M.S. THESIS

A Data Retailer Based Decentralized Oracle Design for Smart Contracts in Blockchain

블록체인 내 스마트 컨트랙트를 위한 데이터 리테일러 기반 분산형 오라클 디자인

FEBRUARY 2019

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Park Soobin

M.S. THESIS

A Data Retailer Based Decentralized Oracle Design for Smart Contracts in Blockchain

블록체인 내 스마트 컨트랙트를 위한 데이터 리테일러 기반 분산형 오라클 디자인

FEBRUARY 2019

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Park Soobin

A Data Retailer Based Decentralized Oracle Design for Smart Contracts in Blockchain

블록체인 내 스마트 컨트랙트를 위한 데이터 리테일러 기반 분산형 오라클 디자인

지도교수 권태경

이 논문을 공학석사학위논문으로 제출함

2018 년 11 월

서울대학교 대학원

컴퓨터 공학부

박수빈

박수빈의 석사학위논문을 인준함

2018 년 12 월

위 원 장	김종권	(인)
부위원장	권태경	(인)
위 원	전화숙	(인)

Abstract

A Data Retailer based Decentralized Oracle Design for Smart Contracts In Blockchain

Soobin Park
School of Computer Science Engineering
Collage of Engineering
The Graduate School
Seoul National University

Smart contracts, a program that enables the automated execution of terms from the pre-defined condition establishment, became a core component of distributed applications (Dapps) leveraging the decentralized blockchain technology. Although many researchers keep trying adopting them to solve problems in the centralized systems such as the single point of failure (SPOF) and the need for excessive trust, there are multiple technical challenges in the practical use of them.

Amongst the challenges, we focus on "the oracle problem" in this paper, which means how to feed external data with ensuring the trustworthiness to the smart contracts. There are two primary technical considerations: (1)how to make the smart contracts to access the external data (2)how can measure the reliability of the data. In this paper, we propose a useful oracle design and implementation with an investigation of target platforms and related works to analyze the technical requirements and available design choices.

The proposed design focuses on the trustworthiness of the data sources

by separating the roles of the data provider into "publisher" and "retailer". The former, the data sources, can gain rewards by publishing data on their networks, and the latter gains rewards by relaying the data to the target blockchain network. A reasonable incentive model motivates them to behave honestly to maintain the oracle stable. Consumer contracts can receive the resolved data from the multiple retailers by using the oracle deployed as an on-chain smart contract. In the case of data inconsistency, the proposed reputation-based data consensus protocol determines a value from the collected data pools. The selected retailers and publishers can gain rewards from the deposit pre-paid by the consumer contract, and the rejected ones get penalties.

The main goal of this design is providing sufficient high-quality data to the blockchain by relieving the data sources' burden of participating in blockchain networks of potential consumers. The plenty of data supply may decrease the data price, that leads to great inter-operability between blockchain systems and the rest of the Internet. We also implement the protocols on Ethereum network and prove the practicality of the proposal in cost and time aspects, while most of the precedent researches are stuck with the protocol design. We expect that the design to contribute to the practical use of Dapps as the first working decentralized oracle through the code optimization process.

Keywords: SNU, Computer Science Engineering, thesis

Student Number: 2017-21931

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Smart contracts are programs that enforce an execution of terms automatically under a pre-defined condition. The concept was initially proposed by Nick Szabo in 1997 [1], but could not be realized due to the lacks of necessary technologies. Ever since Satoshi Nakamoto presented the idea of Bitcoin in 2008 [17], they are typically executed on blockchains of which immutability property enables a trustless consensus. Leveraging the strong integrity assurance of events, smart contracts allow credible transactions without the trusted third parties which require an excessive trust to them and have a single point of failure (SPOF) threat. The advent of specialized platforms such as Ethereum [16] and EOS [13], smart contracts became a core component of distributed applications (Dapps).

However, despite the potential of Dapps, there have been some reported technical challenges in the practical use of them, e.g., lack of the transaction privacy [2], irreversible insecure contracts [6], and the corruptibility of data sources. Among the problems, we focus on "the oracle problem". As the most compelling applications of smart contracts (e.g., insurance, payment, and logistics applications) necessarily require access to data about real-world events, ensuring the data trustworthiness is a critical technical requirement for Dapps.

There are two primary technical considerations in the oracle design. The first one is building a data transfer channel between the network of data sources to another of smart contracts. The latter is a blockchain, while the former can be any off-chain network. Smart contracts cannot perform a self-triggering by themselves or access external network. Therefore, an oracle is required to perform the 'aggregation' and 'pushing' data to smart contracts.

The second concern is how the smart contracts measure the trustworthiness of external data. In a centralized approach, a smart contract developer can use a trusted third party service exists on the outside of the blockchain. However, it can be a single point of data corruption. The decentralized approach seems worthwhile in this sense, but it accompanies other complex design considerations for robustness such as data consensus mechanism and an incentive model enforcing the participants be honest. As the models should consider relationships among the multiple stakeholders with possible security threats, a set of built decentralized oracle protocols requires to be proven to work in an intended way.

In this paper, we introduce and report the technical requirements details and possible design choices for them from investigating the existing platforms and services. Then we propose a scalable design of the decentralized oracle, which runs as an on-chain smart contract. A new entity, called *a retailer*, plays the role of setting up a "channel" between a network of a data source and another of a consumer. The retailer seeks to relay the (external) data that can satisfy the consumer's data requirements. In this way, smart contracts become more practical, and the sources do not need to participate in multiple blockchains of potential consumers. We also formally analyze the total cost and expected processing time models to evaluate. Our implementation of the design and running simulation using the models shows that the design is worthwhile as the first practical decentralized oracle.

Chapter 2

Background

2.1 Smart Contract Platforms

2.1.1 Bitcoin

Bitcoin [17] is the first practical public blockchain network designed to serve as the public ledger for the asset transferring transactions. The basic idea of public blockchain is preventing illegal modification to digital records by maintaining them in multiple copies with integrity validation. To achieve this goal, Bitcoin is made up of three main components: transactions, consensus protocol, and the peer-to-peer (P2P) communication network [7]. Bitcoin network maintains a stable blockchain on these main components as followings.

- Participating nodes generate a transaction message to transfer their assets using an ad-hoc script (i.e., OPCODE).
- The message contains a sender's address and signature, receiver's address, and a locking script that only can be unlocked by the receiver.
- The transactions are propagated to peers along over the P2P communication network, then aggregated as a new block by each peer node.

- The decentralized peers make a consensus to decide a corresponding block to be the latest block of the blockchain using the consensus protocol (i.e., the Proof-of-Work protocol for Bitcoin).

In this scenario, the script verification can be considered as a kind of smart contract that allows the transactions which satisfy the signature verification (i.e., condition) to be accepted by any nodes (i.e., terms). A reliable asset transfer is carried out automatically since the blockchain guarantees the trustworthiness of input data. Bitcoin scripts are limited to perform a general task as smart contracts due to the functional restrictions. For example, it does not support the iterative loop or state storing to prevent the code from falling into an unexpected state (e.g., infinite loop). Though, the Bitcoin is the first implementation which eliminates the need of a trusted third party of the untrusted system entirely by introducing a concept of the smart contract. This invention has significantly leveraged the decentralized blockchain technology.

2.1.2 Ethereum

Motivated by the Bitcoin's technical potential but poor expandability, Vitalik Buterin et al. presented Ethereum [16] in 2014, a decentralized platform to run the custom-built smart contracts. The concept of "smart contracts" began to be used commonly since Ethereum uses the word. Currently, Ethereum is the largest eco-system of smart contracts built on the public blockchain.

The most significant progress in Ethereum is that they not only deal with transactions but also perform general computational tasks including iterative loop and storing the states. Therefore, developers became to make various types of Dapps for their purposes. Ethereum provides a built-in Turing-complete programming language "Solidity" that help the developers to handle the Ethereum specific events, e.g., account handling, message authentication, token transferring, and so on. Furthermore, there are several newly introduced concepts introduced by Ethereum. In the rest of this section, we introduce some essential

features among them since they significantly inspire the other succeeding smart contract platforms.

Accounts: In Ethereum, not only the users but also the smart contracts own accounts. The former is called a "externally owned account", and the latter is called an "contract account". When a developer deploys a smart contract on the Ethereum network, a contract account is assigned to the compiled binary code and stored in the Ethereum blockchain.

Message and State: Message in Ethereum is a similar concept of "transaction" in Bitcoin, but it is used for more general purposes as they can transfer additional data with or without the value transfer. Another important difference is that the contracts also can be senders or recipients of the message as they have accounts. Ethereum introduces a concept of "state" and "state transition" on these characteristics, which can be widely adopted to the general programming. Following features define the "state" in Ethereum.

- All accounts maintain a state such as Eth balance, variable states, and storage contents.
- Only the message sent to the account can trigger a transition of the state. The transition requires the corresponding amount of gas, which is relative to the required amount of the task for the transition.
- As messages are recorded in the Ethereum blockchain like Bitcoin's transactions, state transitions are immutable and credible.

Consequently, a message is a "trigger" to functions of smart contracts, and the transferred data is an input data to them. Contract-to-contract messaging is also allowed in Ethereum. It helps the modulized implementation of a complex system. In other words, the Ethereum smart contracts cannot trigger a

function of themselves and even cannot be triggered by an event occurred in the blockchain without a message delivery. Therefore, typical Ethereum Dapps are composed of two parts: the smart contract as a back-end core service, and the web-based front-end part that can listen to events and interact with the smart contract.

Ethereum virtual machine (EVM) EVM refers the whole Ethereum network which works to build a consensus for the final state defined as a descriptive tuple of all accounts like a state machine. Participating P2P nodes act like the Bitcoin nodes under the consensus protocol. They are incentivized by contributing to maintaining the network through the message propagation. In comparison to the traditional centralized server-client model, EVM can be regarded as a server infrastructure, but it provides free to maintain, non-stop, trustworthy execution environment which a centralized system cannot guarantee.

The biggest weakness of Ethereum as a platform is the limited scalability. Only 20-30 transactions (assumed to be value-transferring transactions without an additional data handling) are processed per second on the network currently. Ethereum foundation is preparing to change its core consensus mechanism from the PoW (proof of work) way to the PoS (proof of stake) way to improve the transaction throughput.

2.1.3 EOS

EOS.IO [13] was introduced in 2017 for scaling of the Dapp ecosystem. The major goal of EOS is making the development of Dapps easy by providing a set of scalable services and functions that Dapps can make use of. Being compared with Ethereum, it has two significant differences. The first is its consensus algorithm, a deligated proof of stake (DPOS) mechanism. This choice is for

enabling the commercial scalability handling 10,000-100,000 transactions per second.

The second difference is an economic model. In Ethereum, amount of gas is required to all network resource usage including messaging, storage, and computation power. Therefore, the users of Dapps should pay the fees as much as they use the applications. In contrast, EOS utilizes an ownership model, in which holding EOS tokens allows the contract operators (developers) a proportional share in the network resources. For example, if someone owns N% of the total EOS tokens, it has access to N% of network bandwidth to operate its application. Consequently, the entry barriers to the normal users in EOS is lower than theirs of Ethereum since they do not need to own enough stake, but developers have higher barriers to deploy and run their Dapps.

2.2 The Oracle Problem

Regardless of the platform dependencies, data trustworthiness is critical to smart contracts since data-related conditions trigger their executions which are irreversible. There are two kinds of data : (1) internal data published in the ledger, (2) external data coming from sources outside the blockchain. Transactions inside the blockchain are internal, which are credible since they are verifiable and trackable. The problem is how to ensure the trustworthiness of the external data.

This issue is called "the oracle problem", which means where the blockchain can get the feed of trustworthy external data from. The blockchain network run in a deterministic way so that the miner nodes should have the same view for an event. Since web-like data sources are usually not guaranteed to be consistent, a reliable oracle system to decide an exact data is necessary. It is quite complicated to design a set of the oracle protocols because multiple components in the off-chain and on-chain network operate the oracle without trust. Technical challenges exist on the all the steps of data feeding as followings.

- Data source: Are the data sources trustworthy? How to authenticate the trusted sources?
- Data delivery: How the obtained data can be delivered to the blockchain network securely? Is there any possibility of forgery or corruption during data transmission?
- Data reliability: How the smart contracts ensure the data is "fact" by themselves?

Without being connected to the off-chain real-world data, use of smart contracts is strongly restricted. In the following section, we introduce some related works to solve the oracle problem.

2.3 Related Works

2.3.1 Oraclize

Oraclize [8] is a representative centralized oracle for smart contracts which is currently working. Contracts willing to use Oraclize API need to send a query to Oraclize's contract on the same platform with paying the price corresponding to the data size and retrieval complexity. The destination of a data source and a ruleset to parse the response to get an exact part should be specified in the query string, e.g., URL with the structure of the XML/JSON, InterPlanetary File System (IPFS) address and file name. Once the Oraclize's system detects a requesting event emission, the server on the cloud (outside of the blockchain) performs the delegated data retrieval using its network capacity to access the sources. Then, the result is delivered to the requester contract via the call back interface implemented on it.

2.3.2 SchellingCoin

Vitalik Buterin proposed the concept of "SchellingCoin" as a decentralized data feeding protocol in Ethereum white paper [16]. It assumes that all the Ethereum nodes will try to report correct data so that they can be included in the honest group for a reward. Its median based value decision mechanism is not proved to be stable yet, but the basic concept inspires the following studies.

2.3.3 Augur

Augur [10] built a prediction market on the Ethereum blockchain that provides correct outcomes in a decentralized fashion. Users of Augur can participate in the market by betting to an expected result for a future event such as weather, political election, and growth of a company. The winners gain proper Ethereum tokens as prize money, and also earn some reputations tokens which are re-distributed from the losers'. Augur is valuable in that it is the first working example of a collective intelligence based data decision protocol on a blockchain network.

2.3.4 Witnet

Witnet is a design for decentralized oracle that is still under being implemented and tested. Witnet runs on a side chain with a native protocol token (called Wit), which miners and the witnesses earn by retrieving, attesting and delivering web contents for clients. A new entity "witness" is assigned to be delegated to do a data retrieving task from a specified web URL destination. Another new role in Witnet is "bridge" nodes which are the subset of the witness nodes and specialized in fulfilling the observation events on other blockchain networks and bring the result to Witnet network. Its consensus mechanism assigns the roles based on their reputation. Therefore, the participants can have more opportunity to earn the fees by gaining the higher reputation from behaving honestly.

Chapter 3

Design

From the investigation of target systems and the related works, the critical design considerations in oracle are categorized. In this chapter, we first present the organized design choices with trade-offs. On this basis, we propose a new decentralized oracle model with an overview.

3.1 Design Choices and Considerations

3.1.1 Centralized vs. Decentralized

The advantages of using the centralized oracle model is a competitive price and the relatively low system complexity. As the service flow requires only a few interactions between the contracts, processing fee is low. However, it requires an excessive trust to the oracle, to its infrastructure, and even to the data source (i.e., URL destination) since the data is not authenticable in the customer's side. Introducing a trusted hardware (e.g., Intel's Software Guard Extensions) to the oracle server can relieve this tampering threats by securing authentication [4]. Nevertheless, still there is a threat of a single point of failure (SPOF) or compromise that defeats the ultimate purpose of using Dapps. The

SPOF threat may defeats the blockchain philosophy.

A decentralized approach resolves the centralized trust issue, but more costly and complex since multiple entities participate in the oracle. There are different kinds of critical considerations for implementing a oracle in the decentralized fashion as listed below.

Participants and Roles

A decentralized oracle is built up with multiple participants. The Augur community is composed of the market creator and the reporters, and the Witnet network is composed of the clients, witnesses, and bridges. Roles and responsibilities of those participants significantly affects the overall system design regarding to security model, cost model, rewarding system, and the consensus design. From the nature of a decentralized network, the system will be more robust as more participants are involved since a few adversaries cannot easily harm the system. Though, the larger amount of fees and the more fine-grained distribution of them are required to a customer (the market creator in Augur and the clients in the Witnet) as much parties join the data resolution task.

Incentive and Punishment

Reasonable incentive and penalty are a core component of a decentralized oracle model that makes the oracle practical. It enforces the participants behave honestly. The fundamental security requirement of it is making the sum of the benefits from a fraud behavior must be less than a value he can gain from an honest behavior. In addition to an economic system, introducing the concept of reputation system is helping to prevent the predicted misbehaving. An adversary who keeps behaving to blame the system becomes to lose the opportunities to participate in the system gradually as they lose the reputation.

Consensus

Data consensus mechanism is used to determines the exact value of data from the multiple ones submitted by multiple parties (like in Augur). In other words, it is the consensus design that decides the honest and dishonest actors. Therefore, the mechanism must to be designed carefully with enough consideration of the incentive system. For the deterministic types of data such as 'true or false', 'yes or no', and noun type of data, a standard or weighted majority rule can be adapted as the Schelling coin propose. However, if the data content is a numeric type, the consensus algorithms can be implemented in the various ways: mean approach, median approach, voting based approach and so on.

3.1.2 Architecture

On-chain Architecture

Building an on-chain oracle application is the most straightforward way to implement an oracle system that demands the less attention to securing the untrusted communication channel between the environment of the blockchain network and the off-chain network. All the oracle-related events happen on-chain, thus the processes are transparent as they are trackable and verifiable. Furthermore, the system is easy to deploy since it does not require a modification to the platform protocols. One weakness of on-chain oracle is that it may be expensive as all the computation tasks and data handling require network resources which are quite costly.

Off-chain architecture

Off-chain architecture is more lightweight than the on-chain approach by delegating the tasks to the off-chain resources which are cheaper. However, the design necessarily needs the additional trust and security robustness to the off-chain infrastructures. The hybrid way to relieve the trust problem is a side-chain

approach which the Augur and the Witnet adopt. The side-chain approach is reasonable because it is a kind of off-loading model widely used in traditional computing resource management systems. In these case, it can be challenging to integrate the ad-hoc currency to the target platforms one by one. Additionally, as the participants gain the monetary rewards as the ad-hoc tokens assumed to be exchangeable to the major currencies, the economic value of the tokens must be maintained in a stable mood to prevent the participants leaving.

3.1.3 Service Scope

Service scope is defined by what methods the oracle supports to retrieve the data. If it only delivers data from the specified source such as URL and file system, a built-in data specifying grammar can be implemented in the protocol, e.g., the query specification of Oraclize and Witnet. The responsibility of the data submitter(s) is accessing the source, getting the data, and submitting it to the oracle. Augur like a market-based model can serve data from various sources. However, informing and understanding the exact market topic is critical, e.g., Augur users are actual human-beings who can understand the market title and join the poll. An oracle also can be specialized in a specific topic and provide a persistent service only for the use, e.g., an oracle for the real-time diesel price feeding. This model is different from the market-based model in that the latter is only valid during the market is running and only for the market creator. The data submitters who are interested in the specialized topic can concentrate on customers using the oracle.

3.2 Design Proposal

In this paper, we propose a new design and protocol for an on-chain decentralized oracle. we introduce a new entity, called *a retailer*, plays the role of setting up a "channel" between the external data sources and the smart contracts on blockchain network. The retailers take over the overhead to cross the network

border which is data providers suffers in existing models. The motivation here is that the data sources are considered as a critical participant in the proposed design; other existing systems do not consider them as a dependent entity and give no incentive to them. Attracting enough numbers of high-quality data sources is expected to invigorate this kind of data market, and lower the cost to use the data. In retailers' view, the design is compatible to existing models which of all nodes can freely submit data after getting it from the other sources (like the Witnet model) or creating their own opinions (like the Augur model). However, there are strong improvements for data providers and customers; (1) the data providers in real-world can more effectively provide the data to various blockchain networks for monetary reward (2) the data providers can easily gain proper reputation by revealing their identity in the blockchain via trusted off-chain channel (3) the customers (i.e., developers of Dapps) can specify a fine-grained data retrieval condition that improves the reliability. Consequently, the increase in high-quality data in oracle networks will lead to the decrease of the

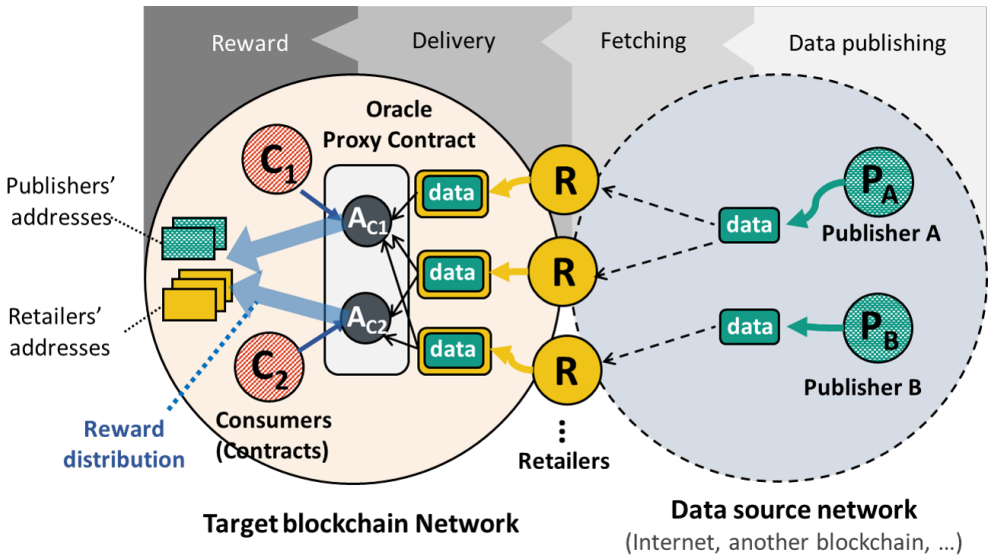

Figure 3.1 Data delivery from an off-chain network to consumers in the target blockchain over the retailers.

price.

Figure 3.1 illustrates how the proposed oracle model works. The data source can be any off-chain network, e.g., Internet, IoT, or another blockchain. The publishers, who publish data over their off-chain channels to one or more data retailers, need to create identities on the target blockchain network in advance to get the rewards. They are motivated by getting enough reward consists of economic gains and the reputation that is related to the future incomes. A consumer specifies the data requirements in her running smart contract. Then, the retailers try to fetch the suitable data from the publisher's off-chain channel. For this, the retailers must have the capability to access both of the on-chain and off-chain networks as they serve as a bridge. The data delivering transactions will be propagated through P2P connections to be verified by miners as same as the normal transactions (omitted in Figure 3.1). When the conditions (i.e., the above transactions) of a smart contract are fulfilled, the contract automatically distributes the reward to the publishers and retailers using their identity information attached to the data of interest.

Chapter 4

Protocol

4.1 Overview

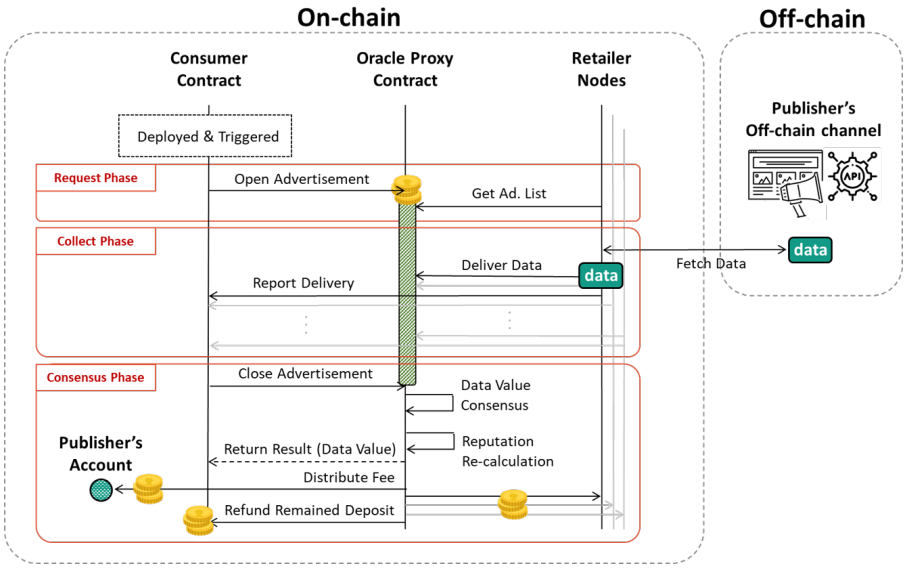

Figure 4.1 Protocol overview

Figure 4.1 illustrates the entire data feeding process with some of main protocols. The whole process is divided into three phases: request phase, delivery

phase, and consensus phase. Consumer contracts can initiate the data retrieval to oracle in the request phase. Then during the delivery phase, the retailer nodes deliver suitable data values that can be fetched from off-chain networks. The collected data will be used to figure out the correct answer in the consensus phase. From the result, the publishers and retailers of 'selected' data gain rewards, but ones of 'rejected' data get some penalties. More details of each phase are provided in the next sections.

4.1.1 Request phase

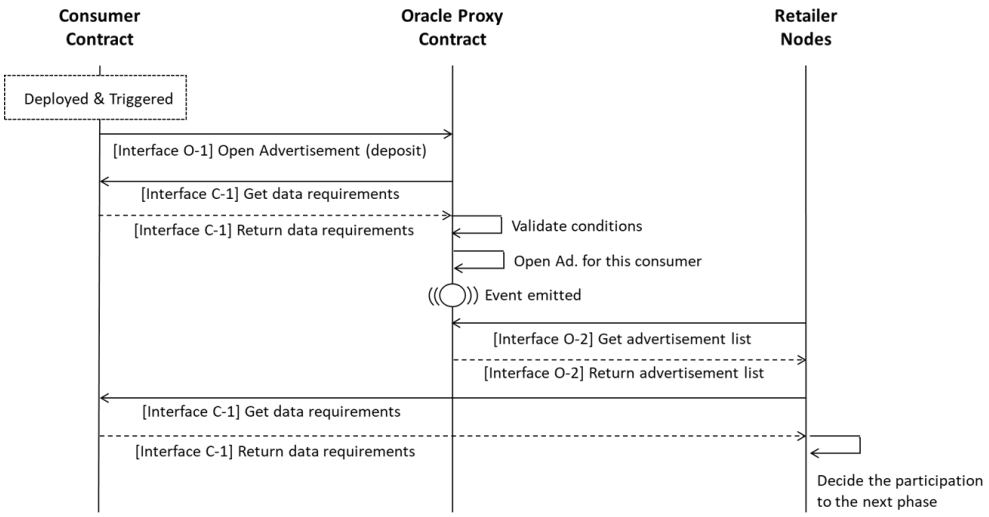

Figure 4.2 Flow of request phase

A series of interactions among participants in the request phase is illustrated in Figure 4.2. It is the first phase to retrieve external data from the consumer contract on the blockchain. The phase starts with a message from a consumer contract to the oracle contract that requests to open an advertisement for data delivery. Enough monetary deposit (i.e. cryptocurrency tokens) to create the advertisement must be enclosed in the message together. By getting the message, the oracle contract will try to fetch requirements of the consumer contract to decide whether it accepts the request or not. The consumer contract should

provide a defined callback interface for this. If the requirements make sense, a new advertisement will be included in an "opened ad" list which is accessible publicly. Retailers, who are expected to keep listening to or polling this event, can obtain the consumers' accounts and try to get the data requirements from them using the same interface that oracle used. If the retailers judge that they can deliver the satisfying data, they will participate in the next phase, a delivery phase. More details about the requirement statement and validation is discussed in 4.3.

4.1.2 Delivery Phase

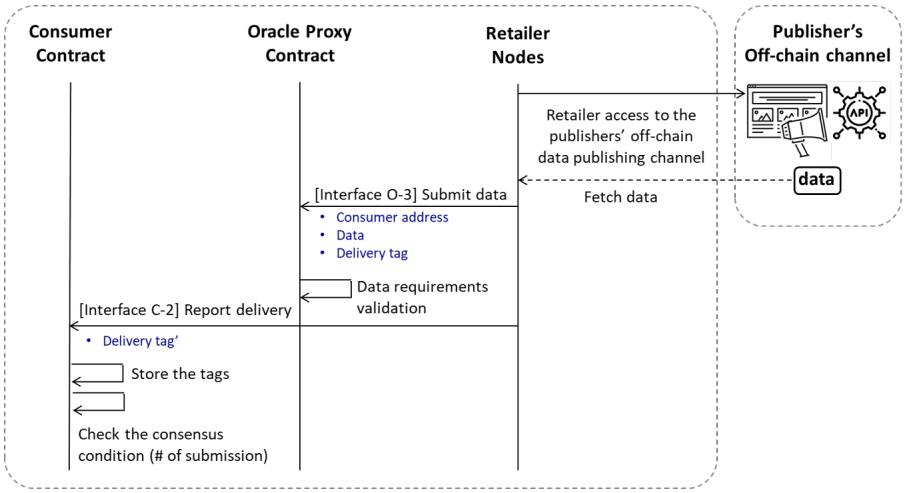

Figure 4.3 Flow of delivery phase

The retailer nodes in delivery phase have two choices: (1) accessing to a proper data publisher to fetch data through the provided channel in off-chain environment, e.g., open API, web page, or a trusted file storage (2) generating corresponding data by themselves. By choosing the second option, retailers can perform both roles of retailer and publisher since the roles are logically defined. Whichever option they choose, the fetched or created data should be formatted to defined structure which consists of two parts: a value part and an

authentication information part for the publisher. The format specification is described in 4.2.

Retailers can submit the data to the oracle contract with consumer address and the delivery tags via "interface O-3" in figure 4.3. The submission will be accepted if its format is valid and it satisfies the consumer data requirements. To report a successful delivery, retailers need to pass the tags to the consumer contracts after transforming it into an unguessable value along with the defined protocol described in section 4.4. The tags which additionally serves a kind of "delivery proof" should be collected in the consumer contracts and presented to oracle in the consensus phase. By introducing this deposit-based 2-way reporting process instead of direct data delivery to the consumer, the system can prevent dishonest consumers from false reporting to lower or snatch the fees of honest retailers after taking the data. Furthermore, in the implementation aspect, such design allows the consumer can decide when to terminate the delivery phase by stating it in itself. As the tag is designed to be only verifiable by the oracle with a message sender information, which is resistant to tampering in platform level, an adversary who only steals the data value from an observation cannot re-use it to incapacitate another retailer's work by replay attack.

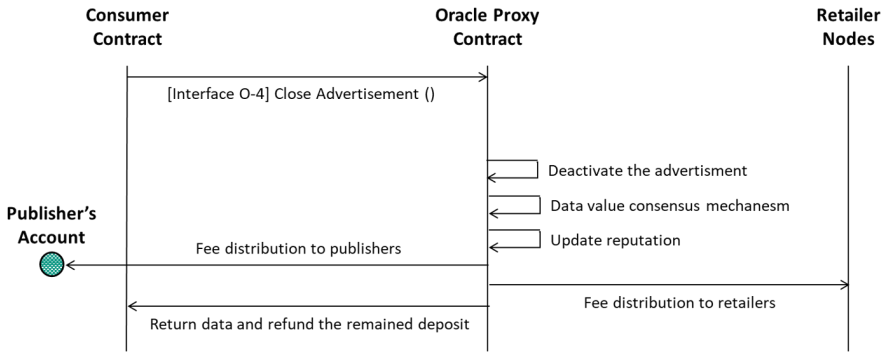

Figure 4.4 Flow of consensus phase

4.1.3 Consensus Phase

When a consumer's pre-defined conditions are fulfilled, the consumer can terminate the delivery phase and getting the finalized data value from the oracle contract. Collected tags (i.e., the converted tag from the original value) from the retailers should be presented to the oracle to close the advertisement and to start the consensus phase. Each tag is paired with a delivery information including publisher identity, retailer identity, and the data value. The oracle can load all the data after validating the tags using tag' and consumer account information.

As the types of the data for each oracle depend on its category, the oracle should use a proper consensus algorithm for its purpose. If an oracle is deployed to provide a diesel price of the United States, for example, the consensus algorithm must be able to handle the numeric types of data. The details of the consensus algorithm is addressed in section 4.6. During the consensus-making process, some of the deliveries are selected, but some of them are rejected to adopt the reports since the algorithm regards them as incorrect ones. For both cases, the oracle updates the reputation information at the end of this phase by granting incentives to the former and giving a penalty to the latter. The incentive and penalty are consist of a monetary reward and reputation as discussed in 4.5. After distributing the fees, the monetary rewards, amount of remained consumer's deposit is returned back to consumer's account. Accordingly, the consumer obtains the data which is estimated as a correct answer over the explained three phases.

4.2 Data Structure and Authentication

From this section to the end of this chapter, we provide the protocol details not covered in the overview section. As the first detail, the basic structure of data for publishing is described in this section. Data authentication is essential in

the proposed model since the roles of data publishing and delivering are separated. Publishing time information also should be included in the data object to validate its freshness if consumers want. For the reasons, a data publisher must include its identity information, a timestamp, and data value to publish. Then she can sign the data object using her private key related to her account address before publishing it[1]. An expected data structure in Ethereum platform is illustrated in Figure 4.5. When the signed data is fetched by the retailers and delivered to the oracle, the oracle can authenticate the publisher's identity using a built-in address verification function. The protocol only requires a little effort to since the most of the well-known platforms support the sign-and-verify process with inherent functions.

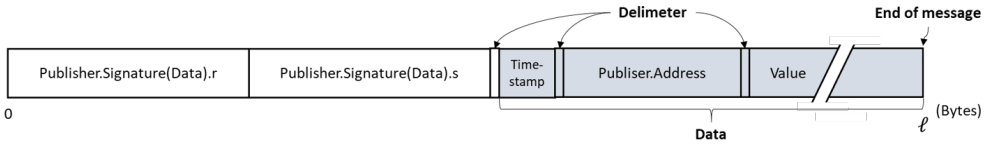

Figure 4.5 Data structure for publishing

4.3 Data Requirements

As explained in the overview section, consumers can specify and present some data requirements which are also used to state the restrictions for data collection. Available fields of data requirements are listed below.

- Blacklist: A list of addresses that a consumer does not want to retrieve the data from. The oracle blocks any data objects published by one of the list, and the retailers may not deliver them since their goal is earning the reward.

- Whitelist: A list of addresses that are only allowed by the consumer to retrieve the data from. If the publisher of the delivered data object is not

[1]In Ethereum, the address refers the first some bytes of the account's public key.

included in the list, the delivery will be rejected.

- Time condition: A time requirement which is represented in a timestamp format. A data object including an unreasonable timestamp (i.e., later than the current timestamp obtained by the platform) or a timestamp that not satisfying the freshness will be rejected.

- Source diversity: If the consumer wants to retrieve data from multiple publishers for cross-checking, it can specify this field in their requirements. It will be useful to mention this field with the whitelist feature to guarantee that the data is coming from the multiple trusted sources.

Data requirements specification helps the proposed model work in an intended way in following two aspects: (1) the decentralized oracle system can be maintained by the law of supply and demand as a kind of market for information (2) the consumers can explicitly block some publishers who are not preferred or known as dishonest. This feature grants an advantage to the well-known publishers who may gain a higher reputation in the real world as they will can be referred much more times than the new entities. Since the publisher reputation model is related to the recent number of successful delivery activities, the publishers can get a higher reputation in this market. This is a very intentional direction for the oracle design as the publishers will be incentivized to participate in the oracle, and they are expected to provide great quality of data.

4.4 Data Delivery Tag

The delivery tag at [Interface O-3] and [Interface C-2] in Figure 4.3 is applied as a delivery proof in the two-way delivery system, that is comparable to a check in a traditional banking system. Figure 4.6 illustrates how the tag works in data delivery process.

1. To generate a tag value, a retailer should make a “secret salt”, an unguessable random value.

2. By hashing a string from concatenating the salt value and the retailer’s address, a tag prime is generated.

3. Then, he can obtain the original tag by hashing the tag prime value after concatenating it with the consumer’s address.

4. This tag is passed to the oracle in data delivery through [Interface O-3].

5. The tag prime is passed to the consumer for noticing the delivery through [Interface C-2].

6. All the tag primes from retailers are collected by the consumer and submitted to the oracle when she closes the advertisement.

7. Then the oracle can reproduce the tag value by hashing the tag primes with the consumer’s address.

8. Finally, this tag value is used as a key for loading a mapped data object from its data container.

As the oracle regenerate the value by referring the consumer authentication information, only authorized entity can access to the data store to retrieve it.

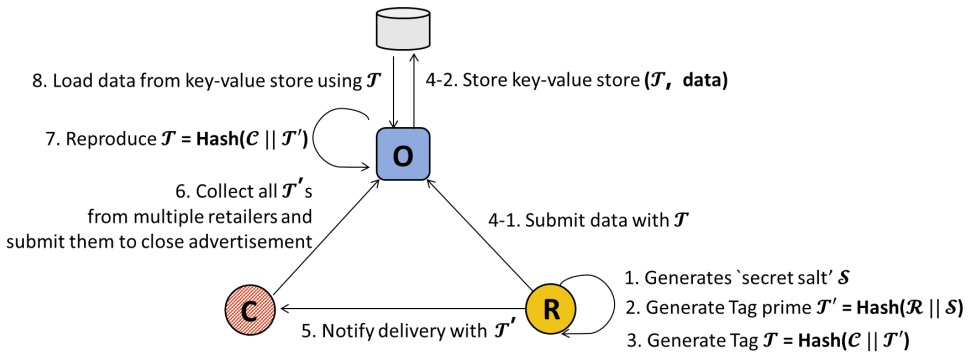

Figure 4.6 Data tag verification during data delivery process

4.5 Incentive and Penalty

Fee as a monetary reward

The first incentive for the contributors to the oracle system is a monetary reward, commonly called fee. The publishers gain the "publishing fee", and the retailers gain the "sales fee" as a reward. The amount of publishing fee may be smaller than the sales fee since the publisher can collect fees from the multiple deliveries by other retailers. The portions of publishing fees for each selected publisher are related to the amount of reputation each publisher has. The sales fee also must be bigger than the sum of the messaging fees that the retailers spend during the delivery phase in advance. The messaging fees are decided by the message size (i.e., data size) and the network status if the platform adopts a competitive fee model for transaction processing. Even for the same size of data, the amount of fees can be various as the price of the platform's token may fluctuate. Thus, the oracle should consider the exact prepaid price. For the penalty, exclusion of the incorrect data contributors from the rewarding pool serves as a penalty for the retailers since they will lose the prepaid expenses. Thus, maintaining reliable and various fetching channels is crucial for retailers. If the delivered data is not adopted by the oracle, they cannot collect their investment.

In the consumer's view, data consumption should be worth more than the sum of the sales fee and the publishing fee.

Reputation

In addition to the monetary reward, a notion of the publisher reputation is one of the design principles that encourage the publishers to report reliable data. Proposed reputation system estimates each publisher's reputation based on his activities with three factors: the total number of submission during recent N advertisement, the number of acceptance, and the number of rejection. Reputation value calculation formula and notations are provided in Figure 4.7.

The first term of the right side reflects the data reliability of this publisher. If all the data from this publisher were accepted, the result of this term always becomes 1. The second term of the right side indicates how much the effective deliveries happened during N of recent advertisements. The number of acceptance is only counted during recent N times, but the number of rejected submissions per a advertisement is counted and accumulated from the first time of oracle deployed. Therefore, a reputation value of publisher whose data have been rejected more than threshold is converged to a low value. By controlling the constant K, the oracle deployer can coordinate the balance between the reliability and activeness. A publisher fully selected during the recent N advertisement may have K+1 times of reputation than a publisher who is just joined.

In our design, each oracle contract maintains own reputation status to be used for data consensus computation. This is from a fundamental idea that the reliability is only valid in its service field. A similar approach is found in an existing knowledge system that are comparable to a decentralized oracle for humans beings, Quora [12]. A concept of reputation system in Quora estimates how much the answerer is specialized for each field, the question category.

$$\blacksquare\ Reputation = \left(\overbrace{\frac{Accepted}{Total + Rejected}}^{Reliability} + \mathrm{K} \times \overbrace{\frac{Accepted - Rejected}{N}}^{Activeness} \right) \times (2 \times N)$$

Total: The number of total advertisements the publisher joined during recent N advertisements

Accepted: The number of data acceptation in total participations

Rejected: The accumulated number of rejections

N: The window size of valid recent advertisements

K: Weight to the recent activeness

Figure 4.7 Reputation calculating formula

4.6 Data Consensus

We adopt a weighted majority-based consensus model to figure out the result value. The reputation points are used as the weight of a publisher, the owner of the reputation. By controlling the value of K, the oracle can decide how much more weight it will give to the "active" publishers than the newcomers.

For the data as non-deterministic values, the numerics, we introduce a new approach named clustering-based mean calculation. As our model run on the on-chain environment, the median approach requires a sorting process which requires much more computation resource in the blockchain network. Therefore, we are motivated to mean based approach with maintaining the rule of excluding out-of-range answers from the rewarding. The clustering-based mean calculation uses a threshold value to group the similar submitted values into the same cluster. The cluster core value is newly calculated in every time a new member is added. The sum of the weight of publishers whose data is in the same cluster is used to select the most trustworthy cluster. As a result, who deliver the value close to the most influential publishers' data have a high probability to be rewarded.

Chapter 5

Proof of Concept Implementation

We implemented an example oracle on the proposed protocol as an Ethereum smart contract to prove the concept. We suppose that the oracle is designed to provide the Korean-won to US-dollar exchange rate (won per 1 dollar), which is a numeric type of data. By implementing a working example on the Ethereum, the most widely used smart contract platform, we proved that the feasibility of the proposed model. Furthermore, we evaluate the practicality of the model in two aspects, the first one is a reasonable total cost that should be paid by a consumer, and the second is the total processing time.

5.1 Code

The code is written in the Solidity language and under 500 lines including four interfaces, sixteen of helper functions, and one abstract contract for the consumer contracts to implement to use the oracle. Figure 5.1 shows the variables of the contract commonly used as the memory or storage spaces. As changing the variables' state induces gas consumption, maintaining minimum spaces is one of the implementation goals. 'AdResult', 'PulblisherActivity', 'DataRequirement', 'DataPacket' refer the custom structures to handle the appropriate

data. There are relevant comments for each variable on the figure. The pseudocode of interface functions are described in figure 5.2 and figure 5.3.

Memory/Storage on contract

- address[] public **advertisers** // A list of consumers who open advertisements
- uint256 **windowPointer** // A pointer to the oldest advertisement
- AdResult[windowSize] **adResults** // windowSize of recent advertisement results
- mapping(address => PublisherActivity) public **recentActivities** // Publishers' recent activities
- mapping(address => DataRequirement) public **requirements** // Data requirement of consumer retrieved at opening
- mapping(address => uint256) public **deposit** // Amount of deposit each consumer put
- mapping(address => uint) public **results** // Query results
- mapping(bytes32 => DataPacket) public **delivery** // Data delivery pool having delivery tag as a key

Figure 5.1 Memory/storage variables on the contract

Interface O-1: openAdvertisement() payable

```
Consumer _consumer = Consumer(msg.sender);
requirements.push(consumer.getRequirements());
advertisers.push(msg.sender);
deposit[msg.sender] = msg.value;
return index;
```

Interface O-2: getAdvertisements() view returns (address[])

```
return advertisers;
```

Interface O-3: submitData(consumerAddress, Data, Proof)

```
publisherAddress = verifyDataSignature(data);
Authenticate(publisherAddress, data);
Filter(requirement.blacklist, requirement.whitelist, publisherAddress);
delivery.push(this);
```

Figure 5.2 Pseudocode of oracle interfaces(1/2)

Interface O-4: closeAdvertisement(Proofs)

```
tags = genTags(msg.sender, proofs);
value = consensusForNumeric(tags);
    for ( tag of tags) {
        reputation = getReputation(delivery[tag].publisher);
        value = deliver[tag].value;
        clustering (clusters, reputation, value, THRESHOLD);
    }
    return findTargetCluster(clusters).core.value;
slideWindow (windowPointer, adResults, recentActivities);
for ( delivery of deliveries ) {
    if( |value – delivery.value| < THRESHOLD )
        selectPublisher(delivery.publisher);
        selectRetailer(delivery.retailer);
    else
        rejectPublisher(delivery.publisher);
}
remove (advertisers, msg.sender);
```

Figure 5.3 Pseudocode of oracle interfaces(2/2)

5.2 Evaluations

5.2.1 Experimental environment

We use a smart contract simulator "Truffle suites [14]" to estimate the amount of gas usage for running the code. Based on the gained gas amount used to run each function, we calculate the fee values in US dollar if it is consumed on the main Ethereum network. The currency exchange is referred out in [15].

5.2.2 Transaction Cost and Processing Time

Total time consumed to process a data advertisement is composed of transaction processing time for each interface. Since the processing time is related to the gas price the senders pay for the transaction as Ethereum adopts the price racing model, we calculate the cost and time consumption for three cases: the fastest but most expensive case, the average case, and the cheapest case. Recall that the estimation to the interface C-1 and O-2 are omitted as the "call" transactions used to read state do not consume the gas in Ethereum. Table 5.1 lists the results.

For the fastest case, we set the gas price as 21 Gwei (Giga wei). This means that the transaction pays 21 nano Ether per 1 gas consumption to be processed. The gas price is set as 2.4 Gwei and 2.3 Gwei for each of the average case and cheapest case. The time is evaluated as an "expected time to be confirmed" considering the main network status at the evaluation time (i.e., Nov. 2018). The table shows that the interface O-3 and O-4 require the more massive amount of gas than other interfaces as they transfer the extended data. A transaction for interface O-4 to close advertisement requires a particularly tremendous amount of gas and processing time, as the function also carries out a complex computation including clustering and reputation handling. The time estimation from the transaction results in about 4,000-5,000 seconds which means the consumer can confirm the data after about 80 minutes when she pays the average price.

For interface O-1 that used to open advertisement and interface C-2 that for

Interface		O-1	O-3	O-4	C-2
Gas consumption (per transaction)		148472	298594	5323497	48318
Fastest (21 Gwei)	**Cost($)**	0.327	0.659	11.742	0.107
	Time(s)	35	35	1773	35
Average (2.4 Gwei)	**Cost($)**	0.051	0.103	1.841	0.017
	Time(s)	2016	2016	4608	2016
Cheapest (2.3 Gwei)	**Cost($)**	0.047	0.094	1.675	0.015
	Time(s)	2533	2533	5345	2533

Table 5.1 Expected processing time and cost to the gas price in each interface transaction (Simulated on Nov. 2018)

delivery reporting, the required gas amount may not fluctuate as the message sizes are fixed. The delivering data size is the main factor to decide the cost in case of the interface O-3. In the evaluation, we set the pure data part (excluding the signatures and timestamp parts) as a simple six bytes string to indicate the exchange rate multiplied by one hundred (for handling the decimal point). As the data length is longer, the cost required to send it and store it becomes higher. Another main factor related to the cost is the number of deliveries the consumer want to refer. In addition to that the retailing and publishing fees that are directly correlated to the number, the total length of proofs and computational tasks in interface O-4 increase so that the transaction becomes costly. The amount of gas in table 5.1 is for two deliveries. Figure 5.4 shows the relation between the number of deliveries and gas consumption required to transfer the request message and make a consensus from them.

5.2.3 Practicality

We now demonstrate the proposal's practicality by analyzing the cost and time required to take the whole protocols. We build a model to calculate the total cost and time required to the consumer in Ethereum network. A total cost model described in figure 5.5 reflects the sum of transaction cost, sales fees, publishing fees, and the oracle fee which can be optionally required. For the

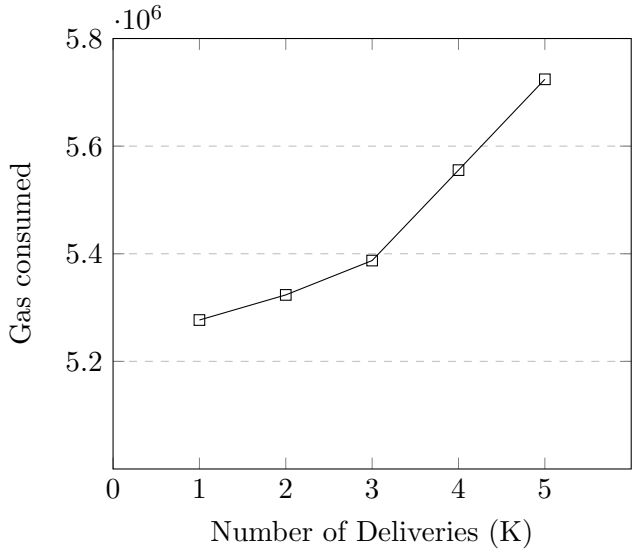

Figure 5.4 Gas consumption to the numbers of deliveries in transaction for interface O-4

total processing time, we provide a model is designed as a lower bound model in figure 5.6 because the actual time consumption is decided how the retailers actively participate in the advertisement. Therefore, we ignore the time consumed for observing the ads and fetching data in the model and assume that the retailers try to deliver data as soon as an advertisement opens.

■ $\boldsymbol{Total\ Cost = TC_{O-1} + K(TC_{O-3} + TC_{C-2} + SF + PF) + TC_{O-4} + OF}$

$\boldsymbol{Total\ Cost}$: Total cost a consumer needs to pay

$\boldsymbol{TC_{Interface}}$: Transaction processing cost for using the interface

$\boldsymbol{K}$: The number of deliveries to use

$\boldsymbol{SF}$: Sales fee for retailers

$\boldsymbol{PF}$: Publishing fee for publishers

$\boldsymbol{OF}$: Oracle fees (optional)

Figure 5.5 Total cost for an advertisement

Figure 5.7 shows the simulation result using the models above. By setting the most expensive gas price for all the transactions to accelerate them, the

total cost a consumer should pay for 15 deliveries is about 30$ and it may takes more than 31 minutes to end up the process. If all the transactions pays the gas price as an average value, the total price is decreased to under 5$ under the same condition, but the lower bound of processing time goes up to about 215 minutes. To compromise between the price and time, the consumer can optimize the cost model by setting gas prices selectively. If she allows all the transactions to adopt the fastest option except the transaction for interface O-4, the most expensive transaction to close her advertisement, she can use the oracle protocol under 15$. The lower bound of the time delay is shortened to about 177 minutes.

The prices are reasonable even if 0.3$ of the sales fees and publishing fees are included. We suppose that the retailers and publishers to be implemented in an automated system like the Oraclize, consequently the market price will go down in the competitive environment.

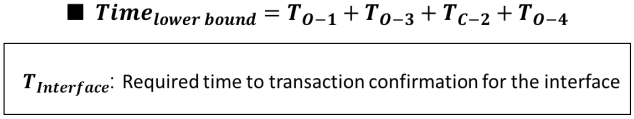

Figure 5.6 The lower bound of processing time for an advertisement

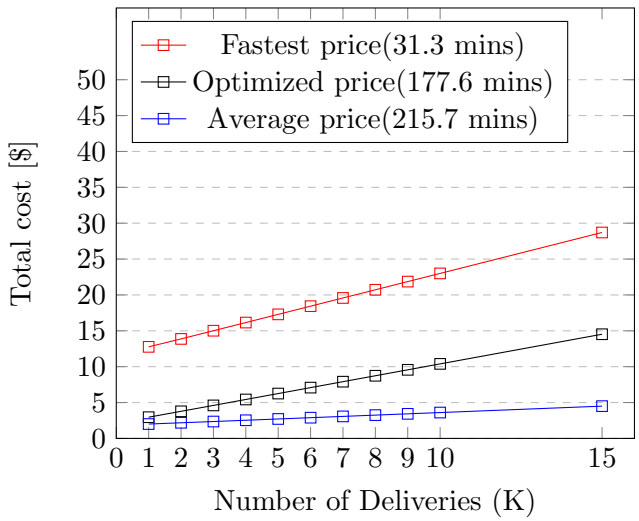

Figure 5.7 Total data delivery transaction cost to number of deliveries

Chapter 6

Conclusion

We have introduced a decentralized oracle which separates the data provider's role into two sub-roles: the publisher and the retailer. The publisher is in charge of publishing a data in an authenticable format. The reatiler takes off the delivery overhead by relaying the data from the off to the on-chain oracle to gain fees. In the design, the data trustworthiness is estimated based on the sum of reputations of the publishers who submit the similar data values. As a result, the publishers who have maintained the continuously high reputation have more opportunity to be selected and to earn the money. Therefore, the existing powerful data providers are motivated to provide a proper data for smart contracts since their off-chain reputation can be reflected to the oracle from the consumer requirements. Through an implementation on Ethereum environment, we prove the practicality of proposed model. We will continue optimization of the code to make it use less memory and reduce the calculation complexity to lower the total cost consumption. Using a complete set of protocol, we expect to provide the first working decentralized oracle to the typical smart contract platforms.

Bibliography

[1] Szabo, N. "The idea of smart contracts," *Nick Szaboś Papers and Concise Tutorials*, vol. 6, 1997

[2] KOSBA, Ahmed, et al. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proceedings of IEEE symposium on security and privacy*, 2016. p. 839-858.

[3] Kalra, Sukrit and Goel, Seep and Dhawan, Mohan and Sharma, Subodh "Zeus: Analyzing safety of smart contracts," in *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2018.

[4] Zhang, Fan and Cecchetti, et al. "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security (CCS)*, 2016. p. 270-282.

[5] Haber, Stuart, et al. "How to time-stamp a digital document." in *Conference on the Theory and Application of Cryptography*, Springer, Berlin, Heidelberg, 1990.

[6] Kalra, Sukrit et al. "Zeus: Analyzing safety of smart contracts." in *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2018.

[7] BONNEAU, Joseph, et al. “Sok: Research perspectives and challenges for bitcoin and cryptocurrencies.” In: Security and Privacy (SP), 2015 IEEE Symposium on. IEEE, 2015. p. 104-121.

[8] Oraclize - blockchain oracle service, enabling data-rich smart contracts, http://www.oraclize.it/

[9] Reality Keys - Facts about the future, cryptographic proof when they come true, https://www.realitykeys.com/

[10] Augur - A Decentralized Oracle and Prediction Market Protocol, https://www.augur.net/

[11] Verity - Real-time decentralized oracle, https://verity.network/

[12] Quora - Quora is a place to gain and share knowledge, https://www.quora.com/

[13] Ian Grigg, Dan Larimer and others, “EOS.IO Technical White Paper v2,” https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md

[14] Truffle Suite, Sweet Tools for Smart Contracts, https://truffleframework.com/

[15] Eth Gas Station — Consumer oriented metrics for the Ethereum gas market https://ethgasstation.info/calculatorTxV.php

[16] Buterin, Vitalik and others, “A next-generation smart contract and decentralized application platform,” white paper, 2014.

[17] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.”, 2008.

요약

스마트 컨트랙트는 입력 값이 사전에 약속된 조건을 만족하면 자동으로 계약을 이행하는 일종의 프로그램으로서, 분산 블록체인 기술의 발달과 함께 분산 어플리케이션(Distributed Applications, Dapps) 구현의 핵심 요소로 자리잡았다. 기존의 중앙화된 시스템에 존재했던 단일 장애점 문제, 과도한 신뢰 요구 등의 문제를 Dapp의 도입을 통해 해결하고자 하는 노력들이 계속되었음에도 불구하고, 기존 시스템을 대체할만한 수준의 솔루션을 구현하기 위해 해결해야 하는 기술적 난제들이 아직 상당수 존재한다.

본 논문에서는 이들 중 외부 데이터 연동 문제를 해결하기 위한 기법을 제안한다. 이러한 문제는 일반적으로 "오라클 문제"로도 불리는데, 블록체인 내부에서 발생하는 데이터는 모두 검증 가능하므로 신뢰할 수 있는것에 비해 외부 네트워크에서 생성된 데이터의 신뢰성은 내부에서 보장할 수 없기 때문에 신뢰할 수 있을만한 데이터 공급처인 오라클을 어떻게 구현할 것인지에 대한 문제로 귀결된다. 오라클의 기술적 요구사항은 크게 두가지로 나누어 지는데, 외부 접근성이 없는 스마트 컨트랙트로 하여금 어떻게 외부 데이터에 접근할 수 있게 할 것인지에 대한 문제와, 해당 데이터의 신뢰성을 어떻게 보장할 것인지에 대한 문제가 함께 고려되어야 한다. 본 논문에서는 대표적인 스마트 컨트랙트 플랫폼 및 관련 서비스 분석을 통해 오라클 디자인에서 고려되어야 하는 특성들과 디자인 결정사항들을 도출하고 이를 기반으로 효과적인 오라클 디자인을 제안 및 구현하고자 하였다.

제안된 기법은 기존 관련 연구들과 달리 데이터 자체의 신뢰성에 집중하기 위해 외부의 데이터 "발행자(Publisher)"와 이를 블록체인 네트워크로 전달하고 수수료를 얻는 "리테일러(Retailer)"로 네트워크 참여자의 역할을 분리하고, 적절한 수익 배분 모델을 구성하여 이들로 하여금 정직한 활동을 유도함으로서 오라클 시스템을 동작케하도록 설계되었다. On-chain 스마트 컨트랙트로 배포된 오라클을 통해 소비자(Consumer) 컨트랙트는 원하는 정보를 다수의 리테일러로부터 전달받을 수 있다. 전달받은 데이터들이 값이 서로 다른 경우 이들은 발행자의 Reputation

에 따른 가중치를 고려한 제안된 합의 프로토콜을 통해 신뢰할 수 있는 하나의 값으로 결정된다. 정직한 데이터를 제공한 발행자와 리테일러는 오라클을 통해 소비자가 미리 지불한 금액의 일부를 보상으로 획득할 수 있으며, 그렇지 않은 경우 합당한 패널티를 얻게 되어 향후 데이터 공급시 불이익을 받게 된다.

본 기법은 궁극적으로 데이터 발행자에게 보상을 지급하고 직접적인 블록체인 네트워크 참여 없이도 데이터를 발행할 수 있게 함으로써 기존 시스템들이 보유한 다수의 양질의 데이터가 안전하게 블록체인으로 공급될 수 있도록 설계되었다. 풍부한 데이터의 공급은 데이터 조달에 요구되는 비용의 감소를 통해 블록체인 네트워크와의 원활한 상호 운용성 보장으로 이어질 수 있다. 또한 현재까지 제안된 기법들이 주로 Whitepaper를 통한 아이디어 중심의 제안이거나 단순 분산 투표를 구현한 집단 지성 플랫폼인 것에 반해, 본 연구에서는 실질적인 데이터를 전달하는 오라클을 이더리움 네트워크에서 구현함으로서 제안된 모델이 실제 운용 가능하며 합당한 수준의 비용과 처리 시간으로 동작함을 증명하였다. 제안된 기법은 향후 내부 메모리 사용량과 연산량을 최소화하는 최적화 과정을 통해, 최초의 상용 분산 오라클 모델로서 더욱 실용적인 Dapp들의 구현에 기여할 수 있을 것으로 기대된다.

주요어: 서울대학교, 컴퓨터 공학부, 졸업논문
학번: 2017-21931