A Blockchain-based system for credible Reputation

Moiz Raza Amir (24100296@lums.edu.pk)
Department of Computer Science, Lahore University of Management Sciences

Abstract

The increasing popularity of online review systems has made it easier for consumers to make informed decisions about the products and services they purchase. However, the ease with which these systems can be gamed through the use of fake reviews has led to a proliferation of dishonest and misleading content. In this research paper, we propose a blockchain-based solution for combating fake reviews. By leveraging the decentralized and immutable nature of blockchain technology, our solution allows for the creation of a secure and transparent review system that is resistant to tampering, and manipulation and can deal with Sybil attacks better than the current centralized review and reputation-based systems. We provide a detailed analysis of the potential benefits and limitations of using blockchain in this context, by implementing a smart contract-based solution that thrives on user ratings, participation, and consensus to reduce the credibility of fake reviews, encourage unbiased and authentic reviews and provide a fair rating to the service providers while keeping the computational cost on the blockhain low.

Introduction

Communication technologies have allowed users to exchange information and content easily. Due to this boom in mass communication, there has been a rise in user reviews and websites that are dedicated to processing and publishing these reviews. Business and eCommerce giants like Amazon, Airbnb and Trip Advisor are only a few examples that employ and encourage the use of reviews. For a lot of users, a helpful review can influence their opinion of a product or a service, so it is natural that the credibility of these posts comes under question. The reviewed score and the user reviews of service can sway people to either avoid it or try it out simply because of how highly ranked it is by others. So far, there is no way to quantifiably verify these reviews beyond restricting the ability to leave one until the purchase and use of a service or product by a customer can be verified. Despite verification efforts, these reviews and the score associated with them can be manipulated to present a false image to the customer. Ideally, the reviewers should provide unbiased and accurate reviews of service. However, that is not the case for most of the review systems where reviewers are biased, and paid by the service provider to provide fake reviews. In the past, websites like Yelp, Trip Advisor, and Airbnb have been accused of actions like changing the content of the reviews, deleting negative reviews, or adding fake ones to portray a false idea about the service. Most of the times these websites are not held accountable for such malicious actions either. This is a bi-product of these review systems being centralized, making them prone to manipulation and even attacks by hackers; all of which illustrate the constraints and vulnerabilities that come with the way current systems. Currently, there are no evaluation mechanisms for the reviewers in most of the rating systems. Furthermore, there is no transparency behind how reputation scores are calculated and the criteria these ranking algorithms are based on, only adding onto the ambiguity surrounding these reputation systems.

This paper is meant to address these issues with a proposed solution that employs the use of a reputation system based on a decentralized blockchain. Attributes of this system would include being able to trace the user behind the original review, determining if the reviewer is known to be malicious by the reviews they have provided in the past and an immutable, secure and decentralized database. The solution proposed in this built upon the use of Ethereum Network and smart contracts with the deployment of the Inter-Planetary File System which is a P2P file system trusted for being secure and reliable. Expanding on the use of the Ethereum Network, the proposed system would reward users to incentivize the act of leaving a review and interacting with other reviews. This approach would place an emphasis on participation while combining community-based and incentive-based cooperation into one failproof system. Once a user posts a review, it will be rated through upvotes and downvotes by other participants in the system after which the reviewer would be rewarded with discount tokes.This smart contract based solution would ensure a trustworthy, transparent and publicly verifiable decentralised review system. The rest of the paper is organized as follows.

Related Work

The idea of designing a reputation system over blockchain is not new. Several uniquely different approaches have been used to implement a transparent, publicly verifiable and efficient reputation system over a blockchain network. In this section we analyze some of the previous work done so far.

Lina.Review [10] is a hybrid architecture of Ethereum bridged with a private permissioned blockchain named as Lina Core. The hybrid architecture is used to store fully detailed review transactions on Lina Core while benefiting from the security provided by Ethereum but at the same time avoiding heavy transaction fees on and reducing the load on the public network. Lina's review system consists of four participants: Merchants, Advertisers, Helpers and Common users. Merchants are service providers who are required to register on the platform to receive reviews and Advertisers are businesses that wish to advertise their products. A Merchant can also be an Advertiser. Merchant and Advertisers are the main revenue providers of the platform. Helpers are a group of reviewers who must register by providing a CV and proof of

domain knowledge. The reviews from helpers are assumed to be credible and each Helper must submit a specific number of reviews in a month to maintain their status. Common users do not need to register but they can look up the reviews provided by Helpers and submit their own reviews. However, the core difference between Helpers and Common users is that common users do not receive the revenue generated by the platform only Helpers do. The limitations in this system can be seen by the fact that in order to share the platform's revenue, one has to submit a valid proof of their identity violating the anonymity of users. Moreover, merchants must go through the process of registration which can be slow, and it is possible that some merchants are not profitable enough to pay the platform fees. Another problem is that all rewards are distributed in a global currency token called LINA. LINA's price is highly volatile which may rise or fall monetarily providing unreliable source of income to the Helpers.

Another currently active reviewing platform utilizing blockchain is Revain [11]. Unlike Lina, Revain is built completely on Ethereum and have two native tokens RVN and REV. RVN is pegged to dollar and is used to reward and penalize users whereas REV is a utility token utilized by companies to advertise and encourage reviews. Reviews are filtered off-chain through a Review Automatic Filtering system to detect spam reviews. After passing through this filter, reviews are sent to the company to accept or object to publishing a review, and in case of a dispute, moderators are assigned who investigate and can penalize companies or authors for malicious activity. One problem with this solution is that the process requires manual checking, which is slow and sometimes inaccurate. Since the review rejection process is done off-chain, it might lead to unfair rejections which are not publicly verifiable. Assigning moderators to accept the review defeats the purpose of decentralizing the reviews on the blockchain.

A smart contract-based solution built on Ethereum is proposed in [1]. This solution provides a smart contract that is uploaded by every service provider where users can provide reviews. The reviews are stored on IPFS, and reviewers are rewarded. This is a simple solution relative to Lina and Revain however, the authors do not go in much detail about how the reviews are filtered and how malicious users are detected. Our design builds upon previous blockchain and smart contract-based solutions to provide a system tolerant to malicious attacks, along with providing fair ratings for service providers and reviewers.

System Design and Implementation:

Main contract

In this section we present the main architecture of our review system. Our system comprises of three contracts. First, there is one main contract which is manages all the other contracts that service providers control. The main contract is responsible for creating, and keeping track of all the functioning service provider contracts. Main contract has several functions which helps in providing this functionality. The *registerFunction()* is called when a service provider, like a restaurant owner, wants to become a part of the rating system. The amount, name and symbol of the tokens to be minted are passed as parameters. This is a public function and any EOA can call this to register itself in the rating system. This function deploys a *serviceProvider* contract with *msg.sender* as the owner of the contract. The main contract maintains a map, called *activeContracts*, of currently active public addresses of service providers as keys. Each key has another map containing the address of deployed contract of the service provider as keys, and the amount of ETH submitted when they mint new tokens as the value.

Each user's reputation is crucial for the system as it shows how reliable the review provided by them is and is used to reward the user. The main contract has the function *updateReputation()*, which is called by the user every time they want to update their reputation based on the latest activity. Since reputation is dependent on user reviews, number of upvotes and downvotes, updating the reputation each time there's a change in any of these variables could consume a lot of gas. Hence, the user may call this function only once before posting a review, so he is rewarded based on their updated reputation. The reputation function uses a weighted average method to compute the reputation of the reviewer. In the implementation of this function, we have used percentages as weights. Since Solidity does not support floating point integers, we have used Basis Points. Basis Points are a well-known measurement unit in finance where 1 basis point is one-hundredth of 1 percentage point. Percentage is calculated in the *calculate*() function with basis points and then used to update the reputation. Further details of updating reputation are mentioned in this paper's later section.

Database operations

As the number of EOAs which interact with our system grows, it gets difficult to keep track of every user's score and status in one contract as that would take a lot of storage. To cater to this problem, our design uses Inter-Planetary File System (IPFS) as a decentralized database. IPFS uses a decentralized approach to file sharing, meaning no central authority or server controls the distribution of content [5]. It uses content-based addressing where each file can be retrieved by using its hash called content identifier (CID). A separate contract named *dbHandler* is deployed to deal with all the functions involving retrieving or storing user data or uploading reviews. This contract provides an interface for our system to perform operations like querying, increasing the reputation of users, the number of reviews, and upvotes or downvotes of an address. The purpose of this contract is to store the content identifier (CID) of the data stored on IPFS and provide functions which are called by the service provider and main contract to interact with the database. The connection between blockchain and IPFS is established through a NodeJs script. We use 'Infura', which provides an API to connect to the Ethereum blockchain network. Solidity allows the users to emit Events from the smart con-

tract, which are broadcasted the to all nodes in the public Ethereum network. In our system, we use Events to broadcast all the database operations. Through Infura and web3.js library, these events are detected by NodeJs script, allowing us to make corresponding changes to our database. The connection with IPFS is established through the 'ipfs-http-client' library, which allows us to add the data from the emitted event to IPFS. The updated hash is then stored in the database contract through 'set' function calls.

Service Provider's contract

Each service provider is assigned a contract that allows them to make payments, get reviews and reward users. The service provider contract is deployed through the *registerFunction*(), owned by the EOA, which called the function. There is a mint function call in the constructor of contract, with parameters that include the details to mint new ERC20 tokens for this service provider. This function interacts with OpenZeppelin library to mint ERC20 tokens. Since each service provider will have its own ERC20 token to reward its customers and offer discounts, the value of these tokens must be backed by ETH. While registering in the Main contract, the service provider specifies the number of tokens to be minted and sends ETH with their transaction. The value of each token can then be calculated by dividing the number of tokens minted by the amount of ETH submitted. The ETH is then stored as a balance of the service provider in the *activeContracts* map. This ensures the amount of ETH is locked by the main contract, and the service provider cannot issue tokens without any value. The value of each token the service provider provides is essential as that can significantly affect how the users behave and interact with the system. The reviewers are incentivised to provide honest reviews about a service provider, so their reputation increases, and they are rewarded with tokens. The token value must be enough to attract users to use the service and write a review. However, the service provider must carefully set the exchange rate of the tokens so there is no overall loss when users are rewarded for their reviews. The exchange rate is calculated by dividing the total ETH deposited in the main contract at the time of the creation of tokens by the total amount of tokens the service provider wants to create. The customers who own the tokens provided by the service provider could be given the option to withdraw the equivalent amount of ETH based on the exchange rate from the main contract as their reward.

This contract has a *payment*() function, which is used once a consumer uses the service and wants to pay on the blockchain. This is a payable function which receives the payment in terms of ETH and has a parameter indicating how many discount tokens the customer wants to use. If the consumer has the tokens, the discount is applied, and the remaining ETH is transferred back to the consumer's account and ERC20 tokens to the owner's wallet. When a customer has made the payment, *allowReview*() function is called by the contract which will grant the permission to post a review. This ensures that the consumers who have actually used the service are allowed to post a review. An array of *current_customers* is maintained by the contract. For example, in a restaurant, there may be several customers at one time. If a customer is eligible to post a review, its address is added to that array. When the consumer decides to post a review, it calls the *postReview*() and enters the review. This function checks if the address of the consumer is present in the array it removes it to ensure same consumer does not post multiple reviews. This function then interacts with the *dbHandler* contract to upload the review on IPFS. A function call in the *dbHandler* fires an event with the address of the user and service provider's contract, the review and the unique id of the review. After the review is uploaded the user is rewarded based on its current reputation by calling the *rewardUser*() function. This function receives the user reputation as a parameter. For rewarding the user, a simple tier-based rewarding system can be assigned where having a reputation above a specific threshold puts the address in a tier, and as the reputation increases, the consumer advances up the tiers. The higher the tier, the greater number of tokens that will be awarded to the user. This algorithm is shown in algorithm 1 where tiers and reward amount are determined by the user.

Algorithm 1 Reward User

$tier \leftarrow 0$
if $reputation > threshold_1$ **then**
  $tier \leftarrow 1$
end if
if $reputation > threshold_2$ **then**
  $tier \leftarrow 2$
end if
if $reputation > threshold_3$ **then**
  $tier \leftarrow 3$
end if
if $reputation > threshold_4$ **then**
  $tier \leftarrow 4$
end if
$reward \leftarrow getReward(tier)$
award *reward* **tokens to user**

The *upVote* and *downVote* functions are also part of the service provider contract. To upvote or downvote a review the user provides the unique id of the review as a parameter. This function interacts with *dbHandler* which queries the database to check if caller's address is in the database and if false it does not execute the command. This ensures that only those users who have used the service agree or disagree with the review. The database is then queried for the review with the provided unique id, checks the address which uploaded the review and increases the number of upvotes or downvotes for that address.

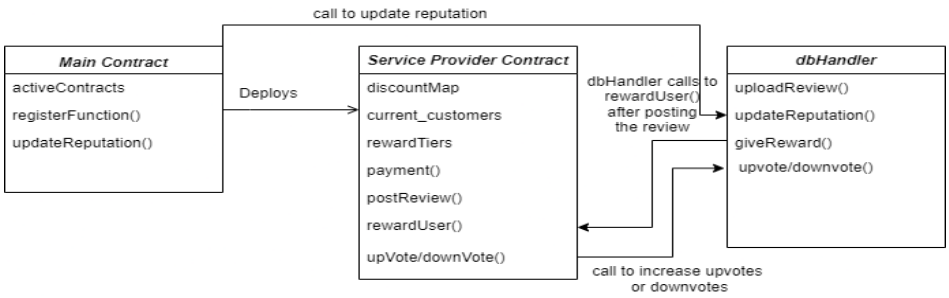

Figure 1: Workflow of three contracts

Workflow

To fully explain how the proposed design works assume the following scenario: there are three participants Ali, Alex, Bob and Bill. Ali is the owner of the system, so he deploys the main contract on the blockchain. Alex owns a restaurant, and he wishes to register himself to promote it, so people are incentivised to visit and review the restaurant. Alex calls the *registerFunction*() provides the token details and creates a new smart contract which assigns him as owner and mints the tokens. Details of the Alex's contract are added to the array of *activeContracts*, and Alex can now use his contract to receive payments and allow users to review his restaurant. Bob is a regular customer of Alex and wants his friend to try out this restaurant, so he transfers Bill some of the discount tokens which were rewarded to him. Bill visits Alex's restaurant for the first time and decides to pay through blockchain. Bill calls the payment function, and since he has the discount tokens, the discount is applied, the balance is transferred back to Bill, and the rest of the account tokens are transferred back to Alex. After seeing the payment has been made, Alex calls the *allowReview*() which makes Bill eligible to post a review and appends Bill's public address in the array of current_customers. Bill decides to post a positive review so he calls postReview(),with his review as the parameter of this function. After ensuring Bill is eligible to post a review, the review is uploaded to the database on IPFS and Bill is rewarded. Bob upvotes the review provided by Bill as he agrees with what Bill has posted. Upvoting adds credibility to Bill's review and increases his reputation in the system.

Evaluation

Functions	Cost in ETH	Cost in USD
Main contract deployment	0.008512	10.37
registerFunction	0.004751	5.79
updateReputation	0.00008459	0.10
Payment	0.0002446	0.30
Upvote/downvote	0.00007496	0.09
postReview	0.0001490	0.18
rewardingUser	0.00036	0.44

In this section we implement the basic prototype of proposed solution to compute the cost of main function calls in the system. The contracts were deployed and tested on Ethereum's Goerli test network. The computations done off-chain are not considered in this evaluation. The main operations which were tested include the deployment of Main contract, call to *registerFunction*() to deploy the service provider's contract and calls to the *updateReputation*() function, payment() function, *postReview*() function, *rewardUse*r() function and finally *upVote*() and *downVote*() functions. The results are shown in the table above.

Each function call was tested ten times and average cost computed in ETH and in U.S dollars is shown in the table. This conversion is done based on the ratio 1ETH : 1217 USD (price at the time of writing this paper). Although the gas fees on the public network are usually higher and are likely to increase as more users start interacting with the contract, the fees shown in the table gives us a general idea on how much the end users would have to pay in case. The most expensive operations are deploying the main contract, which also deploys the database contract, and deploying the service provider contract. However, the prices are reasonable given that these are only one-time costs which the owner of the system and the system provider would pay initially. The more frequently used operations which are called by the consumers are cheaper and cost only few cents. Posting the review and upvoting or downvoting costs the consumer only a few pennies and this cost is compensated when the consumers are rewarded the discount tokens. The reason behind these low costs is that these functions generally only interact with the database contract to emit an event, rest of the computation is done off-chain. This evaluation of the basic prototype of our system shows that the barrier to entry is very low and proposed solution can be used by small scale service providers.

Assigning Reputations

Several statistical and probabilistic algorithms have been proposed to for a fair rating mechanism. Although this problem is still being researched, weighted average methods are commonly used to compute the reputation of a consumer or a service. One of the sophisticated weighted average algorithm is proposed in [2] uses reviews leniency or strictness in ratings as the weights. Another approach mentioned in [8] gives an

iterative algorithm which uses prior successful reviews and belief propagation algorithms to evaluate the trustworthiness and reputation of the consumer. However, it is not scalable as the complexity of algorithm increases linearly with the number of raters. Finally, one of the superior and accurate approaches proposed in [9], uses a machine learning model along with weighted average approach to determine how reliable the rating provided by the consumer is. This approach takes in account the fluctuations, experience and reliability of the reviewer to provide precise ratings. However, such complex computations are not suitable in a smart contract deployed on Ethereum blockchain as the gas fees limit us. There is a chance a complex computation might fail entirely as the user might not provide enough gas for the transaction. There is no right or wrong solution for this and several current rating systems uses an already adopted approach by slightly modifying it as required by their domain.

Moreover, it is also seen that a simple weighted average approach can be compared to more complex approaches, as shown by the authors in [4]. Hence in our system, we use the weightage average approach to rank the reviewer's performance and differentiate them from malicious reviewers writing false reviews. Our database keeps track of the number of reviews provided and the number of upvotes and downvotes by each participant. We use these variables in the *updateReputation*() function and assign them weights to compute the reputation. The difference between number of upvotes and downvotes is assigned a weight of 70% and the number of reviews is assigned the weight of 30%. This function computes the difference and adds the result after multiplying the weights to give the result. For example, if the number of reviews are 2, number of upvotes are 8 and number of downvotes are 3, the result would be: 0.7*5+0.3*2 = 4.1. The reason behind the number of reviews being assigned the weight of 30% is that our system prefers quality over quantity. This ensures that the users who are posting false reviews are punished severely. This idea is similar to that of Reddit which uses a system of "karma" to help distinguish good content from bad. Karma is a numerical representation of how positively a user's content has been received by the Reddit community. It is calculated based on upvotes and downvotes on posts and comments. There is no single algorithm that determines how karma is calculated, but it is believed to be based on factors such as the age of a post, the number of upvotes or downvotes, and the overall engagement of the post. In our system, it is not possible to engage with the review, except through upvotes and downvotes, and it does not matter how old the review is. As long as the review is receiving upvotes and downvotes, credibility of the review can be determined.

So far, our system only provides an algorithm to rate the reviewers in the system. If a new customer would like to have an idea about the quality of services provided by the service provider, they would have to read other reviews. The intuition behind adding rating to the reviewers instead of the service providers was that the readers judge the review's credibility based on the reviewers' reputation. However, reading reviews provides more of a qualitative and subjective idea of the services provided. A numerical value representing the service provider's reputation will give a quantitative measure of what the majority thinks about the product. To compute the rating of the service provider we can use the similar approach which we used for computing the rating of the reviewer. The reputation of the user can be used as weights to compute a weighted average rating of the service provider. This ensures that the rating by the fake and unreliable reviewers are less likely to affect the overall rating of the service provider. The algorithm for this is given below. For example, if a restaurant has two reviews where customer A gives a rating of 8 and has a reputation of 26, and customer B gives a rating of 6 and has a reputation of 14. The total weight would be 40 (26 + 14). Then the weighted average rating would be 7.

Algorithm 2 Calculate Restaurant Rating

$totalWeight \leftarrow 0$
$weightedTotal \leftarrow 0$
for $i \leftarrow 1$ to n **do**
  $rating \leftarrow customerRatings_i$
  $reputation \leftarrow reputations_i$
  $totalWeight \leftarrow totalWeight + reputation$
  $weightedTotal \leftarrow weightedTotal + rating * reputation$
end for
$weightedAvg \leftarrow weightedTotal / totalWeight$
$mappedRating \leftarrow 10 * weightedAvg / 10$
return $mappedRating$

Threat Model

Sybil attacks: A known threat to any system which relies on user reviews to evaluate the quality of product or service is a Sybil attack. Sybil attack occurs when one user creates multiple fake identities to leave numerous fake reviews for one service provider to manipulate that service provider's rating. These attacks are common in a system where obtaining fake identities is easier. Although in a blockchain based review system getting a new identity only requires creating a new public address, our design is tolerant to sybil attacks to some extent as it assigns reputation to each reviewer. Every new address which interacts with our system will have an assigned score based on their performance and participation in the system. These scores can be used to evaluate how credible the reviewer is. For example, if service provider A tries to make fake accounts to leave a positive rating for their business in our system, the low reputation scores would show that the address is newly created and have started interacting with the system recently. Additionally, the malicious behavior of reviewers can be further confirmed by tracking their on-chain activity through their public address. If the recently created address has been used to interact with our system only or if the address is leaving same unoriginal reviews for every service provider, the address can be reported as malicious. In

case of Sybil attack, proposed design causes the fake identities to have minimum effect on the system. However, during initial stages when the number of participants is low and all participants are relatively new, a Sybil attack could be successful and severely affect the system.

Unauthorized reviewers: In several marketplaces, people may leave biased reviews for a service which before using the service. Imagine a scenario where a group of users are biased towards a competitor brand, so they start providing false negative reviews. This is the main reason why our design ensures that only the customers who have made the payment and have used the service are allowed to review. The *allowReview*() function helps us ensure this condition.

Malicious service providers: Many service providers try to game the reputation systems by providing high quality services initially to gain a higher rating and then degrading the quality over time either to save costs. Our system is capable of handling such cases as with degrading quality, reviewers will start posting negative reviews for the service provider. As the majority would agree with such reviews, negative reviews will start receiving more upvotes indicating that there is a change of opinion among consumers and the reputation of the service providers will eventually decrease over time.

Discussion, Challenges and Future Improvements

The main architectural design choice of this reputation system is also its limitation. Although building a reputation system on a blockchain, like Ethereum, provides public verifiability and eliminates any central authority changing reviews, it opens up other challenges like storing data and processing transactions on blockchain. The problem of storing data on blockchain is solved in our implementation through storing the data on a decentralized file system (IPFS). However, this choice comes at the cost of deploying a separate contract to deal with database operations and multiple transactions to that contract. This causes a large overhead in frequent database operations, which would have been relatively simple if a centralized database had been used. Due to high transaction frequency, gas fees can become expensive. Another potential problem is that processing transaction takes time on a blockchain like Ethereum. For example, in a restaurant during peak hours when there are large number of customers, the owner would want instant transactions to process all the payments quickly. One improvement could be deploying contracts on another EVM supporting chain like Avax or Fantom with high throughput and low transaction costs.

This reputation system adopts a community-based approach to maintain the credibility of reviews. Hence, user participation is necessary to prevent reviews from malicious users from affecting the service providers' ratings. To further incentivize participation, users can be rewarded for upvoting and downvoting the reviews. Due to the implementation of the rating function is crucial that the number of trustworthy active users remain higher than the non-malicious users. Otherwise, several malicious users together can portray a false reputation about the quality of services provided. Therefore, it would take some time before the proposed system becomes effective as driving traffic, allowing reviewers to provide their input so other participants can make informed decisions, and generating a fair reputation for the participants cannot be done instantly after deployment. To attract honest participants, the system owner can use promotional schemes like offering discount tokens of selected service providers to users. Moreover, developing an advanced system like this completely over blockchain in Solidity requires several design considerations. For example, since Solidity does not support floating point integers, we used basis points for the calculation of weights.

However, for complex computations like designing an algorithm for the reputation function, we were forced to keep the algorithm simple to keep the computational cost low. This forces us to ignore other factors which should be considered before giving a reputation score to the consumer and the service provider. For example, we are not considering how consumers tend to rate a service; for consumers, A 6/10 might be a very good score for a service provider, whereas the equivalent score for consumer B could be 8/10. Our algorithm fails to consider the scenario where one of the consumers with a higher reputation might turn malicious and start giving a service provider a false or negative rating. For this scenario, there must be a mechanism that measures the variance of the rating of the malicious user with previous credible ratings. One improvement that caters to such factors could be computing the reputations off-chain. This approach would give us the flexibility to run machine-learning algorithms, similar to the algorithm proposed in [9], to observe how each reviewer behaves in the system and provide a fair rating to the reviewer and the service provider. For public verifiability, the algorithm can be made public and open-sourced for further improvements.

One improvement to make the system more tolerant of Sybil attacks could be using decentralized identity identifiers to authenticate each user before allowing them to interact with the system. Digital identities are issued by a trusted central authority and are unique for everyone. They provide selective information for authentication while maintaining the privacy of that individual. Instead of allowing reviews from every public address, we can only allow authenticated public addresses to post reviews, hence reducing the frequency of Sybil attacks.

Conclusion

This paper proposes a decentralized credible review system on top of Ethereum, using smart contracts, IPFS and JavaScript libraries. We designed the system to be tolerant to malicious attacks by detecting malicious users while keeping the cost of operations as low as possible. For future research, a more sophisticated ranking algorithm could be implemented on or off the blockchain for the system to allow fair rankings

and quick detection of fake reviews by malicious users. The goal should be to minimize the tradeoff between the cost of operations on blockchain, public verifiability and providing credible reviews. To make the system more accessible, frontend with seamless UI can be implemented which makes all the function calls less confusing and easier to call.

[1] Salah, K., et al. "A Blockchain-based System for Online Consumer Reviews." IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE, Apr. 2019, https://doi.org/10.1109/infcomw.2019.8845186.

[2] H. W. Lauw, E. P. Lim, and K. Wang, "Quality and leniency in online collaborative rating systems," ACM Trans. Web, vol. 6, no. 1, pp. 1–27.

[3] Lisi, Andrea, et al. "Rewarding Reviews With Tokens: An Ethereum-based Approach." Future Generation Computer Systems, vol. 120, Elsevier BV, July 2021, pp. 36–54. https://doi.org/10.1016/j.future.2021.02.003.

[4] M. McGlohon, N. Glance, Z. Reiter, Star quality: Aggregating reviews to rank products and merchants, in: Proceedings of the International AAAI Conference on Web and Social Media, Vol. 4, (1) 2010, pp. 114–121 (why simple average weighted approach is comparable to that of complex solutions)

[5] Ipfs paper J. Benet, "IPFS - Content Addressed, Versioned, P2P File System", arxiv.org, July 14th, 2014. [Online]. Available: https://arxiv.org/pdf/1407.3561.pdf.

[6] 2015 bitcoin review R. Dennis, G. Owen, Rep on the block: A next-generation reputation system based on the blockchain, in: 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), 2015, pp. 131–138

[7] Bharadwaj, Kamal K., and Mohammad Yahya H. Al-Shamri. "Fuzzy Computational Models for Trust and Reputation Systems." Electronic Commerce Research and Applications, vol. 8, no. 1, Elsevier BV, Jan. 2009, pp. 37–47. https://doi.org/10.1016/j.elerap.2008.08.001.

[8] Ayday, Erman, et al. "An Iterative Algorithm for Trust and Reputation Management." 2009 IEEE International Symposium on Information Theory, IEEE, June 2009, https://doi.org/10.1109/isit.2009.5205441.

[9] Alqwadri, Ahmad, et al. "Application of Machine Learning for Online Reputation Systems." International Journal of Automation and Computing, vol. 18, no. 3, Springer Science and Business Media LLC, Mar. 2021, pp. 492–502. https://doi.org/10.1007/s11633-020-1275-7.

[10] www.allcryptowhitepapers.com. "Lina Whitepaper." The Whitepaper Database, 24 Nov. 2020, www.allcryptowhitepapers.com/lina-whitepaper.

[11] Revain R Whitepapers - whitepaper.io. whitepaper.io/coin/revain.