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

Dissertation in partial fulfillment of the requirements for the degree of

MASTER PROGRAMME IN COMPUTER SCIENCE

UPPSALA
UNIVERSITET

Uppsala University
Department of Information Technology

SUJATA TAMANG

August 13, 2018

BLOCKCHAIN BASED DECENTRALIZED REPUTATION MODEL AND GENERAL TRUST FRAMEWORK

Dissertation in partial fulfillment of the requirements for the degree of

MASTER PROGRAMME IN COMPUTER SCIENCE

Uppsala University
Department of Information Technology

Approved by

Supervisor, Jonatan Bergquist

Reviewer, Björn Victor

Examiner, Prof. fName lName

August 13, 2018

Abstract

The purpose of this thesis is to study blockchain technology and smart contracts and demonstrate its use for modeling a general trust framework and an online reputation system. The problems of existing online reputation systems and motivation behind using a blockchain based approach are discussed. Methods of graph properties, network flow, and graph-based algorithms are studied and analyzed to motivate a solution. The result is a PoC design for an Endorsement network where entities can endorse each other. The endorsement model aggregates the interaction and via the use of simple computation steps, can assign a global trust score to entities. This model is applied to an existing real dataset to demonstrate the functionality, correctness, and usability. Various threat models on a P2P network are analyzed, and an interaction graph is simulated to show how the endorsement model handles them. It concludes with a discussion on generalization and further improvements as future works.

Table of Contents

List of Tables

List of Figures

List of Acronyms

EDS	Endorsement
nEG	Number of Endorsements Given
nER	Number of Endorsements Received
CP	Consumable Points
TRP	Total Received Points
TEI	Total Endorsement Impact

1 Introduction

This chapter provides the definition and discussion of various concepts and terminologies that will be used throughout the report. It gives a general introduction to the problem at hand along with the motivation to solve it. The research questions and scope of the project is discussed here.

1.1 Definition

1.1.1 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 [1] [2]. Using the classification from McKnight et al., 1996, 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 personality attribute of the trusting party. It can be seen as a sense of basic trust(attitude) and is cross-contextual. While the trust mentioned above are implicitly directed towards a person, Impersonal/structural trust is more likely to refer to an institutional structure such as a judiciary system.

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. [3] "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, the reputation of 'B' plays a central role. Broadly defined, Reputation is the perception of an individuals character or standing. Like Trust, reputation is context-dependent. e.g., Alice may be trusted to answer or use Linux questions efficiently but not Windows related questions [4]. 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 [5] [6].

The classification of trust and reputation measures based on previous survey [7]:

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

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. As mentioned in [8], trust and reputation are soft security mechanisms where it is up to the participants rather than the software/system to maintain security. Unlike hard security mechanism such as access control, capabilities, authentication where a user can be allowed or rejected access to the resource. Reputation system aids in calculating the probability of success or risk of failure of a transaction between interacting parties [9] [10].

1.1.2 Blockchain

Blockchain can be defined as a distributed record of state changes that let anybody on the network audit state changes and prove with mathematical certainty that the transactions transpired according to the blockchain rules [1]. There exist several definitions of blockchain technology each specific to their closest use case. A formal standard definition of Blockchain is under development as ISO/TC 307 [2].

Vitalik Buterin, the founder of Ethereum, puts it this way:

> "A blockchain is a magic computer that anyone can upload programs to and leave the programs to self-execute, where the current and all previous states of every program are always publicly visible, and which carries a very strong cryptoeconomically secured guarantee that programs running on the chain will continue to execute in exactly the way that the blockchain protocol specifies." [3]

This definition provides a broad overview of what blockchain does. As a continually developing discipline, it keeps adapting to a new definition while maintaining the essence. The major innovation of blockchain as an architecture is distributed, decentralized trustless transactions [11]. It completely removed the need for an intermediary trusted third party by building trust in the system itself. One dimension of trust as mentioned by [12] is trust in data which is based on stored data's integrity. Trusting data ensures

[1] https://media.consensys.net/time-sure-does-fly-ed4518792679
[2] https://www.iso.org/committee/6266604/x/catalogue/p/0/u/1/w/0/d/0
[3] https://blog.ethereum.org/2015/04/13/visions-part-1-the-value-of-blockchain-technology

that the data is appropriate for use: accurate, precise, available, and uncorrupted [12]. Blockchain achieves this by use of cryptographic schemes mentioned in section 3.2 assuring tamper-resistant, fault-tolerance, zero-downtime characteristics [13].

1.2 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.
(i) **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, the integrity of all these relevant data.
(II) **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.[4] Reports by Experia [5] and Javelin [6] 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).
Additionally, reports on fake news [7,8] that leads to spread of misinformation from malicious users or portals, attack on an existing system continues.
A recent report on Finland's data breach exposed [9] 130,000 users login details while Facebook has admitted to the compromise of 2.2 billion of its user's data [10] 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 measuring the reliability of interacting entities so that users can make an informed decision before participating in any transactions. 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.

[4]https://www.experian.com/assets/decision-analytics/reports/global-fraud-report-2018.pdf
[5]https://www.experian.com/blogs/ask-experian/the-state-of-online-shopping-fraud/
[6]https://www.javelinstrategy.com/press-release/identity-fraud-hits-all-time-high-167-million-us-victims-2017-according-new-
[7]https://journalistsresource.org/studies/society/internet/fake-news-conspiracy-theories-journalism-research
[8]https://www.prnewswire.com/news-releases/84-percent-of-businesses-could-reduce-fraud-risk-if-certain-about-customers-id
html
[9]https://securityaffairs.co/wordpress/71136/data-breach/finnish-user-data-breach.html
[10]https://thenextweb.com/facebook/2018/04/05/zuckerberg-facebooks-2-billion-users-assume-data-compromised/

1.3 Purpose and research questions

The primary goal of this thesis is to use blockchain technology and smart contracts to simulate an endorsement network where entities can endorse each other based on physical or digital acquaintance. The endorsement will be quantified to infer reputation score which in turn can yield a value that can represent the impact the agent has made on the network. The nodes and their relationship will be studied to analyze honest or malicious participation. Generalization of this endorsement network to serve other use cases shall be discussed as well.
The research questions that this thesis aims to address are :

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?

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?

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?

1.4 Scope

This masters' thesis work attempts to answer all the research questions mentioned in section 1.3.
Research Question 1
To answer research question 1, literature survey will be performed on various reputation algorithms and existing models. This survey should follow with the discussion on various analysis metrics and threat models eventually leading 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 will be presented. Comparative analysis of on chain vs. off-chain storage requirements will be studied and analyzed. Overall system design and architecture will be presented.
Research Question 3
The endorsement network will be 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 will be presented.

1.5 Structure of Report

This paper is structured as follows. Chapter 2 will perform a literature survey on the existing algorithms and their implementations. Chapter 3 will provide 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 Literature Review

The earliest and most known internet reputation is that of eBay[1]. It uses a feedback based rating system where a user can rate a transaction along with some textual feedback. The range of values used being {1, 0, -1}, positive, neutral and negative respectively. The final aggregated score is computed by subtracting the total of positive and negative ratings. This system [14] [15] could be judged as working based on the sales volume and the observation that more than half the buyers usually engage in providing feedback. However, this method fails to address issues such as Sybil attack, inactive participation(e.g., users fear retaliation from giving negative feedback), whitewashing etc. There are many none E-commerce online systems such as StackExchange [2], Yelp [3], Quora [4], Reddit [5] e.t.c, that make use of similar reputation mechanism to filter the participating users and avoid serving malicious participation. Most of the E-commerce systems employ a client-server architecture which lets a central entity in control of stored data. A single point of control is a single point of failure. In light of this, there have been several research and proposals on decentralized reputation methods for the distributed network discussed briefly in the following sections. Especially significant for P2P systems such as file-sharing, content delivery applications, etc. for detecting the quality of file/content and the owner of those files.

Resnick et al. [16]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 [17] 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. [18] 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. Thus, Sybil identities is not a concern here as a customer is allowed to make multiple identities and in fact, a unique identifier is recommended for a unique transaction. The obvious problem here is the unfair rating attack. A buyer can transact with a service provider and provide negative feedback with

[1] https://www.ebay.com/
[2] https://stackexchange.com/
[3] https://www.yelp.com/
[4] https://www.quora.com/
[5] https://www.reddit.com/

an intention to damage the service provider's reputation despite their honest behavior in the network.

2.1 Reputation Algorithms

The most known and widely used reputation algorithm in a P2P network is EigenTrust [19] 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., peer's 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. 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 [20] also uses this notion. HonestPeer [21] 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. [22], 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++ [22] 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 [23] 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[6] is a BitTorrent based torrent client that uses Bartercast to rank its peers. PowerTrust [24] 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.

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

3 Background

To understand the interpretation of interaction graph and design of endorsement model, basic concepts about graph properties and metrics is required. Similarly, Blockchain relies a lot on cryptographic functions to maintain its property of the secure, immutable and attack-resilient structure. This chapter, therefore, provides a general overview of the background concepts that is required for the subsequent sections.

3.1 Graph properties

A graph, as the name suggests can be used to represent objects and their relationships graphically. Formally, a graph 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 that connects two vertices and $v \in V, e \in E$. φ_G is an incidence function that assigns pair of vertices to each edge of the graph G. φ_G(e) = uv represents that e is an edge that joins vertices u and v [25]. 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 micro, meso, and macro level [26]. At a micro-level, the local properties of a particular node such as its activity, connection degree, its neighbors, and interactions can be observed. On a meso-level, the node's relative position in a given graph that can help to determine its centrality and connectivity can be observed. On a macro-level, one can view the overall network as the topology of the graph itself. It helps to study the global properties of the graph.
Network metrics that are helpful in analyzing the complexity of interactions at macro and meso-level are [26]:
Degree Connectivity: The number of connections a node has is the degree of its connectivity. 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.

3.2 Cryptography

3.2.1 Basic Concepts

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 [27].
A cryptosystem can be defined as a five-tuple (P, C, K, E, D) where,
P, is a finite set of plaintexts and C, 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 *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 3.2.3. RSA [28] is one example of a public-private cryptosystem.

3.2.2 Hash functions

Cryptographic hash functions are a one-way function, 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 its 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. Other significant properties of hash function that contributes to reliability in digital security are [29]:
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^{24} [30]. 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 [31] 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. This cryptographic hash functions is relevant for understanding blockchain data structure. Especially significant for verification of transactions that make use of hash tree, namely Merkle Tree for including list of transactions in the block and chaining the blocks together. The process of getting a root hash is given by 3.1 Two significant use of Merkle tree are:
i) Data integrity verification:An attempt to change any transaction data would completely change the root hash of the block.
ii) Data inclusion verifcation:It is easy to verify the inclusion of a transaction in a block without requiring to include all the transactions. From figure 3.1, to verify inclusion of transaction HK, only the hash values that are shown in blue needs to stored.

The PoW consensus mechanism, relies on SHA-256 hash(in Bitcoin) applied to the block header and nonce to produce a verifiable fingerprint of data. This is discussed in more detail in 3.3. Proof-of-work was originally used in 1997 by Adam Back as a anti-spam system for the then proposed Hashcash [32]. The general idea being that the sender of

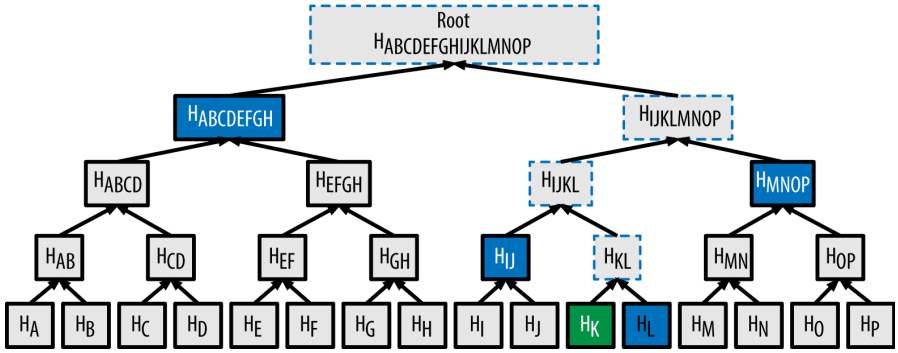

Figure 3.1: Merkle Tree

message would have to compute some number of sha operations before sending the message. The generated fingerprint could be checked by anyone to see if the sender has actually done the required number of computations. If a legitimate user sending out one email had to spend 'x' seconds to do the sha operations, a malicious user intending to send thousands of emails had to spend 1000x seconds.

3.2.3 Digital Signature

A digital signature acts as an intermediary to prove that an entity A, has the password without ever requiring A to reveal it. As discussed earlier, public-key cryptography uses key-pairs that correspond with each other. In context of Blockchain, If Alice wants to send a value(transaction) to Bob, she can create a transaction message, sign it using her private key and broadcast the transaction over the network. Her signature and the transaction message will be publicly available on the network(assuming a public blockchain network). Anyone on the network can verify that the signature corresponds to Alice's public key. Thus, Alice can always prove that she is the owner of the public key from where the message originated. The signature is dependent on the message, and therefore any attempt to modify the message will refute the signature.

3.3 Blockchain Technology

Blockchain Technology is a variant of distributed database implemented on a P2P network. Every participating node in the network has the same copy of history of records of database. Blockchain is essentially a chain of blocks, as the name suggests. Each block consists of list of valid transactions(signed messages) collected by validators/miners of the network. The linking and ordering of transactions are also the responsibility of validators. The validators proposes a block they have unlocked to the network which can either be accepted or rejected. If accepted, the newly created block gets linked to the existing blockchain with a hash pointer pointing to the previous block. As such, blockchain provides the transaction ordering that every node agreed on at a given time.

Consensus metric helps to establish and maintain the integrity of a blockchain system[1]. The primary attributes that constitute a blockchain system are distributed, decentralized, time-stamped transactions. There have been P2P protocols deployed as a file-sharing or other form of content delivery services before Blockchain. What separates Blockchain from them is that for the first time, it made possible to transfer values online on a P2P network without a double-spending problem. i.e., If a peer 'A' sends a file or a value 'v' to 'B,' then the file should be owned by 'B' and removed from the account of A. 'A' should not be able to send the same file 'v' to other entities on the network. This was the significant contribution of the technology.

3.3.1 Evolution & Categories

Bitcoin was the first application that made use of Blockchain technology which was a peer-to-peer electronic cash system. The major contribution of this application was solving distributed trust at scale without using a trusted intermediary. Along the dimension of validation and access control [33], blockchains 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 trusted entity. The records are publicly verifiable.
- Private Permissioned: This is similar to a 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 doesn't need to know about other business policies or customer information etc.

[33] provides a detailed discussion on various blockchain types and their uses.

3.3.2 Consensus Mechanisms

As a distributed database with multiple writers, there has to be a way for everyone 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[2]:

- Leader Based System: In this case, there is a pre-selected leader that collects all the transactions and appends 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

[1] https://github.com/ethereumbook/ethereumbook/blob/develop/consensus.asciidoc
[2] Categorization based on Hashgraph's presentation [45]

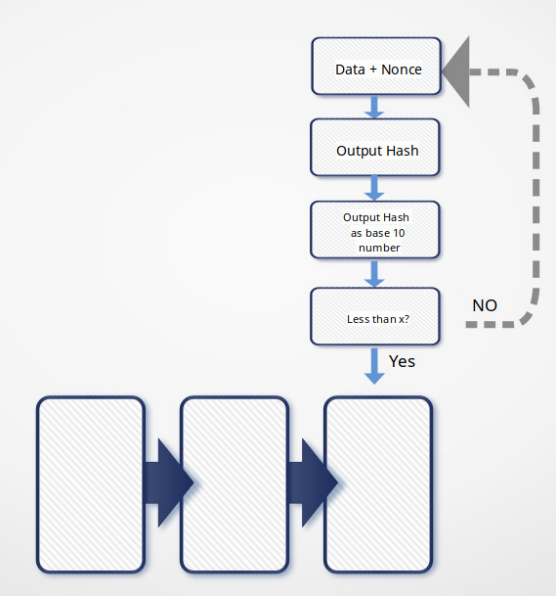

Figure 3.2: Cryptographic Puzzle

other centralized system, this system is susceptible to DDOS attacks and third-party(leader) interference. Generally used in a private or permissioned blockchain setup, it offers higher throughput compared to public permissionless blockchains. Examples include Hyperledger Fabric, R3 Corda etc.

- Proof-Of-Work: This is the most widely used consensus mechanism in a public permissionless setup. As the name suggests, a validator/miner needs to provide the proof to the network that it has done a significant amount of work. This work requires miners to invest a substantial amount of computational resource. The reason for this is that everyone(all miners) compete to be the writer of the next block for which they need to solve a cryptographic puzzle. Mainly, they need to find a hash value that can be associated with the proposed block. The only way to find this value is by brute-forcing. The simplified overview of this mining process is given by figure 3.2 The apparent advantage of such consensus mechanism is that it makes the system DDOS resistant while offering immutable audit trail and scalability. However, miners can still decide upon the order of transactions to include in the block although they cannot modify the transaction. As such, one could term this as 'unfair' since the transaction doesn't get picked up in order of when it was broadcasted to the network.

- 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 don't compete with each other to be the writer of 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 highest value at stake gets to write the next block. If the participation turned out to be a malicious one, then all the tokens that were at stake get lost. As such, it puts scarce resource at stake. However, this includes problem such as nothing-at-stake [34]. i.e., a node could vouch for two forks of the same blockchain with nothing to lose. Other drawbacks of this approach are that there is no certainty of consensus, and often has no total ordering of transactions. Examples include Casper 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. A smart contract, however is a computer program that can codify the interaction between participating entities and self-executes when triggered by an event. It doesn't necessarily need to be legally binding or even associated with the outside world [3]. The term Smart contact was first coined by Cryptographer Nick Szabo, in 1994 [35] 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 [36]:
Observability, ability to observe the 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 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, and objective mentioned above.
Privity refers to minimization of third-party vulnerability by limiting knowledge and control whereas, 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. [37]. Following the invention of Bitcoin and several decentralized blockchains, the definition of smart contracts have evolved. Ethereum being the first platform to offer programmable blockchains, introduced a virtual machine, EVM, where the contract code can be executed that results in a deterministic output provided the same transaction context and blockchain state [4]. 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 platform. Examples include solidity, serpent, LLL, etc. For the sake of relevance to this

[3] https://universe.ida.dk/media/23422289/fritz-henglein.pdf
[4] https://github.com/ethereumbook/ethereumbook/blob/develop/smart-contracts.asciidoc

project, solidity as the smart contract language and Ethereum as the blockchain will be used as a point of view henceforth. Contract's code resides in the blockchain as an immutable form. They are not autonomous self-executing program 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, there exists a possibility to include killable function that can be executed by the owner which when called executes an EVM opcode called SELF-DESTRUCT and deletes the contract from the blockchain. As in any Turing-complete language, solidity is affected by the halting problem. To address this, Ethereum introduces the concept of gas. 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 [38].

4 Methodology and Implementation

The problem of measuring the trustworthiness of communicating entities is an essential aspect of any online system where they interact with each other for any purpose, be it shopping, content delivery or file sharing. This chapter follows on a discussion of a proposed endorsement network where physically or digitally acquainted entities can endorse each other or their presented information. The model will address several concerns such as the roles and requirements of participants as endorser and endorsee, why a participant would play by the rule and what is to stop them from not doing so, threat models, etc. With a system of smart contracts, PoC design will confer interaction between entities, aggregation of information and assignment of scores for final computation. The storage of data both on and off-chain will be 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 immutable, trustless, decentralized and distributed attribute of blockchain protocol is a recommended solution on a public permissionless network.

4.2 User stories & Requirements

Anyone can join the network and become a participant in the endorsement system. The two notable roles of a user are endorser and endorsee. An endorser can initiate the transaction by sending an endorsement to the participant they wish to. The same user can assume both the roles of endorser and endorsee as long as a set of predefined requirements are met.

The user stories for each role that defines the system requirements for each user type is presented in table 4.2.

The functional requirements can be listed in points as :

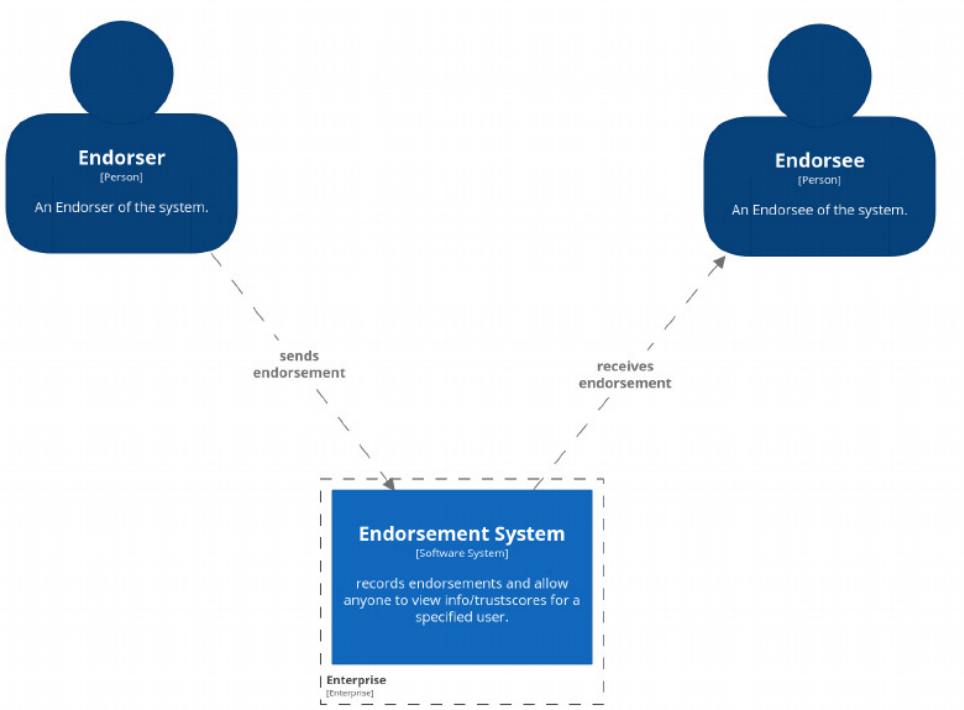

Figure 4.1: Context Layer

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 to whom i have sent 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/receiving endorsements.	R4.2

Table 4.1: User Stories and Requirements

1. It must be impossible to make an Endorsement if the endorser and endorsee is same address or not a registered participant.
2. It must be impossible to remove an endorsement if the endorser doesn't belong to the list of endorsers for the given endorsee.
3. All the endorsements must be stored such that, it is possible to see:
   - 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 public key hashes to the corresponding computed trust scores.
5. It must be possible for a participant to edit their own profile if the editor is the same as the profile owner.

The non-functional system requirements are :

1. Security: There should be minimal security considerations taken into account while writing the smart contract that can avoid simple bugs.
2. Reliability: The data stored on the blockchain should be immutable and verifiable by anyone publicly.
3. Trust metrics should correctly describe the actual trust score/impact of the nodes.

4.3 The Model - Endorsement Network

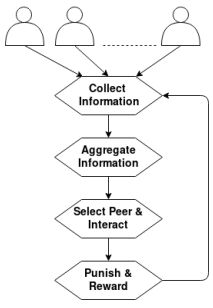

Figure 4.2: Trust and Reputation Model steps taken from [39]

Complete trust and reputation model consists of four different steps as depicted in figure 4.2. Among these steps, endorsement network concentrates on the first two steps, collection, and aggregation of information. For 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 network. As implementing transaction network is not the main task of this project, feedback based on success or failure of a transaction will be mostly based on the assumption that can be used by endorsement system. Thus, most of the section in this chapter is based on the smart contract logic for endorsement model and discussion on the possibility to be used by other transactional systems.

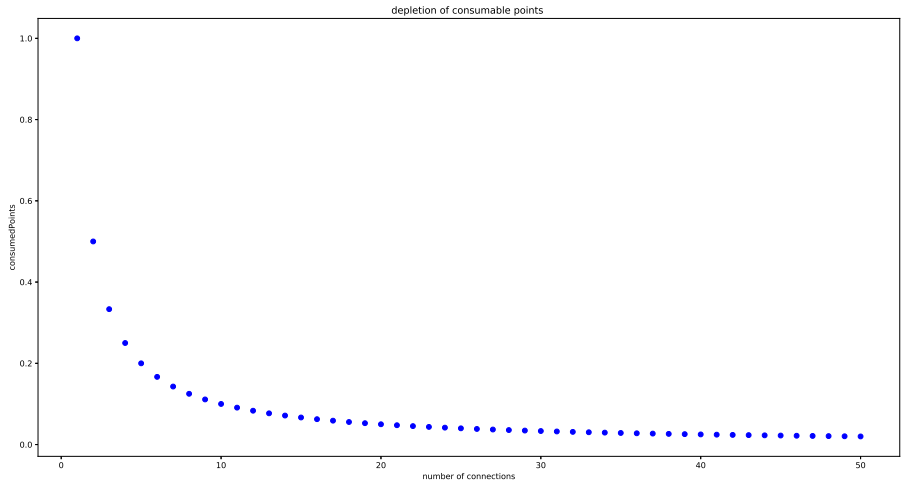

Figure 4.3: Convergent behaviour of consumable points as 'n' increases

The initial assumption on endorsement system is that all nodes are honest and as such receive equal points that they can spend at will once registered on the network. This received points are the consumable power that keeps depleting with every endorsement connection made along the way. As can be seen in figure 4.3 , these points follow a convergent sequence that converges to the limit 0 as the number of connection 'n' increases. As such, increasing the number of connection alone will not be enough to achieve a higher impact on the network.

4.3.1 Computation of Total Endorsement Impact(TEI)

The collection of information refers to the collection of endorsement interaction between endorsers and endorsees. Aggregation of them to get the global trust score, total endorsement impact for all the nodes is made based on various factors. A short definition of them and the terminology used throughout the endorsement model is given below.

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

nER$_A$: Number of Endorsements Received (nER) by a pariticipant 'A'.
$ratio_A$: ratio of nEG_A to nER_A. This value ensures that sent and received endorsement are not far off from each other. $ratio_A$ is always assumed to be less than 1 and is given by:

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

CP$_A$: Consumable Points (CP) of a participant 'A' that updates with each new endorsement interaction. 1 being the initial consumable points received by everyone who joins the network,$CP_A$is given by $1\ /nEG_A$.

TRP_A: This corresponds to A's Total Received Points (TRP) which is the accumulated sum of consumable points by all endorser of A. If a peer 'A' receives an endorsement from 'n' number of peers, then the TRP_A is calculated as:

$$TRP_A = \sum_{i=0}^{n} cp_i \tag{4.2}$$

Finally, the Total Endorsement Impact (TEI) made by 'A' is given by:

$$TEI_a = ratio_a * TRP_a \tag{4.3}$$

4.4 Design of PoC

The PoC design for Endorsement (EDS) system is based on the requirements mentioned in section 4.2. This section explains the design considerations and the setup of smart contracts used for the EDS system. A high-level overview of the overall system is presented in figure 4.4. The visualization is based on an architectural model by [40]. It essentially uses three-layer to visualize a software architecture. i.e., context layer, container layer, and component layer. The context layer attempts to describe several user types and their interaction with the system on a high level. The container layer takes it further by expanding on the software system and shows a high-level overview of the components involved. The component layer then goes on to explain in detail the connected component. For endorsement system, the component that is most relevant and thus explained in detail is the smart contract. After the design details, the section ends with the discussion on graph algorithms that can be used by the transactional network, blockchain infrastructure, governance model, and storage of information on and off-chain.

4.4.1 Design Consideration

The design considers possible behavior that can result from the interaction between nodes, both honest and malicious. The endorsement relation exists based on a subjective measure from an endorser. Thus, they are known entities based on physical or digital acquaintance.

The acquaintance could be of the following form:

- Alice and Bob go to the same school/workplace, have worked on multiple projects together and is confident of Bob's reliability.
- Alice has dealt many times with Bob in an online shopping website and always had an excellent transaction with him. In this interaction, Alice is sure that Bob is an honest seller and Bob is confident that Alice is a reliable buyer.

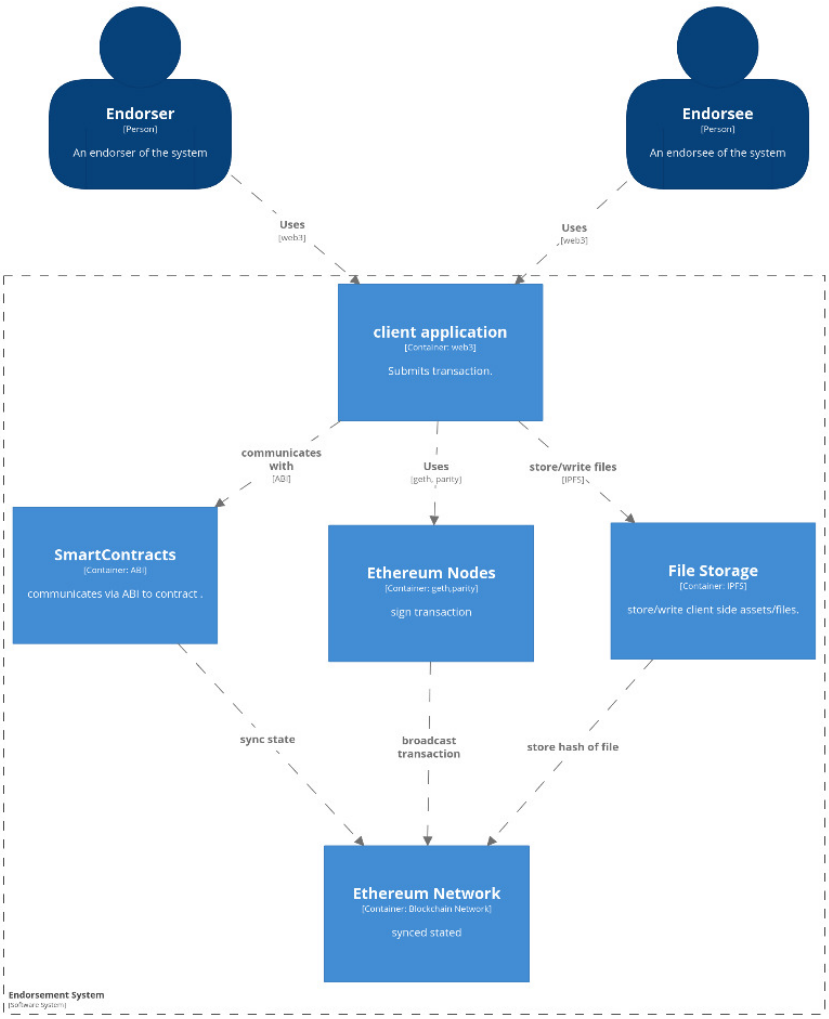

Figure 4.4: Container Layer

- 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.

Thus, Alice is likely to endorse Bob and vice-versa on the endorsement network. They can use secure messaging channels to exchange their keys and endorse each other. There is no objective way to verify if Alice and Bob are separate entity sending out an honest evaluation of each other. However, the endorsement network can receive feedback from the transaction network. Therefore, if any entity that has received endorsement turns out to be a bad actor in an online transaction, they can be punished along with the node that sent the endorsement. i.e., both endorser and endorsee get punished by decreasing the total endorsement impact by 50%. From a game-theoretic perspective of a behavioral outcome, following definition were made for network influencing factors.

1. **Fake endorsements with pseudonymous identities:** Endorsement system being on a distributed, public permissionless blockchain network allows anyone to join and start sending endorsements immediately to whoever they wish to. This creates the possibility that an entity could create multiple pseudonymous identities with an aim to inflate their impact on the network by increasing nEG or nER for their associated persona. There is no straightforward way to detect and stop this behavior right away. However, if doing so doesn't provide any significant advantage, then the assumption is that a rational decision would be not to do it. One factor that is believed to stop a participant from making too many endorsements is the convergent behavior of consumable points. While, there is no limit to the amount of connection a participant can form, as the number of connection increases, the value of consumable points decreases.

2. **Transaction cost:** Ethereum is a programmable blockchain that supports a Turing complete programming language. Thus, to avoid running in infinite loops or DDOS attacks, gas was introduced. Every operation has a gas cost. One could write a program to do anything they wish to do on the network as long as the account that initiates the transaction can pay the gas cost of all operations. The gas consumption is an imperative aspect of endorsement system for two reasons.

    - *Standard transactions on Ethereum:* A participant that makes a call to 'sendEndorsement' function is responsible for paying all the required gas costs. This function updates the state variable nEG and nER. While the price may not seem too high for making one transaction, a malicious node with multiple pseudonymous identities has to pay for all the operations initiated by all personas. For instance, given the interaction graph in the figure, if Alice is an honest node, then she only needs to pay for the operation of one transaction. Whereas if both Bob and Charlie are the pseudonymous identities of Alice, she needs to pay for six transactions. Thus, the assumption is that they may

make ether transfer at some point between their accounts. This information is publicly available for anyone on the ethereum blockchain network to view the chain of ownership. If some interactions in the endorsement network look suspicious, one could look up this detail. This method is not guaranteed to detect a Sybil node on the endorsement network but is just another additional factor that might be useful before making a decision.

- *Local information of all neighbouring nodes:* Whenever an endorser makes a new connection, the nEG, and consumable point change accordingly. This change in consumable point has to be reflected for the list of all endorsees. This is not to be confused with a one time update. Every new connection made by an endorser changes the state for all his/her old endorsees. Therefore, all the neighboring nodes of an endorser should be stored previously. The impact of a participant is dependent on his/her direct interaction as well as the endorsers(the participants that have endorsed them). There is no way to make constant cost lookups and updates for this operation. It requires iterating through the list of arrays and computing the impact of every endorsee based on the updated state variable. While it is possible to iterate through items in the array, the general recommendation is to avoid them if possible. One could surely assume that the list will not grow too big for two reasons: (a)A rational node will not make too many connections for reasons mentioned earlier in 1 (b)Dunbar's number suggests a cognitive limit of 150-250 stable social relationships for humans.
  One way to approach this problem is to store the list of endorsees and endorsers for a participant but not change the state. The computation can be done on the client-side using language such as javascript. The final score will be done by the client. However, all the variables necessary to compute the final score will be stored and updated on blockchain as a publicly verifiable information.

3. **Dynamism:** The dynamic social behavior of human is that trust between two entities is not perpetual. Alice may have trusted Bob yesterday but refuses to endorse him today. Trust is dynamic and so is the endorsement decision that an entity can take. Therefore, the design also considers removal of endorsement previously assigned. The removal of endorsement is captured by the figure **??**.

4. **Free-Riders Problem:** Free riders problem is addressed by making it necessary to maintain the ratio between nEG and nER. A peer without a balanced proportion cannot have a significant impact score on the endorsement network. This method also discourages Sybil nodes because each identity needs to have an almost equal bi-directional connection. If they are only receiving from their own pseudo identity that don't have too many connections, then the impact is ignorant and thus useless and not worth the time.

4.4.2 SmartContract

Smart contracts can be either deterministic which doesn't require any information from outside blockchain or non-deterministic which does need to get oracles from external sources[41]. For this PoC, endorsement contract is deterministic and so all the data and variables required are stored and executed on the blockchain.
Figure 4.5, represents the system of smart contracts as a way to visualize the component layer and is based on [42]. The main contracts written for this PoC are :

- Ownable: tracks the owner of the contract. i.e., the creator of contract.
- killable: inherits from ownable and can be killed by owner only.
- Participants: set participant and store their information. An index to access each participant.
- Endorsement: It inherits from participants and can be called by participants only. Endorsement contract handles the core logic of endorsement system, accesses/-queries addresses from Participants and is used for storage of data along with CRUD operations on them.
- Computation: inherits from Endorsement contract and allows anyone to get the final score by accessing the current state from an Endorsement contract.
- Marketplace: stores the buyers, sellers information and allows them to buy or sell the product. Also, allows buyer/seller to compute the score of the involved entity before doing a transaction.

'Marketplace' contract was written to test the endorsement network on a transactional network. However, when deploying endorsement system in the real world, other transactional network/online systems are assumed to have their reputation platform. The reputation platform should have assigned a score to the corresponding users based on the behavior on that network. The endorsement system can act as additional conformity for deciding on a transaction. Say, Alice is registered on Endorsement network and has made a decent score. If she wants to sell a product on Marketplace(or any other transactional network), she can claim about her score and anyone who wants to buy from Alice 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 on the blockchain, they can send a pre-transaction message to each other to verify that Alice is who she claims to be and vice-versa. 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.

4.4.3 Data and variables on and off blockchain

For this PoC, the data required to identify the users are stored on the blockchain. But, it does preserve the anonymity requirement mentioned in section 4.2, as the only public

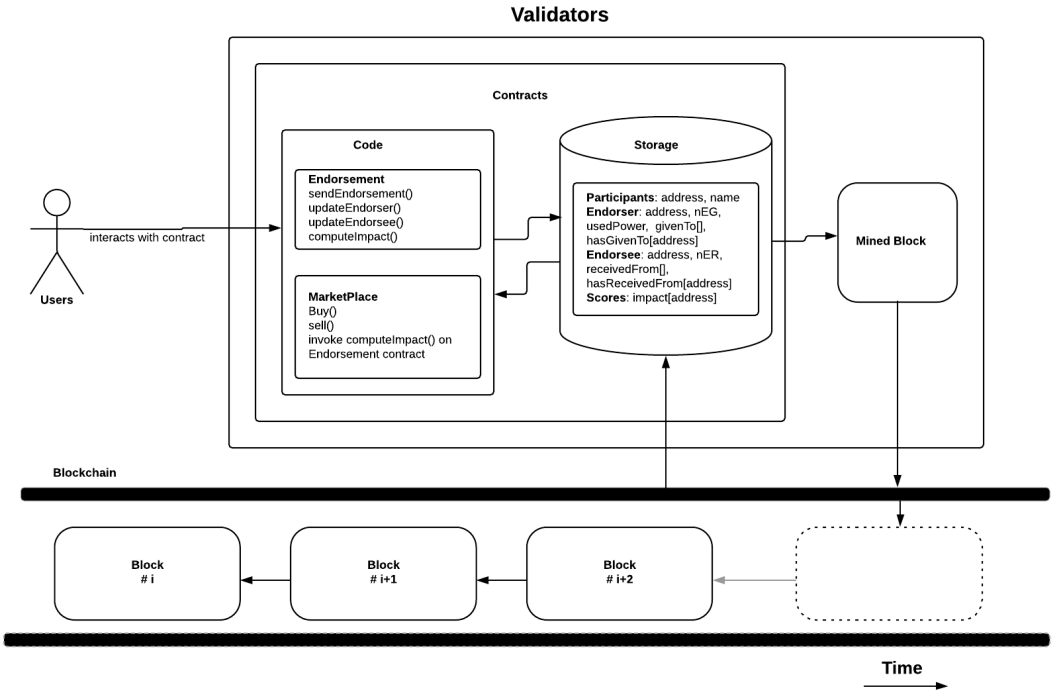

Figure 4.5: Smart contract system

information is the link between the public key hash and individual trust score. Even though the users are required to register with a pseudonym, it is not needed that the aliases be linked to real-world identity. A user might like to share more information(other online account ids, address, etc.). As mentioned earlier in the gas consumption section, every non-zero byte data or code of a transaction costs a certain amount of gas [43]. Storing this data can become an expensive operation for real-world usage. The right approach can be to use an off-blockchain storage solution such as IPFS, swarm [44]. The hash that points to the file in IPFS can then be stored on the blockchain. Generally, client-side assets (HTML, js) are stored on these distributed off-chain file system that can communicate to the contracts registered on blockchain network.

4.4.4 Blockchain and Consensus algorithms

The proposed platform for this PoC is Ethereum, continually developing open source blockchain ecosystem. The process of starting up the nodes and deploying the contract on the network is shown in figure 4.6. As a permissionless system, it allows any nodes to collect transactions and act as a writer. Consensus mechanism that is generally used in a permissionless setting is Proof-of-Work or Proof-of-Stake. 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. In the endorsement network, an honest endorsement is supposed to be between known

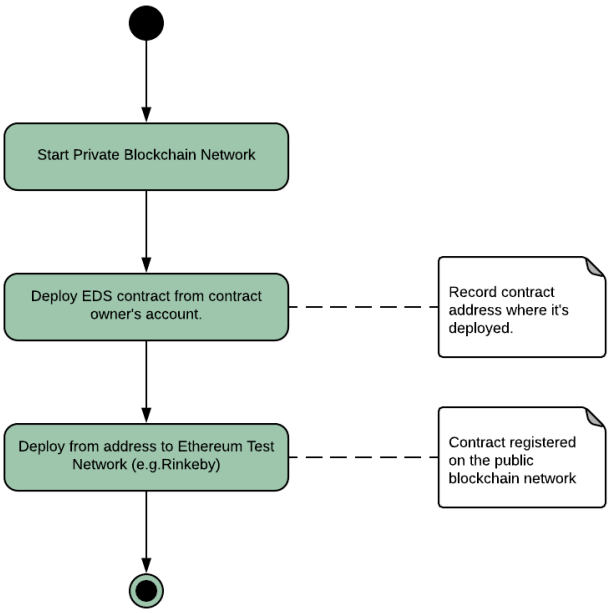

Figure 4.6: Startup activity for registering contract on the network

entities. However, all participating entities do not know each other, and no one node can be trusted to collect everyone's transaction and make a final commit. Gochain [1] 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 can 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. It is not recommended to start with it. However, PoR can be used along with PoW from the beginning. As such, the recommended consensus mechanism for the current design is PoW despite its limitations. Various alternatives to PoW and consensus algorithms related research that can solve specific problems(e.g., transaction speed, transaction size, size of nodes, etc.) are under development. Hashgraph [45] proposes the recent advancement in consensus engine that claims to be fair, fast and Byzantine fault tolerant. It is based on gossip protocol(nodes gossip about transactions and gossip) and offers mathematical proof on 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

[1] https://gochain.io/

be used on a permissioned setting, much like sovrin [46] does. Sovrin introduces a steward node who is trusted based on a signed agreement with sovrin [2] and has received approval as trusted institutions. The concept of steward nodes might be questionable regarding full decentralization, 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 organizations that could agree on a specific set of protocols. Validation of transactions can be based on a voting mechanism that requires the consent of 2/3 of the trusted members. Doing so can increase the transaction validation time compared to PoW.

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

5 Evaluation & Results

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 will be presented.

5.1 Evaluation Criteria

To assess the fulfillment of requirements, a descriptive approach based on the design evaluation method of [47] 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 will be simulated using an interaction graph simulation tool neo4j [1]. It will follow the study of graph topology and relevant subgraphs to study the interactions and base the results on it. For this interaction, an existing dataset from SNAP[48] is used. The details on the dataset are presented in the respective section. Given the time constraints, no form of structural/unit testing was performed. The purpose of this project was a PoC design to demonstrate reputation model as a use case via the use of smart contracts and blockchain technology. Therefore, extensive experiment to simulate a real-world usage was not performed. However, manual testing was done during development using ganache [2] and remix environment [3]. Additionally, front-end for endorsement interaction was also deployed to test simple function calls and communicate from the web 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 [49] that uses Hevner's descriptive design evaluation approach[47]. The table 5.2 presents the motivation for how the PoC fulfills the functional requirements. For the relevant smart contract code, refer to the appendix section.

The fulfillment of non-functional system requirements is discussed below:
SmartContract Security: Given the limited timeframe of this project ($\sim$ 4 months), not all security considerations were taken into account. As part of fail-early principle[4], the

[1] https://neo4j.com/
[2] https://github.com/trufflesuite/ganache
[3] https://remix.ethereum.org/
[4] https://blog.zeppelin.solutions/onward-with-ethereum-smart-contract-security-97a827e47702

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

contracts function code was ordered as conditions, actions, interactions where relevant. A snippet of a function to demonstrate this is presented below.

```
1   function joinNetwork(string _userName) public{
2     //conditions: only allow unregistered participant
3     require(!joined[msg.sender]);
4
5     //actions
6     joined[msg.sender] = true;
7
8         //Interaction
9     Participant memory newParticipant = Participant({
10        identifier: msg.sender,
11        name: _userName
12        });
13    participants.push(newParticipant);
14        LogJoinNetwork(msg.sender, _userName);
15        numberOfParticipants++;
16        participantIndex[msg.sender] = numberOfParticipants-1;
17    }
```

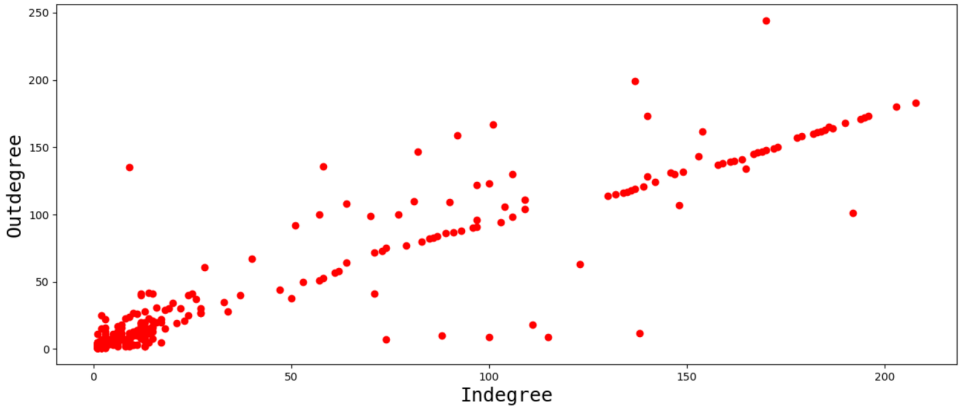

Figure 5.1: Given Vs. Received

Reliability: The information of endorser, endorsee and their interaction is collected as transaction data by the miners. A miner that is first to solve the hash value of a block gets to write the list of transactions into a block and chain it to the existing chain of blocks. The block-chain protocol ensures the immutability of these list of transactions. i.e., Each block has a timestamp and a pointer to the previous block. The public ledger ensures the verifiability of data which can be checked by every node on the network. Thus, the reputation score stored on endorsement system can be said to be reliable.

Trust metrics correctly describe the actual trust of the nodes: For this requirement, an interaction graph is simulated, and different scenarios are modeled that represents different possible behavior in the network. Refer to section 5.3 for this.

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 [5] weighted signed network which was mainly 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 consists of 3,783 nodes that made 24,186 edges out of which 93% of the edges were marked as positive edges[50].

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 [51]. Unlike the Bitcoin Alpha network

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

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 doesn't 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 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 edge weights.
The scores along with new findings are presented below:
Total Endorsement Impact: This value is based on the degree of connections and total received points for each node. Surprisingly, there were 1175 nodes(70%) whose final computed score was equivalent to zero. On examining the nodes, it was found that all of these nodes only had one incoming or outgoing connections. One requirement that was required by the implementation was that a node needs to have a strictly greater than one number of connections to be considered for impact score computation. The distribution of indegree and outdegree can be seen in figure 5.1. Having one connection is only considered a start in the network. The zero, in this case, is not a representation of a non-trustworthy node, but a starting node. Therefore, it can be assumed that 70% of the nodes in the network are new users. This computation leaves the network with only 502 nodes to account for having a considerable impact value.

Node Label	nEG	nER	ratio	TRP	TEI
430	5	3	0.6	0	0
761	2	2	1	0	0
448	3	3	1	0	0
676	2	2	1	0	0
936	2	2	1	0	0

Table 5.2: Nodes with Impact zero because of the receivedpoints

Total Received Points: The remaining 502 nodes had connections more than 1. Nevertheless, there were still few nodes among them with impact value equivalent to zero. This is due to another factor, received points, that the endorsement impact takes into account. The relation between Total received points, and total endorsement impact is shown in figure **??**. This factor is more or less similar to the prestige centrality metrics in a graph network. However, in this case, the significance is not directly associated with the total endorsement impact of the node it is connected to. Instead, the depleting consumable point of a giving node accumulates to the received points to a receiving node. The interaction subgraph is shown in figure 5.2 for the discussed nodes behavior. The figure makes it clear that even though the nodes 430, 761, 448, 676, 936 have a degree

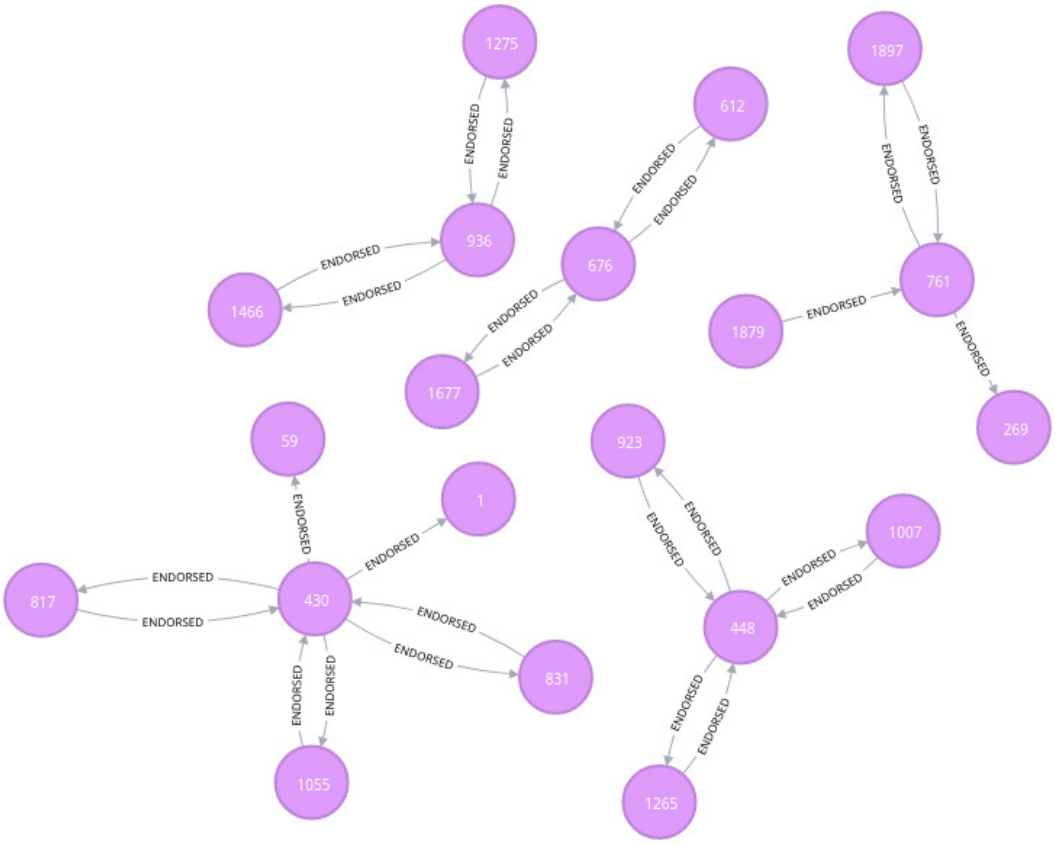

Figure 5.2: Interaction subgraph of nodes with impact zero

of connection that is more than 1, their respective endorsers do not. Table 5.2 displays the state and scores of these nodes.

Ratio: Another significant factor that the Impact computation takes into account is the ratio of indegree and outdegree. The table 5.2 makes it clear that ratio alone is not enough to have a high impact. The figure 5.5 shows the relation of ratio and total endorsement impact over all nodes. One can infer that higher ratio does not imply a higher impact but a higher impact most probably implies a higher ratio. As such, it is a contributing factor in the long run.

Among the nodes that have maintained an above zero value for total endorsement impact, the distribution of ranges is shown in table **??**. Majority of nodes have impact value within the range of 0-25. Only a few nodes have managed to obtain a higher impact value. For simplicity, one could classify the impact value ranges as less trustworthy, average trustworthy and more trustworthy. Based on the distribution table **??**, 0-55, 55-105 and 105-155 can be used for the mentioned classification. The term maximum impact henceforth refers to the more trustworthy nodes, average impact to average trustworthy and minimum impact refers to less trustworthy nodes. Figure 5.4 gives the interaction subgraph of the most impactful node for the provided network.

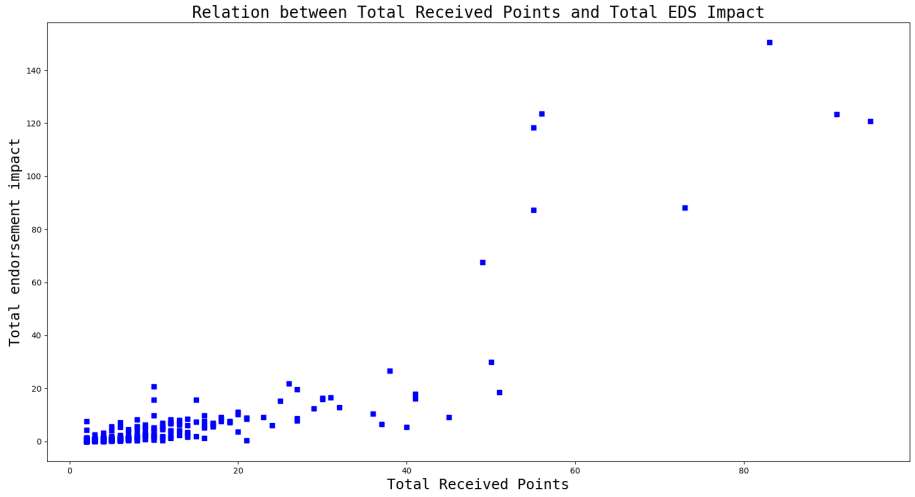

Figure 5.3: TRP vs. Total EDS Point

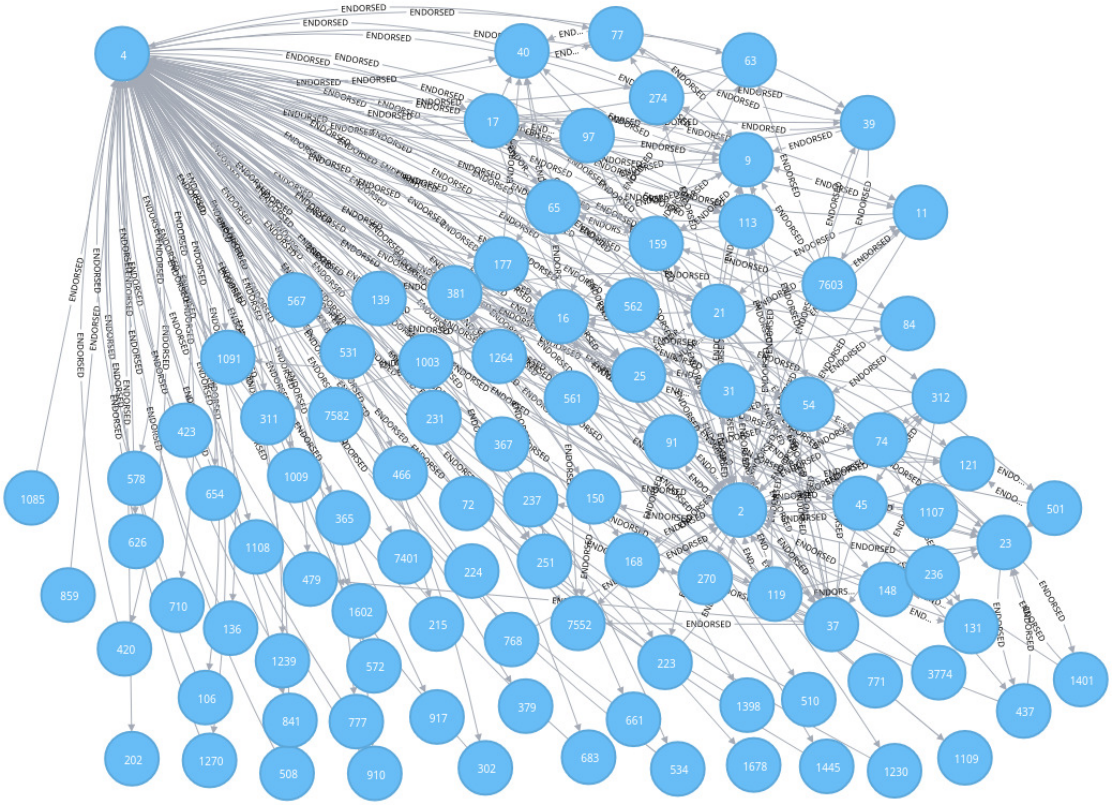

Figure 5.4: High Impact Node

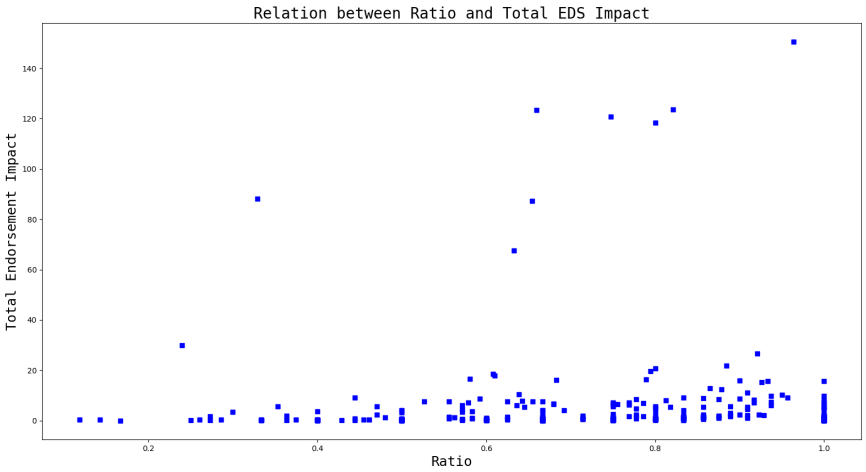

Figure 5.5: Relation of Ratio and Total endorsement impact

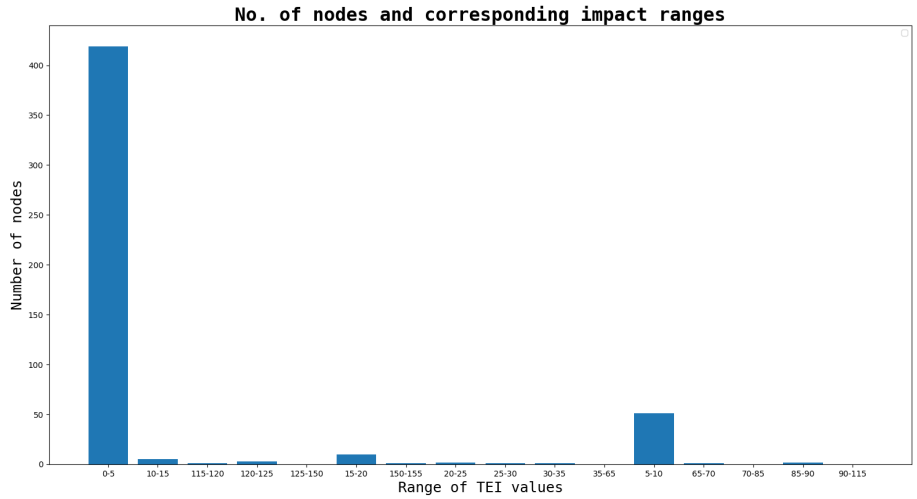

Figure 5.6: table:totalimpact

5.4 Total impact across several factors with different scenarios

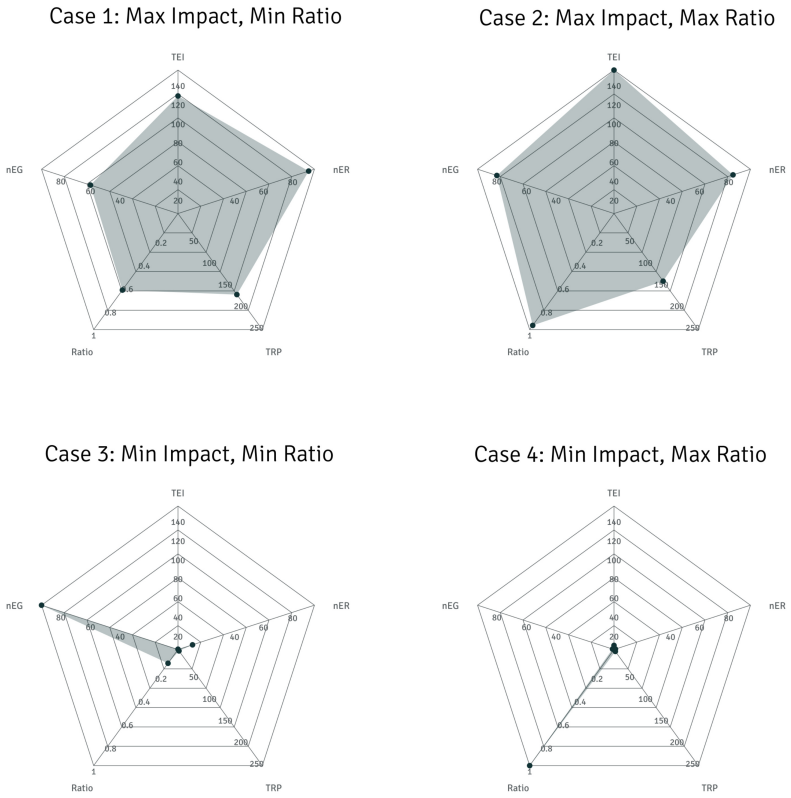

Figure 5.7: Total Impact Across all factors

Different scenarios are considered to clarify further the relationship between ratio and total endorsement impact. It seeks to answer the question such as why a high ratio can both contribute to a high and low total endorsement impact and vice-versa. Although only these two factors are mainly focused on creating different cases, the study will show the effect of other factors and their state as well.

5.4.1 Case1: Maximum Impact, Minimum Ratio

All the nodes that got maximum impact value when computing the endorsement impact are considered in this case. Among them, the nodes with the low ratio are extracted to examine the behavior. As can be seen in figure 5.7, the lowest ratio of a maximum impact node is 0.6. This illustrates the behavior mentioned earlier that a maximum impact node most certainly has a maintained ratio.

5.4.2 Case2: Maximum Impact, Maximum Ratio

A maximum ratio of a node with a maximum impact should be the maximum ratio possible in the network. This behavior is the best case scenario where a node is actively

participating both as an endorser and an endorsee. The data is taken using the same approach as in case1 for this scenario as well. The behavior of such a node is depicted in figure 5.7 Case2. The value across all dimension is equally distributed.

5.4.3 Case3: Minimum Impact, Minimum Ratio

A minimum impact node is the one that is not an active participator in the network across all dimension as required. For this category, all the nodes from minimum impact region were taken. From those nodes, the node with the minimum impact is taken to show the behaviour as shown in figure 5.7 Case3. It can be seen that the node has only actively involved in giving out endorsements but received relatively low connections. As such, neither the ratio or the impact is significant for this node. This behavior is typical of an extreme one-way connection.

5.4.4 case4: Minimum Impact, Maximum Ratio

Table 5.2 and figure 5.5 has already demonstrated this form of behavior as well. From the minimum impact nodes, the node with the maximum ratio is taken to simulate an interaction that looks like figure 5.7 case 4.

5.5 Threat Model

An honest peer in the endorsement system has scores distributed across all factors as shown by figure 5.7 in case2. Similarly, a malicious peer is the one that has high score across one dimension which is shown by case 3.
The relevant threat scenarios for the proposed system can be described as following:
Individual and collective malicious peers[39]: A malicious peer is always supposed to provide inferior services in the transactional network. Similarly, a malicious collective is formed by a group of malicious peers that give a good rating to each other with the intent of inflating their trust score. This can be detected by other peers based on feedback/rating. Once detected, the peer will have an endorsement impact reduced by 50% and the endorsers of the node will have the total impact reduced by 25%.
Sybil Attack: The endorsement network addresses Sybil attack by requiring the endorsers of a peer to have high TEI as well. A peer can create multiple Sybil identities and self-interact to send a large number of endorsements directed to themselves. However, being endorsed by the set of endorsers with a zero or low impact does not contribute towards a better global score for that node as discussed earlier. Thus, a peer that creates multiple identities is required to maintain the score across all factors for all pseudonymous nodes thereby increasing the cost of maintaining global value for all nodes.
Whitewashing: Whitewashing [52] is when a peer with a bad reputation score on a P2P network tries to exit the system and starts afresh as an entirely new user. In case of endorsement system, a new user can never have an impact score higher than an old user.

Assuming that a malicious peer was detected and penalized, the penalty is a certain percentage of the obtained score. As such, an old user will always an impact value greater than zero. The new user, on the other hand, will start with a full consumable point but a zero impact value. To be considered for impact computation step, the peer needs to make strictly larger than one connection. Since the value is decreased in percentage. A new user will only receive a consumable point but has a zero impact when entering the network. The design requires that a peer make strictly more than one connection to be considered for impact computation step. Thus, white-washing is a discouraging attack in the endorsement model.

Freeriders: In a P2P network that provides service, free-riders indicates the peers that only take services but do not participate equally in uploading files. In endorsement system, they can denote the peers that have high outdegree and negligibly low incoming connections. As shown by 5.3, doing so doesn't contribute to a high total impact value.

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, 4.4.1 4.4.

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.

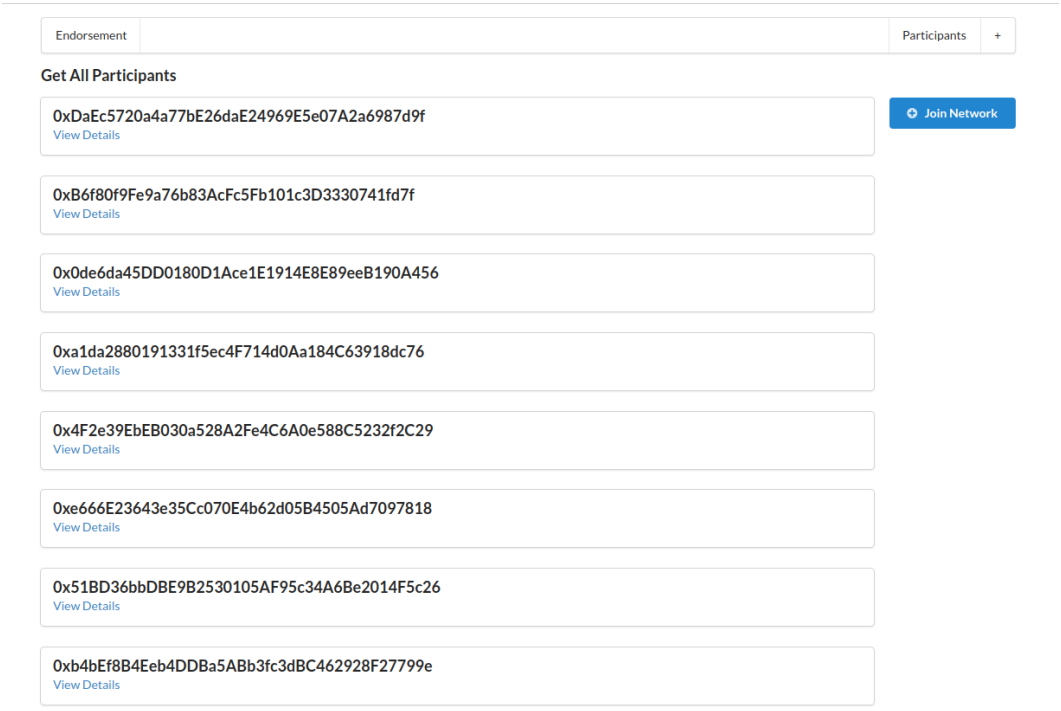

Figure 6.1: List of all participants

- 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.

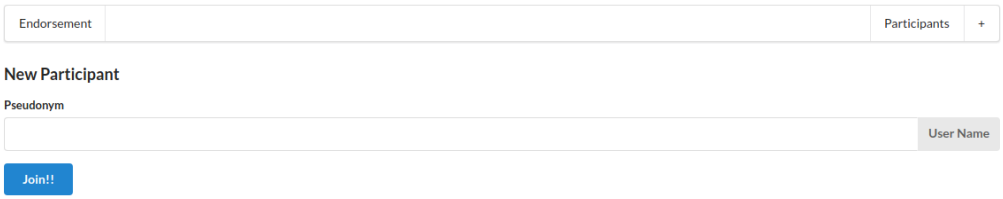

Figure 6.2: Join Network using Pseudonym

- 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.

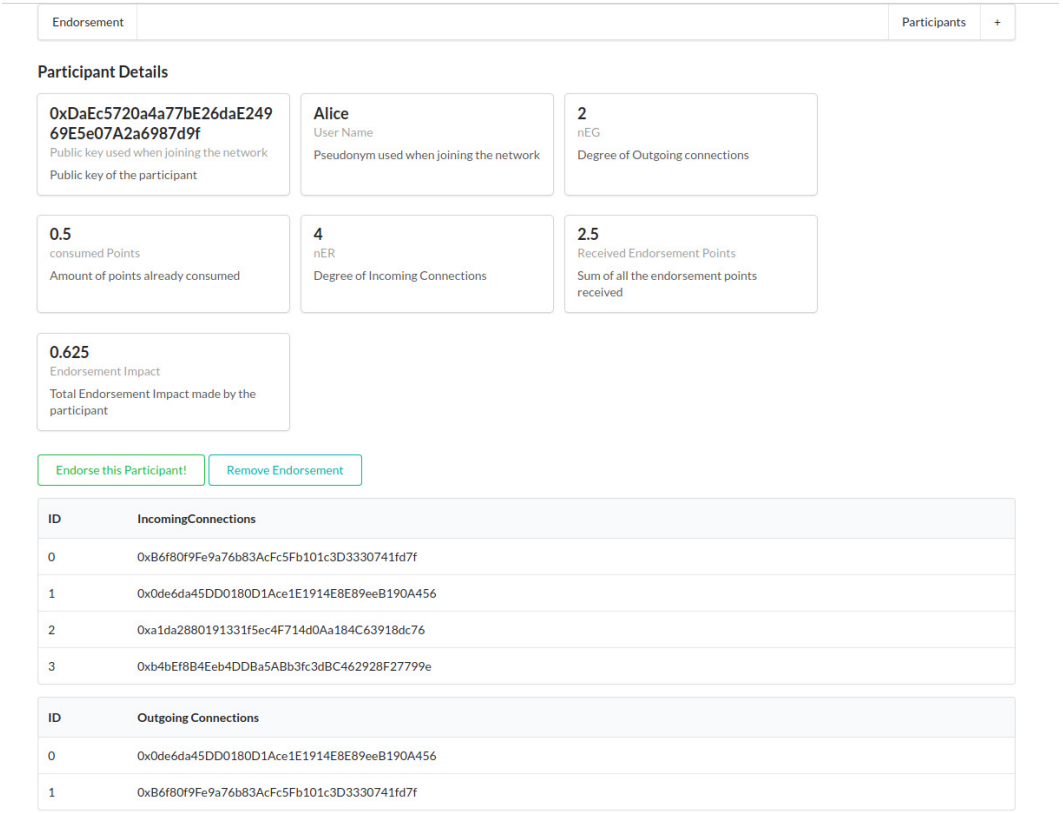

Figure 6.3: View Details of participants and endorse them

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

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

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

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

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

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

[7] 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.

[8] 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.

[9] L. Mui, M. Mohtashemi, and A. Halberstadt, "Notions of reputation in multi-agents systems: A review", in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, ACM, 2002, pp. 280–287.

[10] M. Carbone, M. Nielsen, and V. Sassone, "A formal model for trust in dynamic networks", in *Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on*, IEEE, 2003, pp. 54–61.

[11] 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.

[12] K. W. Miller, J. Voas, and P. Laplante, "In trust we trust", *Computer*, vol. 43, no. 10, pp. 85–87, 2010.

[13] M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

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

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

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

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

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

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

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

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

[30] 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.

[31] 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.

[32] A. Back *et al.*, "Hashcash-a denial of service counter-measure", 2002.

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

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

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

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

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

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

[39] 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.

[40] S. Brown, "Software architecture for developers", *Coding the Architecture*, 2013.

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

[42] 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.

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

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

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

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

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

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

[49] 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.

[50] 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.

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

[52] 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.

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     Endorser storage endorser = endorsers[msg.sender];
193     Endorsee storage endorsee = endorsees[_endorsee];
194
195     //proceed only if endorsee is in the endorser's list of endorsees
196     if (endorser.hasGivenTo[_endorsee]) {
197       endorser.hasGivenTo[_endorsee] = false;
198       endorser.nEG--;
199
200       //remove endorsee from endorser.givenTo array
201       endorsee.hasReceivedFrom[msg.sender] = false;
202       endorsee.nER--;
203
204       //remove endorser address from endorsee.receivedFrom array
205     }
206     return endorsers[msg.sender].index;
207   }
208
209   //computation of total received points of an endorsee
210   function computeReceivedPoints(address _endorsee) public view returns(uint)
          {
211
212     //get list of endorsers addresses from whom _endorsee has received eds
           from
213     address [] memory receivedFrom = getReceivedFrom(_endorsee);
214
215     //aggregate total received points from  the accumulated receivedFrom list
216     uint receivedPoints;
217
218     for (uint i=0; i<receivedFrom.length; i++) {
219       receivedPoints = receivedPoints + endorsers[receivedFrom[i]].usedPower;
220     }
221
222     return receivedPoints;
223   }
224
225   //computation of total endorsement impact of a participant
226   //the degree of connection should be strictly greater than 1 to be
        considered for
227   //impact computation, else, the impact by default should be ignorant, i.e.,
         0
228   function computeImpact(address _participant) public view returns (uint) {
229
230     uint nEG = endorsers[_participant].nEG;
231     uint nER = endorsees[_participant].nER;
232
233     uint _RE = computeReceivedPoints(_participant);
234
235     uint impact;
236     uint totalImpact;
237
238     if (nEG <=1 && nER <=1 ) {
```

```
239       impact = 0;
240       return impact;
241       //return impact and exit here
242     } else {
243
244       uint minval = min(nEG,nER);
245       uint maxval = max(nEG,nER);
246
247       uint ratio = Division(minval, maxval,9);
248       uint usedUpByParticipant = endorsers[_participant].usedPower;
249       uint RE = _RE;
250
251       impact = ratio * RE;
252     }
253
254     // call feedback function here
255     totalImpact = transactionFeedBack(_participant, impact);
256     return totalImpact;
257   }
258
259   //Receive feedback from Transaction Network and penalize the nodes
260   function transactionFeedBack(address _participant,
261                 uint _impact )
262                 public returns (uint) {
263
264     if (flagCount[_participant] >= 1) {
265       //Decrease the current impact by 50 %
266       uint res = Division(_impact,2,9);
267       uint penalty = _impact - res ;
268
269     } else {
270       penalty = _impact;
271
272     }
273
274     return penalty;
275   }
276
277   //Single function to get all the details of a registered participant
278   function getProfile(address _participant) public view returns (
279     uint,
280     uint,
281     address[],
282     uint,
283     uint,
284     address[] )
285     {
286
287     uint outDegree = endorsers[_participant].nEG;
288     uint usedPower = endorsers[_participant].usedPower;
289     address[] outConns  = endorsers[_participant].givenTo;
```

```
290
291     uint inDegree =  endorsees[_participant].nER;
292     uint receivedPoints = computeReceivedPoints(_participant);
293     address[] inConns   = endorsees[_participant].receivedFrom;
294
295     return (
296         outDegree,
297         usedPower,
298         outConns,
299
300         inDegree,
301         receivedPoints,
302         inConns
303     );
304   }
305
306   //get connections and degree of connections - helper function
307   function getConnections(address _participant) public view returns (
308     address [],
309     address []
310   ){
311
312     address [] inConns = endorsees[_participant].receivedFrom;
313     address [] outConns = endorsers[_participant].givenTo;
314
315     return (inConns, outConns);
316   }
317
318   //count total number of registered participants
319   function getCount( ) public view returns (uint) {
320
321     return numberOfParticipants;
322
323   }
324
325   //return array of all endorser accounts
326   function getEndorsers() view public returns (address []) {
327
328       return endorserAccts;
329
330   }
331
332   //return the total consumable power used by an endorser
333   function getUsedPower(address _endorser) view public returns(uint) {
334
335       return (endorsers[_endorser].usedPower);
336
337   }
338
339   //return list of addresses that an endorser has sent endorsement to
340   function getGivenTo(address _endorser) view public returns(address []) {
```

```
341         return (endorsers[_endorser].givenTo);
342     }
343
344     //return number of endorsees an endorser has sent endorsement to
345     function getGivenToCount(address _endorser ) view public returns (uint) {
346       return (endorsers[_endorser].givenTo).length;
347     }
348
349     //return a boolean value from the matrix of hasGivenTo, quick access for
            checking
350     //if an endorsee's address is in the list of endorsee addresses of the
            particular endorser.
351     function gethasGivenTo(address _endorser,
352                 address _endorsee) view public returns(bool) {
353         return (endorsers[_endorser].hasGivenTo[_endorsee]);
354     }
355
356     //return an array of all endorsee accounts - front end
357     function getEndorsees() view public returns (address []) {
358         return endorseeAccts;
359     }
360
361     //return address of all the endorsers for an endorsee, helper function to
362     //compute total received point
363     function getReceivedFrom(address _endorsee) view public returns(address [])
            {
364         return (endorsees[_endorsee].receivedFrom);
365     }
366
367     //count total number of endorser for an address of endorsee
368     function getReceivedFromCount(address _endorsee) view public returns (uint
            ) {
369       return (endorsees[_endorsee].receivedFrom).length;
370     }
371
372     //return a boolean value from the matrix of hasReceivedFrom, to check if
373     // an endorser's address is in the list of endorser address of the
            particular endorsee
374     function gethasReceivedFrom(address _endorser,
375                   address _endorsee) view public returns(bool) {
376         return (endorsees[_endorsee].hasReceivedFrom[_endorser]);
377     }
378
379
380     //some helper  functions for floating point calculation
381     function Division( uint _numerator,
382               uint _denominator,
383               uint _precision) internal pure returns (uint _quotient) {
384
385       uint numerator = _numerator * 10 ** (_precision + 1);
386       uint quotient = ((numerator / _denominator) + 5  ) / 10;
```

```
387
388     return (quotient);
389   }
390
391   //some helper maths function to compute max, min value.
392   //used for computing ratio and ensuring that the ratio is always less than
          1.
393   function max (uint x, uint y ) internal pure returns (uint) {
394     if (x < y) {
395       return y;
396     } else  {
397       return x;
398     }
399   }
400
401   function min (uint x, uint y ) internal pure returns (uint) {
402     if (x < y) {
403       return x;
404     } else {
405       return y;
406     }
407   }
408 }
```

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

```
1  pragma solidity ^0.4.18;
2
3  contract MarketPlace {
4
5    struct Product {
6      uint id;
7      address seller;
8      address buyer;
9      string name;
10     string description;
11     uint price;
12
13     //feedback rating from 0 - 5
14     //uint rating;
15   }
16
17   //store list of products accessible through uint key
18   mapping(uint => Product) public products;
19   uint productCounter;
20
21   //ratings
22   mapping(address => uint) public ratings;
23
24   //flags
25   mapping(address => bool) public flag;
26   mapping(address => uint) public flagCount;
27
28
29   //events - informative message stored in each ethereum block transaction
         log
30   //trigger events from contract to notify watchers that something just
         happened
31   //associated to address of contract that triggered it
32
33   event LogSellProduct(
```

```
34      uint indexed _id,
35      address indexed _seller,
36      string _name,
37      uint _price
38    );
39
40    event LogBuyProduct(
41      uint _productId,
42      address indexed _seller,
43      address indexed _buyer,
44      string _name,
45      uint _price
46    );
47
48
49    // sell product
50    function sellProduct(string _name, string _description, uint _price) public
          {
51
52      productCounter++;
53
54      //storing the product
55      products[productCounter] = Product (
56        productCounter,
57        msg.sender,
58        0x0,
59        _name,
60        _description,
61        _price
62      );
63
64      LogSellProduct(productCounter, msg.sender, _name, _price);
65    }
66
67    //get number of products
68    function getNumberOfProducts(  ) public view returns (uint ) {
69      return productCounter;
70    }
71
72    //get product that has not been sold yet
73    function getProductsForSale( ) public view returns (uint [] ) {
74      //productIds cannot have more identifiers than number we have in contract
75      uint [] memory productIds = new uint[] (productCounter);
76
77      uint numberOfProductsForSale = 0;
78
79      for (uint i=1;i <= productCounter; i++ ) {
80        if(products[i].buyer == 0x0){
81          productIds[numberOfProductsForSale]  = products[i].id;
82          numberOfProductsForSale++;
83        }
```

```
84      }
85
86      //copy productIds array into smaller for sale array
87      uint[] memory forSale =  new uint[] (numberOfProductsForSale );
88
89      for(uint j=1; j<= numberOfProductsForSale; j++ ) {
90        forSale[j] = productIds[j];
91      }
92
93      return forSale;
94    }
95
96    function buyProduct(uint _productId ) payable public {
97      //at least one product exists
98      require(productCounter > 0  );
99
100     require(_productId > 0 && _productId <= productCounter);
101
102     //retrieve product from mapping
103     Product storage product = products[_productId];
104
105     require(product.buyer == 0x0);
106
107     require(msg.sender != product.seller);
108     require(msg.value == product.price);
109
110     product.buyer = msg.sender;
111     require(product.seller != 0x0);
112     product.seller.transfer(msg.value);
113
114     LogBuyProduct(_productId, product.seller, product.buyer, product.name,
          product.price );
115   }
116
117   //leave rating for the seller as a buyer
118   function leaveRating(uint _productId, uint _rating ) public {
119     //only buyer can rate the product
120     require(msg.sender == products[_productId].buyer);
121     require(msg.sender != products[_productId].seller);
122     require(_rating >= 0 && _rating <= 5);
123
124     ratings[products[_productId].seller] = _rating;
125   }
126
127   function flagSeller(uint _productId ) public {
128     address seller = products[_productId].seller;
129     address buyer = products[_productId].buyer;
130
131     require(msg.sender == buyer );
132
133     flag[seller] = true;
```

```
134     flagCount[seller] == flagCount[seller]++;
135
136     //if flagCount is more than 5, eliminate the seller
137   }
138
139 }
```

Listing A.4: MarketPlace 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');
```

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

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


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

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

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

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

Listing B.3: Compile script

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

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

const deploy = async (accountNumber = 0) => {
  const accounts = await web3.eth.getAccounts();
  const deployAccount = accounts[accountNumber];
  const data = compiledEds.bytecode;
  const gas = 4000000;
  const gasPrice = web3.utils.toWei('2','gwei');

  console.log('Attempting to deploy from account', deployAccount);

  const result = await new web3.eth.Contract(JSON.parse(compiledEds.interface
      ) )
  .deploy({
    data
  })
  .send({
    gas,
```

```
    gasPrice,
    from: deployAccount
  });

  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>
```

```
35      return (
36        <Layout>
37        <div>
38          <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}>
```

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

export default ParticipantShow;
```

Listing C.3: Show all details of Participants

Application Components:

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

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

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

```
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" >
27            +
28          </a>
29        </Link>
30
31        </Menu.Menu>
32      </Menu>
33
34    );
35
36  };
```

Listing C.4: Header

```
1  import React from 'react';
2  import { Container } from 'semantic-ui-react';
3  import Head from 'next/head';
4  import Header from './Header';
5
6  export default (props ) => {
7    return (
8      <Container>
9        <Head>
10       <link
11         rel="stylesheet"
12         href="//cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.12/semantic.
               min.css">
13       </link>
14       </Head>
15
16       <Header />
17       {props.children}
18     </Container>
19
20   )
21
22 }
```

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(() => {
```

```
13    createServer(handler).listen(3000, (err ) => {
14      if (err) throw err;
15      console.log('Ready on localhost:3000');
16    });
17  });
```

Listing C.8: Server

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

Listing C.9: Routes