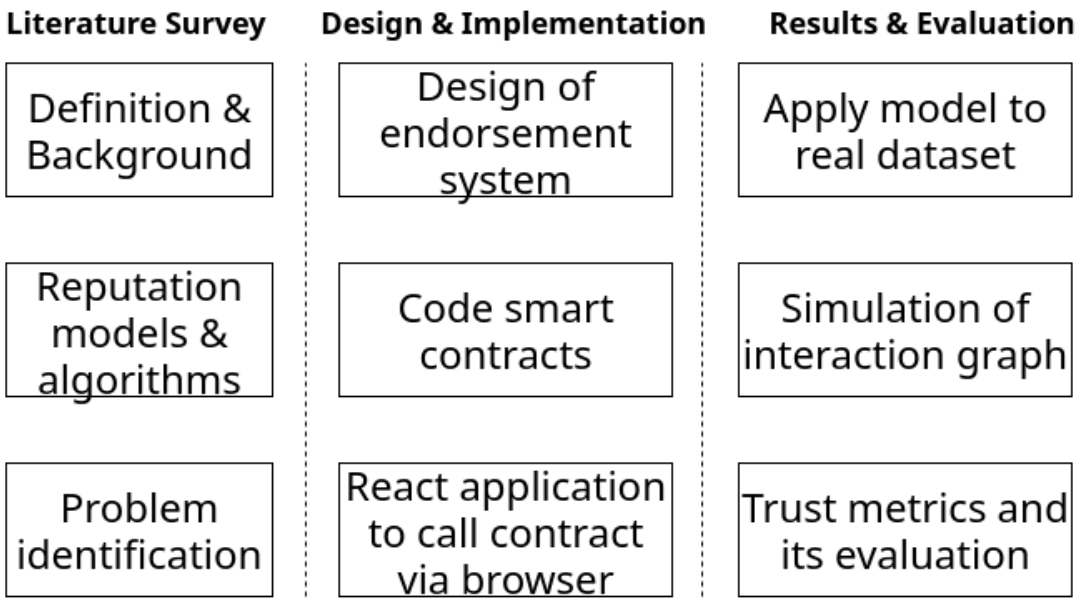

Figure 1.2: Workflow of the project's phases

Thus, if one node fails then, clients can still request data from other nodes in the network. This architecture solves the problem by P2P data distribution. However, the governance of data is still centralized such that clients still need to wait for a particular node that has the privilege to add new data. This leaves the possibility for an online service running on these networks to be shut down or data to be modified by the central authority. Examples include the shutdown of P2P services such as KaZaA [4] and Napster [5]. Therefore, this system is still vulnerable to internal modifications as it is reliant on a trusted entity (special node to add data). Blockchain [6] solves these issues by providing P2P data distribution along with decentralized data governance. These models can be seen in figure 1.1 in section 1.

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 performed. 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 each other's information 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 model.

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, unknown entity to Alice, the success or failure of the transaction is dependent on two factors that may or may not be transparent.

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

Platform's reputation: Reputation of the platform can also be inferred similarly based on the history of services it has been able to provide, a general perception in the community, etc. Here, the platform in use acts as the trusted third party that Alice must trust to present correctly computed, untampered data about Bob. The entity claiming to be Bob could be Eve, who found a way to bypass the platform's security and inflate his reputation. Eve could delete the ad and associated account when the payment is complete, or she could gather Alice's details 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 [7]. Reports by Experia [7] and Javelin [8] indicate that E-commerce fraud has risen to 30% in 2017 from 2016 while identity fraud victims have risen by 8% in 2017 (16.7 million U.S. victims). A recent report of data breach on Finnish Enterprise Agency for Helsinki [9] [4] exposed 130,000 users' login information. While there are several security reasons that have led to attack at such scale. One major reason is the client-server architecture where everything is stored on a centralized server and data flows in and out from the same source. On the other hand, distributing information over a decentralized network would require a simultaneous attack to achieve the same effect, thereby increasing the difficulty level of attack. Similarly, Reputation models can help in

[4] https://liiketoimintasuunnitelma.com/

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 trustworthiness of online entities with 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 a system of interaction. The interaction system that allows entities to endorse each other. To design a method to aggregate this interaction information such that global trust scores can be assigned to every participating entities.

The research questions that this project aims to address are:

- How can graph theories and relevant reputation algorithms be used to model the interaction between entities and detect/identify honest and malicious nodes in the network? How can the interaction graph be modeled?
- 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?
- 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.?

1.3 Scope

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

Research Question 1 To answer research question 1, literature survey is performed on various reputation algorithms and existing models. This survey follows with the discussion on various analysis metrics and threat models that eventually leads to graph simulation of endorsement network.

Research Question 2 Interpretation of nodes connections and quantification of scores for individual nodes that represents trustworthiness based on score range is presented. Comparative analysis of on chain vs. off-chain storage requirements is studied and analyzed. Overall system design and architecture is presented.

Research Question 3 The endorsement network is analyzed against various network metrics to show resilience to threat models. Discussion on other use cases and how the endorsement model can be used on top of other systems is also presented.

1.4 Structure of Report

This paper is structured as follows. Chapter 2 performs a literature survey on the existing algorithms and their implementations. Chapter 3 provides a background overview of relevant concepts necessary to understand the following sections. In chapter 4, system requirements and the approach taken for the model design is shown. It shows the overall system design and architecture. Chapter 5 follows on with discussion of evaluation metrics and test methods and present results representative of the designed model. Finally, conclusion and future work is presented in chapter 7.

2 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. As such, trust and reputation system to detect and limit those kinds of behavior has been a subject of interest ever since. There is a significant amount of research literature in trust frameworks and reputation management systems. This section aims to offer an overview of the research literature and the implementations of trust and reputation as it exists.

2.1 Reputation systems and algorithms

Resnick et al. [10] points out that 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 dishonest behavior. TrustDavis [11] is a reputation system that addresses these concerns. 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's transaction in exchange for a reward. The system relies on the insurer's capability of estimating failure probability. Schaub, Alexander, et al. [12] proposes a block-chain based reputation model that recommends the use of blind signature to disconnect the link between customer and ratings. Doing so lets a customer freely rate/review the transaction without fear of retaliation. 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. A buyer can use same or different identity for different transactions. The system recommends the latter. As such, a merchant cannot trace back the identity of the buyer from whom he might have received the negative feedback. Therefore, an unfair rating attack is possible in this system. A buyer who intends to damage the merchant's (service provider) reputation can do so by giving dishonest negative feedback.

The most known and widely used reputation algorithm in a P2P network is EigenTrust [13] 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 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, 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 this algorithm is that peers are more likely to center around a static set of pre-trusted peers that joined the network and thus has limited reliability if they leave the network. Pre-trusted peer is a notion of trust, where few peers that join the network are assumed trustworthy by design. Advogato's trust metric [14] also uses this notion. HonestPeer [15] proposes to enhance EigenTrust algorithm further by selecting reputable nodes dynamically and not just relying on pre-trusted nodes. It claims to have a better success rate in quality file serving and lower malicious participation compared to EigenTrust.

As mentioned by Alkharji, Sarah, et al. [16], a reputation system can serve one of the following purpose, a peer-based reputation system or file-based reputation system. The peer-based system allows peers in the network to be rated and assigned a 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++ [16] is a trust management system for P2P networks that combines both, i.e., it shares the notion of both trusted peers and trusted files. As such, 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 [17] is a distributed reputation mechanism designed for P2P file-sharing systems. It creates a graph based on data transfers between peers and uses max flow algorithm to compute the reputation values for each node. Tribler [1] is a BitTorrent based torrent client that uses Bartercast to rank its peers. PowerTrust [18] proposes a robust and scalable reputation system that makes use of Bayesian learning. It uses Bayesian method to generate local trust scores and a distributed ranking mechanism to aggregate reputation scores.

2.2 Trust Model

Ries, S., Kangasharju, J. and Mühlhäuser, M., 2006, [19] have analyzed 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, an 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 [20] mentions in their research 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 [21] that addresses direct trust and recommender trust. Direct trust refers to an agent's belief

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

in another agent's trustworthiness whereas recommender trust refers to an agent's belief in another agent's ability of recommendations. It uses discrete labeled trust values as $\{vt, t, u, vu\}$ for direct trust where vt, t, u, vu represents very trustworthy, trustworthy, untrustworthy, and very untrustworthy respectively. Similarly, it uses $\{vg, g, b, vb\}$ for recommender trust where vg, g, b, vb represents 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 [22] 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. The degree of confidence in the trustworthiness of public-key uses a discrete labeled value as {undefined, marginal, complete}. Undefined refers to uncertainty, marginal refers to maybe the key is valid whereas complete refers to full confidence in that the key is valid. Similarly, the trustworthiness of the introducer is given by {full, marginal, untrustworthy}. In this case, full refers to the user's full confidence in the key being able to introduce another public key, marginal refers to uncertainty if the public key is fully competent, and untrustworthy represents that the key cannot be trusted to introduce another public key. Jøsang, Audun [23], proposes the trust model that uses subjective logic to bind keys and their owners in public key infrastructure.

2.3 Blockchain applications

Section 3.3 provides the technical details and discussion on blockchain technology. This section aims to discuss the applications and use cases of the technology and explain its relevance to the project.

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 is 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 [24] is believed 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 [25]. 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 [26], [27] 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 third party. Proof-of-work is discussed in more details by section 3.3.1. Examples of other platforms that extends bitcoin are Namecoin [28], which offers decentralized name resolution system, counterparty [2], which provides a platform for creating P2P financial applications. Hyperledger Fabric [29] is an open source permissioned distributed ledger technology platform that allows the applications to have blockchain components as a plug-and-play. Everledger [30] is a blockchain platform that provides a digital, global ledger to track and protect valuable assets. Ethereum [31] 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 [32] is a smart contract language employed by various blockchain platforms such as Ethereum [31], Tendermint [33], 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/

3 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 [34] or as a way to protect personal data [35]. 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.

3.1 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 [36]. Using the classification from McKnight et al., 1996 [37], 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. [38] "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 $B's$ trustworthiness, *reputation* is the perception of an individuals character or standing. Like trust, reputation is context-dependent. e.g., Alice may be trusted to answer linux questions efficiently but not windows related questions [39]. 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., transaction history, ratings)

as an input to yield a resulting score that can aid in detecting reliability/trustworthiness of an entity [40], [41].

Previous survey [42] has classified trust and reputation measures as either subjective or objective measure. This classification is further divided into specific or general. A subjective measure is based on an individual's perspective and has no formal metrics. Specific, subjective measures imply a subjective perception of an entity for a specific ability, such as package delivery time of a seller or average response time. 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 between positive and negative ratings used by eBay to give an average rating.

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 particular 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.

The table below shows the classification of trust and reputation measures as discussed above based on [42]:

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

Table 3.1: Classification of trust and reputation measures.

Individuals in online systems are identified by their online identities which can be anything and not necessarily linked to their real-world identities. 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. By definition, a security mechanism [43] 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 system 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, Lars and Jansson, Sverker [44] first used this term, to describe the idea of identifying malicious users and preventing harm to other users in the context of secure open electronic commerce. Reputation system needs to continuously receive feedback about the user's behavior and maintain an updated record of users. It provides a way to calculate the probability of success or risk of failure of a transaction between interacting parties.

3.2 Cryptography

Cryptography offers algorithms to achieve confidentiality, integrity, authenticity, and non-repudiation. Confidentiality and integrity ensure that the information being communicated is not disclosed or has been modified to or by any unauthorized parties. The data is hidden/encrypted such that only the authorized parties can make sense out of it. i.e., decrypt using the previously agreed upon key. Authenticity relates to confirming the truth of an attribute claimed by an entity. Non-repudiation is associated with the property that any entity who has previously sent the message cannot deny their authorship [45].

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.
- E, set of encryption rules such that $e_k : P \Rightarrow C$.
- D, set of decryption rules 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, also known as public-key cryptography makes use of key pairs, public and private keys. The public key can be publicly distributed, and an entity *'A'* wishing to send a confidential message to other entity *'B'* can encrypt the message using *B's* public key k_p to form ciphertext c. Upon receiving c, B can decrypt it using the private key k_s that is only known to B and corresponds to the respective public key that was used to form c. The computation of k_s given k_p is computationally infeasible in a secure system. Besides public-key encryption, public-key cryptography also has use in digital signature as discussed in section 3.2.2. RSA [46] is one example of a public-private cryptosystem.

3.2.1 Hash functions

Cryptographic hash functions are one-way functions, also known as mathematical trapdoor function that transforms 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 inefficient. An important characteristics of hash function 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.

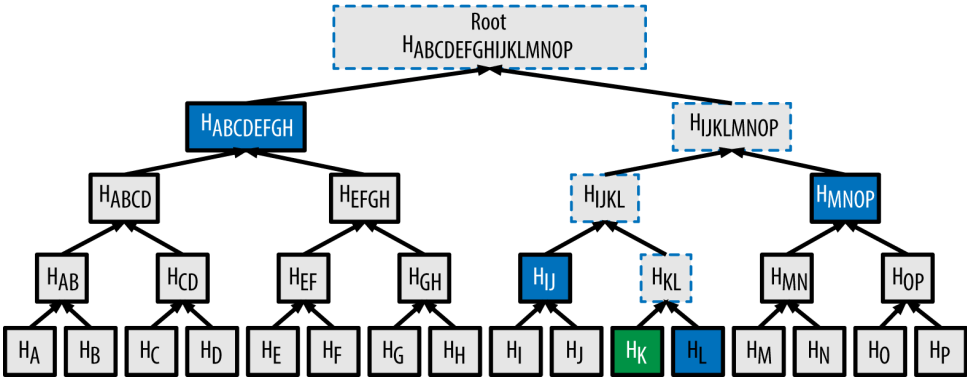

Figure 3.1: Merkle root and data inclusion verification

Other significant properties of hash function that contributes to reliability in digital security are [47]:

One-way: Given a key k, and an output w, it should be hard 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 hard for an attacker to find y such that $H_k(x) = H_k(y)$.

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

Earlier hash functions include MD5 which produced an output of size 128 bits. Collision-resistant attack in MD5 is possible within seconds with a complexity of 2^{39} [48] MD5 operations. NIST published 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 it's predecessor algorithms. Collision attack for SHA-1 was shown to be practically pssible [49] by creating two colliding pdf files that produced same SHA-1 digest. It took equivalent processing power of 6,500 years of single CPU computation time and 110 years of GPU computation time. Applications of hashing algorithms include digital signature, ISO checksums, fingerprinting data etc.

These cryptographic hash functions are relevant in understanding blockchain data structure. Merkle Tree [50], [51] is a hash tree that is used by blockchain to create a root hash of the list of transactions. Transactions are explained in detail by section 3.3. For understanding the process of root hash creation, one can assume the list of transactions to be a list of items. Assuming a binary tree of height n with 2^n leafs, the transactions are leaf nodes at height 0. All the leaf nodes are paired and hashed together. These hashes are again hashed together repeatedly until the number of hash is only one. The final hash value is the hash of the root node at height n. Figure 3.1 illustrates the process of Merkle root creation. This root hash value is included as a block header and is significant for two things in blockchain:

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 easy to verify the inclusion of a transaction in a block without requiring to include all the transactions. As shown in figure 3.1, 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. The second preimage-resistance attribute ensures that the proof is not fake.

3.2.2 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 [46]:

- 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. Transaction data and necessary fields are explained in detail by section 3.3. For this example, data can be Bob's address and value that Alice wants to send to

Bob. 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 case of ethereum [1], Alice would compute the ECDSA signature. This signature (components) 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 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. Data serialization and transaction validation are discussed in detail by section 3.3.1.

3.3 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 [52]. 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 [53]. As pointed out by Antonopoulos, Andreas M., and Wood, Gavin [54], 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.

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

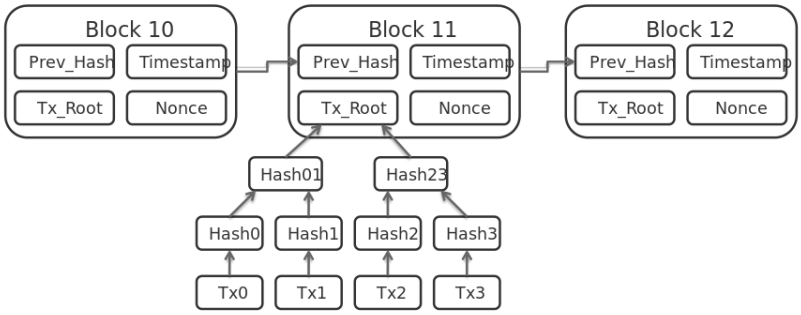

Figure 3.2: BlockchainStructure

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 3.3.1. 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 3.2. 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. A malicious node attempting to change the transaction data in Block 11 will need to find a hash value equivalent to the associated transaction root with a modified transaction data. The second pre-image resistant property of a hash function ensures that finding such a value is computationally infeasible.

3.3.1 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 [55]:

Leader Based System: In this case, there is a pre-selected leader that is responsible for collecting all the transactions and appending new records to the blockchain. Having a small group or consortium, 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 include Hyperledger Fabric [56], R3 Corda [57]. It is important to note that, in Hyperledger Fabric, the entire transaction flow (proposal, endorsement, ordering, validation, and commitment of transactions) is considered to be part of the consensus algorithm [58], [59], unlike in other systems where consensus relates to the ordering of transactions. However, it makes use of a leader election mechanism to elect a leader that is responsible for ordering transactions.

Proof-Of-Work (PoW): This is the most widely used consensus mechanism in a public permissionless setup. As the name suggests, a validator node (miners) 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 the miner to find a specific nonce value that is less than a target value. This target value is also known as difficulty target because it is responsible for setting the difficulty level of the block. Lower the difficulty target, higher is the difficulty of the puzzle. The difficulty level is adjusted based on the average block time between two blocks. As can be seen in Figure 3.3, the hash function applied to the concatenation of transaction data and the nonce should be less than the specified difficulty target. An important 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 the nonce value that can generate an output hash less than the target value is by brute forcing multiple possible values. Thus, they have to compute many hash operations before finding the valid hash value that satisfies the function requirement. The first validator node to solve the problem gets to add the proposed 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

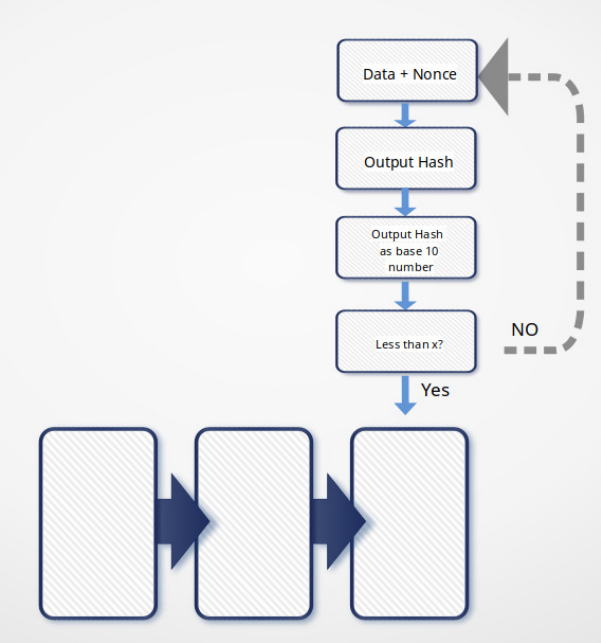

Figure 3.3: Cryptographic Puzzle

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 simply 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 [24], Ethereum [60].

Economy based systems: Consensus mechanisms such as Proof-Of-Stake or delegated proof-of-stake 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. 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 [61], 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 Casperr [62]. Casper tries to solve this problem by penalizing the node (destroy the stake) that is detected to have validated two blocks at the same height.

3.3.2 Categories

Along the dimension of validation and access control [63], 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 doesn't 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.

3.3.3 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 [64] 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 [65]:

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 [66].

Ethereum being the first platform to offer programmable blockchains, introduced a virtual machine, EVM (Ethereum Virtual Machine), where the contract code can be executed that results in a deterministic output provided the same transaction context and blockchain state [54]. 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 [32], LLL [67]. Contract's 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. Doing so doesn't 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. 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 [68].

3.3.4 Ethereum

3.4 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 [69], [70] 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 an identity's reputation score 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 nonnegative 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 negatives 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 of 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 [71]. 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. [72] discusses the use of an autonomous reputation scheme by prioritizing resource allocation to peers with higher reputation.

3.4.1 Graph properties

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

Formally, a graph [73], 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.

Network metrics that are helpful in analyzing the complexity of interactions at different levels [74] and used for evaluation of results in this project are:

Degree Connectivity: The number of connections a node has is the degree of its connec- tivity. The number of inflow is referred to as indegree whereas the number of outflows is the outdegree

of a node. Usually, a higher degree of connectivity implies a higher likelihood for information (relevant data to the network) to pass through that node.

Network Centrality: Centrality refers to the significance of a node in the network. i.e., how important the node is in the overall network. The degree of connectivity is one way to measure centrality of a node. Similarly, there are other centrality measures which includes: Closeness centrality, Betweenness centrality, Prestige centrality.

Closeness Centrality: refers to how close a node is to other nodes in the network.

Betweenness Centrality: refers to the number of nodes to which the given node acts as a connector. i.e., how many nodes passes through this node.

Prestige Centrality: refers to the significance of the node based on the significance of the adjacent nodes(nodes one is connected to). To observe and analyze the behavior at the macro-level, one needs to look at the overall structure of the graph. i.e., Network topology that shows how constituent parts are interconnected to form the graph as a whole. They can form ring, star, tree, or mesh structure or be a fully connected graph where each node are connected to each other.

4 Methodology and Implementation

The problem of measuring the trustworthiness of communicating entities is an essential aspect of any online system. This chapter 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 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 requirements.

4.2 User stories & 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 endorsement) and endorsee (who receives 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.

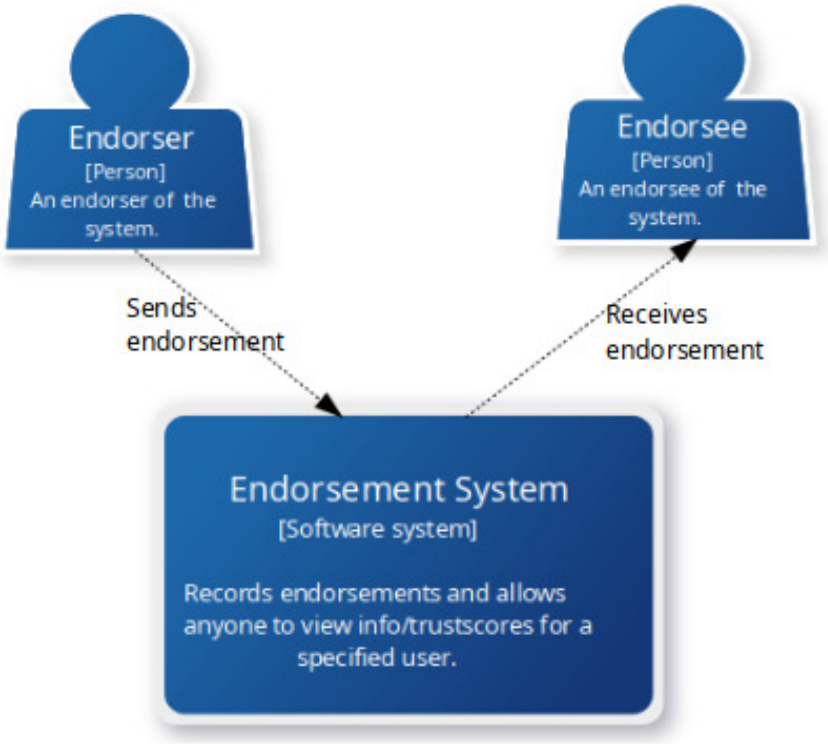

Figure 4.1: Context Layer

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 doesn't engage in the 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 meaningful value to infer one's 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 in Section 4.2 shows the endorsement interaction between users (endorser and endorsee) and the endorsement system. In software development paradigm, user stories [75] provide an informal description of the feature that the system can have from an end users perspective. Sketching user stories for the endorsement system can help to define the roles and the associated specific features for a given user type. Table 4.2 presents the user stories for different user types of endorsement system and follows a

As an	I need to be able to..	Traceability
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

role-feature-reason template [76]. The traceability column is used to trace back to that specific feature when checking the fulfillment of requirements in Section 5.2 in Chapter 5.

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 and the rationale for user interaction.

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 to their 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's architecture and can be listed as:

1. Security: Solidity has a list of known bugs [77] and recommendations on security considerations [78] for writing smart contract code. The smart contract code for endorsement system should take into account minimal security considerations to avoid the relevant possible bugs.

2. Reliability: The data stored on the blockchain should be immutable and publicly verifiable.

3. Trust metrics should correctly describe the actual trust score of the nodes.
   The definition of honest or malicious interaction 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 better score than the malicious ones.

4.3 The Model - Endorsement Network

Complete trust and reputation system should consist of four separate steps as shown by Figure 4.2. 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, selecting and interacting with a peer for transaction based on which they get punished or rewarded is based on a transactional system. As implementing the transaction based system is the not the main task of this project, feedback based on success or failure of a transaction is based on the assumption about a transaction network that can be used by endorsement system. Thus, the system implementation is based on the smart contract logic for endorsement model. Updating a value based on transactional outcome is purely based on an assumption about the transaction network.

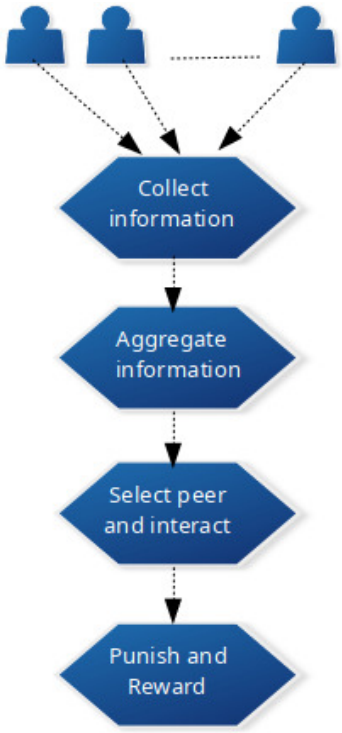

Figure 4.2: Trust and Reputation Model steps based on [70].

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 **??** in Section **??** shows different components of the endorsement system and how the users interact with them. The endorsement 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 endorsement system. Abdul-Rahman A, Hailes S. [21] 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 exists 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 [13], 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 [79] 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.

ratio$_A$: represents the ratio of nEG$_A$ to nEG$_A$. This value can be used to ensure that the sent and received endorsement are not far off from each other. ratio$_A$ is always assumed to be 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 point 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 $E = \{e_1, e_2, e_3, ..., e_n\}$ is a 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 Total Endorsement Impact (TEI) made by A is given by:

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

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 [80] 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 concern 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 point 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 4.3, 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

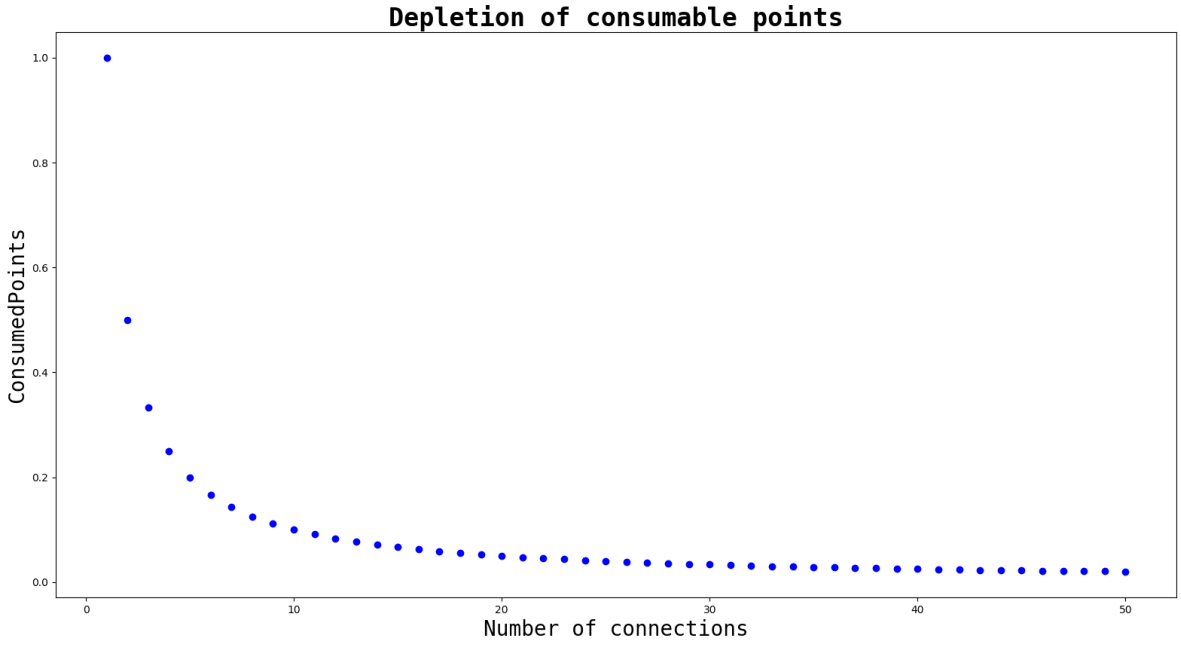

Figure 4.3: Convergent behaviour of consumable points as 'n' increases.

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.

Transaction Cost: 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 user 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 4.4, if Alice is an honest node, then she only needs to pay for the operation of one 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) 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 endorsement), 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

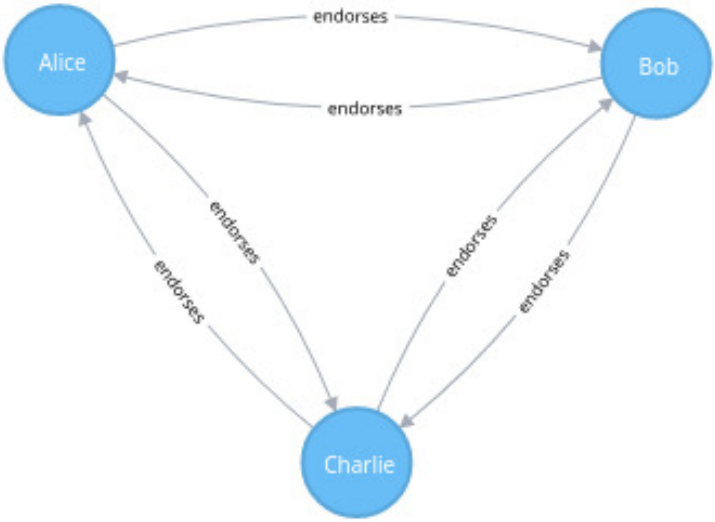

Figure 4.4: Interaction between participants

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 account. The larger the size of this list, 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 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 connection for contributing 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 [81] [1] says that there is a limit of 150-200 stable social relationships that humans can maintain. A study by Dunbar, R. I. (2016) [82] 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. The final score can be computed by the client.

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

[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.

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.

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.

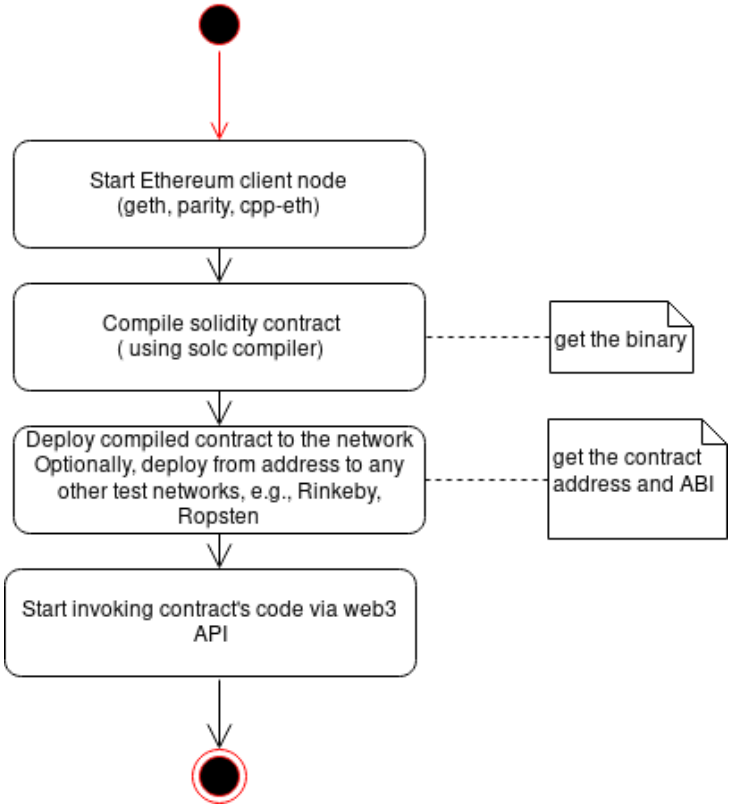

Figure 4.5: Deploy solidity contract on the blockchain network

4.4.1 Smart contracts

The process of starting up the node and deploying the contract to the blockchain network is given by Figure 4.5. 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 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 [83] 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. address.

Killable: inherits from Ownable and allows the owner of the contract 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

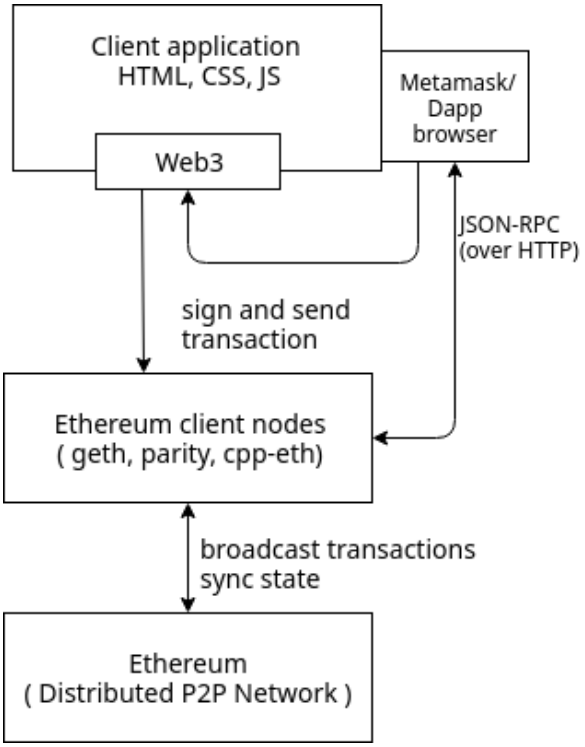

Figure 4.6: Client Application

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 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 endorsed method is invoked by a user account, the state variables nEG on the account that initiated the transaction gets incremented. Similarly, the nER for the account specified by the transaction data. 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 4.6 in Section 4.4.2. Reactjs [2] was used for front-end development of endorsement application. The

[2]https://reactjs.org/

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 [84], 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 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 [87]. 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 Proof-Of-Work (PoW). As mentioned earlier, PoW is computationally expensive and wasteful. Using other consensus mechanisms such as delegated proof of stake 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

[3] https://www.rinkeby.io/explorer

wish to. The participants do not necessarily know each other. Therefore, no "one" node can be trusted to collect everyone's transaction and make a final commit. Gochain [4] has mentioned the use of PoR (Proof-Of-Reputation) 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 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 [88] 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 [89]. 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 PoW.

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

5 Results & Evaluation

This chapter presents the evaluation criteria for the endorsement model proposed in chapter 4. Based on which, the overall system design and functionality will be evaluated and final results is presented.

5.1 Evaluation Criteria

To assess the fulfillment of requirements, a descriptive approach based on the design evaluation method by Hevner A, Chatterjee S. [90] is used when necessary. The system is not only reliant on the smart contract code and its execution but also on the interaction theory discussed. For this, the endorsement interaction is simulated using the graph simulation tool ne04j [1]. The dataset used to simulate the interaction is taken from SNAP [91]. The details on the dataset are presented in Section 5.3. 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 [2] 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.

5.2 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 [90]. The table 5.2 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: The security considerations by Solidity [77] was followed for the contract code. Not all the recommendations presented there were relevant for the endorsement contract. For instance, recommendations on restricting the amount of ether or patterns for sending/receiving ether in a function call is not relevant as endorsement contract does not require any function call to include any amount of ether as the message value. As part of fail-early principle, the contract function code was ordered as conditions, actions, and interactions

[1] https://neo4j.com/
[2] 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

where relevant. Doing so can avoid Re-Entrancy bug. The function call can be made by both externally owned account or a contract address in ethereum. 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 bugs.

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, permissionless blockchain network which allows any node to commit a block of transactions to 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. But, he does not broadcast the blocks he solved to the network and instead keeps solving the puzzle in isolation with the network. The transactions that he 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 a race with the rest

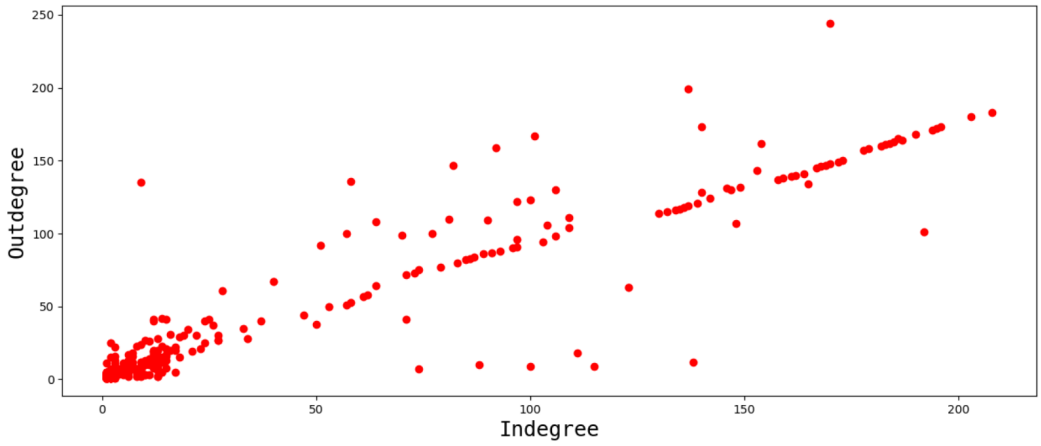

Figure 5.1: Given Vs. Received

of the network, this form of attack is only possible if he owns more than 51% of the hashing power compared to the rest of the network. Therefore, this attack is called 51% attack. 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 5.3 presents the details regarding this requirement.

5.3 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 [3] 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[93]. 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 [94]. Unlike the Bitcoin Alpha network that let users rate on a scale of -10 to +10 to demonstrate the strength of their trust towards other users. The endorsement is more of a boolean decision problem, 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 a bit more relevant to endorsement interaction, the existing dataset was filtered only to have edges

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

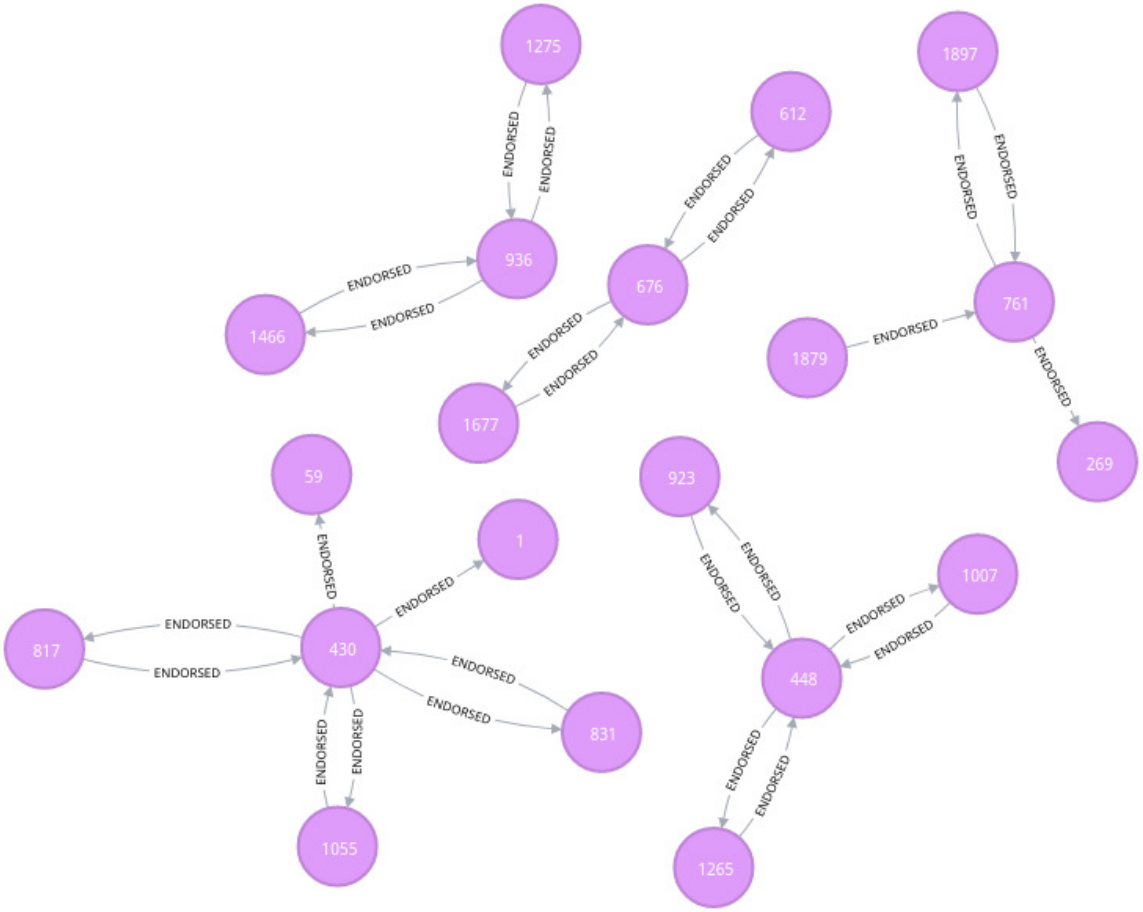

Figure 5.2: Interaction subgraph of nodes with impact zero

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 and TRP for each node. 70% (1175 nodes) turned out to have a TEI score of zero. On examining the nodes, they were found to have only one incoming or outgoing connections. As such, the TEI score 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 the 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.

Total Received Points: Among the remaining 502 nodes, there were 5 participants 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,

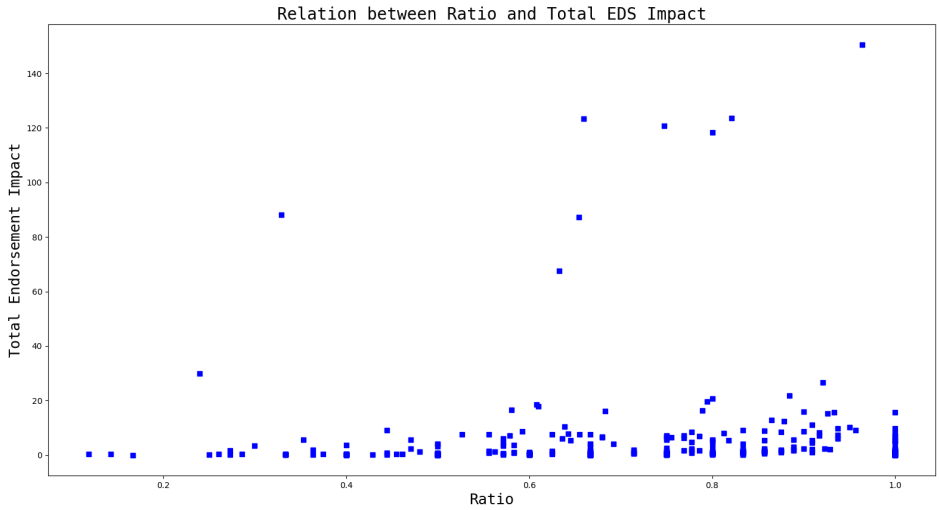

Figure 5.3: Relation of Ratio and Total endorsement impact

their TRP was zero because the endorsers were not impactful in the network. The interaction subgraph for these five nodes is shown in Figure 5.2. This factor corresponds to the prestige centrality metrics of a graph network where the significance of its 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. The CP of the endorsers for these nodes can be seen in Figure 5.2. The table **??** shows the value for each relevant variables required to compute TEI for the nodes.

Ratio: The information presented by table **??** shows nodes that have the maximum possible value for ratio which is 1 and still have the lowest TEI score. It shows that maintaining the ratio between outgoing and incoming connections is not enough to have a significant impact on the network. The Figure 5.3 shows the relation between ratio and total endorsement over all nodes. Based on this relation, we can say that higher ratio does not mean a higher impact but a higher impact should have a higher (maintained balance) ratio. Thus, the ratio is a contributing factor to maintain a significant score in the long run.

There Figure 5.4 shows the number of nodes and the range of TEI values distributed across the nodes in the network. There are very few nodes with a higher impact value. The ranking of nodes based on the impact value can be made. Higher the value of TEI that a node has, higher is its trustworthiness. There are very few nodes which have managed to make a significant impact on the network. The Figure 5.5 shows the interaction graph structure of impactful node for the given nodes.

5.4 Total impact across several factors with different scenarios

This section considers different scenarios to analyze how several factors such as nEG, nER, ratio, TRP is distributed and what contributes to having a higher or a lower TEI value. First

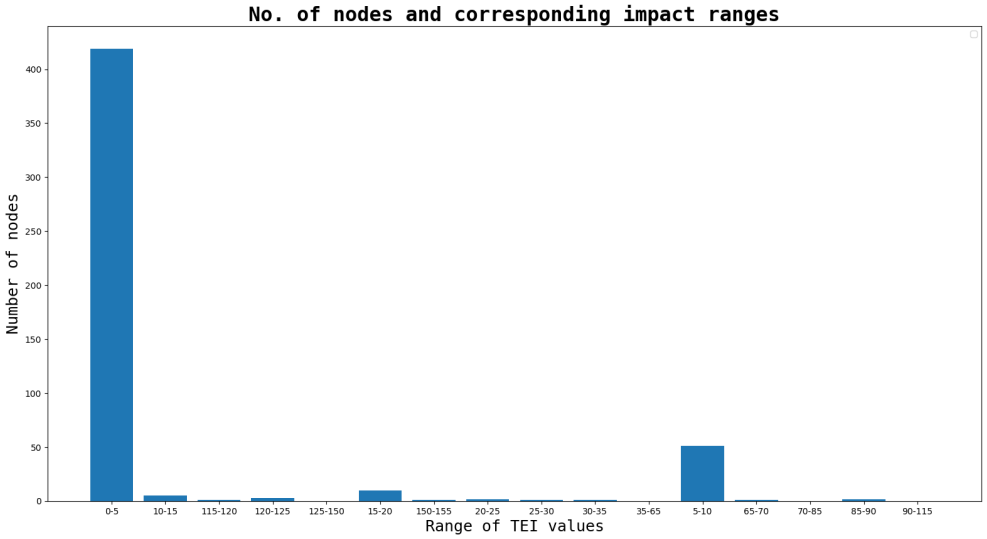

Figure 5.4: Total endorsement impact vs. number of nodes

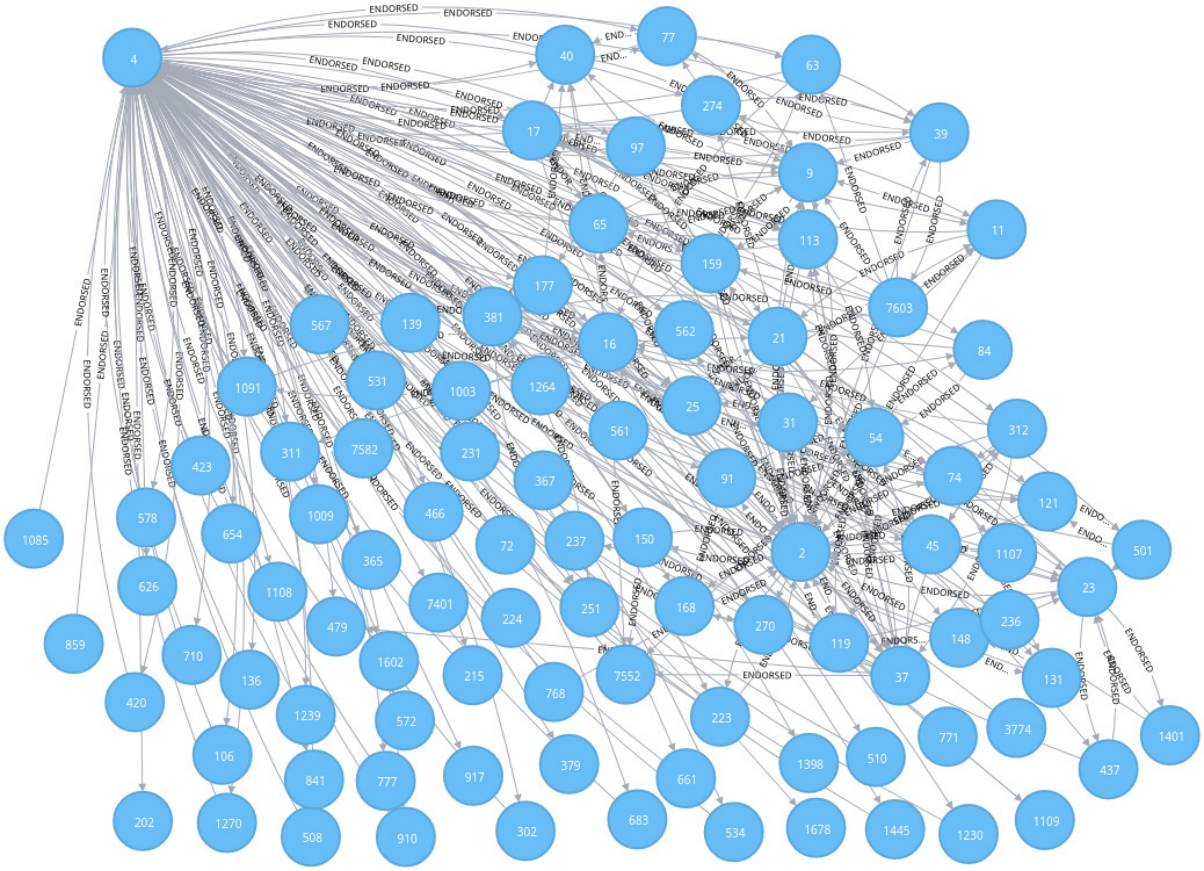

Figure 5.5: High Impact Node

two cases look at the nodes with maximum impact value. As mentioned earlier, a node that does not have a maintained ratio cannot have a higher impact value. As such, it is interesting to see the minimum ratio that a maximum impact node has. Case1 and Case 2 shows this behavior. Similarly, the third and fourth case also makes the same analysis but with nodes having minimum impact value. The Figure 5.6 shows the distribution across several factors for all four cases mentioned.

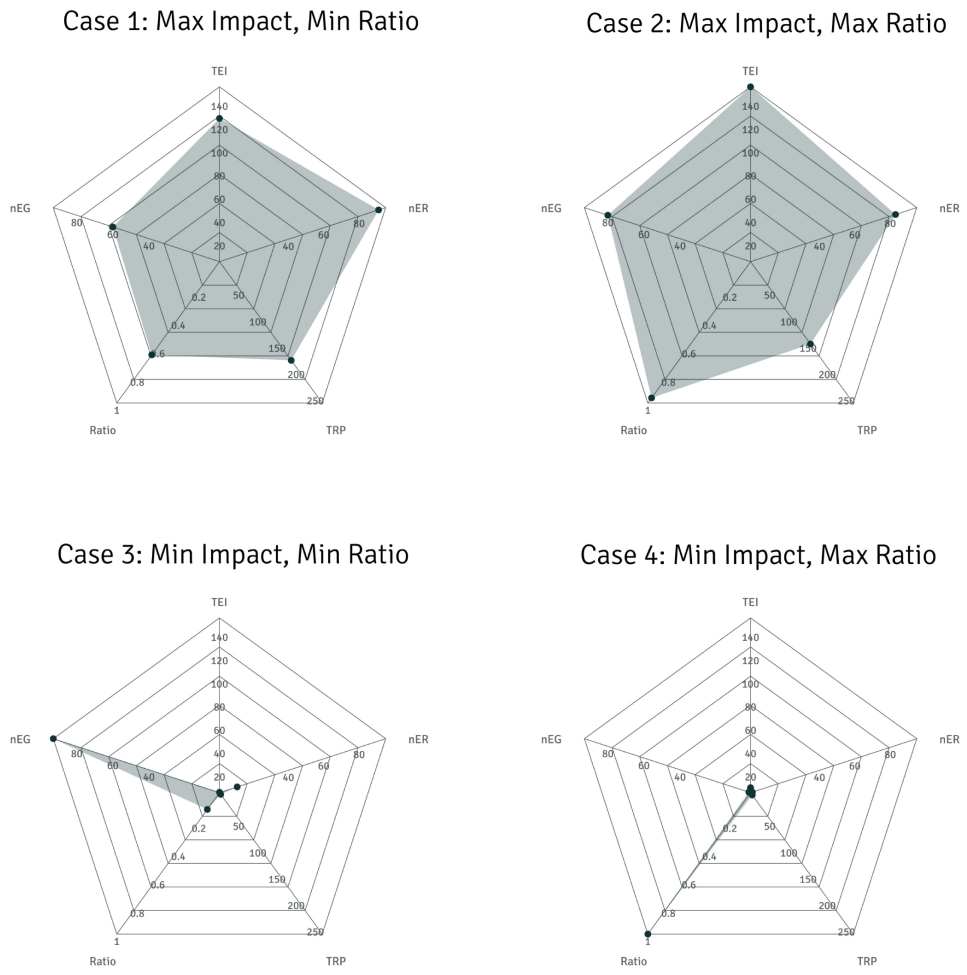

Figure 5.6: Total Impact Across all factors

Case 1 & Case 2: The lowest ratio of the node with maximum impact value is 0.6 and has a significant number of outgoing and incoming connections which looks balanced as expected for a higher impact node. Similarly, the maximum ratio that a node with the maximum impact has is 1, which is the highest possible value. As such, the nodes with higher TEI score represents the honest nodes with expected interactions behavior.

Case 3 & Case4: We can see from the Figure 5.6 the node has the lowest impact value because of an extreme one-way connection in case 3. It has too many outgoing connections and compar-

atively very few incoming connections making it evident why the node has such a low impact value.

5.5 Threat Model

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 5.3. 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 can aid 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 can be addressed. The punishment of a node requires communication with the transaction network to receive the feedback on a transactional outcome.

Freeriders: 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 identities. The Sybil attack has been discussed earlier.

Malicious collective

5.6 Summary of Results

6 Discussion & Analysis

This chapter discusses the fulfillment of the project goal. The answers to the research questions posed at the beginning of this project are answered theoretically as well as implementation wise as necessary.

Research question 1: How can graph theories and relevant reputation algorithms be used to model the interaction between entities and detect/identify honest and malicious nodes in the network? How can the interaction graph be modeled?

This is answered by section 5.3, 5.5.

Research question 2: 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?

This is answered by section 4.3, 4.2, **?? ??**.

Research question 3: How can the discussed endorsement network ensure trustworthiness while also preserving users 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.?

This is answered by section 4.4.4, 6.2.

6.1 Contract calls with web browser

Additionally, frontend was deployed using Reactjs and web3 API to communicate with smart contracts on blockchain directly from the browser.

The features implemented were:

- View list of all participants: Anyone can see the list of all participants registered on endorsement network. This feature relates to requirement R3. Figure 6.1 demonstrates this feature.

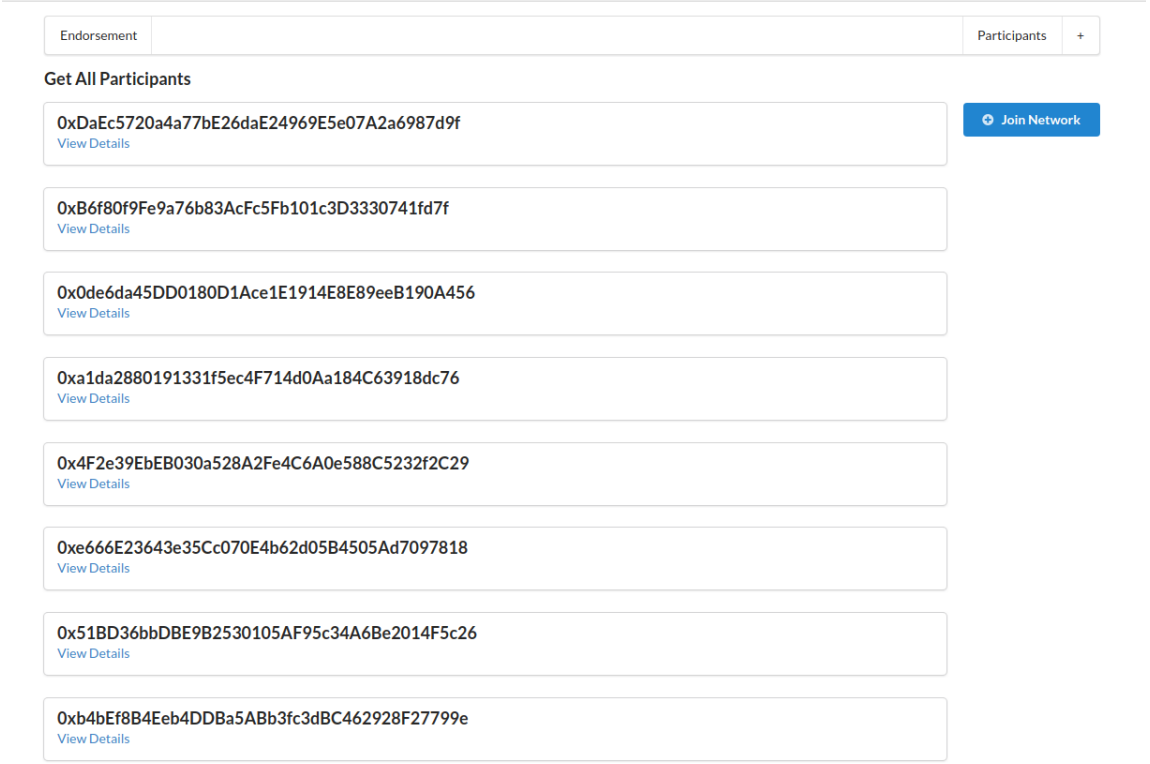

Figure 6.1: List of all participants

- Join Network with a Pseudonym: Anyone can join the endorsement network and start calling the send/remove endorsement functions from the endorsement contract. This feature relates to requirement R4.2. Figure 6.2 demonstrates this feature.

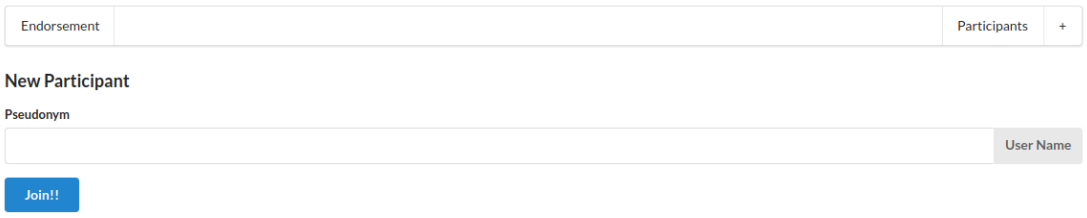

Figure 6.2: Join Network using Pseudonym

- Send and Remove Endorsement: Anyone can view the details of all registered participants on the network which is a requirement mentioned by R3, R4. However, sending and removing endorsement checks if the message sender is a registered participant or not.

Only if it's true, then the sending or removing of endorsement is called without error. This feature refers to requirement R1 and R2. Figure 6.3 shows the view details of participants along with the send and remove endorsement feature.

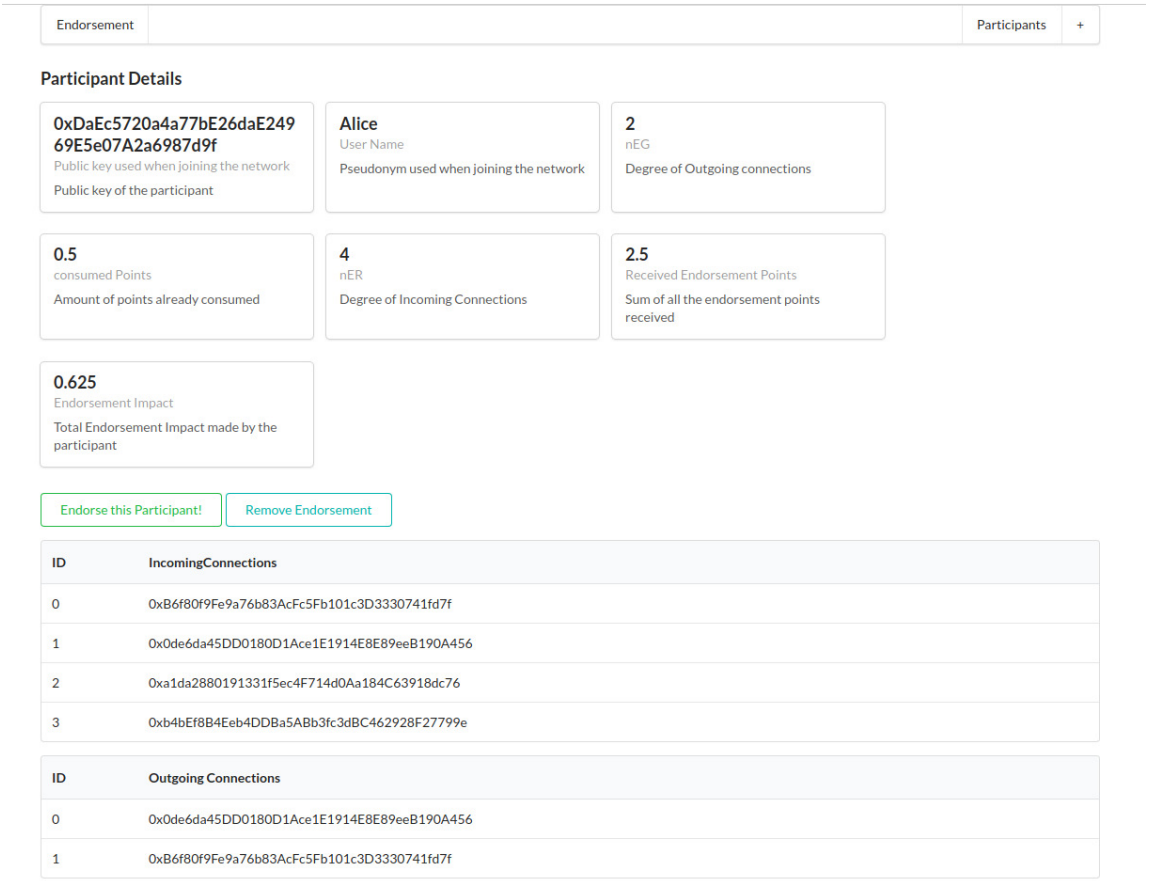

Figure 6.3: View Details of participants and endorse them

6.2 Generalization

The endorsement PoC is a general model that aggregates and assign reputation scores to individuals based on their interaction in a network. As such, any transactional network, e-commerce, content-serving platform, etc. can use it. The platform-specific reputation system can still co-exist alongside the endorsement model. It is useful to get an objective measure of the actual transaction feedback(history, quality of services, etc.). Consider a buy/sell system scenario where two unknown entites, A and B are attempting to transact with each other. The entities can check each other's rating/feedback from the platform's reputation system in use. If that is not enough for deciding on the transaction, they can check the endorsement network for the global reputation score of the entity in question. As such, the decision is no more reliant only on A's reputation or the platform's reputation. There is a third option that guarantees a reliable and immutable data stored in a decentralized manner. From the view of the platform, say, B/S, it allows users on endorsement network to transact on B/S. The users of B/S gets output from decentralized trust score storage system. The platform itself provides input to the

endorsement system in case of a failed transaction outcome to help penalize the peer in the endorsement network. Doing so increases the accuracy in both systems.

7 Conclusion

Trust system running on an interaction network can aid in a successful interaction by pre-evaluation(i.e., evaluate the outcome of the transaction beforehand) before actually engaging in a transaction. Collection and aggregation of information to infer the trustworthiness of online entities are vital for any online system. Having a reliable and trustworthy reputation system that models users interaction helps both the reputation of the platform in use and the honest users. The PoC designed for this project followed the definition of a trust and reputation system closely during the implementation phase. Specifically, encouraging honest behavior while making it difficult to maintain a good trust score with malicious behavior. Various threat models that concern the reputation model was addressed by game-theoretic assumptions as well as codified restrictions. The endorsement model described in this project can be used by any interaction model with certain modifications to meet the requirement of the specific transaction network.

While the designed PoC provided a way to measure the trustworthiness of an entity, there is no way to measure the actual trust value. The accuracy is based on the constant feedback from the peer's interacting with an actual transactional network. Trustworthiness is not a boolean factor that can just be solved by a 1 or 0 for an honest or malicious peer. The task of assigning a reputation score and inferring trustworthiness is a vague problem. The PoC, therefore, provided a value based on several transparent factors that a user can see for themselves and verify. Based on the evaluation of results, trust metrics have been able to successfully represent the actual trust score of the entity in question. From the evaluation criteria mentioned, this project has fulfilled the goal and answered the research questions as per plan.

7.1 Future Works

Given the time constraints, the project was not able to cover all the aspects and contexts of a reputation system. Further improvements can be made by including the logic to directly retrieve the information from a transaction network that can act as a feedback loop for the endorsement system. Use of complex graph algorithms can be further implemented and evaluated to analyze the transaction behavior on several levels. For instance, using a Ford-Fulkerson algorithm to compute the maximum flow on a network can be used. Doing so can set a bound on the number of edges that an honest cluster can make with the malicious one. Thus, detecting a malignant node to further improve on the punishment aspects can help in the measure of more precise

scores. Since the trust and reputation are dynamic by nature, it needs to receive feedback and update the state continually. However, use of a sophisticated algorithm also implies a higher computation overhead. Besides the implementation, the project could also use more evaluation from blockchain aspects. The PoC was deployed on Ethereum network, and transactions were mostly tested out using ganache, which creates an in-memory nodes for generating transaction receipts and blocks. Therefore, the speed and size of transactions, cost of computations, etc. were taken for granted from Ethereum network. Future works could perform a more general evaluation for the transactions and speed of the network.

The primary limitation of blockchain at the current stage is its scalability problem. Being able to handle request from a large number of users without confirmation delays for simple transactions can help to create more adaptable use cases in Blockchain space. It is a continually developing domain, and research on consensus mechanisms, latency, scalability aspects of the technology is being carried out by various sectors.

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] A. Mlcakova and E. A. Whitley, "Configuring peer-to-peer software: An empirical study of how users react to the regulatory features of software", *European Journal of Information Systems*, vol. 13, no. 2, pp. 95–102, 2004.

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

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

[7] 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.

[8] 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.

[9] H. Times, *130,000 login credentials compromised in large-scale hacking*, Accessed Aug 17, 2018, [Online]. Available: http://www.helsinkitimes.fi/finland/finland-news/domestic/15448-130-000-login-credentials-compromised-in-large-scale-hacking.html/, 2018.

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

[11] 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.

[12] 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.

[13] 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.

[14] R. Levien, *Advogato's trust metric*, 2003.

[15] H. A. Kurdi, "Honestpeer: An enhanced eigentrust algorithm for reputation management in p2p systems", *Journal of King Saud University-Computer and Information Sciences*, vol. 27, no. 3, pp. 315–322, 2015.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

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

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

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

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

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

[32] (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.

[33] (2018). 'What is tendermint?' Accessed Aug 31, 2018, [Online]. Available: https://github.com/tendermint/tendermint/blob/master/docs/introduction/introduction.md.

[34] 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.

[35] 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.

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

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

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

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

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

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

[42] 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.

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

[44] 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.

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

[46] 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.

[47] I. Mironov *et al.*, "Hash functions: Theory, attacks, and applications", *Microsoft Research, Silicon Valley Campus. Noviembre de*, 2005.

[48] 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.

[49] 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.

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

[51] B. at Berkeley. (). 'Workshop 5: Data structures, consensus algorithms, bitcoin scripting'. Accessed Aug 22, 2018, [Online]. Available: https://drive.google.com/file/d/0ByBe1QJVC-EJVVBZeEFJNGFTbTA/view/.

[52] 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.

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

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

[55] (2018). 'Hashgraph: Consensus in blockchain'. Accessed Aug 22, 2018, [Online]. Available: https://www.foundery.co.za/blog/hashgraph-consensus-in-blockchain/.

[56] 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.

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

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

[59] ——, (2017). 'Hyperledger fabric model'. Accessed Aug 31, 2018, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/fabric_model.html.

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

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

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

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

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

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

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

[67] (2017). 'Lll introduction'. Accessed Aug 31, 2018, [Online]. Available: https://lll-docs.readthedocs.io/en/latest/lll_introduction.html.

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

[69] 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.

[70] 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.

[71] 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.

[72] 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.

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

[74] D. Gkorou, "Exploiting graph properties for decentralized reputation systems", 2014.

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

[76] 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/.

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

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

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

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

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

[82] 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.

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

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

[85] M. Alharby and A. van Moorsel, "Blockchain-based smart contracts: A systematic mapping study", *arXiv preprint arXiv:1710.06372*, 2017.

[86] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab", in *International Conference on Financial Cryptography and Data Security*, Springer, 2016, pp. 79–94.

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

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

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

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

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

[92] J. Bergquist, *Blockchain technology and smart contracts: Privacy-preserving tools*, Presentation held externally at Technical University Munich on the 29-5-2017 10.50., 2017.

[93] 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.

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

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

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

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

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

```
300
301     return (
302         outDegree,
303         usedPower,
304         outConns,
305
306         inDegree,
307         receivedPoints,
308         inConns
309     );
310   }
311
312   //get connections and degree of connections - helper function
313   function getConnections(address _participant) public view returns (
314     address [],
315     address []
316   ){
317
318     require (joined[_participant]);
319
320     address [] inConns = endorsees[_participant].receivedFrom;
321     address [] outConns = endorsers[_participant].givenTo;
322
323     return (inConns, outConns);
324   }
325
326   //count total number of registered participants
327   function getCount( ) public view returns (uint) {
328
329     return numberOfParticipants;
330
331   }
332
333   //return array of all endorser accounts
334   function getEndorsers() view public returns (address []) {
335
336       return endorserAccts;
337
338   }
339
340   //return the total consumable power used by an endorser
341   function getUsedPower(address _endorser) view public returns(uint) {
342
343       return (endorsers[_endorser].usedPower);
344
345   }
346
347   //return list of addresses that an endorser has sent endorsement to
348   function getGivenTo(address _endorser) view public returns(address []) {
349       return (endorsers[_endorser].givenTo);
350   }
351
```

```
352   //return number of endorsees an endorser has sent endorsement to
353   function getGivenToCount(address _endorser ) view public returns (uint) {
354     return (endorsers[_endorser].givenTo).length;
355   }
356
357   //return a boolean value from the matrix of hasGivenTo, quick access for checking
358   //if an endorsee's address is in the list of endorsee addresses of the particular
          endorser.
359   function gethasGivenTo(address _endorser,
360               address _endorsee) view public returns(bool) {
361       return (endorsers[_endorser].hasGivenTo[_endorsee]);
362   }
363
364   //return an array of all endorsee accounts - front end
365   function getEndorsees() view public returns (address []) {
366       return endorseeAccts;
367   }
368
369   //return address of all the endorsers for an endorsee, helper function to
370   //compute total received point
371   function getReceivedFrom(address _endorsee) view public returns(address []) {
372       return (endorsees[_endorsee].receivedFrom);
373   }
374
375   //count total number of endorser for an address of endorsee
376   function getReceivedFromCount(address _endorsee) view public returns (uint ) {
377     return (endorsees[_endorsee].receivedFrom).length;
378   }
379
380   //return a boolean value from the matrix of hasReceivedFrom, to check if
381   // an endorser's address is in the list of endorser address of the particular
          endorsee
382   function gethasReceivedFrom(address _endorser,
383                address _endorsee) view public returns(bool) {
384       return (endorsees[_endorsee].hasReceivedFrom[_endorser]);
385   }
386
387
388   //some helper  functions for floating point calculation
389   function Division( uint _numerator,
390             uint _denominator,
391             uint _precision) internal pure returns (uint _quotient) {
392
393     uint numerator = _numerator * 10 ** (_precision + 1);
394     uint quotient = ((numerator / _denominator) + 5  ) / 10;
395
396     return (quotient);
397   }
398
399   //some helper maths function to compute max, min value.
400   //used for computing ratio and ensuring that the ratio is always less than 1.
401   function max (uint x, uint y ) internal pure returns (uint) {
```

```
402     if (x < y) {
403       return y;
404     } else  {
405       return x;
406     }
407   }
408
409   function min (uint x, uint y ) internal pure returns (uint) {
410     if (x < y) {
411       return x;
412     } else {
413       return y;
414     }
415   }
416 }
```

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

```
1  import web3 from './web3';
2  import Endorsement from './build/Endorsement.json';
3
4  //many different addresses as user visits different addresses
5  export default (address) => {
6    return new web3.eth.Contract(
7      JSON.parse(Endorsement.interface ),address);
8  };
```

Listing B.2: Participants addresses

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

```
4
5  const buildPath = path.resolve(__dirname, 'build');
6  //1.Delete entire build folder
7  fs.removeSync(buildPath);
8
9
10 const edsPath = path.resolve(__dirname, 'contracts', 'Endorsement.sol');
11 //2. Read Endorsement.sol from contracts folder
12 const source = fs.readFileSync(edsPath,'utf8');
13
14 //3. Use solidity compiler to compile the contract
15 const output = solc.compile(source, 1 ).contracts;
16
17 //check if dir exists, if not create it
18 fs.ensureDirSync(buildPath);
19
20 for (let contract in output ) {
21   fs.outputJsonSync(
22     path.resolve(buildPath,contract.replace(':','') + '.json'),
23     output[contract ]
24   );
25 }
```

Listing B.3: Compile script

```
1  const HDWalletProvider = require('truffle-hdwallet-provider');
2  const Web3 = require('web3');
3  const compiledEds = require('./build/Endorsement.json');
4
5  const provider = new HDWalletProvider(
6    'http://localhost:7545'
7  );
8  const web3 = new Web3(provider);
9
10 const deploy = async (accountNumber = 0) => {
11   const accounts = await web3.eth.getAccounts();
12   const deployAccount = accounts[accountNumber];
13   const data = compiledEds.bytecode;
14   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
```

```
  console.log('Contract deployed to', result.options.address);
};
deploy();
```

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

C [Appendix: Application]

```
import React, { Component } from 'react';
import { Card, Button } from 'semantic-ui-react';
import eds from '../ethereum/eds';
import Layout from '../components/Layout';
import { Link } from '../routes';

class ParticipantIndex extends Component {
  static async getInitialProps() {

    const participants = await eds.methods.getAllParticipants().call();
    return {participants: participants};

  }


  renderParticipants(){
    const items = this.props.participants.map(address => {
      return {
        header: address,
        description: (
          <Link route={`/participants/${address}/` }>
            <a>View Details </a>
          </Link>
        ),
        fluid: true
      };
    });

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

  }

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

```
39
40          <Link route="/participants/new">
41          <a>
42            <Button
43              floated="right"
44              content = "Join Network"
45              icon  = "add circle"
46              primary = {true}
47            />
48          </a>
49          </Link>
50
51          {this.renderParticipants()}
52        </div>
53        </Layout>
54      );
55    }
56  }
57
58  export default ParticipantIndex;
```

Listing C.1: Application Index

```
1  import React, { Component } from 'react';
2  import { Form, Button, Input, Message } from 'semantic-ui-react';
3  import Layout from '../../components/Layout';
4  import eds from '../../ethereum/eds';
5  import web3 from '../../ethereum/web3';
6  import { Router } from '../../routes';
7
8  class ParticipantNew extends Component {
9    state = {
10     pseudonym: '',
11     errorMessage: '',
12     loading: false
13   };
14
15   onSubmit = async (event ) => {
16     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

```
1  import React, {Component } from 'react';
2  import { Card, Button, Form, Input, Message, Group, Grid, Table } from 'semantic-ui
       -react';
3  import Layout from '../../components/Layout';
4  import Endorsement from '../../ethereum/participants';
5  import ConnectionRow from '../../components/ConnectionRow';
6  import OutRow from '../../components/OutRow';
7  import eds from '../../ethereum/eds';
8  import web3 from '../../ethereum/web3';
9  //import Endorse from '../../components/Endorse';
10
11 class ParticipantShow extends Component {
12   static async getInitialProps(props) {
```

```
13
14      const accounts = await web3.eth.getAccounts();
15      //props.query.address=address of the actual participant that we show
16      //console.log(props.query.address);
17      const address = props.query.address;
18      const user = Endorsement(props.query.address);
19
20      const summary = await eds.methods.getProfile(props.query.address).call();
21      const impact = await eds.methods.computeImpact(props.query.address).call();
22      const index = await eds.methods.getParticipantIndex(props.query.address).call()
            ;
23      const name = await eds.methods.participants(index).call();
24
25      const givenTo = await eds.methods.getGivenTo(props.query.address).call();
26      const receivedFrom = await eds.methods.getReceivedFrom(props.query.address).
            call();
27
28      const givenToCount = givenTo.length;
29      const receivedFromCount = receivedFrom.length;
30
31      const outConns = await Promise.all(
32        Array(givenToCount)
33          .fill()
34          .map((element,index) => {
35            return givenTo[index];
36          })
37      );
38
39      //console.log(outConns );
40
41      const inConns = await Promise.all(
42        Array(receivedFromCount)
43          .fill()
44          .map((element,index) =>{
45            return receivedFrom[index];
46          })
47      );
48
49      //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}>
205             Endorse this Participant!
206           </Button>
207           <Button color="teal" basic onClick={this.onRemove} >
208             Remove Endorsement
209           </Button>
210           </Grid.Column>
211         </Grid>
212       <Table>
213         <Header>
214           <Row>
215             <HeaderCell>ID</HeaderCell>
216             <HeaderCell>IncomingConnections</HeaderCell>
217           </Row>
```

```
218         </Header>
219         <Body>
220           {this.renderRows()}
221         </Body>
222       </Table>
223       <Table>
224       <Header>
225         <Row>
226           <HeaderCell>ID</HeaderCell>
227           <HeaderCell>Outgoing Connections</HeaderCell>
228         </Row>
229       </Header>
230       <Body>
231         {this.renderOutRows()}
232       </Body>
233       </Table>
234       </Layout>
235       );
236   }
237 }
238
239 export default ParticipantShow;
```

Listing C.3: Show all details of Participants

Application Components:

```
1  import React from 'react';
2  import { Menu } from 'semantic-ui-react';
3  import { Link } from '../routes';
4
5  export default ( ) => {
6    return (
7      <Menu style={{ marginTop: '15px' }}>
8
9        <Link route="/" >
10         <a className="item" >
11           Endorsement
12         </a>
13
14       </Link>
15
16       <Menu.Menu position="right">
17
18       <Link route="/" >
19         <a className="item" >
20           Participants
21         </a>
22
23       </Link>
24
25       <Link route="/participants/new" >
26         <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;
    console.log('Ready on localhost:3000');
  });
});
```

Listing C.8: Server

```
const routes = require('next-routes')();

routes
  .add('/participants/new','/participants/new')
  .add('/participants/:address','/participants/show' );

module.exports = routes;
```

Listing C.9: Routes