IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received January 10, 2020, accepted January 18, 2020, date of publication January 21, 2020, date of current version January 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2968492

Fabric-iot: A Blockchain-Based Access Control System in IoT

HAN LIU, DEZHI HAN, AND DUN LI
College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China

Corresponding author: Dezhi Han (dzhan@shmtu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61672338 and Grant 61873160.

ABSTRACT IoT devices have some special characteristics, such as mobility, limited performance, and distributed deployment, which makes it difficult for traditional centralized access control methods to support access control in current large-scale IoT environment. To address these challenges, this paper proposes an access control system in IoT named fabric-iot, which is based on Hyperledger Fabric blockchain framework and attributed based access control (ABAC). The system contains three kinds of smart contracts, which are Device Contract (DC), Policy Contract (PC), and Access Contract (AC). DC provides a method to store the URL of resource data produced by devices, and a method to query it. PC provides functions to manage ABAC policies for admin users. AC is the core program to implement an access control method for normal users. Combined with ABAC and blockchain technology, fabric-iot can provide decentralized, fine-grained and dynamic access control management in IoT. To verify the performance of this system, two groups of simulation experiments are designed. The results show that fabric-iot can maintain high throughput in large-scale request environment and reach consensus efficiently in a distributed system to ensure data consistency.

INDEX TERMS Blockchain, IoT, ABAC, hyperledger fabric, distributed system.

I. INTRODUCTION

With the developments of Internet and computer hardware, more and more devices are connected with each other through wireless network, making the scale of Internet of Things(IoT) larger and larger. IoT is a distributed network composed of a large number of sensors and gateways. IoT devices interact with the environment all the time, producing different types of data resources, such as image, audio, video, digital signal, etc. All systems and applications of IoT can access the Internet to effificiently share resources and information [1]. That is to say, we are living in a world where everything is connected [2]. However, due to the distributed deployment of the IoT devices and their large number and scale, the access control of device resources is facing great challenges. The resources produced by IoT devices often contain privacy and sensitive data, so there will be serious consequences when they are obtained illegally. The access control technology is an important means to protect resources, which has been widely used in various systems and environments [3]. Traditional access control methods include discretionary access control (DAC), identity-based access control (IBAC), and mandatory access control (MAC), etc. But these methods are all centralized designs, which have the disadvantages of single-point failure, difficult to expand, low reliability and low throughput. In fact, IoT devices may belong to different organizations or users, and probably have mobility and limited performance, which make centralized access control difficult to meet the requirements of access control in IoT environment. Attributed based access control (ABAC) is a logical access control model, which controls the access between subjects and objects, according to the attributes of entries, operations and related environments [4]. ABAC firstly extracts the attributes of user (subject), resource (object), permission and environment respectively, then combines the relationship of these attributes flexibly, and finally transforms the management of permission into the management of attribute, providing a fine-grained and dynamic access management method. Blockchain [5] is another kind of emerging data management technology, it ensures the reliability of data through distributed storage. The reading or modification record of data is written into a block as a transaction, and the blocks are linked by hash algorithm as a chain to ensure the integrity of the data. It synchronizes data between nodes through P2P network and consensus algorithm, which makes all parties involved in blockchain network reach a consensus, thus

The associate editor coordinating the review of this manuscript and approving it for publication was Hong-Ning Dai.

ensuring data consistency. Hyperledger Fabric [6] is an open-source blockchain development platform, which not only has the characteristics of blockchain such as decentralized ledger, immutable, and group consensus, but also provides more efficient consensus mechanisms, higher throughputs, smart contracts, and support for multiple organizations and ledgers.

In this paper, we apply blockchain technology to IoT access control, design and implement an access control system named fabic-iot, which is based on Hyperledger Fabric and ABAC. By using distributed architecture, fabric-iot can trace records, provide dynamic access control management and solve the access control problem in IoT.

The main contributions of this paper are:

1) We define a device resource sharing model according to the data production of the IoT devices in real life. The model makes the data resources generated by the device correspond to the URL one by one, greatly simplifying the sharing mode and storage structure of the device resources.

2) We propose a blockchain-based access control system for IoT named fabric-iot and describe its workflow and architecture in detail. The system uses distributed architecture to separate users and devices, and implemented the dynamic management of permissions to support efficient access.

3) We design three kinds of smart contracts based on the Hyperledger Fabric platform. The first one implements the ABAC model. The second one implements the ABAC policy management. The last one implements the device resource management.

4) We introduce the network initialization, chaincode installation, and smart contract invoking of fabric-iot in detail.

5) We design two groups of comparative experiments to verify the system performance and consensus speed.

The remaining of this paper is organized as follows. Section II introduces the related work. Section III details the structure and operating mechanism of Hyperledger Fabric, as well as ABAC model. The design of resource model, policy model, system's structure, workflow, and the implements of smart contract are presented in Section IV. In Section V, we demonstrate how to setup fabric-iot system and how to work with it to manage the access control of the IoT resources. We set two groups of comparative experiments, and then analysis the results. In section VI, we make a conclusion for this paper and give a preview for further work.

II. RELATED WORK

The rapid development of IoT requires a higher standard for distributed access control. The blockchain technology has four advantages for assess control which are decentralization, data encryption, scalability, and non-tampering. At present, the blockchain technology has evolved to 3.0 version. As its core technology, smart contracts built a safe and reliable operating environment for applications, and give blockchain more powerful functions. Therefore, based on the existing access control methods, many scholars proposed a variety of IoT access control methods by combining it with blockchain and smart contracts.

Reference [7] proposed a secure Fabric-based data transmission technique and realized a power trading center in industrial IoT, which solved the problems of low security, high management cost, and difficulty in supervision. In reference [8], a decentralized access control scheme for the IoT devices was proposed. The scheme used a single smart contract to reduce the communication cost between nodes. A node called management hub is designed to interact with devices, avoiding direct interaction between blockchain and IoT devices. It has six advantages, which are Mobility, Accessibility, Concurrency, Lightweight, Scalability and Transparency. In reference [9], an access control scheme based on Ethereum smart contract was proposed, which included three smart contracts: access control contract (ACC), judge contract (JC), and register contract (RC). ACC implemented policy-based authorization by checking the behavior of objects. JC was used to judge the wrong behavior and return the corresponding punishment. RC was used to register the above two smart contracts and provide update, delete, and other operations. Finally, the architecture was implemented with one PC, one notebook, and two Raspberry Pi. In reference [10], a distributed application (DAPP) based on Ethereum was proposed. It combines the SaaS business model with blockchain, which was used to buy and sell sensor data. Reference [11] proposed an improved blockchain structure. In the enterprise network, the private chain was used to manage the device configuration files of IoT in a distributed way. It stored the device configuration files on blockchain and monitors the operation with a smart contract. In reference [12], the routing information of router nodes are saved to blockchain to ensure that the routing information can't be tampered and traced. In reference [13], an attribute-based access control method for IoT was proposed, which saved attribute data through blockchain. This method avoided data tampering and simplified the access control protocol to meet the computing power and energy constraints of the IoT devices. In reference [14], a multi-blockchain distributed key management architecture (BDKMA) was proposed, and a fog computing method was introduced to reduce the delay of the multi-chain operation, which can better guarantee the privacy and security of users. In reference [15], blockchain was used for the trust management of the Internet of vehicles. By combining two consensus mechanisms, proof of work (PoW) and proof of stake (PoS), the difficulty of mining nodes can be changed dynamically, so the consensus can be reached more efficiently. Reference [16] proposed an efficient and safe road condition monitoring authentication scheme based on fog computing. In reference [17], a kind of Edge-chain was proposed, which used the currency system of blockchain to link the edge computing resources with the resource usage between accounts and IoT devices. Edge-chain established a trust model based on the behavior of devices to control the IoT devices to obtain resources from edge servers. In reference [18], a scalable security management architecture of IoT was designed for wireless sensor networks, and its performance was compared with the existing management solutions of IoT. In reference [19], a rights management system based on bitcoin was proposed, which allowed users to publish and transfer access policies. The advantage was that the access policies would be opened to all users and prevent any party from denying the authenticity of the policies. Reference [20] combined blockchain and RBAC and implemented a cross-organizational RBAC based on Ethereum, which allowed small organizations to participate, so the users can fully control their roles. In reference [21], the Fair-Access privacy protection and rights management framework was proposed, which used smart contracts to manage access policies in a fine-grained way and checked policy reuse. In reference [22], a scheme named EduRSS was proposed. It used a Ethereum-based blockchain to store and share educational records. Reference [23] proposed a novel trust-based recommendation scheme (TBRS) to ensure security and real-time data transmission in a vehicular CPS network. In reference [24], a blockchain model based on hypergraphs was proposed. This model used hyperedge structure to organize storage nodes, transformed the data storage of the whole network into local network storage, reduced storage consumption, and improved the security level.

III. PRELIMINARIES

A. HYPERLEDGER FABRIC

Digital currencies represented by bitcoin have achieved great success and attracted worldwide attention to blockchain technology. However, this kind of public chain has many disadvantages which are listed as follows:

1) Low transaction throughput. Only about 7 transactions can be accepted per second.

2) Long transaction confirmation time. Each transaction takes about 1 hour to be finally confirmed.

3) Waste of resources. The PoW mechanism consumes a lot of computing resources and power.

4) Consistency issues. Blockchain is easy to form a branch. When a branch is formed, only the longest chain takes effect, and all transactions in other chains are invalid.

5) Privacy issues. As the ledger of bitcoin is open, there is no privacy in the transaction.

To solve those problems, the Linux Foundation launched the Hyperledger project in 2015, which built an enterprise blockchain developing platform. As one program of them, Hyperledger Fabric uses a modular structure to provide scalable components including encryption, authentication, consensus algorithm, smart contract, data storage and other services.

All the programs of the Hyperledger Fabric run in the docker containers. The container provides a sandbox environment, which separates the application program from the physical resources, and isolates the containers from each other to ensure the security of the application. Hyperledger Fabric is a kind of alliance chain, in which all the nodes need to be authorized to join the blockchain network. On this basis, Fabric provides a consensus mechanism based on Kafka message queue, which can quickly reach consensus in large-scale application scenarios. Hyperledger Fabric overcomes those shortcomings of public chain.

1) MAIN COMPONENTS: CA, CLIENT, PEER, ORDERER

CA provides unified management for digital certificates of member nodes, and generates or cancels identity certificates of members.

Client is used to interact with the peer node and operate the blockchain system at the same time. The operation is divided into two categories. The first one is management category which is mainly used to manage the nodes, including start, stop, configure nodes, etc. The other is chaincode category which is mainly used for the life cycle management of chaincode, including installation, instantiation, upgrade and execution of chaincode, etc. The Client is generally a command-line client or an application developed by SDK.

Peer is an equal footing node in the distributed system, this component stores ledgers and chaincodes of blockchain. Applications connect to peer and query or update ledgers by invoking chaincodes. There are two types of peers: endorser and committer. The endorser node is responsible for verifying, simulating and endorsing transactions. The committer is responsible for verifying the legitimacy of transactions as well as updating the blockchain and ledger status.

Orderer is responsible for accepting the transactions sent by the peer node, sorting the transactions according to certain rules, packaging the transactions in a certain order into blocks, and sending them to the peer node. The peer node updates the local ledger with new blocks, and finally reaches a consensus.

2) CHANNEL

An important requirement of most enterprise applications is data privacy and confidentiality. The Fabric designes a channel system to isolate the blockchain data of different organizations. In each channel, there is an independent private ledger and a blockchain. Therefore, Fabric is a system with multi-channel, multi-ledger, and multi-blockchain.

3) LEDGER

The data of Fabric is stored as a distributed ledger, and the data items in this ledger are stored in the form of key-value pairs. All key-value pairs constitute the state of the ledger, which is called 'World State', as shown in Fig.1.

4) CHAINCODE

The smart contract in Fabric is called chaincode. Chaincode is a program written in golang (supporting other programming languages, such as Java) and implements predefined interfaces. Transactons can be generated by chaincode, which is the only way for the outside to interact with the blockchain system. By submiting or evaluating a transaction, the outsiders can change or read the data of state database (SDB), while the transaction would be written to ledger of Fabric. Business logic can be implemented by writing chaincode,

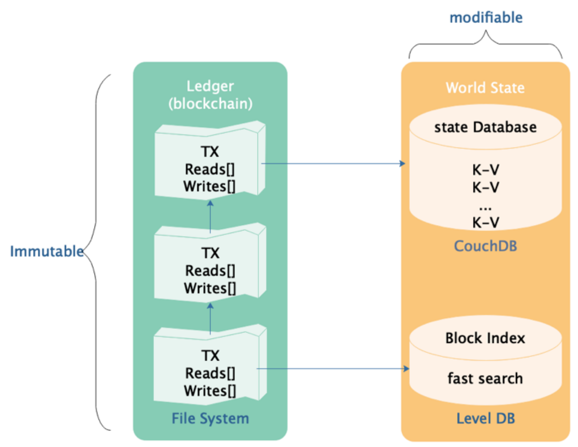


FIGURE 1. Structure of the ledger in Hyperledger Fabric.

thus developers can write different chaincodes to implement different applications.

B. ABAC MODEL

Attributed based access control (ABAC) is a kind of access control technology, which takes the attribute of subject, object, permission, and environment into account. It determines whether to grant access to the requester by judging whether its request contains the correct attributes. Because the attributes of subject and object are defined separately, ABAC can separate policy management and access control efficiently. The policy can be changed according to the actual situation, such as adding or reducing the attributes of it, for the scalable purpose. Besides, the attributes of entities can be defined from various perspectives to achieve fine-grained access control. Attributes are the core of ABAC, which can be defined to a set with four elements: $A \in \{S, O, P, E\}$. The meaning of each field is explained as follows:

A represents the attribute, $A = \{name : value\}$. The value of a attribute has a key name.

S represents the attribute of subject, which means the identity and characteristics of the entity that initiates the access request, such as a person's ID, age, name, position, etc.

O represents the attribute of the object, which means the attribute of the accessed resource, such as resource type, service IP address, network protocol, etc.

P represents the attribute of permission, which means the operation of the subject on the object, such as reading, writing, executing, etc.

E represents the attribute of environment, which means the environment information when the access request is generated, such as time, location, etc.

ABACR (Attributed based access control request) is defined as: $ABACR = \{AS \wedge AO \wedge AP \wedge AE\}$, which is a set contains the above four attributes. It represents the AP(Attributes of Operation) of the AS(Attributes of Subject) on the AO(Attributes of Object) under the AE(Attributes of Environment).

ABACP (Attributed based access control policy) is defined as: $ABACP = \{AS \wedge \ or \ \vee AO \wedge \ or \ \vee AP \wedge \ or \ \vee AE\}$, which represent the access control rules of the subject to the object. It expresses the needed attribute set for the protected resources accessing.

IV. SYSTEM MODEL AND DESIGN

A. RESOURCE AND POLICY MODEL

There are many kinds of data produced by IoT devices, and most of them are unstructured data [25]. For example, the camera can capture the real-world images and produces pictures or video data, the microphone can capture the external sound and produces audio data. Sensors can capture physical signals such as temperature, humidity, light, and convert them into digital signal data. Most of these data are unstructured, so they can't be directly stored in the relational database. And because they are all real-time data, they need to be pushed to the authorized users in time. In general, voice and video data are all streaming data. The data collected by the device is encoded and then pushed to the cloud server through WiFi or 4G. Finally, a resource URL is generated. The users can pull streaming data according to the video transmission protocols such as HLS and RTMP by URL. For sensor data, the device sends the data to a topic through the MQTT-based [26] service or other protocols. After the client is authorized, it subscribes the corresponding topic (which can be represented by URL), and the server pushes the message under the topic to the client. Besides, this kind of data is mainly used to control IoT devices to excute binding, unbinding, opening, closing, adjusting and other operations. Generally, clients can send requests to the server through restful API based on HTTP (s). After the server verifies the permissions, it can send control signals back to the device through MQTT or other protocols. Table 1 shows the URL formats for several different types of resource. To sum up, this paper defines a device resource model: $\{Device\} \rightarrow \{resource\} \rightarrow \{url\}$.

TABLE 1. Examples of resource URL.

URL	Meaning
https://www.xxx.com/live/test_video.m3u8	URL of HLS video resource
rtmp:// www.xxx.com/live/test_video.flv	URL of RTMP video resource
tcp://www.xxx.com/mqtt/test_topic	Real time data URL with topic as "test_topic"

The access rights of the subject (user) to the object (resource) are defined by the access policy. Instead of requesting resources from the device directly, users get the resource data according to the URL from the blockchain system with permission verification.

The connection between device resources and users is shown in Fig.2. The brief workflow is shown in the 1-5 serial number in the figure.

1) The device distributes the resource to the Internet and generates the resource URL.

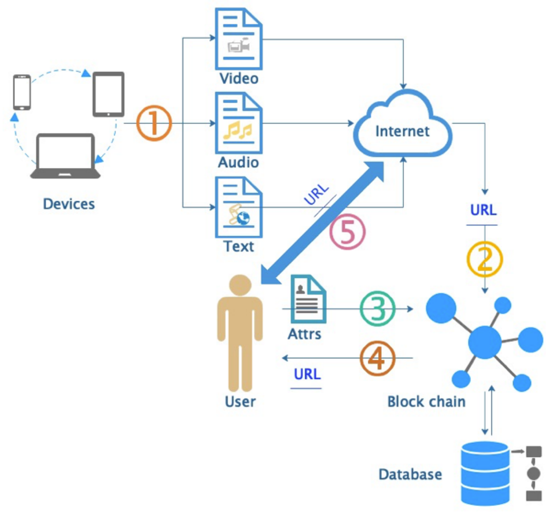


FIGURE 2. Connections between users and resources.

2) The device saves the resource URL to the blockchain system.

3) Users request the blockchain system by attributes to get authorization.

4) Blockchain sends URL to the authorized users.

5) Users download or pull resource data on the Internet according to the URL.

Combined with ABAC model and the characteristics of data generated by the IoT devices, the device access controll policy model is defined as follows:

$$P = \{AS, AO, AP, AE\} \quad (1)$$

$$AS = \{userId, role, group\} \quad (2)$$

$$AO = \{deviceId, MAC\} \quad (3)$$

$$AP = \begin{cases} 1, & \text{allow} \\ 0, & \text{deney} \end{cases} \quad (4)$$

$$AE = \{createTime, endTime, allowedIP\} \quad (5)$$

P (Policy): It represents a policy of attributed access control. This set contains four elements: AS, AO, AP, and AE.

AS (Attribute of Subject): It represents the attributes of a subject (user) and includes three types: userID (unique identification user), role (user role), and group (user group).

AO (Attribute of Object): It represents the attributes of an object (resource), which consist of a device ID or a MAC address of device. In this model, we do not regard the resource URL as an attribute directly. Instead, we use the unique identifications as attributes of a device. Because in reality, the network of the device is variable and the data produced by the device is dynamic. We assume that the function of a device in the system is single, and each ID or MAC can only correspond to one resource URL at one time.

AP (Attribute of Permission): It indicates whether user have access to resources. The value 1 stands for "allow" and 2 stands for "deny". When AP is initialized, the default value is 1. Admin can revoke access authorization according to the situation by setting the value of AP to 0.

AE (Attribute of Environment): It indicates the attributes of environment which required for access control. AE has three kinds of attributes: time, end time, and allowed IP. Time stands for the creation time of policy. End time stands for the expiration time of policy. The policy will be invalid when current time is later than end time. Allowed IP is aimed to prevent the IP address outside the network segment from accessing the system.

B. SYSTEM STRUCTURE

Fabric-iot, a blockchain-based access control system for IoT, consists of four parts: users, blockchain, smart gateway, and devices, which are shown in Fig.3.

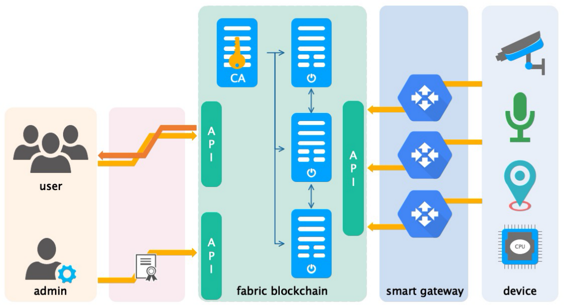


FIGURE 3. Architecture of fabric-iot.

1) USERS

This system divides users into two types: admin, and common user.

The admin is responsible for managing the blockchain system and maintaining the program of smart gateway. The admin needs to provide a certificate to access the blockchain system. The specific operation allowed are as follows.

1) Add new smart contracts. Admin can deploy new smart contracts on the blockchain through API.

2) Upgrade contracts. The admin can upload a new smart contract to nodes and install it, while the old one will be upgraded to a new version.

The common user, which means the owner of the device, gets the resource URL by sending attributed based authorization request to the blockchain system.

2) BLOCKCHAIN

It is the core of system. All nodes need to obtain CA authentication before joining the blockchain system. The blockchain is developed based on Hyperledger Fabric, which implements access control by smart contracts. Blockchain system exposes API for users and smart gateway to access. It mainly implements three functions as follows.

1) Device resource URL data storage.

2) Attribute-based user rights management.

3) Authentication of user access to resources.

3) SMART GATEWAY

As the bridge between devices and blockchain system, it can receive the message from the device and put the URL it contained to blockchain, avoiding the pressure on blockchain system caused by direct access of devices.

4) IoT DEVICES

As the largest group, the IoT devices generally do not have strong computing ability, enough storage, and durable battery. Therefore, it is impossible to deploy IoT devices as peer nodes of blockchain directly. IoT devices has a unique MAC address or product ID, which can be distinguished from other devices. Generally, the devices may belong to both some users or groups. Whenever a new resource is generated by the device, a message contains the URL of resource is sent to the smart gateway. In this system, MQTT protocol is used as the message transmission protocol.

C. WORKFLOW

As shown in Fig.4, the whole system workflow mainly includes four parts. This section details the intermediate steps of each part. The used symbols are described in Table 2.

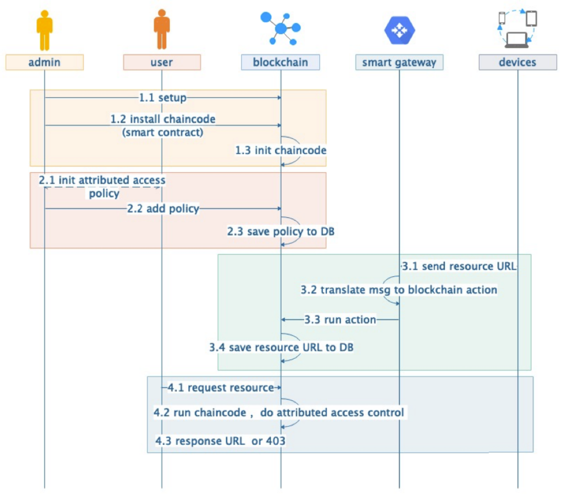

admin
user
blockchain
smart gateway
devices
1.1 setup
1.2 install chaincode (smart contract)
1.3 init chaincode
2.1 init attributed access policy
2.2 add policy
2.3 save policy to DB
3.1 send resource URL
3.2 translate msg to blockchain action
3.3 run action
3.4 save resource URL to DB
4.1 request resource
4.2 run chaincode , do attributed access control
4.3 response URL or 403

FIGURE 4. **Workflow of fabric-iot.**

Part1 The blockchain network initialization and the chaincode installation are the basis process of the system. These operations require the admin to work in the intranet. Flow 1 mainly including three steps.

Step1 Before setting up a Hyperledger Fabric network, we need to create certificates for all members such as peer nodes, orderer nodes, channels, users, etc. All certificates are generated by CA.

$$CA \rightarrow \{Cert_{peer}, Cert_{order}, Cert_{channel}, Cert_{user}\} \qquad (6)$$

The peer node and the order node run in the docker container. The certificate needs to be packaged into their docker image before running.

$$Build(conf, Cert) \xrightarrow{build} Image \xrightarrow{run} Container \qquad (7)$$

After all the peer nodes and orderer nodes are set up successfully, we start to create the channel. Every channel joined an independent blockchain and ledger.

$$\{blockchain, ledger\} \xrightarrow{join} Channel \qquad (8)$$

Step2 So far, a basic Hyperledger Fabric network is set up. In order to build applications, we need to design chaincode. Our source code of chaincode is written in Golang.

$$Code(F(x)\ldots) \rightarrow CC \qquad (9)$$

Admin uses Hyperledger Fabric SDK or client to install CC (chaincode). All CC will be installed to peer nodes.

$$Install(CC) \xrightarrow{SDK/Client} Peer \qquad (10)$$

Step3 Once the chaincode is installed, it needs to be initialized. Invoke function is used to initialize chaincode. Every instantiated chaincode will be saved in the container as an endorsement.

$$Invoke(Init) \xrightarrow{SDK/Client} Peer \qquad (11)$$

Part2 Develop access control policy and save them to blockchain system. This process requires the user and admin work together to decide and customize the access policy in advance and uploads them to the blockchain system by admin.

Step1 Admins and users jointly make access policies, which are defined based on the attributes of subject (user), object (device resource), operation, and environment.

$$Decide(AS, AO, AE, AP) \rightarrow ABACP \qquad (12)$$

Step2 After the access policy is defined, the admin uploads it to the blockchain network.

$$Upload(ABACP) \rightarrow contract \qquad (13)$$

Step3 Admin connects to blockchain to add, modify and delete the policy by running the PolicyContract. The value

TABLE 2. **Symbol descriptions.**

Name	Meaning
CA	Certificate Authority
$Cert$	Certificate file
$conf$	Config file of the node
$F(x)\ldots$	Functions defined in source code
CC	Chaincode in Hyperledger Fabric
AC	Access Contract
PC	Policy Contract
DC	Device Contract
$Image$	Docker Image
$Container$	Docker Container
TX	Transaction in blockchain
AS, AO, AE, AP	Attributes of subject, object, permission, and environment
$Ledger$	Ledger in Hyperledger Fabric
SDB	State Database in Hyperledger Fabric
SG	Smart Gateway
BA	Blockchain Action
Cli	Blockchain system client
$ABACP$	Attribute based Access Control Policy

of the policy is saved in the SDB and the record of action is written into the ledger.

$$PolicyContract(ABACP) \rightarrow \{Ledger, SDB\} \tag{14}$$

Part3 Device reports resource URL to smart gateway, and after that, smart gateway uploads it to blockchain system.

Step1 The device generates a message included device ID and URL and sends it to smart gateway by MQTT.

$$\{deviceId, URL\} \rightarrow Msg \xrightarrow{MQTT} SG \tag{15}$$

Step2 Smart gateway parses messages and generates an action for blockchain.

$$Translate(Msg) \rightarrow BA \tag{16}$$

Step3 Smart gateway connects to blockchain client to run the action.

$$Run(BA) \xrightarrow{Cli} DeviceContract \tag{17}$$

Step4 Blockchain saves the URL of device resource by invoking functions of DC.

$$DeviceContract(deviceId, URL) \rightarrow \{Ledger, SDB\} \tag{18}$$

Part4 The process of acquiring resources based on attributes is the core of system. It has three steps as follows:

Step1 User initiates attributed based requests.

$$userId \rightarrow Request\{AS \wedge AO \wedge AP\} \tag{19}$$

Step2 Invoking the functions of AC after receiving the requests.

$$AccessContract(Request) \rightarrow \begin{cases} 1, & \text{OK} \\ 0, & \text{Forbidden} \end{cases} \tag{20}$$

Step3 If the validation passes, blockchain system query URL by invoking the function in DC and return it to users. If it failed, 403 error (403 stands for forbidden in HTTP status codes) will be returned to users.

$$result = \begin{cases} 1, & \xrightarrow{DC} URL \\ 0, & \rightarrow \ 403 \end{cases} \tag{21}$$

D. SMART CONTRACT DESIGN.

Smart contract is the core of access control implementation. There are three kinds of smart contracts in this system: Policy Contract(PC), Device Contract(DC), and Access Contract(AC).

1) POLICY CONTRACT

It provides the following methods to operate ABACP.

Auth(): Admin defines the ABACP for users, and sends the request for adding ABACP to the blockchain system. An example of ABACPR(attributed based access control policy request) is shown in Table 3. Admin encrypts the data with the public key of the PC node, and then signs the request with the private key. PC invokes Auth() to verify the identity of admin with its public key and decrypts the data with self's private key.

TABLE 3. An example of ABACPR.

ABACPR.json
{"*Action*" : "*add*","*Data*" : {"*AS*" : {"*userId*" : "10001","*role*" : "*manager*","*group*" : "*g*1"},"*AO*" : {"*deviceId*" : "*B*230011001*xxx*01","*MAC*" : "48 : *e*2 : *xx* : *xx* : 5*f* : *f*9"},"*AP*" : 1,"*AE*" : {"*createTime*" : "1572607208","*endTime*" : "1575199208","*allowedIP*" : "10.10.100. ∗ /10.10.255. ∗ "}}}

CheckPolicy(): As shown in Algorithm 1. PC needs to check the validity of ABACP. A legal ABACP needs to contain the above four attributes, and the type of each attribute also needs to meet the requirements.

Algorithm 1 PolicyContract.CheckPolicy(): Check ABAC Policy Before Putting it Into DB

```
Input: ABACP
Output: True or False
1: $< AS, AO, AE, AP > \leftarrow ABACP$
2: $IsOK = True$
3: for $item$ in $AS$ do:
4:     if $item \notin < userId, role, group >$ then
5:         $IsOK = False$
6:     end if
7: end for
8: for $item$ in $AO$ do:
9:     if $item \notin < deviceId, MAC >$ then
10:        $IsOK = False$
11:    end if
12: end for
13: if $Val(AE)! = 1\ or 0$
14:     $IsOK = False$
15: end if
16: for $item$ in $AP$ do:
17:     if $item \notin < createTime, endTime, allowedIP >$ then
18:         $IsOK = False$
19:     end if
20: end for
21: return $IsOK$
```

AddPolicy(): As shown in Algorithm 2. After ABACP is verified to be legal by CheckPolicy(), the PC invokes AddPolicy () to add ABACP to the SDB, at the same time, all the action records will be written to the ledger.

UpdatePolicy(): In some cases, admin needs to modify ABACP. The function UpdatePolicy() implements the interface of updating SDB, and the operation record of updating will also be written to the blockchain. UpdatePolicy() is similar to AddPolicy(), which also invokes the put method of the application interface to overwrite the old value.

DeletePolicy(): ABACP has an expiration time and it can be canceled by admin. There are two cases where deletion occurs. One occurs when admin actively delete a policy by

Algorithm 2 PolicyContract.AddPolicy(): Add ABAC Policy to Blockchain

Input: ABACP
Output: Error or null

```
 1: $@implement\ SmartContract\ Interface$
 2: $APIstub\ ChaincodeStub \leftarrow Invoke()$
 3: if $CheckPolicy(ABACP) == False$
 4:     return $Error('BadPolicy')$
 5: end if
 6: $Id \leftarrow Sha256(ABACP.AS + ABACP.AO)$
 7: $err \leftarrow APIstub.PutState(Id, ABACP)$
 8: if $err! = null$ then
 9:     return $Error(err.Text)$
10: end if
11: renturn $null$
```

invoking this function. And the other occurs when the Check-Access() method is executing, if the attribute "endTime" is expired, then it will invoke this function in PC to delete the related policy. As shown in Algorithm 3.

Algorithm 3 PolicyContract.DeletePolicy(): Delete ABAC Policy From Blockchain

Input: AS, AO
Output: $Error\ or\ null$

```
 1: $@implement\ SmartContract\ Interface$
 2: $APIstubChaincodeStub \leftarrow Invoke()$
 3: $Id \leftarrow Sha256(AS + AO)$
 4: $err \leftarrow APIstub.GetState(Id)$
 5: if $err! = null$ then
 6:     return $Error(err.Text)$
 7: end if
 8: $APIstub.DelState(Id)$
 9: if $err! = null$ then
10:     return $Error(err.Text)$
11: end if
12: renturn $null$
```

QueryPolicy(): It implements the interface of database querying which provides a function to get ABACP for other chaincode. We choose CouchDB as SDB. Although it is a key-value document database, CouchDB supports complex queries similar to mongoDB. In that case, QueryPolicy() supports querying ABACP by AS or AO.

2) DEVICE CONTRACT

DC is mainly responsible for storing the resource URL of the device into SDB. DC has 2 input parameters $\{DeviceId, URL\}$. The functions provided are as follows.

AddURL(): It uses DeviceId as a key and URL as a value to store in SDB.

GetURL(): It queries the corresponding URL value from SDB according to DeviceId.

3) ACCESS CONTRACT

It verifies whether the user's ABACR matches the ABAC policy. Like PC, the request data is signed by the user's private key, after that, AC verifies the signature to check the user's identity by public key of user. The methods provided are as follows.

Auth(): Similar to the PC method of the same name, it verifies the request with the user's public key and check the authenticity of its identity.

GetAttrs(): It parses the property data field after the signature is verified. Only part of ABAC attributes are included in ABACR:$\{AS, AO\}$, while AE needs to be determined by AC. Finally, these attributes are combined as $\{AS, AO, AE\}$.

CheckAccess(): It is the core function to achieve access control management, as shown in Algorithm 4. Firstly, it gets the attribute set by GetAttrs(). Secondly, it invokes the Query-Policy() method of PC to query the corresponding ABACP according to AS and AO. If the returned result is empty, which means that there is no policy to support the request, it will return a 403 error directly (indicating that there is no permission). If the returned result is not empty, it indicates that one or more ABACP will be obtained. Thirdly, it starts to judge one by one whether the AE of request matches the AE of ABACP and whether the value of AP is 1 (stands for allow). If all attributes match the policy, the verification passes. Finally, it invokes the GetURL() function of DC to get the URL of the resource and return it to the user, otherwise 403 error will be returned.

Algorithm 4 AccessContract.CheckAccess(): Check User's Access

Input: $ABAC\_Request$
Output: $URL\ or\ Error$

```
 1: $< A_uS, A_uO, A_uE > \leftarrow GetAttrs(ABAC\_Request)$
 2: $P =< P_1, P_2, \ldots P_n > \leftarrow PC.QueryPolicy(A_uS, A_uO)$
 3: if $P == Null$ then
 4:     return $Error(403)$
 5: endif
 6: for $P\ in < P_1, P_2, \ldots P_n >$ do
 7:     $< \ldots, A_pP, A_pE > \leftarrow P$
 8:     if $Value(A_pP) ==$ '$deney$' then
 9:         $continue$
10:     if $A_uE \cap A_pE ==$ then
11:         $continue$
12:     $URL \leftarrow DC.GetURL(A_uO)$
13: end for
14: if $URL! = Null$ then
15:     return $URL$
16: else
17:     return $Error(403)$
18: end if
```

V. EXPERIMENT AND COMPARISON

This section introduces the experiment process and the result comparison to demonstrate the function and performance of fabric-iot we proposed. In first part, we list the hardware and software used in this experiment. In second part, we show the process of setting up the system and the realization of access control based on it. In last part, we give the performance test, comparative experiment, and results analysis.

The source code of fabric-iot project is open-source on GitHub: https://github.com/newham/fabric-iot.

A. ENVIRONMENTS

The experiment of this paper is carried out on two PCs. The hardware and software environments are listed in Table 4.

TABLE 4. Hardware and software environments.

Hardware	
CPU	i7 7500u 2.9GHz, i7 8700k 3.7GHz
Memory	8G,8G
Hard Disk	256G, 1T
Software	
OS	Mac OS 10.14.6, Deepin Linux 15.11
docker	v19.03.2
docker-compose	v1.24.1
node	v12.12.0
golang	v1.12.9
hyperledger fabric	v1.4.3

B. SYSTEM BUILDING PROCESS AND REALIZATION

The experiment is divided into three parts. The first part mainly introduces the structure of fabric-iot, as well as initialize configuration and start-up steps. The second part introduces the process of chaincode installation. The third part describes how to implement the IoT resource access control method based on ABAC by invoking three kinds of smart contracts(PC, AC, and DC).

1) STRUCTURE AND INITIALIZATION OF FABRIC-IoT

Fabric-iot consists of eight kinds of docker nodes which are shown in Table 5. The steps of system initialization are as follows.

TABLE 5. Docker images of nodes.

Node Name	Description	Number
fabric-iot/couchdb	database node	4
fabric-iot/ca	CA node	2
fabric-iot/peer	peer node	4
fabric-iot/orderer	orderer node	1
hyperledger/fabric-tools	tools of hyperledger	1
fabric-iot/chaincode/PC	PolicyContract node	4
fabric-iot/chaincode/DC	DeviceContract node	4
fabric-iot/chaincode/AC	AccessContract node	4

Step1: Use the Hyperledger cryptogenic tool to generate root certificates and secret key pairs for nodes (peer, orderer, etc.).

Step2: Move those certificates and secret key pairs to the directory specified, which will be mounted by a docker image of CA and take effect when the container running. Other nodes can authenticate their identity to the CA with their signatures.

Step3: Use the configtxgen tool to generate a genesis block, which is used to package transactions contains the configuration of nodes and channels. When fabric-iot is up, the genesis block is written into the blockchain, ensuring that the identity information of each node in the whole system can't be tampered with. After that, the image of other nodes will be started according to the initialization configuration of docker compose. When all containers run successfully, the peer nodes will be added to a channel.

2) INSTALLATION AND UPGRADE OF CHAINCODE

After the network is successfully set up, we start to install chaincodes. The chaincodes are installed by executing commands with the hyperledger client. The steps are as follows.

Step1: Copy the source code of chaincodes to a directory the client node mounted.

Step2: Run the command to package the chaincodes to a peer node in a channel.

Step3: Transport each compiled chaincode to other peer nodes and instantiate it. A copy of each chaincode is saved to a separate container as an endorsement.

The upgrade process is similar to installation. However, only the node which first installed chaincode will be upgraded to the new version immediately, the other peer nodes will be upgraded synchronously only when a transaction is generated.

3) IMPLEMENT OF ABAC

There are some ways to invoke chaincode in Hyperledger Fabric platform. The outsiders can use client or SDK(support Java, golang, node) to invoke chaincode in Hyperledger Fabric platform.

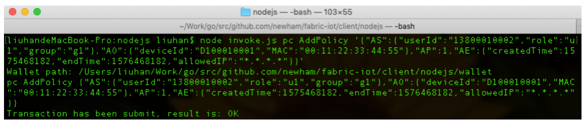
Fabric-iot uses the client written by node SDK to invoke chaincode. The steps are as follows.

Step1: CA node generates secret key pairs for client, which are saved in a wallet of user.

Step2: Admin runs a client to connect to the peer node to submit or evaluate (corresponding write and read operations) a transaction.

Step3: Peer node queries or updates the SDB by making a consensus with other peer nodes under the service of orderer node.

To add ABAC policy, the AddPolicy() method of PC is invoked, which is shown in Fig.5.

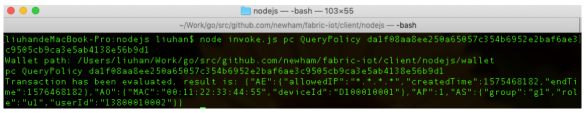


nodejs — -bash — 103×55
~/Work/go/src/github.com/newham/fabric-iot/client/nodejs — -bash

```
[liuhandeMacBook-Pro:nodejs liuhan$ node invoke.js pc AddPolicy '{"AS":{"userId":"13800010002","role":"u
1","group":"g1"},"AO":{"deviceId":"D100010001","MAC":"00:11:22:33:44:55"},"AP":1,"AE":{"createdTime":15
75468182,"endTime":1576468182,"allowedIP":"*.*.*.*"}}'
Wallet path: /Users/liuhan/Work/go/src/github.com/newham/fabric-iot/client/nodejs/wallet
pc AddPolicy {"AS":{"userId":"13800010002","role":"u1","group":"g1"},"AO":{"deviceId":"D100010001","MAC
":"00:11:22:33:44:55"},"AP":1,"AE":{"createdTime":1575468182,"endTime":1576468182,"allowedIP":"*.*.*.*"
}}
Transaction has been submit, result is: OK
```

FIGURE 5. Result of invoking AddPolicy() in PC.

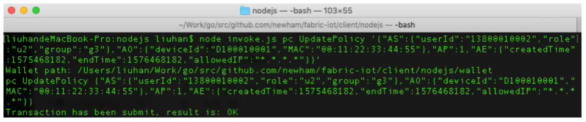
To verify whether the policy is added successfully, we invoke the PC.QueryPolicy() method to query. The key used in the query is obtained according to the formula: $sha256(UserId, DeviceId) \rightarrow key$. A query is shown in Fig.6.



```
nodejs — -bash — 103×55
~/Work/go/src/github.com/newham/fabric-iot/client/nodejs — -bash
liuhandeMacBook-Pro:nodejs liuhan$ node invoke.js pc QueryPolicy da1f08aa8ee250a65057c354b6952e2baf6ae3
c9505cb9ca3e5ab4138e56b9d1
Wallet path: /Users/liuhan/Work/go/src/github.com/newham/fabric-iot/client/nodejs/wallet
pc QueryPolicy da1f08aa8ee250a65057c354b6952e2baf6ae3c9505cb9ca3e5ab4138e56b9d1
Transaction has been evaluated, result is: {"AE":{"allowedIP":"*.*.*.*","createdTime":1575468182,"endTi
me":1576468182},"AO":{"MAC":"00:11:22:33:44:55","deviceId":"D100010001"},"AP":1,"AS":{"group":"g1","rol
e":"u1","userId":"13800010002"}}
```

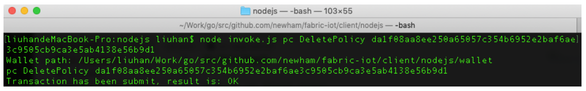
FIGURE 6. Result of invoking QueryPolicy() in PC.

If the properties in the real environment are changed or the policy needs to be updated, the PC.UpdatePolicy() function could be invoked, as shown in Fig.7. Delete operation can be realized through PC.DeletePolicy() method, as shown in Fig.8.



```
nodejs — -bash — 103×55
~/Work/go/src/github.com/newham/fabric-iot/client/nodejs — -bash
liuhandeMacBook-Pro:nodejs liuhan$ node invoke.js pc UpdatePolicy '{"AS":{"userId":"13800010002","role"
:"u2","group":"g3"},"AO":{"deviceId":"D100010001","MAC":"00:11:22:33:44:55"},"AP":1,"AE":{"createdTime"
:1575468182,"endTime":1576468182,"allowedIP":"*.*.*.*"}}'
Wallet path: /Users/liuhan/Work/go/src/github.com/newham/fabric-iot/client/nodejs/wallet
pc UpdatePolicy {"AS":{"userId":"13800010002","role":"u2","group":"g3"},"AO":{"deviceId":"D100010001","
MAC":"00:11:22:33:44:55"},"AP":1,"AE":{"createdTime":1575468182,"endTime":1576468182,"allowedIP":"*.*.*
.*"}}
Transaction has been submit, result is: OK
```

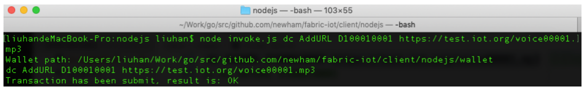
FIGURE 7. Result of invoking UpdatePolicy() in PC.



```
nodejs — -bash — 103×55
~/Work/go/src/github.com/newham/fabric-iot/client/nodejs — -bash
liuhandeMacBook-Pro:nodejs liuhan$ node invoke.js pc DeletePolicy da1f08aa8ee250a65057c354b6952e2baf6ae
3c9505cb9ca3e5ab4138e56b9d1
Wallet path: /Users/liuhan/Work/go/src/github.com/newham/fabric-iot/client/nodejs/wallet
pc DeletePolicy da1f08aa8ee250a65057c354b6952e2baf6ae3c9505cb9ca3e5ab4138e56b9d1
Transaction has been submit, result is: OK
```

FIGURE 8. Result of invoking DeletePolicy() in PC.

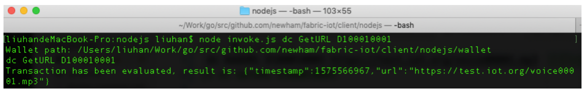
As shown in Fig.9, the device can report the resource address by invoking DC.AddUrl(), the URL is associated with the policy by DeviceId or MAC address.



```
nodejs — -bash — 103×55
~/Work/go/src/github.com/newham/fabric-iot/client/nodejs — -bash
liuhandeMacBook-Pro:nodejs liuhan$ node invoke.js dc AddURL D100010001 https://test.iot.org/voice00001.
mp3
Wallet path: /Users/liuhan/Work/go/src/github.com/newham/fabric-iot/client/nodejs/wallet
dc AddURL D100010001 https://test.iot.org/voice00001.mp3
Transaction has been submit, result is: OK
```

FIGURE 9. Result of invoking AddURL() in DC.

The URL can be queried by invoking DC.GetURL(), as shown in Fig.10.



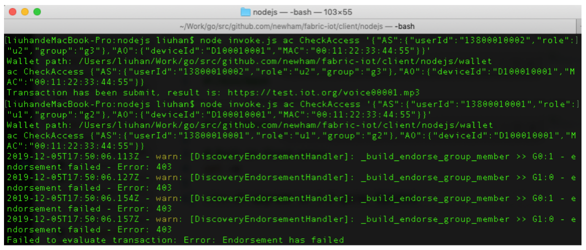
```
nodejs — -bash — 103×55
~/Work/go/src/github.com/newham/fabric-iot/client/nodejs — -bash
liuhandeMacBook-Pro:nodejs liuhan$ node invoke.js dc GetURL D100010001
Wallet path: /Users/liuhan/Work/go/src/github.com/newham/fabric-iot/client/nodejs/wallet
dc GetURL D100010001
Transaction has been evaluated, result is: {"timestamp":1575566967,"url":"https://test.iot.org/voice000
01.mp3"}
```

FIGURE 10. Result of invoking GetURL() in DC.

After receiving the user's request, AC invokes the Check-Access() method. It first invokes the PC to query the relevant policy according to the AO and AS, and then checks whether the AE and AP attribute meet the conditions. If the condition is met, it is proved that the property constraint of ABAC is met. Finally, the GetURL() method of DC is invoked to get the device resource URL. If the property condition is not met, an error will be returned, as shown in Fig.11.

C. RESULT AND COMPARISON

In order to test the performance of fabric-iot system, two groups of comparative experiments are designed by simulating concurrent access to the system with multithreaded



```
nodejs — -bash — 103×55
~/Work/go/src/github.com/newham/fabric-iot/client/nodejs — -bash
liuhandeMacBook-Pro:nodejs liuhan$ node invoke.js ac CheckAccess '{"AS":{"userId":"13800010002","role":
"u2","group":"g3"},"AO":{"deviceId":"D100010001","MAC":"00:11:22:33:44:55"}}'
Wallet path: /Users/liuhan/Work/go/src/github.com/newham/fabric-iot/client/nodejs/wallet
ac CheckAccess {"AS":{"userId":"13800010002","role":"u2","group":"g3"},"AO":{"deviceId":"D100010001","M
AC":"00:11:22:33:44:55"}}
Transaction has been submit, result is: https://test.iot.org/voice00001.mp3
liuhandeMacBook-Pro:nodejs liuhan$ node invoke.js ac CheckAccess '{"AS":{"userId":"13800010001","role":
"u1","group":"g2"},"AO":{"deviceId":"D100010001","MAC":"00:11:22:33:44:55"}}'
Wallet path: /Users/liuhan/Work/go/src/github.com/newham/fabric-iot/client/nodejs/wallet
ac CheckAccess {"AS":{"userId":"13800010001","role":"u1","group":"g2"},"AO":{"deviceId":"D100010001","M
AC":"00:11:22:33:44:55"}}
2019-12-05T17:50:06.113Z - warn: [DiscoveryEndorsementHandler]: _build_endorse_group_member >> G0:1 - e
ndorsement failed - Error: 403
2019-12-05T17:50:06.127Z - warn: [DiscoveryEndorsementHandler]: _build_endorse_group_member >> G1:0 - e
ndorsement failed - Error: 403
2019-12-05T17:50:06.154Z - warn: [DiscoveryEndorsementHandler]: _build_endorse_group_member >> G0:1 - e
ndorsement failed - Error: 403
2019-12-05T17:50:06.157Z - warn: [DiscoveryEndorsementHandler]: _build_endorse_group_member >> G1:0 - e
ndorsement failed - Error: 403
Failed to evaluate transaction: Error: Endorsement has failed
```

FIGURE 11. Result of invoking CheckAccess() in AC.

clients. In the first group of experiments, we count the processing time of PC, AC and DC with different numbers of concurrent requests. The numbers of virtual clients are set to 50, 100, 200, 500, 1000. The statistical results are shown in Fig.12-17. It can be seen from the figures:

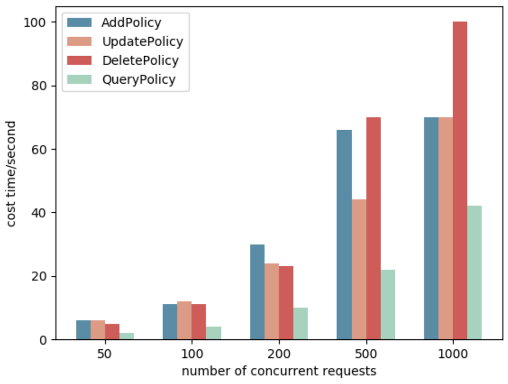


FIGURE 12. Cost time of PC at different numbers of concurrent requests.

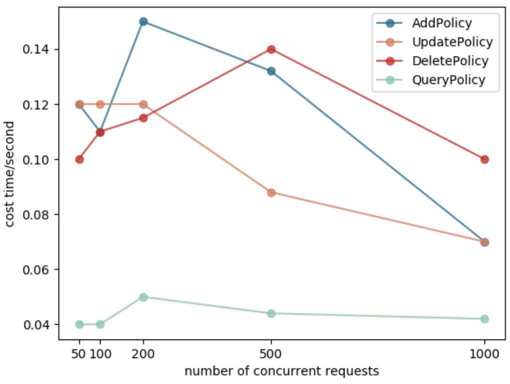


FIGURE 13. The trend of average cost time of PC at different numbers of concurrent requests.

1) Write (such as "add", "update") operations cost longer time than read (such as "get", "query") operations.

2) The throughput of the system grows up with the increase of the requests' number. When the throughput reaches a certain value, it tends to be stable. And with the further increase of the clients' number, there is no obvious downward trend of throughput.

In the second group of experiments, we use MQTT message queue to sort transactions according to the principle of consensus mechanism in Hyperledger Fabric,

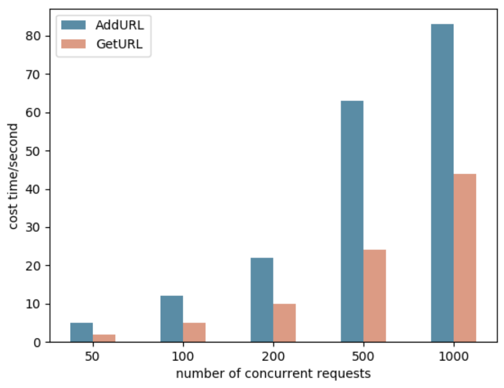


FIGURE 14. Cost time of DC at different numbers of concurrent requests.

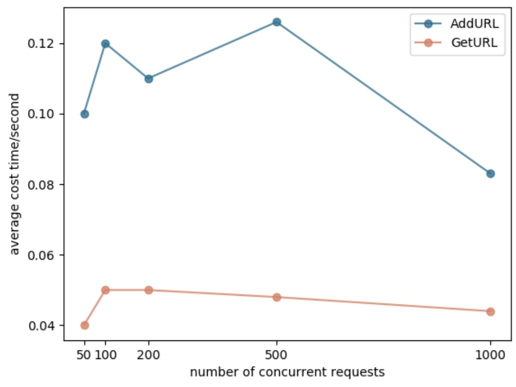


FIGURE 15. The trend of average cost time of DC at different numbers of concurrent requests.

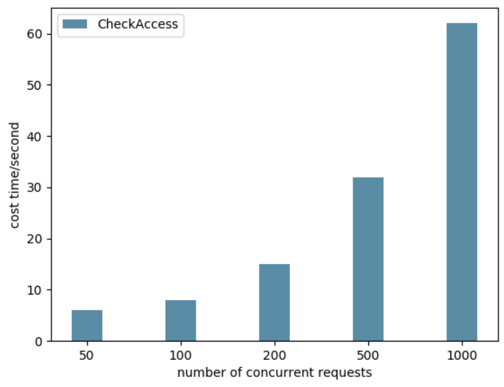


FIGURE 16. Cost time of AC at different numbers of concurrent requests.

simulating the consensus algorithm of fabric-iot. We implement PoW consensus algorithm in golang language, and set reasonable difficulty to fit the experimental environment. We test the efficiency of data consistency of distributed system by comparing the cost time of fabric-iot and PoW consensus mechanism under different node numbers. The numbers of nodes in the experiment are set from 5 to 100. The results are shown in Fig.18. It can be seen from the figure that under the difficulty of ensuring the PoW security, the cost time of consensus in fabric-iot is far less than the PoW.

The above two groups of experiments can prove that fabric-iot can maintain high throughput in large-scale request

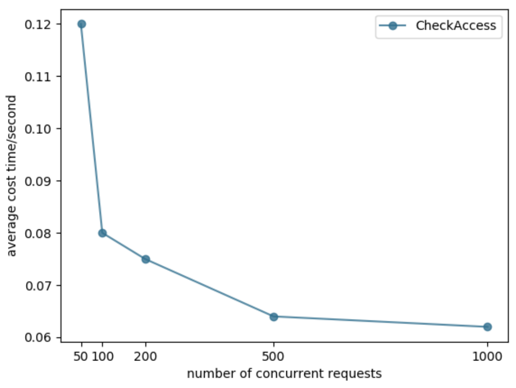


FIGURE 17. The trend of average cost time of AC at different numbers of concurrent requests.

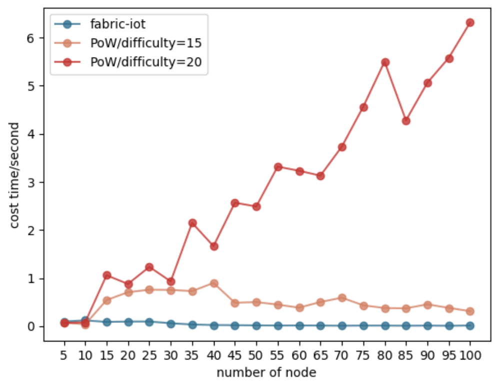


FIGURE 18. Comparison of the consensus speed between fabric-iot and PoW.

environment, and can effectively reach consensus in distributed system to ensure data consistency.

VI. CONCLUSION AND FURTHER WORK

This paper combines the blockchain technology with the ABAC model, takes advantages of the blockchain technology such as decentralization, tamper-proof and trace-ability, solves the problem that the traditional access control method based on the centralized designs is difficult to meet the access control requirements in IoT. Firstly, according to the actual production data of the IoT devices, we proposes a device authority model. Secondly, according to the ABAC model, we implements the ABAC policy management and ensures the access security of the device resources by implementing the smart contract application. Furthermore, an open-source access control system named fabric-iot based on Hyperledger Fabric is designed and implemented. This system adopts distributed architecture, which can provide fine-grained and dynamic access control management for the physical network. Finally, the steps of blockchain network building, chaincode installation, smart contract invoking are described in detail, and the experiments show a convincible results. Above all, this paper provides a practical reference for other researchers to carry out relevant research.

Future works can be improved in the following aspects:

1) The experiments in this paper is carried out on two PCs. In the future, we consider using the cluster or the edge

computing service to deploy, and further verify the distributed performance of this system.

2) In the future, more physical devices can be used to test the reliability and throughput of the system.

3) Future research can try to improve the scalability of fabric-iot and to support more IoT application integration.

REFERENCES

[1] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[2] K. Chopra, K. Gupta, and A. Lambora, "Future Internet: The Internet of Things–a literature review," in *Proc. Int. Conf. Mach. Learn., Big Data, Cloud Parallel Comput. (COMITCon)*, Feb. 2019, pp. 135–139.

[3] J. Wang, H. Wang, H. Zhang, and N. Cao, "Trust and attribute-based dynamic access control model for Internet of Things," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery (CyberC)*, Oct. 2017, pp. 342–345.

[4] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, Feb. 2015.

[5] N. Satoshi. (Nov. 2019). *Bitcoin—Open Source P2P Money*. [Online]. Available: https://bitcoin.org/en/

[6] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, and Y. Manevich, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, p. 30.

[7] W. Liang, M. Tang, J. Long, X. Peng, J. Xu, and K.-C. Li, "A secure fabric blockchain-based data transmission technique for industrial Internet-of-Things," *IEEE Trans. Ind. Inf.*, vol. 15, no. 6, pp. 3582–3592, Jun. 2019.

[8] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.

[9] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1594–1605, Apr. 2019.

[10] G. Papadodimas, G. Palaiokrasas, A. Litke, and T. Varvarigou, "Implementation of smart contracts for blockchain based IoT applications," in *Proc. 9th Int. Conf. Netw. Future (NOF)*, Nov. 2018, pp. 60–67.

[11] K. Košt'ál, P. Helebrandt, M. Belluš, M. Ries, and I. Kotuliak, "Management and monitoring of iot devices using blockchain," *Sensors*, vol. 19, no. 4, p. 856, Feb. 2019.

[12] J. Yang, S. He, Y. Xu, L. Chen, and J. Ren, "A trusted routing scheme using blockchain and reinforcement learning for wireless sensor networks," *Sensors*, vol. 19, no. 4, p. 970, Feb. 2019.

[13] S. Ding, J. Cao, C. Li, K. Fan, and H. Li, "A novel attribute-based access control scheme using blockchain for IoT," *IEEE Access*, vol. 7, pp. 38431–38441, 2019.

[14] M. Ma, G. Shi, and F. Li, "Privacy-oriented blockchain-based distributed key management architecture for hierarchical access control in the IoT scenario," *IEEE Access*, vol. 7, pp. 34045–34059, 2019.

[15] Z. Yang, K. Yang, L. Lei, K. Zheng, and V. C. M. Leung, "Blockchain-based decentralized trust management in vehicular networks," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1495–1505, Apr. 2019.

[16] M. Cui, D. Han, and J. Wang, "An efficient and safe road condition monitoring authentication scheme based on fog computing," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 9076–9084, Oct. 2019.

[17] J. Pan, J. Wang, A. Hester, I. Alqerm, Y. Liu, and Y. Zhao, "EdgeChain: An edge-IoT framework and prototype based on blockchain and smart contracts," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4719–4732, Jun. 2019.

[18] O. Novo, "Scalable access management in IoT using blockchain: A performance evaluation," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4694–4701, Jun. 2019.

[19] D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *Proc. IFIP Int. Conf. Distrib. Appl. Interoperable Syst.* Springer, 2017, pp. 206–220.

[20] J. P. Cruz, Y. Kaji, and N. Yanai, "RBAC-SC: Role-based access control using smart contract," *IEEE Access*, vol. 6, pp. 12240–12251, 2018.

[21] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "FairAccess: A new blockchain-based access control framework for the Internet of Things," *Security Commun. Netw.*, vol. 9, no. 18, pp. 5943–5964, Dec. 2016.

[22] H. Li and D. Han, "EduRSS: A blockchain-based educational records secure storage and sharing scheme," *IEEE Access*, vol. 7, pp. 179273–179289, 2019.

[23] W. Liang, J. Long, T.-H. Weng, X. Chen, K.-C. Li, and A. Y. Zomaya, "TBRS: A trust based recommendation scheme for vehicular CPS network," *Future Gener. Comput. Syst.*, vol. 92, pp. 383–398, Mar. 2019.

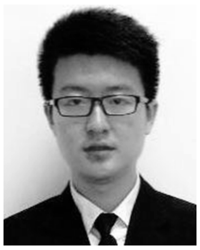
[24] C. Qu, M. Tao, and R. Yuan, "A hypergraph-based blockchain model and application in Internet of Things-enabled smart homes," *Sensors*, vol. 18, no. 9, p. 2784, Aug. 2018.

[25] W. Liang, K.-C. Li, J. Long, X. Kui, and A. Y. Zomaya, "An industrial network intrusion detection algorithm based on multi-feature data clustering optimization model," *IEEE Trans. Ind. Informat.*, to be published.

[26] N. Tantitharanukul, K. Osathanunkul, K. Hantrakul, P. Pramokchon, and P. Khoenkaw, "MQTT-topics management system for sharing of open data," in *Proc. Int. Conf. Digit. Arts, Media Technol. (ICDAMT)*, Mar. 2017, pp. 62–65.

HAN LIU received the M.S. degree from Shanghai Maritime University, where he is currently pursuing the Ph.D. degree. His main research interests include big data, cloud computing, distributed computing, cloud security, machine learning, the IoT, and blockchain.

DEZHI HAN received the Ph.D. degree from the Huazhong University of Science and Technology. He is currently a Professor of computer science and engineering with Shanghai Maritime University. His research interests include cloud computing, mobile networking, wireless communication, and cloud security.



DUN LI received the M.S. degree from the Macau University of Science and Technology. He is currently pursuing the Ph.D. degree with Shanghai Maritime University. His main research interests include smart finance, big data, machine learning, the IoT, and blockchain.