electronics

MDPI

Article

An IOTA-Based Service Discovery Framework for Fog Computing

Tsung-Yi Tang, Li-Yuan Hou and Tyng-Yeu Liang *

Department of Electrical Engineering, National Kaohsiung University of Science and Technology, 807618, Kaohsiung 807, Taiwan; vazontang@gmail.com (T.-Y.T.); F107154131@nkust.edu.tw (L.-Y.H.)
* Correspondence: lty@mail.ee.nkust.edu.tw; Tel.: +886-7-381-4526 (ext. 15508)

Abstract: With the rise in fog computing, users are no longer restricted to only accessing resources located in central and distant clouds and can request services from neighboring fog nodes distributed over networks. This can effectively reduce the network latency of service responses and the load of data centers. Furthermore, it can prevent the Internet's bandwidth from being used up due to massive data flows from end users to clouds. However, fog-computing resources are distributed over multiple levels of networks and are managed by different owners. Consequently, the problem of service discovery becomes quite complicated. For resolving this problem, a decentralized service discovery method is required. Accordingly, this research proposes a service discovery framework based on the distributed ledger technology of IOTA. The proposed framework enables clients to directly search for service nodes through any node in the IOTA Mainnet to achieve the goals of public access and high availability and avoid network attacks to distributed hash tables that are popularly used for service discovery. Moreover, clients can obtain more comprehensive information by visiting known nodes and select a fog node able to provide services with the shortest latency. Our experimental results have shown that the proposed framework is cost-effective for distributed service discovery due to the advantages of IOTA. On the other hand, it can indeed enable clients to obtain higher service quality by automatic node selection.

Keywords: distributed ledger technology; masked authenticated message (MAM); time-difference registration addressing; tree-based MAM indexing; node selection

Citation: Tang, T.-Y.; Hou, L.-Y.; Liang, T.-Y. An IOTA-Based Service Discovery Framework for Fog Computing. *Electronics* **2021**, *10*, 844. https://doi.org/10.3390/electronics10070844

Academic Editor: Sunghyun Cho

Received: 28 February 2021
Accepted: 27 March 2021
Published: 1 April 2021

1. Introduction

With the rise in the Internet of Things, the concept of fog or edge computing was proposed for minimizing the response time of cloud services. By bringing resources from distant data centers to the end of networks, fog-computing nodes can provide services for user applications with lower network latency and less bandwidth consumption compared with cloud-computing nodes. Many fog-computing toolkits or platforms have been proposed in recent years. For example, Cloudlet [1,2] was proposed by Carnegie Mellon University to support user applications with a network distance of one hop. Fog Computing [3] was created by CISCO to analyze time-sensitive data at the network edge instead of sending massive IoT data to the cloud. ETSI standardized mobile Edge Computing (MEC) [4] to provide cloud-computing services close to mobile users with the radio access network. Cisco, ARM, DELL, INTEL, Microsoft, and other companies cooperatively proposed OpenFog [5] to move computation, storage, communication, and decision making closer to the users along a cloud-to-thing continuum. The Linux Foundation proposed EdgeX [6] to be the edge computing gateway of IoT and software development.

Among the projects and standards mentioned above, the OpenFog standard clearly states that containerization is used at the software layer to provide software and application microservices, which require an efficient mechanism of service discovery. The software development documents of MEC show that it is recommended to use microservice and

DesOps to develop and deploy applications. EdgeX adopted a microservice to be the uniform architecture of application development. Amazon Web Services (AWS) launched the IoT Greengrass framework [7], which enables AWS Lambda microservices developed in the cloud to be directly deployed to client nodes. As previously described, adopting the microservice architecture is a trend for the development of fog-computing applications.

Microservices are an evolution of service-oriented architecture (SOA) [8]. The same as SOA, microservices support tandardized service interfaces, service reusability, and loose coupling of services. The difference is that microservices emphasize the granularity of software and services, that is, to combine several single-function and independently operable microservices through a unified interface and protocol to achieve the required applications. In microservices, the service discovery mechanism plays a critical role. Since the web service system is composed of multiple services, each service may have multiple instances, and each service instance also has an access path, such as IP, Port, Path, and Query, which is usually represented by URL. Additionally, the instance itself is dynamically deployed by the container scheduling software. Therefore, the microservice framework usually uses Service Proxy or API Gateway as the external unified entry point (IP, PORT) to reduce certificates, domains, firewall management, and traffic difficulties in monitoring. Service Proxy usually combines the service discovery mechanism to delay binding user requests to service instances. It is useful for solving service access path changes caused by container scheduling and achieving load balancing among multiple service instances.

However, service discovery (SD) becomes a critical and complicated issue when microservices are deployed to fog-computing nodes. In the cloud environment, the service discovery mechanism is usually constructed in Kubernetes and Swarm clusters. The service container regularly registers its information with the service discovery mechanism through the client–server model. When the service agent receives a request, the request is redirected to an available server through the lookup table. Nonetheless, in fog computing, the client is no longer restricted to only accessing resources located in the cloud but can request resources from neighboring fog nodes. Moreover, fog-computing nodes are distributed across different network levels, including the gateway; access network; core network; cloud; and node holders, such as end users, company organizations, Internet Service Provider (ISP) operators, Content Delivery Network (CDN) operators, cloud operators, and service leasing/providers. Therefore, it is challenging to use centralized management mechanisms for service discovery because service instances are scattered among the fog nodes at different network levels, and it is difficult to register and query through the client–server method. Achieving a globally and publicly accessible distributed service registration and discovery mechanism becomes a critical issue for fog computing.

As previously discussed, we propose an IOTA-based service discovery (IBSD) framework in this paper. The proposed framework can support service publishers to publish service information, such as version, ownership, image file location, and service description, by using transactions. On the other hand, it also enables fog-computing nodes to register node information and deploy services into transactions. The entire service registry table is stored in the IOTA transaction and is distributed to all IOTA nodes as copies. When a user application attempts a service, it can query any IOTA node for transaction content to obtain a list of the service instances. Therefore, the proposed mechanism can achieve the goal of a distributed service query and simultaneously ensures correct data transmission through distributed ledger technology. Compared with related work, the main contribution of the proposed framework is that it is the first work that makes use of IOTA for resolving the service discovery problem for fog computing. Due to the advantages of IOTA, it can support a distributed, reliable, and zero-fee service discovery for fog-computing applications and can avoid the cyberattack to Distributed Hash Table (DHT)-based service discovery mechanisms.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 introduces the framework of IBSD. Section 4 discusses the performance evaluation of IBSD. Finally, Section 5 gives the conclusions of this paper and our future work.

2. Related Work

Several researchers have addressed the SD issue for fog computing. Their methods are briefly described as follows. Chii Chang et al. [9] proposed the Indie Fog architecture for service providers to purchases the remaining computing resources from consumers to provide other clients for use. Indie Fog currently supports only the global service registry table but adaptive and joint service registration and service discovery. Simone Cirani et al. [10] designed a peer-to-peer architecture of automatic service discovery for large-scale IoT networks. Their architecture uses Constrained Application Protocol (CoAP)-based service catalogs and exploits the Distributed Location Service (DLS) and Distributed Geographic Table (DGT) for service registration and query. The DLS is a name resolution service based on the Distributed Hash Table (DHT) to access a resource. The DGT is used to retrieve a list of resources matching geographic conditions based on a distributed node location database. Giacomo Tanganelli et al. [11] used the CoRE Resource Directory and DHT for service discovery while integrating service agents and redirection in IoT Gateway. Consequently, the SD requests of clients are transmitted to gateways for looking up the DHT and then are redirected to the CoAP server to retrieve the location of service instances. Sander Soo et al. [12] used the Mobile Ad hoc Social Network (MASN) for service discovery and computation migration among fog nodes. They also used the DHT for information exchange among fog nodes, because clients cannot always access the service registration table. Julien Gedeon et al. [13] adopt a hierarchical architecture including the client, surrogate, broker, and registry server for SD. In this architecture, the surrogate is any device that has additional computation capability to execute client tasks. The broker is responsible for maintaining the local registration table and responding to the SD requests of clients. It also plays one node of the Chord DHT and registers itself into the broker registry for information exchange among brokers. Through the overlay of the DHT, the distributed brokers can collaboratively enable clients to find suitable surrogates for computational offloading.

In addition to DHT-based SD, Kazuya Okada et al. [14] exploited DNS-SD, mDNS, and IP anycast to achieve the cooperation of SD among MEC nodes. Hessam Moeini [15] used applied ontology coding but the hash function on a distributed registry table to build a routing protocol of SD for increasing the correlation among neighbor nodes. Yuuichi Teranishi et al. [16] created an overlay network protocol called the Locality-Aware Service discovery protocol for the K-nearest (LASK) search to support scalable and locality-aware distributed k-Nearest Service Discovery (kNSD).

As previously described, most researchers have used the DHT for the implementation of SD. However, the DHT must face the Sybil attack [17], Eclipse attack, and Pollution attack [18] A Sybil attack interferes with the operation of the DHT to make information retrieval fail by using multiple fake identifiers. An Eclipse attack prevents a target node from communicating with any other peer except the attacker by taking over the target node's routing table. A Pollution attack inserts a large amount of invalid information into the index to prevent users from finding the right resource. Due to these attacks, the DHT-based SD mechanisms face a significant challenge. It is essential to develop a novel SD mechanism that does not rely on the DHT for fog computing.

Therefore, some researchers have recently devoted themselves to developing blockchain-based instead of DHT-based service discovery mechanisms due to the rise in decentralized ledger technology. For instance, Yao Zhao et al. [19] combined Universal Description, Discovery and Integration (UDDI) with smart contracts to store the service registration information of service providers through the alliance chain. The UDDI registry only must manage the authority of registrants and inquirers to operate the license contract. Additionally, Zhenfeng Gao et al. [20] used INKchain to build a consortium chain to form a Decentralized Service Ecosystem (DSES) for achieving decentralization through blockchain and solving problems, such as trust issues, security and privacy, lack of incentives, and maintenance costs. Jun Wu [21] et al. proposed an advanced social networking architecture to secure fog-computing services. In this architecture, they designed a crowd sensing-

enabled security recommendation method to realize security service recommendations. Shreshth Tuli et al. [22] proposed a framework named FogBus for integrating IoT-enabled systems, Fog, and Cloud infrastructure. This framework can help developers to overcome resource heterogeneity and efficiently harnesses edge and remote resources for deploying and executing their applications according to application requirements, and it applies blockchain for data integrity when applications transfer confidential data.

The above research results show that the service discovery based on the distributed ledger technology solves the throughput bottleneck of the centralized architecture and the problem of centralized management. Compared with the DHT, distributed ledger technology has the advantages of transmission security and immutable data. However, the decentralized ledger technologies used in current research are almost all based on alliance chains and private chains, limiting the speed of node joining and information access. The main reason for this is that the need to pay miner fees for sending information on the public chain and the proof of work (POW) under the public chain degrades the throughput of transaction generation. According to the statistics of bitinfocharts.com [23] and ethgasstation.info [24], the service discovery mechanism built on the Ethereum public chain generates one block every 13.4 s, and the average transaction volume per second is 14.78TPS. It has to spend ETH 0.0094 (USD 16.76) on adding the published information into a block within 2 min.

Fortunately, the emergence of IOTA [25] effectively solves the previous problems. It does not require transaction fees and also has high throughput characteristics. It successfully reaches over 1000TPS in the official stress test. Additionally, the IOTA uses Tangle that is different from the blockchain. A transaction does not need to wait for the creation of a block to be incorporated into the chain. Instead, it can be initiated, distributed, and stored as long as the other two transactions are verified. The information storage and dissemination technology used by IOTA is called masked authenticated messaging (MAM) [26]. In recent years, it has also been applied to IoT and medical information research [27–29].

On the other hand, the IOTA Foundation is currently implementing mana, a reputation system to avoid Sybil attacks. When a value transaction is processed, a quantity called mana will be "pledged" to a specified node ID. This quantity is related to the number of tokens moved into the transaction. The mana pledged to each node ID is stored as an extension of the ledger. This process generates a reputation system to distinguish trusted nodes from new, and thus possibly malicious nodes. Therefore, mana provides adequate protection against Sybil attacks.

Moreover, the IOTA Foundation proposes an autopeering mechanism to ensure that the network is secure against Eclipse attacks. The autopeering mechanism is logically divided into two submodules: peer discovery and neighbor selection. The former is responsible for operations, such as discovering new peers and verifying their online status. The latter is responsible for finding and managing neighbors for IOTA's nodes. The neighbors of a node are divided into two groups. One group is chosen neighbors, which the node chooses from its list. Another is accepted neighbors, which choose the node as their peer. The neighbor selection is determined by a distance function of public salt and private salt, which defends against dictionary attacks or their hash equivalent. The public and private salts can help create an asymmetric awareness to discourage an attacker from harming the system. In the autopeering process, the only way to target a node is brute forcing different node identities to become closer (in terms of distance) than an existing neighbor. To prevent brute force attacks, IOTA lets the salts be valid only for a certain amount of time, after which the node updates both its chosen neighbors and its accepted neighbors. This frequent reorganization increases the difficulty of attacking a specific node. Further analysis of the autopeering mechanism can be found in the report in [30].

As previously described, this research adopts IOTA with MAM to implement the proposed service discovery framework under the public chain. This work is focused on developing a distributed architecture to discover fog services securely and efficiently but to recommend which fog service is more secure compared to Jun Wu's work. Different from

Fogbus, this work uses blockchain not only for maintaining data integrity in node-to-node communication but also for building an open, reliable, and decentralized fog-computing service discovery framework.

3. IBSD

IBSD [31] is a service discovery framework designed and implemented for fog-computing microservices. There was a service discovery mechanism in the original microservice framework to solve the mutual conversion relationship between service instances and service namespaces. The service discovery mechanism of the previous microservice framework is mainly to manage the services of the master–slave architecture in the cluster. In contrast, fog-computing routines are distributed under different providers and managers. It is difficult to manage through a master–slave architecture. Therefore, IBSD uses the IOTA distributed ledger technology to achieve the consistency of decentralized service discovery and management through the consensus algorithm.

IOTA is a decentralized ledger technology proposed by David Sønstebø, Sergey Ivancheglo, Dominik Schiener, and Dr. Serguei Popov in 2015. It is dedicated to the Internet of Things and peer-to-peer networks. Unlike Bitcoin and Ethereum, IOTA uses Tangle instead of blocks to store transactions, as shown in Figure 1. Except for the first genesis transaction, the rest of the transactions are created by selecting two transactions based on the Markov Chain Monte Carlo (MCMC) algorithm and verifying if they are legal at first. Since IOTA requires users only to verify the other two transactions by themselves when they initiate a transaction, it eliminates the dependence of the distributed ledger on miners. Additionally, the higher the number of transactions initiated per unit time is, the higher the throughput of transaction verification is.

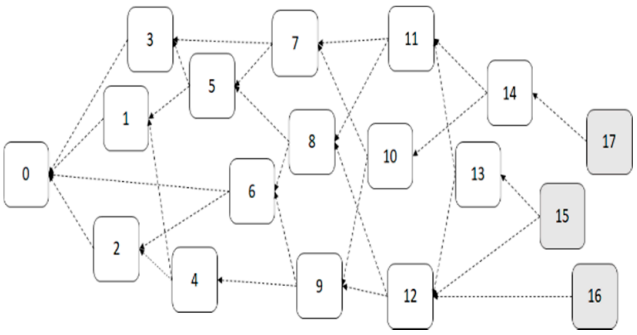


Figure 1. Tangle network.

IOTA transactions (TX) are classified into Input TX and Output TX. Input TX is used to withdraw money from the address, while it cannot be a zero-value transaction. In contrast, output TX is used to collect payments or record data. It can be valued or zero-valued transactions. IOTA node can pack several transactions into a bundle and check the transactions in a bundle unit. The sum of value in a bundle must be zero to prove that the transaction is a break-even one. All transactions in the bundle are either accepted or rejected together. After the bundle is accepted, the internally recorded transactions are added to the Tangle ledger.

IOTA reference implementation (IRI) has been proposed, which is a set of JAVA open-source software. IRI is currently run in public nodes. Clients can use IRI to join the IOTA network and transfer the IOTA token to one another. IRI has three main functions: (1) verifying transactions, (2) storing and disseminating, and (3) interacting with the client library. In the message transmission, IRI uses the gossip protocol to disseminate transactions. The IRI node must compare the database itself when a neighbor sends a transaction to it. If the database does not store the transaction, the IRI node must verify the transaction and store it in the database, then forward the transaction to the neighbor.

When the client wants to initiate a transaction, the following steps will be performed. (1) Prepare Input TX and Output TX then add them to a bundle. The sum of the bundle's value must be zero. (2) Use the sponge function to hash all TXs to generate the Bundle Hash. TXs include Address, Value, Tag, Timestamp, currentIndex, lastIndex, and other information. After that, fill the Bundle Hash into the TX. (3) Generate the Input TX's signature by Bundle Hash and private key, and fill the signature into the Input TX. (4) Use getTransactionsToApprove of API to obtain Tips from IRI, and fill Trunk and Branch into each TX. (5) Perform PoW for each TX of Bundle, and fill Hash into Nonce. (6) Finally, send the Bundle to the IRI network, then perform verify, store and broadcast to the IRI network to complete the transaction.

On the other hand, the transaction query process is executed as follows: (1) Use the Hash value of Bundle, Address, Tags, and Approves to call the findTransaction functionand return the transaction's Hash value (2) Parse the Hash to view the detailed transaction content by the getBundle function. Additionally, IOTA uses the MAM framework to prove the message's ownership and prevent interference from spam. MAM uses the cotyledons of the Merkle hash tree to generate a signature. The Root generates an address. The signature prevents the sender from being impersonated. The Root is used to calculate the address where the message is sent and record the Root of the next generation hash tree. Through the hash tree's generational alternation, IOTA prevents sending messages to the same address for a long time to avoid spam attacks.

3.1. Framework

The architecture of IBSD is shown in Figure 2. The inner components of a fog/edge node are IRI, IBSD services, and a microservice framework for operation. IRI is a node in the IOTA network. The fog node can send transactions through IRI and also receive transactions from other IRI nodes. In this study, the IBSD service combined with IRI was used to distribute the fog node's service information to all IOTA nodes through transactions. The user applications can exploit IBSD-API to search for a service from any IRI node in the IOTA network and send a service request to the fog node that provides the service. On the other hand, the IBSD service also is used to monitor transactions sent by other fog nodes that provide the service, thereby maintaining a list of fog nodes for the same service. The user applications can obtain a complete list of fog nodes through IBSD-API to select the best service node by more detailed node information and obtain better service quality.

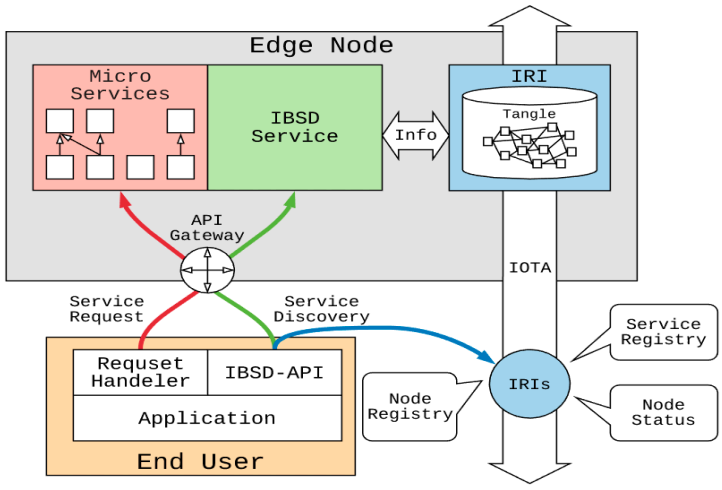


Figure 2. The architecture of IOTA-based service discovery (IBSD).

3.2. Design Issues

How IBSD exploits IOTA transactions for different information exchanges to achieve service discovery is described as follows.

In IBSD, each IOTA transaction has an independent Hash value. User applications can obtain the Hash value of a transaction by searching for transaction address (Address), transaction bundle (Bundle), transaction tag (Tag), and indirectly verified subtransaction (Approves) through IRI API, and then request the details of the transaction through the Hash value. Service nodes use general or MAM transactions to pack the released information. For the general transaction, it can choose the transaction address to be sent. If it (i.e., the fog node) is not the owner address, it cannot issue the Winternitz type one-time (W-OTS) signature.

On the other hand, MAM uses a bundle of transactions to release information. Each bundle contains a W-OTS signature, the Root of the next message, and the released information. The transaction address can be Root or the Hash value of Root according to different MAM modes. Service publishers can register a service through MAM. The Root of the MAM message is the identification code of the service. The signature is used to prevent others from forging it. Similarly, fog nodes also release their node information through MAM, including the service URI and a service identification code list used to record the providing services, and continuously release the update of node information through the message chain's characteristics.

For service discovery, it is necessary to have a method to map between the service namespace and the information of service nodes. As shown in Figure 3, IBSD uses the MAM Root of the node information as the transaction content and sends it through a general transaction to the address calculated by the service ID as the basic concept of service node registration. When a user application intends to search for a service, it can search the transaction address of the service by the service ID and then obtain the registered transaction. Finally, it can obtain the service node information from the MAM message chain recorded in the transaction content.

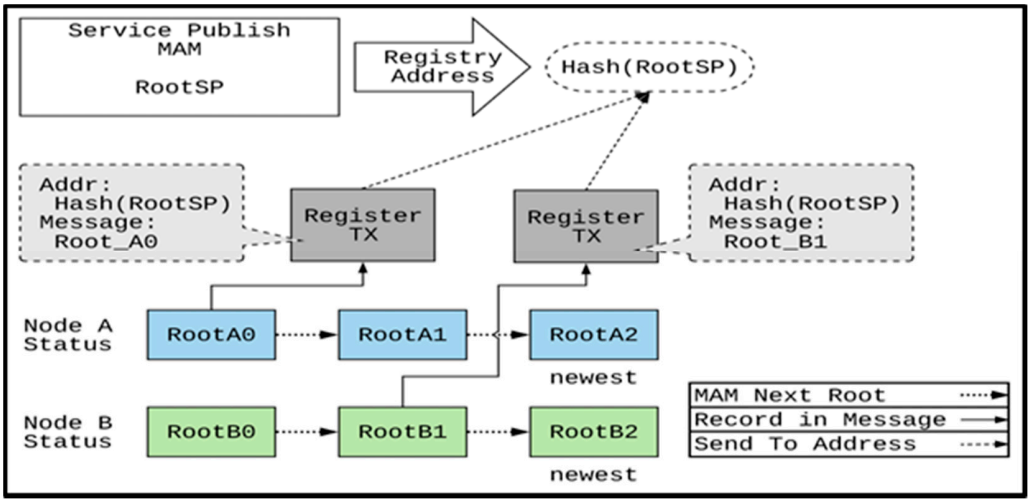


Figure 3. Registration of service nodes.

However, the previous design does not consider the following problems. (1) Service suspension of nodes: when a transaction is released, it will exist in IOTA for a long time. Even if the service is suspended, the registered transaction can still be searched, which reduces service discovery effectiveness. (2) The MAM transaction chain of the node status is too long. Even if a node maintains a service for a long time, it may still release new MAM transactions due to other service changes. Consequently, it may take a long time to traverse the recorded MAM transaction chain to obtain the latest node state. (3) The search range is too broad. Even the timestamp can solve the first problem. The repetitive registration at the same address will still cause the search range to increase over time continuously. We explain how to solve the first problem in Section 3.3 and propose an indexing mechanism to solve the second and third problems in Section 3.4.

3.3. Service Release and Deployment

For service discovery, each service needs a unique identifier. This study uses the Root of the MAM transaction to be the service identification code. The Root of the MAM transaction is a code generated by Merkle Tree, expressed by 27 characters, such as capital A-Z and Arabic numeral 9. When the Root of a message is created, MAM uses W-OTS to sign the message and thereby ensure the ownership of the service registration message and the uniqueness of the service identification code.

The service release and deployment processes are shown in Figure 4. After the service publisher adds the service program into a container image file, it can be uploaded to the image file registry (such as Docker Image Registry) for the service provider to deploy the service. To ensure the consistency of the information between the two parties, the service publisher can publish the service description, service license, deployment method, and other information through MAM. Since the Root of the posted message is the identification code of the service, the information on the service can be found by the service identification code.

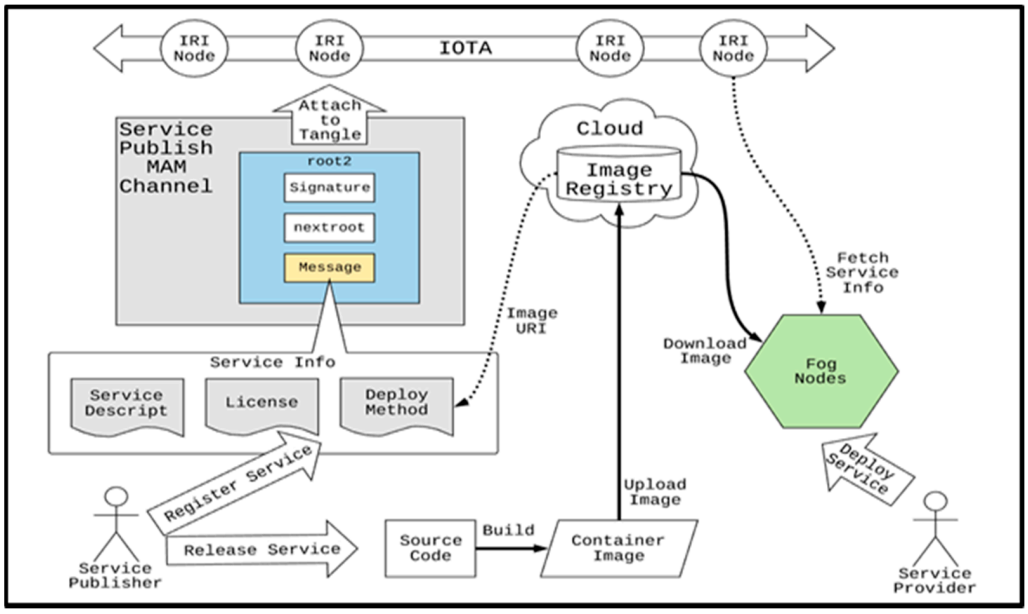


Figure 4. Service release and deployment in IBSD.

The service provider can drive the fog node to deploy the service by the service identification code. When the fog node receives the service's identification code to be deployed, IBSD obtains the service's deployment method from the MAM transaction, deploys the service to the node, and then registers the service node by the mechanism in the subsequent chapter. When user applications use the IBSD API to search for the service identification code, the fog node providing the service can be found.

3.4. Node Information Releasing

This research adopts periodically releasing node information to confirm the accessibility of nodes and services, as shown in Figure 5. The fog-computing nodes periodically release their node status information through CRON. The node status information includes (1) node information, such as IP and node identification number; (2) service list: the identification codes of the services hosted by the fog node; (3) service status: the current operating status of the node service, such as average response time and success rate. Currently, the fog nodes release their information every second. When a fog node does not release its service status, it is regarded as not alive.

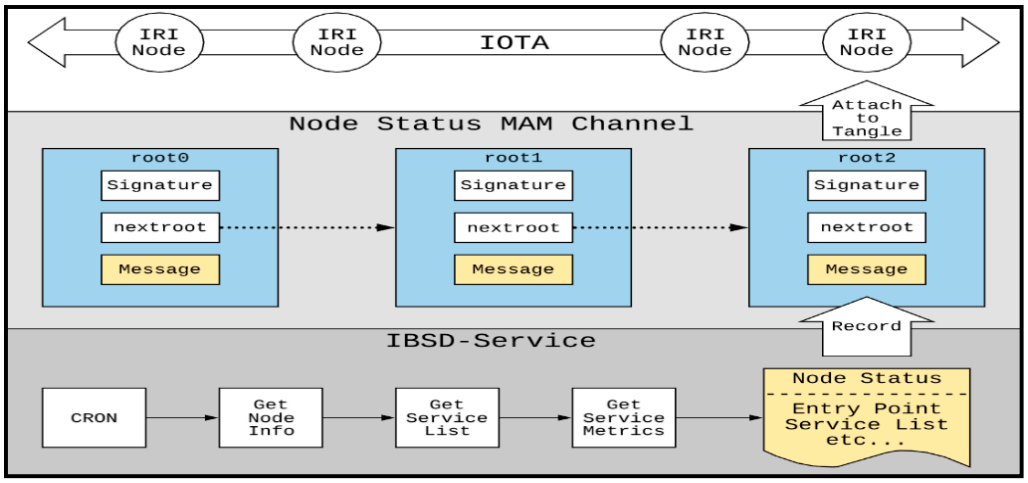


Figure 5. Node information releasing.

3.5. Registration Addressing, and Indexing

This research adopts the registration of service nodes for enabling clients to obtain the status information of the service nodes from the service namespace. The service nodes send the MAM Root of the node status to the service releasing address, and then clients can search this address to find the information of the service nodes. However, the transactions on the IOTA network are stored for a long time. Even if a node has stopped providing a service, the transactions related to the node's service registration information can still be queried by clients later. Therefore, a mechanism of invaliding the out-of-date information is necessary. Additionally, it may take a long time to search from the MAM Root to the latest status because the node status information is continuously released. This study proposes time-difference addressing and tree-based MAM (TBMAM) indexing for transaction searching to solve the previous problems.

3.5.1. Time-Difference Registration Addressing

This study uses a time-difference registration addressing [32] based on IOTA's address generation method, as shown in Figure 6.

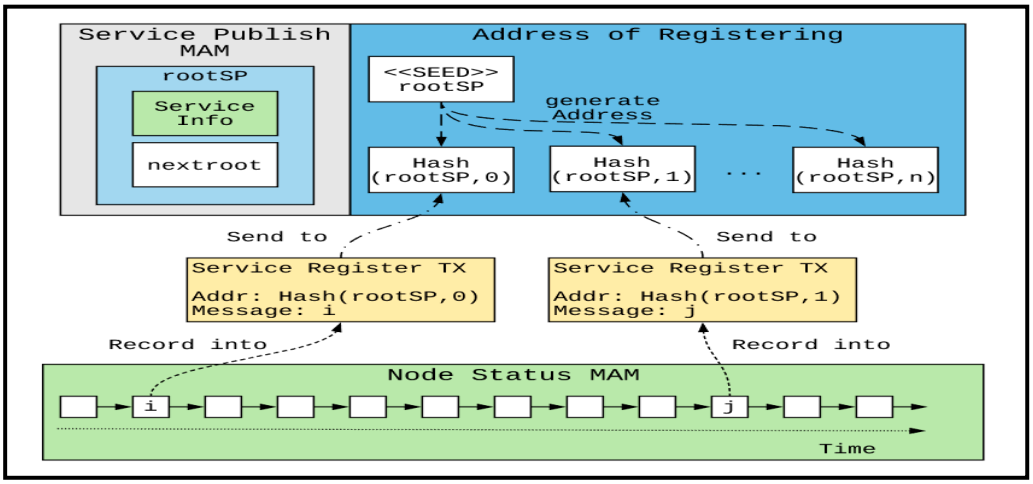


Figure 6. Time-difference registration addressing.

On IOTA, the address is generated by hashing the Seed of 81Tryte and the positive integer index, and each address can only be paid once. The wallet program is usually designed to send the transaction balance to the next index's address for storage. This research uses the MAM Root released by the service as the Seed and uses the time-unit

difference between the service release time and the current registration time as the index. As a result, the registration address of the service node changes with time. When clients search which fog node is available for a given service by the address, they can effectively filter out the registered transactions within the time range.

3.5.2. TBMAM Indexing

MAM transactions are originally linked by a one-way indexing chain. If MAM transactions are used for recording time series data, the transaction-chain length continues to increase with time. It finally results in a long search range from the start index to the latest one. To speed up transaction searching, we implement a tree-based MAM indexing mechanism called TBMAM in this study. Through the hierarchical indexing chains for different time–length intervals, a tree-based indexing mechanism is formed, as shown in Figure 7. Assume that the frequency of transaction generation is one per minute. Every time 60 MAM transactions are released, the latest transaction address in the minute-level chain is added into the hour-level chain. Every time the addresses of 24 MAM transactions are added into the hour-level chain, the latest MAM transaction address in the hour-level chain is added to the day-level chain.

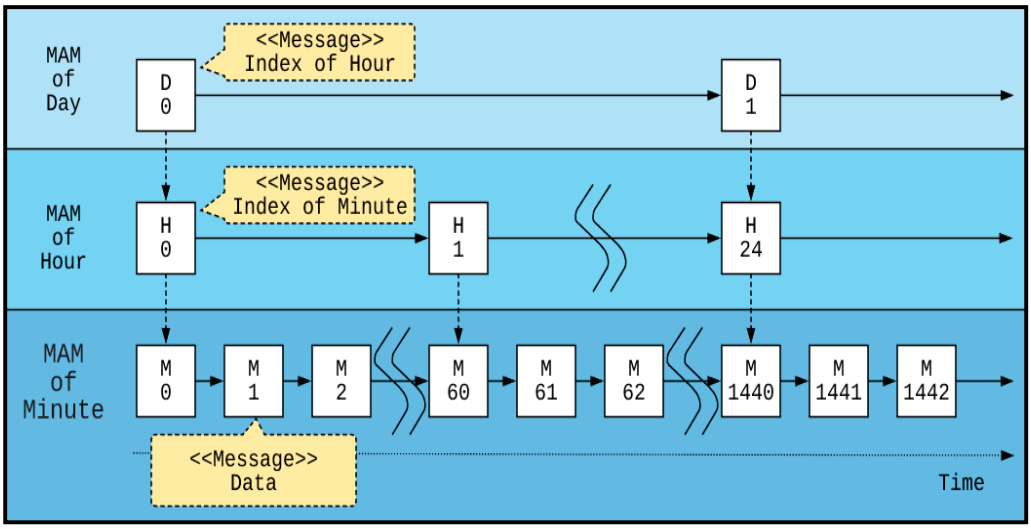


Figure 7. Tree-based masked authenticated message (TBMAM) indexing.

When registering a service node, the latest MAM Root in the top-level chain in TBMAM is used as the entry point for searching the status information of service nodes. As the upper chain in TBMAM has a large time–length interval for horizontal traversal, the tree-like hierarchical traversal reduces the number of steps required to find the latest node state from the transaction of node registration. For example, if a client wants to access the 62nd transaction, the search path is D0-H0-H1-M60-M61. It requires traveling 5 but 62 steps. As a result, TBMAM indexing is useful for reducing the time required to obtain the newest status of a given fog node.

3.6. IBSD Node Composition

The IBSD nodes' composition is shown in Figure 8, divided into four parts: database data collection, dependent software, service composition, and request source.

The database collection is used to record the relevant data required for the operation of IBSD, which is stored and described in the form of Document-Oriented using MongoDB. NodeStatus records node information, settings, and service index values. ServiceList records the services deployed by the node and detailed information on the services. NodeList records other nodes which provide the same services. NamespaceMapping records the correspondence between TBMAM and nodes. ListeningAddress records the

transaction address that is being monitored. MAMChannel keeps track of the current status of the MAM message chain.

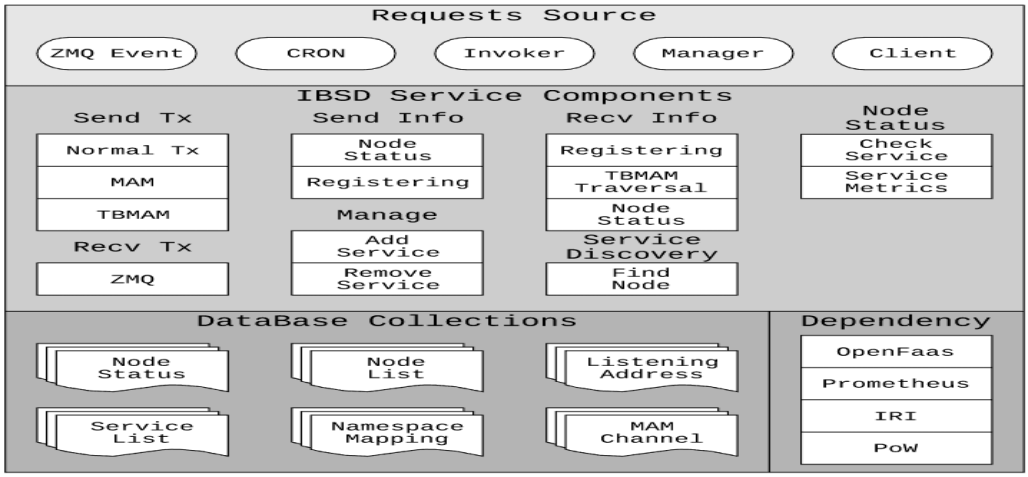


Figure 8. IBSD node composition.

Dependent software is the software or API required by IBSD, mainly including OpenFaas, Prometheus, IRI, and PoW. OpenFaaS is a framework for running and managing all the microservices used by clients and IBSD. Prometheus is a Time Series Database (TSDB) used to store service call information, including service call time, call times, and running time. IRI is the software that runs IOTA nodes, used to verify, store, query, and distribute transactions. In this paper, the node information is packaged as IOTA transactions, and the transactions are sent to IOTA-Mainnet through IRI to exchange data between nodes. PoW is the proof of work required to send IOTA transactions, which can be completed within IRI or handed over to PoW services provided by other computing resources.

The IBSD service composition is a collection of services required to implement the IBSD mechanism. According to different functions, it can be divided into categories such as sending transactions, sending information, receiving transactions, receiving information, node status, system management, and service discovery. Through the information release mechanism described in Section 3.4 and the registration and indexing mechanisms mentioned in Section 3.5, fog nodes can exchange status information with one another through the IOTA distributed ledger technology. In IBSD, the fog node pays attention to the information of other nodes with the same service. The mechanism is as shown in Figure 9.

After the service deployment is completed, IBSD first queries the recent registration records of service nodes and adds the registered address to the monitoring list. For the service node registration transaction found, the TBMAM indexing is performed, and the latest node information is updated or added to the database. Finally, the nextRoot of the updated node status information is also added to the monitoring list. On the other hand, when IRI receives a new transaction, it checks whether the transaction address is in the monitoring list. If it is not, no process is performed. If it is, a new service node registration transaction is performed. Additionally, TBMAM indexing is performed to obtain the latest node status, and the database is updated to monitor the next sending address of MAM. If it is the new node status information, the database is updated, and the next sending address of MAM is monitored.

Finally, the request source is the event source that triggers all IBSD microservices, including the ZMQ event triggered when the transaction is received by IRI, the regularly executed CRON event, mutual calls between services, commands issued by managers, and client requests.

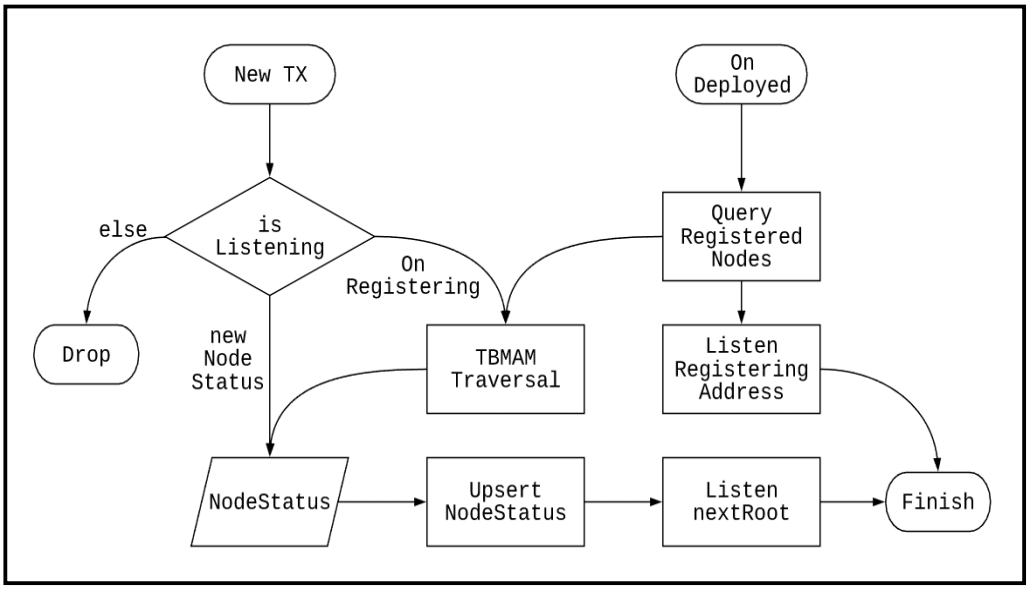


Figure 9. The mechanism of monitoring the nodes with the same service.

3.7. Service Discovery

The service discovery process is divided into two stages. The first is searching for service node registration information through IOTA, and the second is optimizing node selection through IBSD. Their execution flows are shown in Figure 10.

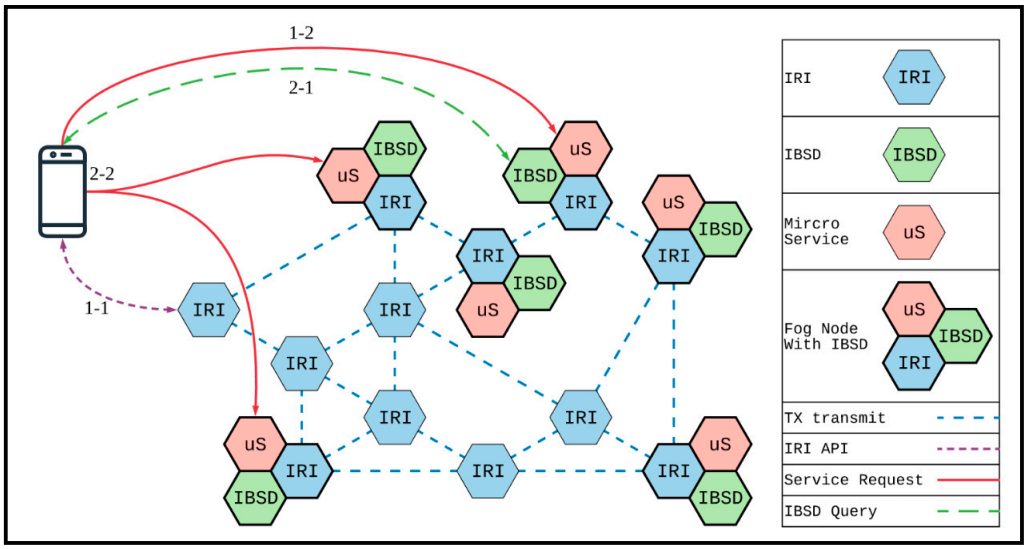


Figure 10. Execution flow of service discovery.

The first stage is mainly aimed at dealing with the case of when no service node is known yet. At first, the client queries any IRI node for the registered service's transaction by the service identifier code. By analyzing the transaction, it can retrieve the information of the registering node by the TMAMA traversal. Then, it sends a request to the registering node and obtains the state of the available service node.

However, it is not guaranteed to find the most suitable node for the client. Therefore, the second stage aims to search for more service nodes and select the best one from the nodes. The first step is to ask the IBSD service about any known service nodes. Since each fog node of IBSD can provide the state information of the nodes deploying the same service, the client can select the best node according to the node-state information and then ask the selected node to provide the required service.

In the first stage, we designed a location-based information filter for finding the service nodes close to the client in terms of geographical location. When a fog node registers itself, the registration transaction's tag field is filled with its geographical location, such as country, state, city, and company. By contrast, the client can translate its IP address into the geographical location by GeoLite [33]. Through geographical location comparison, the client can find nearby service nodes. On the other hand, when service nodes release their states, they add their average execution time into the registered state information. When a client obtains a list of available nodes, it can obtain the average execution time of each service node. Moreover, the client actively estimates the network latency between each service node and itself through an HTTP-ping function implemented by this work. Finally, it can predict the response time of each service node by adding the network latency to the average execution time and then choose the node with the shortest predicted response time to provide service.

4. Performance Evaluation

We evaluated the performance of IBSD in this paper. First of all, we measured the cost of sending transactions to the IOTA Mainnet for propagating service messages. Next, we estimated the cost of service discovery and the impact of TBMAM indexing. Finally, we evaluated the cost and effectiveness of node selection. Our experimental environment was built by six PCs and four virtual machines of Google Cloud Platform (GCP), as shown in Table 1. PC-01, PC-02, and PC-03 were responsible for performing PoW by using GPU. PC-04, PC-05, and PC-06 were the IRI nodes. PC-06, GCP-TW, GCP-SG, GCP-JP, and GCP-US played the fog nodes of providing microservices and the IBSD functions for user clients. During performance evaluation, PC-06 used the local IRI service. By contrast, GCP-TW and GCP-JS used the IRI service of PC-02, while GCP-SG and GCP-US used the IRI services of PC-05.

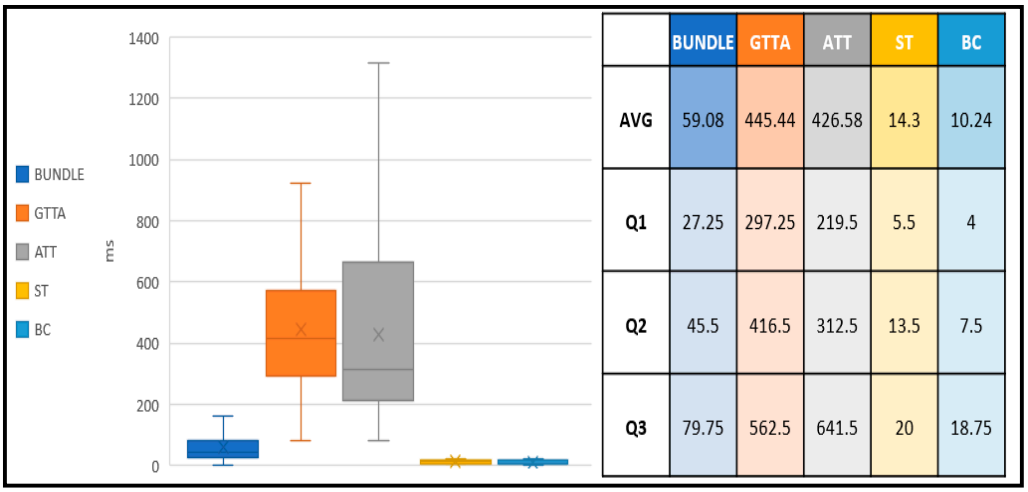
Table 1. Resources used in performance evaluation.

Host	CPU	RAM	Location	Function	Notes
PC-01	Xeon E5645	24 GB	Lab (TaNET)	PoW	Use GTX970 for PoW
PC-02	Xeon E5650	24 GB	Lab (TaNET)	PoW	Use GTX1080 for PoW
PC-03	Core i7-7700	32 GB	Lab (TaNET)	PoW	Use GTX1080Ti for PoW
PC-04	Core i7-8700	32 GB	Lab (TaNET)	IRI, IBSD	static peering
PC-05	Core i7-8700	32 GB	Lab (TaNET)	IRI	static peering
PC-06	Core i7-8700	32 GB	Lab (TaNET)	IRI	dynamic peering (Nelson)
GCP-TW	1vCPU	3.75 GB	Taiwan (GCP)	IBSD	-
GCP-SG	1vCPU	3.75 GB	Singapore (GCP)	IBSD	-
GCP-JP	1vCPU	3.75 GB	Japan (GCP)	IBSD	-
GCP-US	1vCPU	3.75 GB	US (GCP)	IBSD	-

4.1. Transaction Release Cost

IBSD makes use of IOTA transactions for information exchanges. The steps of transaction release are Bundle, getTransactionToApprove (GTTA), attachToTangle (ATT), strore-Transactions (ST), and broadcastTransactions (BC). In this experiment, each bundle has only one transaction. The depth parameter of transactionsToApprove is set as one, and the mwm parameter of attachToTangle is set as 14. We used the IRI of PC-06 to start a transaction and used the GPUof PC-03 for PoW. We measured the cost of a transaction 50 times and then calculated the average time of a transaction. The breakdown of one transaction release cost is shown in Figure 11.

In this table, BUNDLE represents the cost of transaction packing, and GATT is the cost of Tip Select. ATT denotes the cost of PoW. ST is the cost of storing a transaction to IRI. BC is the cost of ordering IRI to broadcast the transaction. Our experimental result shows that Tip Select and PoW are the main factors determining one transaction's cost.



	BUNDLE	GTTA	ATT	ST	BC
AVG	59.08	445.44	426.58	14.3	10.24
Q1	27.25	297.25	219.5	5.5	4
Q2	45.5	416.5	312.5	13.5	7.5
Q3	79.75	562.5	641.5	20	18.75

Figure 11. Breakdown of transaction release cost.

Moreover, the getTransactionToApprove function of IOTA-BT API can change the traversal depth of MCMC by the depth parameter. The influence of traversal depth on the cost of the TIP selection is depicted in Figure 12. The traversal depth influences if the transaction is bound to unreliable tips or not. The shorter the traversal depth is, the easier the transaction is attached to unreliable tips. However, the property of IOTA-BT is zero-fee transactions. The transaction verification does not affect the operation of IOTA-BT. Therefore, we set the traversal depth as one for reducing the transaction cost of IOTA-BT.

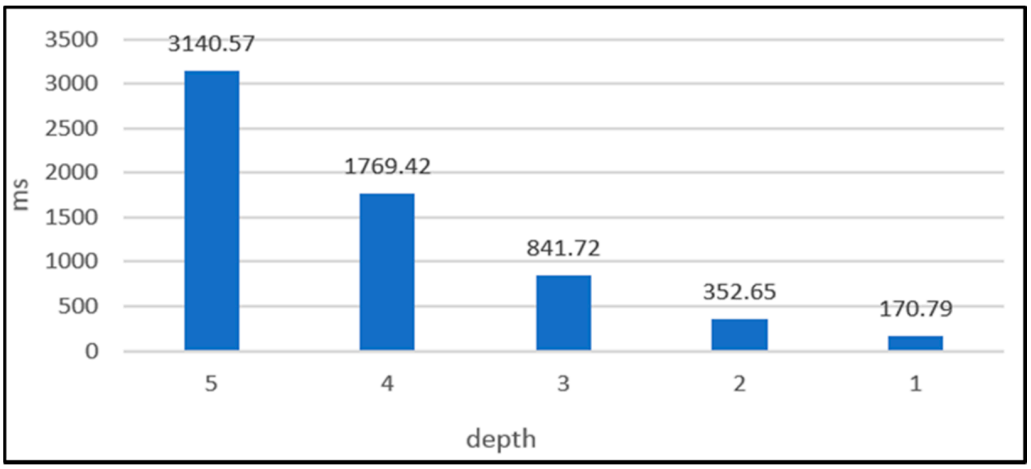


Figure 12. Cost of Tip Select.

In the IOTA Mainnet, the difficulty of PoW is ensured unless the mwm (minimum weight magnitude) parameter's value must be larger than or equal to 14. During this experiment, PCs used CPU to perform PoW through the ccurl library and OpenCL. By contrast, it used the original API of IRI and Java for executing PoW by CPU. The transaction content is statically assigned. We measured the cost of PoW in 500 transactions and obtained the average time cost of PoW. Our experimental result is as shown in Figure 13. In this figure, the orange charts represent the costs of PoW executed by different GPUs, while the blue charts denote the costs of PoW performed by different GPUs. It can be found that GPU is useful for reducing the cost of PoW compared with CPU because of its powerful computation capability.

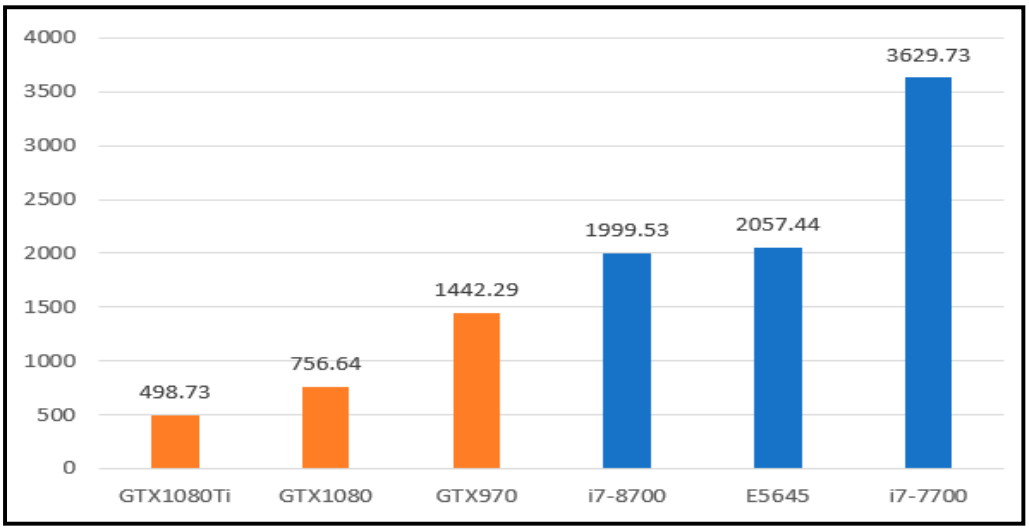


Figure 13. Costs of PoW executed by different resources.

In IBSD, if the hierarchical levels of TBMAM indexing are divided into 1, 3, 12, and 60 min, it is necessary to send 1.43 MAM transactions per minute, and each MAM transaction must consist of 3 typical transactions at least. Therefore, it is essential to send 4.3 typical transactions. Accordingly, the time spent on PoW per transaction must not be greater than 13.95 s, which is the lowest requirement of computation power of the resources used for applying IBSD in IOTA.

On the other hand, we also measured the cost of IOTA transaction propagation. We set up three IRI nodes in the same network section and let them join the IOTA Mainnet. However, none of these three nodes was a neighbor to each other. In this situation, a transaction sent by one node must be transmitted in the IOTA Mainnet through the gossip protocol to arrive at another. Since the time clocks at different nodes are not synchronous, using timestamps is not precise for estimating the cost of transaction propagation. Therefore, we used one node to send transactions and receive the transaction-reception messages coming from the other nodes through Zero Message Queue (ZMQ) to estimate the cost of transaction propagation. As shown in Figure 14, it spent 450 ms for propagating transactions to the tested nodes.

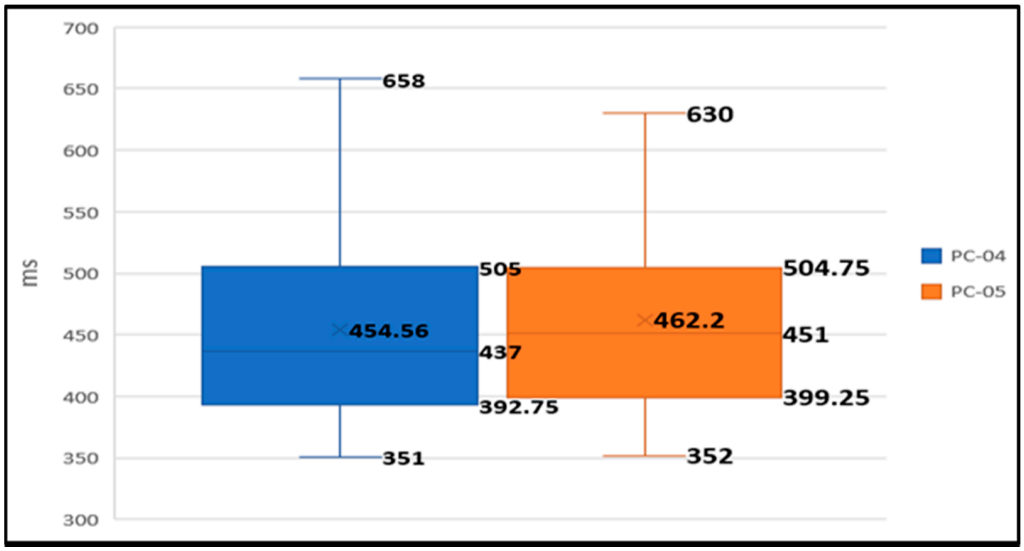


Figure 14. Cost of transaction propagation.

In Ethereum, appending a block into a public chain spends about 13~15 s. The priority of adding a block into the public chain is determined by how much fee is paid. By contrast, IOTA only needs to prove two old transactions for a new transaction. A fog node must spend only 2 s to finish the creation and storage of an MAM transaction with the support of the accelerators and spend only 450 ms to propagate the MAM transaction to other nodes with no pay. As a consequence, a service discovery mechanism based on IOTA is practical and economical.

4.2. Cost of Service Discovery

This experiment is aimed at measuring the cost of fetching and analyzing registration transactions and evaluating the impact of TBMAM indexing. First, we individually added 5, 10, 15, or 20 transactions at the same registered address and measured the fetching cost and analyzed a transaction by the registered address. Figure 15 shows the breakdown of transaction fetching and analysis. In the figure, Count Address denotes the cost of computing the registered address. The symbols of Find Tx and Get Tx represent the searching and fetching costs of a transaction. Decode Tx is the analysis cost of the transaction. The experimental result shows that Count Address (Hash function time) spends the most time cost. The cost of Find Tx and Get Tx is influenced by the IRI state rather than the number of transactions. By contrast, the cost of Decode Tx is increased as well as the number of transactions.

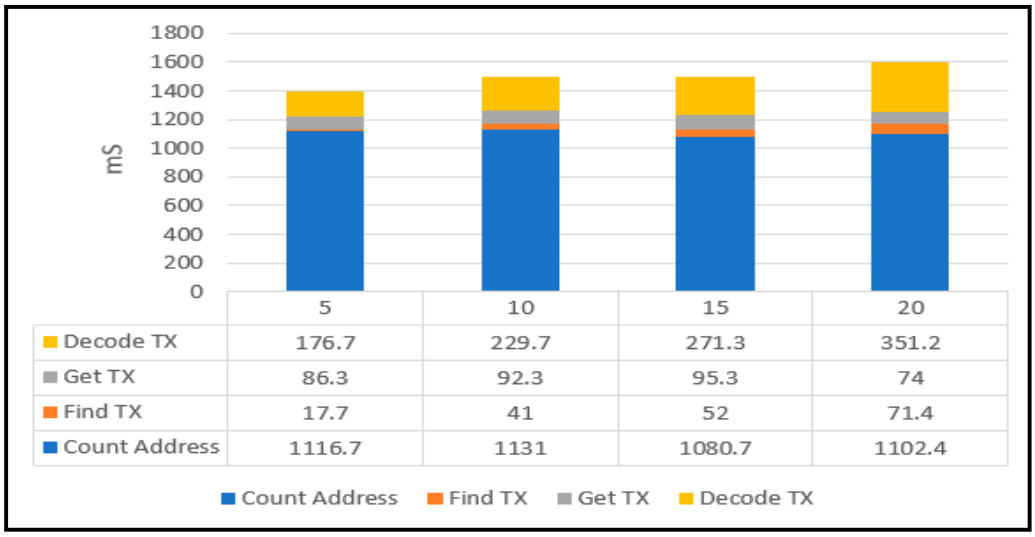


Figure 15. Cost of fetching and analyzing transaction.

On the other hand, we evaluated the impact of TBMAM indexing on the cost of obtaining the newest node state information as follows. The length of the MAM transactions was set as 60. The hierarchical MAM index chains were constructed from the low level to the high level for 1, 3, 12, and 60 min. Additionally, we used a private IRI node without the limit of an API access number and the public IRI node provided by Tangle (https://nodes.thetangle.org:443, accessed on 3 June 2019), which has the limit of an API access number.

As shown in Figure 16, through 1G Fast Ethernet or 4G, a client spent 1262 or 4690 ms to obtain the newest node state information from the private IRI node with the TBMAM indexing. If the IRI node does not support the TBMAM indexing, the client must spend 3548 or 14,131 ms to obtain the newest node state information. When the client queries the public IRI node for the newest node state, it spends 31,728 ms if the IRI node supports the TBMAM indexing. However, it must spend 233,357 ms if the IRI node does not support the TBMAM indexing. The previous result shows that the TBMAM indexing can reduce the cost of obtaining the new node state information.

Figure 16. Cost of tree-based MAM (TBMAM) indexing.

In IBSD, the fog node performs TBMAM indexing only when it searches for a new node. After that, it directly monitors the MAM nextRoot to obtain the node's information and store the node information in the local database. When a client queries the same node's information later, the fog node only needs to retrieve the node information from the local database and return the retrieved data back to the client. By comparing Figure 17 with Figure 16, querying the information of known nodes is much faster than querying the information on a new node, regardless of which IRI node is queried.

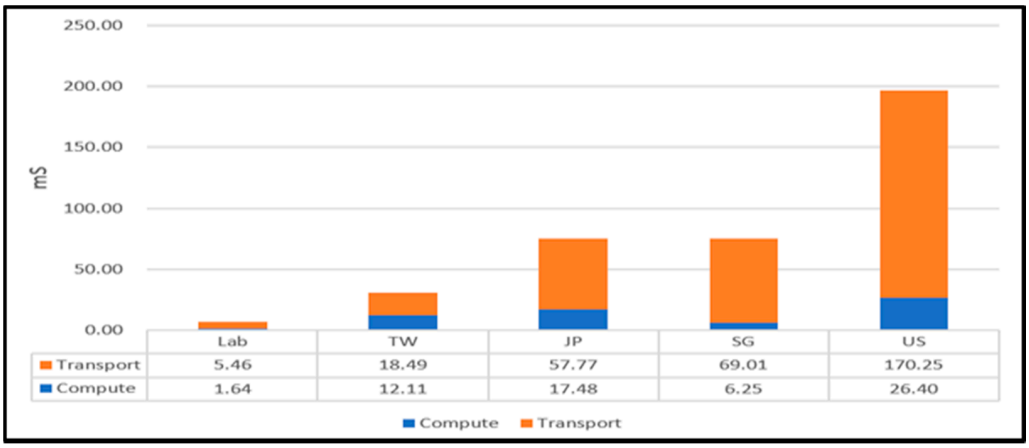


Figure 17. Cost of querying IRI for the information of known nodes.

4.3. Impact of Node Selection

In this experiment, we deployed a simple computing service on two PCs in our lab and four virtual machines in the GCPs of different countries. Additionally, we ran an application to ask the deployed service about a client node (i.e., a PC in our lab). After the client node queried the IBSD node for the service nodes' information, it obtained the recent execution time of each service node. Then, it estimated the network latency from the local to each service node through HTTP-ping and predicted the response time of each service node according to the execution time and network latency of the service node. Finally, it selected the node predicted with the lowest response time to be the service provider. The experimental result is shown in Figure 18. In this figure, the rel string represents real, and the est string denotes predicted. If the client considered only network latency, it would choose the PCs in our lab. By contrast, it would select the virtual machine in Japan's GCP if it considered only execution time. However, it could choose the best node, which is virtual machines in Taiwan if it took both execution time and network latency into account. This

result shows that the node selection mechanism indeed can help clients to obtain a better service quality.

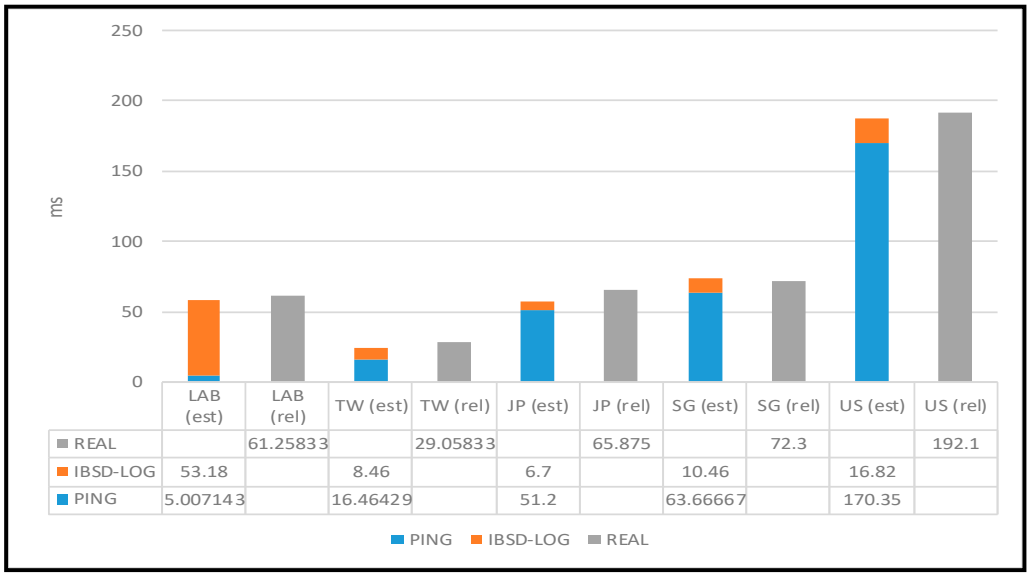


Figure 18. Impact of node selection.

5. Conclusions and Future Work

In this paper, we successfully developed an IOTA-based service discovery framework called IBSD for fog-computing applications. This framework provides higher security than the DHT to defend against cyberattacks, such as Sybil and Eclipse. Our experimental results also show that it also is more cost effective and accessible for service discovery than related work based on Ethereum public or private blockchains, because IOTA resolves the problems of proof fee and throughput in blockchains. Moreover, the proposed framework can provide useful node state information for selecting the best fog node to provide services with the shortest response time.

The IOTA Foundation currently provides digital certificates to all participating parties, and those parties initially trust the Decentralized Identifier (DID) of the IOTA Foundation. Eventually, the participants, such as service requestors or providers, can start to create digital certificates for their own actors. The methods for creating, reading, updating, and deactivating the DIDs and the associated DID documents are described in the respective specification (DID method) [34]. We will make use of the DID method for the authentication of valid service providers and requesters in the future. On the other hand, we will apply the proposed framework to P2P and edge computing applications. For example, we will develop an edge intelligence computing (EIC) environment based on the proposed framework to provide EIC services for IoT and mobile applications at the end of networks. On the other hand, we will develop a P2P crowdsourcing environment for mobile edge computing with the support of IBSD.

Author Contributions: Conceptualization, T.-Y.T.; Data curation, L.-Y.H.; Formal analysis, T.-Y.L.; Funding acquisition, T.-Y.L.; Investigation, T.-Y.T.; Methodology, T.-Y.T.; Project administration, T.-Y.L.; Resources, T.-Y.L.; Software, T.-Y.T. and L.-Y.H.; Supervision, T.-Y.L.; Validation, T.-Y.L.; Visualization, L.-Y.H.; Writing—original draft, T.-Y.T.; Writing—review & editing, T.-Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Technology in Taiwan grant number [MOST 107-2221-E-992-017-].

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Satyanarayanan, M.; Bahl, V.; Caceres, R.; Davies, N. The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Comput.* **2011**, *4*, 14–23. [CrossRef]
2. Satyanarayanan, M.; Chen, Z.; Ha, K.; Hu, W. Wolfgang Richter and Padmanabhan Pillai, Cloudlets: At the leading edge of mobile-cloud convergence. In Proceedings of the 6th International Conference on Mobile Computing, Applications and Services, Austin, TX, USA, 6–7 November 2014; pp. 1–9.
3. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16.
4. Hu, Y.C.; Patel, M.; Sabella, D. Nurit Sprecher and Valerie Young, Mobile edge computing—A key technology towards 5G. *ETSI White Pap.* **2015**, *11*, 1–16.
5. IEEE Standard Association. *IEEE 1934-2018-IEEE Standard for Adoption of Openfog Reference Architecture for fog Computing*; IEEE Standard Association: New York, NY, USA, 2018.
6. EdgeX Foundry. Available online: https://zh.wikipedia.org/wiki/EdgeX_Foundry (accessed on 12 February 2019).
7. IoT Greengrass. Available online: https://aws.amazon.com/tw/greengrass/ (accessed on 23 January 2019).
8. Papazoglou, M.P. Service-oriented computing: Concepts, characteristics, and directions. In Proceedings of the Fourth International Conference on Web Information Systems Engineering, Rome, Italy, 12 December 2003; pp. 3–12.
9. Chang, C.; Srirama, S.N.; Buyya, R. Indie Fog: An Efficient Fog-Computing Infrastructure for the Internet of Things. *Computer* **2017**, *50*, 92–98. [CrossRef]
10. Cirani, S.; Davoli, L.; Ferrari, G.; Leone, R.; Medagliani, P.; Picone, M.; Veltri, L. A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things. *IEEE Internet Things J.* **2014**, *1*, 508–521. [CrossRef]
11. Tanganelli, G.; Vallati, C.; Mingozzi, E. Edge-Centric Distributed Discovery and Access in the Internet of Things. *IEEE Internet Things J.* **2017**, *5*, 425–438. [CrossRef]
12. Sander, S.; Chang, C.; Srirama, S.N. Proactive service discovery in fog computing using mobile ad hoc social network in proximity. In Proceedings of the 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Chengdu, China, 15–18 December 2016; pp. 561–566.
13. Gedeon, J.; Meurisch, C.; Bhat, D.; Stein, M.; Wang, L.; Mühlhäuser, M. Router-based brokering for surrogate discovery in edge computing. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, USA, 5–8 June 2017; pp. 145–150.
14. Okada, K.; Kashihara, S.; Kawanishi, N.; Suzuki, N.; Sugiyama, K.; Kadobayashi, Y. Goedge: A scalable and stateless local breakout method. In Proceedings of the 2018 Workshop on Theory and Practice for Integrated Cloud, Fog and Edge Computing Paradigms, Egham, UK, 27 July 2018; pp. 29–34.
15. Moeini, H.; Yen, I.L.; Bastani, F. Routing in IoT network for dynamic service discovery. In Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, China, 15–17 December 2017; pp. 360–367.
16. Teranishi, Y.; Kimata, T.; Yamanaka, H.; Kawai, E.; Harai, H. Supporting k-nearest service discoveries for large-scale edge computing environments. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–7.
17. Wang, L.; Kangasharju, J. Real-world sybil attacks in bittorrent mainline DHT. In Proceedings of the 2012 IEEE Global Communications Conference (GLOBECOM), Anaheim, CA, USA, 3–7 December 2012; pp. 826–832.
18. Timpanaro, J.P.; Cholez, T. Isabelle Chrisment, and Olivier Festor, Bittorrent's mainline DHT security assessment. In Proceedings of the 2011 4th IFIP International Conference on New Technologies, Mobility, and Security, Paris, France, 7–10 February 2011; pp. 1–5.
19. Zhao, Y.; Tan, W.; Zhao, L. Blockchain-based UDDI data replication and sharing. In Proceedings of the 2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design (CSCWD), Nanjing, China, 9–11 May 2018; pp. 384–389.
20. Gao, Z.; Fan, T.; Wu, C.; Zhang, J.; Chen, C. DSES: A blockchain-powered decentralized service eco-system. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 25–32.
21. Jun, W.; Zhou, S.; Shen, W.; Jianhua, L. Crowd sensing-enabling security service recommendation for social fog computing systems. *Sensors* **2017**, *17*, 1744.
22. Tuli, S.; Mahmud, S.; Tuli, S.; Buyya, R. Fogbus: A blockchain-based lightweight framework for edge and fog computing. *J. Syst. Softw.* **2019**, *154*, 22–36. [CrossRef]
23. Ethereum/Ether (ETH) Statistics. Available online: https://bitinfocharts.com/ethereum/ (accessed on 18 March 2021).
24. ETH Gas Station. Available online: https://ethgasstation.info/ (accessed on 18 March 2021).
25. Popov, S. *The Tangle, Version 1.4.3*; IOTA Found: Berlin, Germany, 2018.
26. Mam.Client.js. Available online: https://github.com/iotaledger/mam.client.js/ (accessed on 12 January 2019).
27. Pinjala, S.K.; Sivalingam, K.M. DCACI: A decentralized lightweight capability based access control framework using IOTA for internet of things. In Proceedings of the 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 15–18 April 2019; pp. 13–18.

28. Lamtzidis, O.; Gialelis, J. An IOTA based distributed sensor node system. In Proceedings of the 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.
29. Brogan, J.; Baskaran, I.; Ramachandran, N. Authenticating Health Activity Data Using Distributed Ledger Technologies. *Comput. Struct. Biotechnol. J.* **2018**, *16*, 257–266. [CrossRef]
30. COORDICIDE Autopeering. Available online: https://blog.iota.org/coordicide-update-autopeering-part-1-fc72e21c7e11/ (accessed on 3 January 2019).
31. Tang, T.-Y. A Service Discovery Framework for Fog Computing Based on Distributed Ledger Technology. Master's Thesis, Department of Electrical Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan, 2019.
32. Hou, L.-Y.; Tang, T.-Y.; Liang, T.-Y. IOTA-BT: A P2P File-Sharing System Based on IOTA. *Electronics* **2020**, *9*, 1610. [CrossRef]
33. MAXMIND GeoLite2 Databases. Available online: https://dev.maxmind.com/geoip/geoip2/geolite2/ (accessed on 18 March 2021).
34. DID Methods. Available online: https://w3c.github.io/did-spec-registries/#did-methods (accessed on 18 March 2021).