BlockONS: Blockchain based Object Name Service

Wondeuk Yoon
Auto-ID Labs and School of Computing
KAIST
Daejeon, Korea
wdyoon@kaist.ac.kr

Indal Choi
Department of Mobile Communications
LG Electronics
Seoul, Korea
indal.choi@lge.com

Daeyoung Kim
Auto-ID Labs and School of Computing
KAIST
Daejeon, Korea
kimd@kaist.ac.kr

***Abstract*— Today, Internet of Things (IoT) technology is applied to everywhere providing tremendous amounts of IoT service such as home control, facility management, and social public services. The GS1, a non-profit international standard organization, standardized an Object Name Service (ONS) which enables users to manage and discover services in the midst of tremendous amounts of service. However, it has a vulnerability in security and fault tolerance of providing service, because the ONS operates based on the DNS protocol. It is weak against data tampering attacks caused by DNS cache poisoning, spoofing, and local DNS cracking. It has a weak fault tolerance from problems with attack or malfunction. In this paper, we propose a BlockONS, which is novel ONS based on a blockchain. It provides a strength in data tampering attacks allowing a fault tolerance for sustainable service. The BlockONS consists of new service data modeling for an off-chain scaling, data tampering validation method, and fault tolerance mechanism. We designed the BlockONS into two parts: a BlockONS Node part to valid data tampering, and a BlockONS Agent part for scaling and fault tolerance. Finally, we implement the BlockONS prototype using a Hyperledger Sawtooth blockchain and intel i5 NUC. We proof the feasibility of the BlockONS by comparing with performance of an existing ONS.**

Keywords—GS1, Object Name Service, Blockchain, Off-chain Scaling, Data Security, Fault Tolerance.

I. INTRODUCTION

The Internet of Things (IoT) technology, introduced in 1999 by GS1's partner Auto-ID Labs, enables various things with sensors and communication modules to collect data and provide services [1]. With the development of IoT, things and things, things and people are connected to each other through the Internet. Variety of IoT services, including home control services, facility monitoring services, and public services, are provided across the society. However, as the number of IoTs and services increases, it becomes difficult for service providers to manage them. It also becomes complex for users to discover IoT services which they want. Therefore, service management and discovery system is indispensable to control and search services easily from numerous IoTs and their associated services.

The GS1, non-profit international standard organization, published the Object Name Service (ONS) standard [2] that manages and discovers things and their associated services using an standardized identifier of things called a GS1 code [3]. A service provider can store ONS records, which are service data of things, in distributed local ONSs using GS1 code as an identifier. A user can request service data of things using GS1 code as an index. The DNS protocol [4] can hierarchically discovers stored service data and return it to the user.

However, because the ONS is designed based on the DNS protocol, it has two disadvantages. First, it is vulnerable to data tampering attacks such as DNS cache poisoning, spoofing, and local DNS cracking. Second, it is hard to provide sustainable service when a data tempering attack occurs. Recently, the IoT services are applied in everywhere such as a payment, an authentication, a healthcare and a management. Therefore, in order to prevent serious problems such as data spill, and economic/human loss due to tampering attack and malfunction, the ONS has to valid tampering of service data stored in the ONS. Also, it has to provide fault tolerance for persistence service.

For several years, many works for the ONS have proceeded. Evdokomov at el. suggested a MONS(Multipolar ONS), which creates multiple replicated ONS Root Nameservers by dividing the data held by an existing ONS root [5]. Kin-Foo at el. proposed a DNS protocol based decentralized federated ONS [6]. Schapranow at el. proposed a P2P-based ONS that uses SSL protocols to prevent cache poisoning, spoofing attacks during ONS Lookup process [7]. Fabian proposed a Distributed Hash Table (DHT) based P2P ONS and provides access control using cryptographic hash [8]. However, they insufficiently deal with fault tolerance of the local ONS. Additionally, they only consider attacks during the process of service querying. They did not consider the data tampering attacks about service data already stored in the local ONS.

In this paper, we propose a BlockONS, a blockchain based Object Name Service that supports strength in data tampering attacks and fault tolerance for sustainable service with off-chain scaling. First, the BlockONS provides tampering validation method. It generates unique hash of original service data and stores it in the blockchain during service data registration. In this way, the attacker cannot easily tamper with the hash of the original service data stored in blocks. The BlockONS can valid whether the tampering attack occurred by comparing the hash stored in blocks with the hash of the original service data. Second, the BlockONS provides fault tolerance mechanism with off-chain scaling. Storing all the original service data in the blockchain to valid data tampering attacks degrades the performance of the blockchain, and requires expensive system maintenance costs. Therefore, we design a new service data modeling for off-chain scaling to improve performance. In addition, we propose a subscribe and synchronize mechanism to tolerate fault caused by off-chain scaling.

The BlockONS is designed into two parts, a BlockONS Node and a BlockONS Agent. The BlockONS Node is implemented based on a Hyperledger Sawtooth blockchain. It handles all transactions including registration, renewal, deletion, and discovery of service data from the BlockONS Agent. The BlockONS Agent stores original service data,

978-1-7281-1328-9/19/$31.00 ©2019 IEEE

which is indexed by GS1 code, in a local database. It handles all requests including management and discovery of service data from service providers or users. It also subscribes and synchronizes original service data among agents for sustainable service. Finally, we deployed a BlockONS prototype testbed using intel i5 NUCs. We proof the data tampering validation method, and fault tolerance mechanism with off-chain scaling. In addition, we show the feasibility of performance by comparing to an existing ONS.

The remainder of this paper is organized as follows. Section II describes the background knowledge of the ONS and the Sawtooth blockchain. Section III describes the architecture and component of the BlockONS. Section IV describes the implementation of the BlockONS prototype and testbed. The performance of the BlockONS is evaluated and discussed in Section V. Finally we conclude this paper and suggest future work in section VI.

II. BACKGROUND

A. Global Standard 1 Identification Keys (GS1 Code)

The GS1 provides 12 types of a GS1 code, an international identification standard for improving the efficiency and visibility of global production, logistics, distribution and consumption networks. The GS1 code is used to global unique identifier of things such as product, location, asset, document, people and etc. Various services can be associated with GS1 codes as shown in Table I.

B. Object Name Service (ONS)

The ONS is a name service that provides ways to globally register and discover services associated with a GS1 Code assigned to objects. It also provides a dynamic service definition model called ServiceType.xml for manging service type. Current DNS based federated model with multiple peer root ONS, that prevents censorship, access block, and tampering by a single operator, has been adopted as a standard version 2.0.1. In addition, a system for registering and discovering services for various urban resources in Smart City utilizing the ONS has been proposed [9]. Fig. 1 shows the entire process of service discovery on the ONS. (1,2) Scan and extract GS1 codes from the IoT devices via barcode, QR code, datamatrix, bluetooth beacon, and WiFi AP. (3,4) Convert the GS1 code to an Application Unique String, and change it to a Fully Qualified Domain Name (FQDN). (5,6) Query a Name Authority Pointer (NAPTR) records using the FQDN. (7) Make service list using returned NAPTR records and ServiceType.xmls. (8) User can acess a service.

C. Hyperledger Sawtooth Blockchain

The Sawtooth is the one of Hyperledger projects managed by the Linux Foundation. It focuses on modularity and flexibility of smart contract called a Transaction Process (TP) and consensus algorithms called POET (Proof-Elapsed-Time). The TP provides flexibility to handle multiple business logics simultaneously. The POET allows the size of the network to scale. It can nearly support limitless nodes in the network. In addition, the Sawtooth manages every distributed ledger of TPs in a single Merkle-Radix tree called a Global State [10]. The Global State provides high visibility and robust read data access of records stored in a distributed ledger.

TABLE I. VARIOUS SERVICES OF GS1 CODES

GS1 Code	Various Services
GTIN (Global Trade Item Number)	commercial product, Smart City service, and etc.
GLN (Global Location Number)	power plant, bus stop, park, city hall, library, hospital and etc.
GRAI (Global Returnable Asset Identifier)	sharing bicycle, sharing car, and etc.
GIAI (Global Indivisual Asset Identifier)	CCTV, bus, metro, street light, healthcare device, and etc.
GSRN (Global Service Relation Number)	doctor-patient, librarian-borrower, smartcity administratorcitizen, and etc.
GDTI (Global Document Type Identifier)	certifications, driving license, tax bill, official document, and etc.

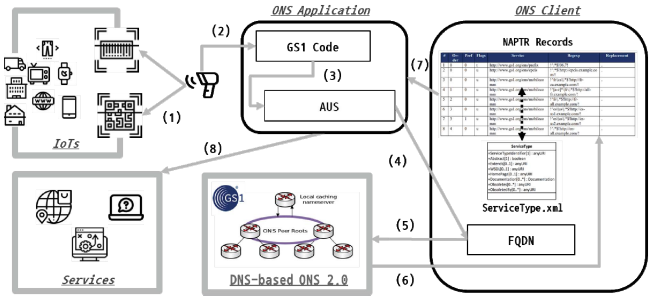

Fig. 1. Overview of GS1 ONS process.

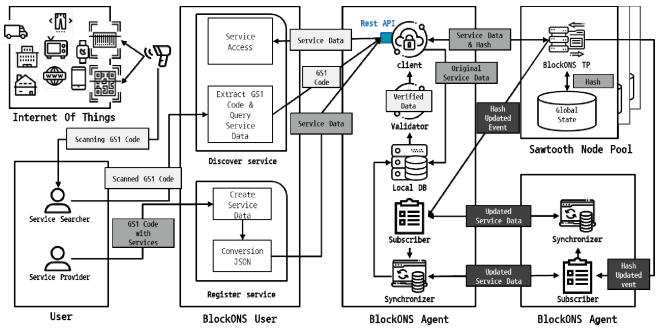

Fig. 2. Blockchain based Object Name Service (BlockONS) Overview.

In order to manage and discovery tremendous amounts of services, the system is required for high network scalability, fast read data access, and flexible business logic implementation. Comparing with other blockchains such as Ethereum, and Hyperledger Fabic, since the Sawtooth satisfies these requirements, it is used for the BlockONS.

III. BLOCKCHAIN BASED OBJECT NAME SERVICE

In the ONS, a service provider registers service data associated by the GS1 code in to distributed local ONS. A service searcher discovery service data by using the GS1 code as an index through DNS protocol. In addition, we should handle the data tampering attacks and fault of service to improve the ONS. Therefore, we propose methods for applying blockchain technology to the ONS. The Fig. 2 shows an overview of the proposed BlockONS. It consists of three main parts. The normal gray part shows how to store the original service data from the service provider to the blockchain. The light gray part shows how to valid data tampering attack using a data stored in the blockchain. The dark gray part shows how to share the service data among the blockchain for sustainable service.

The blockchain protects data stored in the distributed ledger from tampering attacks by sharing and synchronizing same data with all nodes participating in the network. Thus,

if the original service data of all local ONSs stores in the distributed ledger of the blockchain, it can be safe from data tampering attacks. Also, since all blockchain nodes synchronize the same service data, it can continuously provide ONS service when a specific node dies.

However, if all original service data is stored in the blockchain, the size of the blockchain increases exponentially. This rises operation costs and degrades the performance of the blockchain. In order to optimize the size to the blockchain, the original service data must be refined into small size and stored in the distributed ledger called an on-chain. In addition, the original service data before refinement must be stored in a database outside of the blockchain called off-chain. This technical method is called an off-chain scaling. For off-chain scaling, we propose a new service data modeling to refine original service data into small size. Also we propose data tampering validation method that can valid the original service data stored in the off-chain using the refined data stored in the on-chain. Lastly, due to off-chain scaling, the original data is not synchronized over all off-chain databases. Therefore, we propose fault tolerance mechanism to provide sustainable service even if the off-chain databased is broken down.

A. New Service Data Modeling for off-chain scaling

To realize off-chain scaling with data tampering validation, we have to consider a data refinement method to reduce the size of the blockchain, a new service data model to handle the original and refined service data, and an role based off-chain model to manage off-chain database efficiently.

1) Data Refinement Method

In order to reduce the size of original service data ensuring uniqueness of it, we leverage a hash algorithm. The hash algorithm is used to map data of arbitray size onto data of a fixed size. Considering a hash collision problem, we refine the ONSRecord and SeriveType using the SHA-512 [11] as described in Algorithm 1. By using the SHA-512, we reduce the size of the ONSRecord to 512 bits. Also, it is possible to guarantee the uniqueness of the original service data. Additionally, since the hash value must be changed when the original service data is changed, it can be used to valid the original service data.

Algorithm 1. Data Refinement Method

Input: off-chain's *public key*, and *ONSRecord* / *ServiceType*,
Output: HashOR / HashST.
Initialize string *D* to concatenate input data.
if Input is ONSRecord **then**
  for t=1, size of *ONSRecord* **do**
    D = *D* + *ONSRecord.record*(t)
  end for
  return HashOR = {
    gs1Code := *ONSRecord.gs1Code*,
    hash := sha512(*D*),
    agentId := *public key*
  }
else
  for t=1, size of *ServiceType* **do**
    D = *D* + concat(*ServiceType.fields*(t), *ServiceType.types*(t))
  end for
  return HashST = {
    serviceTypeId := *ServiceType.SeriveTypeId*,
    hash := sha512(*ServiceType*),
    agentId := *public key*
  }
end if

2) New Service Data Model

The BlockONS has to fully support features of the ONS standard with off-chain scaling and strength in data tampering attacks. Thus, we redefine the ONS service data models, which are the NAPTR record and the ServiceType.xml as shown in Fig. 3.

For the original service data stored in off-chain database, we design an ONSRecord and a ServiceType data model as shown in Fig. 4. The new service data models basically support compatibility with existing service data model. Additionally, it is designed for flexible service data expand by using key-value paired repeated model. Through key-value paired repeated model, the BlockONS can support broad scalability of the service data. For the refined service data stored in the on-chain, we newly design an HashOR, HashST, and AgentInfo as shown in Fig. 5. In order to reduce the size of the blockchain, the HashOR/HashST are designed to store a single string that is the the ONSRecord/ServiceType hash value created by data refinement method. The AgentInfo is a data model that stores information of off-chain databases. It is designed for data sharing among off-chains.

The on-chain and the off-chain are interconnected through a common identifier. The ONSRecord and HashOR use the GS1 code to guarantee global unique storage and search. The ServiceType and HashST use the serviceTypeId and agentid to guarantee global unique storage and search.

```
struct NAPTR record {
  int order;
  int preference;
  string flags;
  string servicetype;
  string regularExpression;
  string replacement;
}
```

```
struct ServiceType.xml {
  anyURI serviceTypeIdentifier;
  boolean abstract;
  anyURI extends;
  anyURI WSDL;
  anyURI homepage;
  Documentation[] documentation;
  anyURI[] obsoletes;
  anyURI[] obsoletedBy;
}

struct Documentation {
  string languageCode;
  anyURI location;
  }
```

Fig. 3. NAPTR record, servicetype.xml data model.

```
struct RecordField {
  string key;
  string value;
}

struct Record {
  string recordId;
  repeated RecordField fields;
  repeated RecordField types;
}

struct ONSRecord {
  string gs1Code;
  string agentId;
  repeated Record records;
}
```

```
struct ServiceTypeField {
  string key;
  string value;
}

struct ServiceType {
  string serviceTypeId;
  repeated ServiceTypeField fields;
  repeated ServiceTypeField types;
  string agentId;
}
```

Fig. 4. ONSRecord, ServiceType data model.

```
struct HashOR {
  string gs1Code;
  string hash;
  string agentId;
}
```

```
struct HashST {
  string serviceTypeId;
  string hash;
  string agentId;
}
```

```
struct AgentInfo {
  string agentName
  string queryURL;
  string syncURL;
  string infoURL;
  string agentId;
}
```

Fig. 5. HashOR, HashST, AgentInfo data model.

TABLE II. ROLE BASED OFF-CHAIN MODEL.

Off-chain Type	Roles			
	Store own data	Discover own data	Subsidiary roles	
			Copy data	Share data
Full	Operate	Operate	Operate	Operate
Light	Operate	Operate	Operate	-
Isolated	Operate	Operate	-	-

Algorithm 2. Service Data Validation Method

Input: *ONSRecord* / *ServiceType* stored in the off-chain database.
Output: 0 (no tampering), not 0 (tampering detected).
Initialize string *D* to concatenate input data.
Initialize string *H* to store hash value.
if Input is ONSRecord **then**
  for t=1, size of *ONSRecord* **do**
    D = *D* + *ONSRecord.Record*(t)
  end for
  H = find *HashOR* using *ONSRecord.gs1Code* from the on-chain
  return | SHA512(*D*) – *H.hash* |
else
  for t=1, size of *ServiceType* **do**
    D = *D* + concat(*ServiceType.fields*(t), *ServiceType.types*(t))
  end for
  H = find *HashOR* using *ServiceType.serviceTypeId* from the on-chain
  return | SHA512(*D*) – *H.hash* |
end if

3) *Role based Off-chain Model*

We can reduce the size of the on-chain through the off-chain scaling. However, if the off-chain stores all original service data, it will have the same operational cost and performance degradation problem. Therefore, we propose role based off-chain model to manage off-chain database efficiently.

The off-chain is a kind of distributed database managed by the service provider, such as the local ONS. In the ONS, the service provider only stores its own service data in the local ONS. Subsidiary, local ONS can shares data with another local ONS or copies data from another local ONS according to special requests such as backup. Similarly, the off-chain classifies into a Full, a Light, an Isolated types according to the following rolls as shown in Table II. The basic roles are following: (1) Store own data means that service provider can store its own service data to the off-chain, (2) Discover own data means service searcher can discover the service data stored in the off-chain. Subsidiary roles are following: (1) Copy data means off-chain copies all service data from other off-chains, (2) Share data means off-chain allows other off-chains to copy its own data.

The full and light type off-chain can be set for backup of services, which the basis for sustainable services. The light type off-chain can be set if the contents of the service data cannot be stored in other agents due to legal limitations. The isolated type off-chain can be set if the service provider want to operate the off-chain privately.

B. *Service Data Validation Method*

The BlockONS stores the original serive data in the off-chain, and stores the SHA-512 hash value of it in the on-chain as a result of the off-chain scaling. Since the SHA-512 hash value guarantees the uniqueness of the original service data, by using this, we propose a service data validation method in order to valid the tampering of the original service data as . 1 described the Algorithm 2.

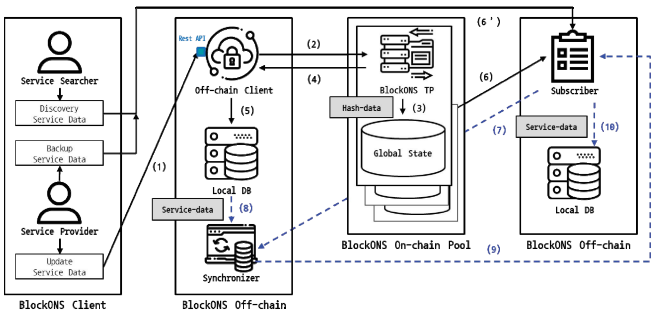

Fig. 6. Flow Chart of Service Data Synchronization among off-chains.

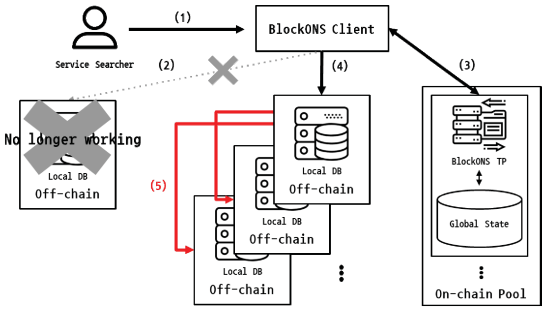

Fig. 7. Flow Chart of Sustainable Service of the BlockONS.

The service data validation method operates on the off-chain. When the service searcher requests a service data, it finds the original service data (*D*) stored in the off-chain database, and generates the hash value by using the original data (*SHA512(D)*). It also finds the hash value (*H*) stored in the on-chain. Lastly, it return a result of subtracting *H* from *SHA512(D)*. Since the hash value is changed when the original data is changed, it returns '0' when the data is normal and 'not 0' when the data tampering occurred.

C. *Fault Tolerance Mechanism*

As a result of the off-chain scaling, all service data is not stored the off-chain, so service fault can occur when a particular node dies. Therefore, we propose subscribe and synchronize based fault tolerance mechanism to prevent service failure.

We design a subscriber and a synchronizer to copy and share the service data stored in the off-chain as shown in Fig. 6. The subscriber performs as the inspector for own off-chain to copy the original service data from other off-chains. It is executed by the service provider's backup request, the service searcher's discovery request, and the on-chain's updated event. The synchronizer performs as the messenger for other off-chains to share their own service data.

The subscription and synchronization process for the service data is following: (1) The service provider request the update transaction to the off-chain. (2) Send the service data to the on-chain. (3) Store hash value in the global state. (4) Return a result. (5) Store the original service data in the local database. (6) The updated event is broadcasted to the subscriber of other off-chains. (6') The backup and discovery request executes the subscriber when the off-chain does not have requested service data. (7) When the subscriber is executed, it requests 'copy data' to the synchronizer that stores the original service data. (8) The synchronizer finds the original service data from the off-chain database. (9) Do 'share data' to send original data to the off-chain. (10) The

subscriber receives the original service data and stores it in the the off-chain database. Through this procedure, all full and light type off-chain can backup up-to-date service data that other off-chains have.

Fig. 7 shows the process of the sustainable service. (1) The service searcher scans the GS1 code to query the service data to the BlcokONS User which is client of the BlockONS. (2) However, the server fails due to some reasons. (3) The BlcokONS User requests and gets other full and light off-chain information from on-chain's AgentInfo. (4) Query the service data on other off-chains that backup the original service data from failed off-chain. (5) Also additional queries can be possible, if target off-chain doesn't have the service data. Through this procedure, the service searcher can experience sustainable service of the BlockONS.

D. BlockONS Architecture

The architecture of the BlockONS is shown in Fig. 8. It consists of the BlockONS Server and the BlockONS Client. The server has a BlockONS Node for the on-chain and a BlockONS Agent for the off-chain. The node includes the TP and global state leveraging the Sawtooth blockchain. The global state manages agent information and hash value of the service data. The TP includes an agent information handler and a record/type hash value handler. The agent includes a local database, a client, a synchronizer, and a subscriber. The local database stores the original service data. The client consists of a manage, discover record/type modules, and a control synchronizer/subscriber module. The synchronizer and subscriber consist of on/off module and a share/copy module. The client consists of the service provider and service searcher. The service provider creates new service data, and requests a register/delete/update query to the agent. The service searcher query the service data by using the GS1 code as an index.

1) BlockONS Node

The node plays a role of the on-chain part in the BlockONS. It handles the AgentInfo, HashOR, and HashST transactions, which are requested by the agent. It also stores results of TPs in the global state.

When registration or update transaction of the AgentInfo is requested, the agent information handler TP processes it the following order: (1) Check the AgentName is already exist in the global state. (2) If it does not exist, save the AgentInfo and return success. (3) If it exists and sender's public key is same to the agentId, update the AgentInfo and return success. (4) If it is different, return transaction error.

When registration or update transaction of the HashOR or HashST is requested, the record/type hash handler TP processes it the following order: (1) Check the HashOR or HashST is already exist in the global state. (2) If it does not exist, generate and store the hash value of the ONSRecord or ServiceType, and return success. (3) If it exists and sender's public key is same to the agentId, update the hash data and return success. (4) If it is different, return transaction error. When deletion transaction of the HashOR or HashST is requested, the record/type hash handler TP can only delete hash value stored in the global state. It cannot delete transactions stored in the blockchain. Therefore, if tracking of deleted service data is needed, we can trace transactions stored in the blockchain.

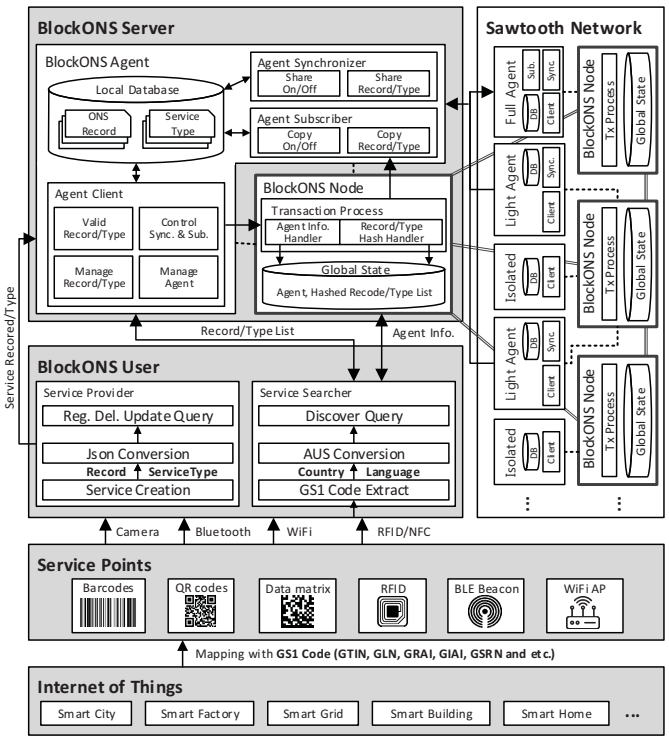

Fig. 8. Blockchain based Object Name Service (BlockONS) Architecture

2) BlockONS Agent

The agent plays a role of the off-chain part in the BlockONS. It handles all requests of the service provider. First, it handles the registration, deletion, and update requests of the AgentInfo, ONSRecord, and ServiceType. Fig. 9 shows a flow chart of registration, deletion, and update requests. (1) The service provider sends the AgentInfo, ONSRecord or ServiceType to the agent through the REST API. (2) The agent forwards received data to the node. (3-5) Receive the result of the node. (6) If the node returns success, dthe client stores original agent information or the service data in the local database. If the node returns transaction error, the client ignore requests.

Second, it handles discover requests of the service searcher. Fig. 10 shows a flow chart of discovery request. (1) The service searcher sends the GS1 code to the agent through the REST API. (2) The client requests transaction of the HashOR/HashST to the node. (3-4) Receive the hash value of the service data. (5) Find the original service data using the GS1Code/ServiceTypeId from the local database. (5') If service data does not exist in the local database, the subscriber requests copy to the synchronizer that has the service data. After sharing the service data, the subscriber store it in the local database. (6) Validate the service data by using the hash value of the HashOR/HashST. (7) Return service data if it is valid.

Depending on roles of the agent, the agent is classified into three types. The client of the agent manages the store and discover roles as basic functions. The subscriber manages the copy role to replicate the original data from other agents. The synchronizer manages the share role to allow the data copy to other agents. As shown in Table III, agent type is determined by module configuration.

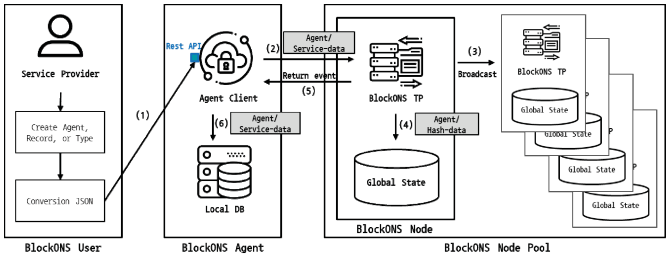

Fig. 9. Flow Chart of Service Registration, Deletion, and Update Request.

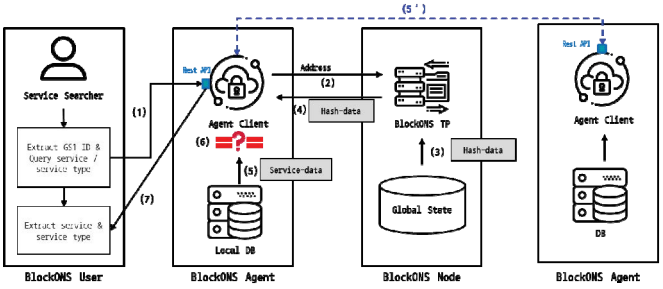

Fig. 10. Flow Chart of Service Discovery Request.

TABLE III. BLOCKONS AGENT TYPES BY RUNNING COMPONENTS.

Agent Type	Client	Synchronizer	Subscriber
Full	Run	Run	Run
Light	Run	Run	-
Isolated	Run	-	-

A Full agent is composed of all three components. A Light agent is composed of the client and synchronizer. An Isolated agent is composed of the client only.

The Full Agent runs all components to handle all requests from the service provider and searcher. It registers and discovers the ONSRecord and ServiceType stored in the local database. It also copies service data from other agents, and stores them in the local database, when the service searcher discovers service data stored in other agents. It allows other agents to share their own service data stored in the local database to other agents' local database.

The Light Agent performs caching by copying service data from other agents. However, it forbids other agents to share their own service data stored in the local database. The Light Agent can be configured if the contents of service data cannot be stored in other agents due to legal limitations.

The Isolated Agent does not copy and the service data of the other agents. It also forbid the other agents to share its own service data. Their own service data is only stored in their local database. If it is difficult to copy service data and the user want to operate the agent privately, the user can configure the Isolate Agent.

IV. PROTOTYPE TESTBED OF THE BLOCKONS

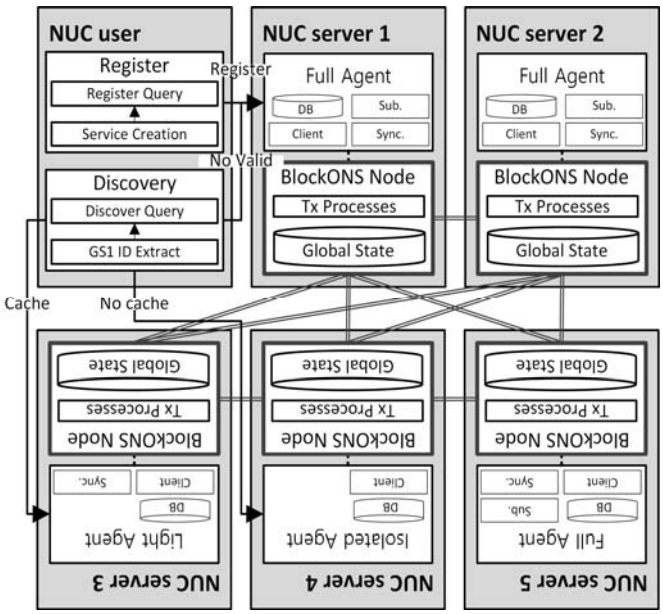

Fig. 11. BlockONS prototype implementation and testbed environment.

Fig. 11 shows the implementation of the BlockONS prototype, and a testbed environment. The BlockONS prototype has the BlockONS User and BlockONS server. The user consists of a register and a discovery module. The server consists of one agent and one node. To evaluate the BlockONS in the real network and device environment, we deploy one BlockONS User and five BlockONS Servers using six intel NUC5i5ryh (SSD 256GB, DDR3 16GB) and iptime a6004ns AP(802.11ac, 700 Mbps).

The BlockONS User uses 'siege [12]' to evaluate the performance of registration and discovery requests. BlockONS Servers are connected with each other through 700 Mbps wireless network. We use the Sawtooth v1.0.5 and Golang to implement the node and TPs. The local database, client, subscriber, and synchronizer of the agent are implemented by using MongoDB v4.0.2 and Golang. Lastly, the testbed includes full, light, and isolated agents for performance evaluation.

V. EVALUATION AND DISCUSSION

To evaluate the performance of the BlockONS, we conducted Transaction Per Seconds (TPS) and a Mean Test Time (MTT) experiment about registration and discovery requests of GS1 codes, ONS records, and ServiceTypes. For realistic load testing, we configured that 592 virtual users sent registration and discovery requests to BlockONS Server every one second.

A. GS1 Code, ONS record, and ServiceType Register

To evaluate registration performance of the BlockONS, we set 40,000 GS1 codes (, 100,000 ONS records, and 40,000 service types. We measured the performance of TPS and MTT by increasing the number of BlockONS nodes from one to five.

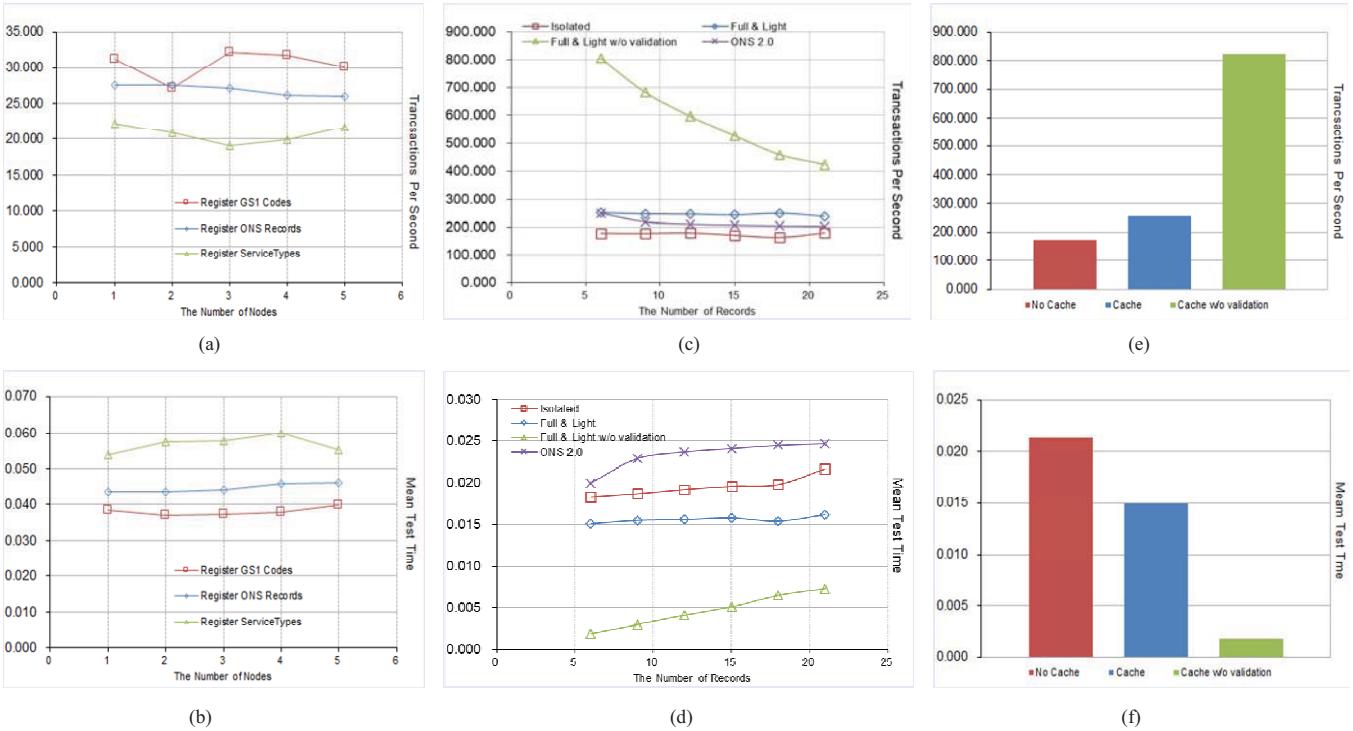

Fig. 12. Performance evaluation of the BlockONS. (a) Register TPS of blockONS by nodes, (b) Register MTT of blockONS by nodes, (c) Discovery TPS of ONS record by records, (d) Discovery MTT of ONS record by records, (e) Discovery TPS of serviceType, (f) Discovery MTT of serviceType.

Fig. 14 (a), (b) show the performance results of registration transactions with the number of nodes. Average TPS results of registration transaction are 30.472 tps (GS1 Codes), 26.895 tps (ONS Records), and 20.782 tps (ServiceTypes). In addtion, average MTT results of registration transaction are 38 ms (GS1 Codes), 45 ms (ONS Records), and 57 ms (ServiceTypes).

The results show that the number of nodes does not significantly affect performance. This is because the transaction performance of the block chain is determined by the consensus algorithm rather than the number of nodes. We leverage the PoET consensus algorithm to implement the BlockONS. Since it likely selects a node as a block minor that can generate a block with the most efficient elapsed time. So performance of TPS and MTT is determined by the hardware resource power, not the number of nodes.

In addition, the results show that performance decreases as the size of the data to be registered increases. In evaluation, since we use 1,202 bytes-sized ServiceType, 258 bytes-sized ONS record, 13 bytes-sized GS1 code, the BlockONS performs in the order GS1 code, ONS record, and ServiceType. Because the larger the size of the data consumes the higher hash generation time.

Thus, to order to improve the performance of registration transactions, we can simply substitute high-performance device for the BlockONS node in the testbed. In a different way, we can survey hash algorithms that generate a hash value without affecting the size of the data.

B. ONS record, and ServiceType Discovery

To evaluate discovery performance of the BlockONS, we used the same data set as the registration performance test. We defined three different scenarios. First, we requests discovery to the isolated agent. Second, we requests discovery to the full & light agent. Third, we requests discovery to the full & light agent without data tampering validation. In ONS record discovery evaluation, we measured performance by increasing the size of the target ONS record, from 6 to 21 records. For the service type performance evaluation, we used 1,202 bytes-sized ServiceType data.

Fig. 14 (c), (d) show ONS record discovery results of Isolated, Full & Light and Full & Light without validation with the number of records. Average TPS results of discovery transaction are 174.982 tps (Isolated), 246.380 tps (Full & Light), 582.112 tps (Full & Light without validation), and 215.615 tps (ONS v2.0.1). In addtion, average MTT results of registration transaction are 20 ms (Isolated), 16 ms (Full & Light), 5 ms (Full & Light without validation), and 23 ms (ONS v2.0.1).

The results show that the smaller size of ONS record has the higher discovery performance. Isolated agent shows similar performance then the result of existing ONS 2.0.1. Full & Light agent shows better performance than the result of the existing ONS 2.0.1. The reason for this results is that the BlockONS changes existing federated ONS query to on/off-chain based direct query through the off-chain scaling. Additionally, if we can omit the service data validation sequence, Full & Light agent without validation shows the best performance then the others.

Fig. 14 (e), (f) show ServiceType discovery results. Average TPS results of discovery transaction are 171.950 tps (Isolated), 254.730 tps (Full & Light), and 823.860 tps (Full & Light without validation). In addtion, average MTT results of registration transaction are 21 ms (Isolated), 15 ms (Full & Light), and 2 ms (Full & Light without validation). For the same reason as ONS record discovery results, Full & Light agent without validation shows the best performance.

Thus, in order to improve the performance of discovery transactions, we can move the data validation process to the

client-side for saving time of hash value comparison. In a different way, since the result of Full & Light better then Isolated's one, we can design incentive economy model for motivating people to operate more full and light agents.

VI. Conclusion and Future Work

We use the distributed ledger technology of the blockchain to provide efficient storage and persistent service discovery ensuring the service data tampering validation. In order to realize the blockchain based ONS called the BlockONS, we design the off-chain scaling scheme including data refinement method, new service data model, and role based off-chain model to manage the service data efficiently. We design the service data validation method using the characteristics of hash algorithm. We design fault tolerance method in the off-chain for sustainable service by subscribing and synchronizing the service data among off-chains. We also implement the BlockONS, and evaluate TPS and MTT performances in the real network and device environment. Average registration result is 26.050 tps, and 47 ms, and average discovery result is average 375.669 tps, and 13 ms, while existing ONS has 215.615 tps and 23 ms discovery performance. Finaaly, we confirm that the proposed BlockONS show better performance in discovery evaluation then existing ONS ensuring data tampering validation and sustainable service.

As the future works, we will consider an access control for the BlockONS. Also we will design incentive economy for the BlockONS eco-system to motivate the user to operate more full and light agents.

Acknowledgment

This work is supported by Smart City R&D project of the Korea Agency for Infrastructure Technology Advancement(KAIA) grant funded by the Ministry of Land, Infrastructure and Transport(MOLIT), Ministry of Science and ICT(MSIT) (Grant 18NSPS-B149386-01), and International Research & Development Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning of Korea (2016K1A3A7A03952054).

References

[1] K. Ashton, "That 'internet of things' thing in the real world, things matter more than ideas", RFID Journal, June 2009.

[2] GS1 Object Name Service (ONS) 2.0.1, "GS1 object name service version 2.0.1", January 2013.

[3] GS1 ID Keys, "https://www.gs1.org/standards/id-keys", Retrieved on Nov. 2018.

[4] MEALLING, M., and DANIEL, R., "RFC 2915: The naming authority pointer (naptr) dns resource record.", Sep. 2000.

[5] Evdokimov, Sergei, Benjamin Fabian, and Oliver Günther. "Multipolarity for the object naming service." In The Internet of Things, Springer, Berlin, Heidelberg, 2008. pp. 1-18.

[6] Kin-Foo, Sandoche Balakrichenan Antonio, and Mohsen Souissi. "Qualitative Evaluation of a Proposed Federated Object Naming Service Architecture." In IEEE International Conferences on Internet of Things, and Cyber, Physical and Social Computing. 2011.

[7] Schapranow, Matthieu-P., Alexander Zeier, Felix Leupold, and Tobias Schubotz. "Securing EPCglobal object name service-Privacy enhancements for anti-counterfeiting." In 2011 Second International Conference on Intelligent Systems, Modelling and Simulation, 2011. pp. 332-337.

[8] Fabian, Benjamin. "Implementing secure p2p-ons." In Communications, 2009. ICC'09. IEEE International Conference on, 2009. pp. 1-5.

[9] Yoon W, Lee Y, Chae H, Seo S, Heo S, Lee N, Kwon K, Kim D. HERMES: GS1-based Smart City Service Intercommunity. In2018 IEEE International Smart Cities Conference (ISC2) 2018 Sep 16 (pp. 1-8). IEEE.

[10] Hyperledger Sawtooth Architecture Guide, "Global State", https://sawtooth.hyperledger.org/docs/core/nightly/master/architecture/global_state.html, Retrieved on Nov. 2018

[11] FIPS P. 180-4. Secure hash standard (SHS)," March. 2012.

[12] Joe Dog Software, "Siege: http load testing and benchmarking utility", https://www.joedog.org/siege-home/, Retrieved on Nov. 2018.