Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.034671
Article

Ether-IoT: A Realtime Lightweight and Scalable Blockchain-Enabled Cache Algorithm for IoT Access Control

Hafiz Adnan Hussain*, **Zulkefli Mansor, Zarina Shukur and Uzma Jafar**

Universiti Kebangsaan Malaysia, Bangi, 43000, Malaysia
*Corresponding Author: Hafiz Adnan Hussain. Email: p97940@siswa.ukm.edu.my
Received: 23 July 2022; Accepted: 01 November 2022

Abstract: Several unique characteristics of Internet of Things (IoT) devices, such as distributed deployment and limited storage, make it challenging for standard centralized access control systems to enable access control in today's large-scale IoT ecosystem. To solve these challenges, this study presents an IoT access control system called Ether-IoT based on the Ethereum Blockchain (BC) infrastructure with Attribute-Based Access Control (ABAC). Access Contract (AC), Cache Contract (CC), Device Contract (DC), and Policy Contract (PC) are the four central smart contracts (SCs) that are included in the proposed system. CC offers a way to save user characteristics in a local cache system to avoid delays during transactions between BC and IoT devices. AC is the fundamental program users typically need to run to build an access control technique. DC offers a means for storing the resource data created by devices and a method for querying that data. PC offers administrative settings to handle ABAC policies on users' behalf. Ether-IoT, combined with ABAC and the BC, enables IoT access control management that is decentralized, fine-grained and dynamically scalable. This research gives a real-world case study to illustrate the suggested framework's implementation. In the end, a simulation experiment is performed to evaluate the system's performance. To ensure data integrity in dispersed systems, the results show that Ether-IoT can sustain high throughput in contexts with a large number of requests.

Keywords: Blockchain; Internet of Things; IoT; access control; ABAC; Ethereum; distributed system

1 Introduction

With the advancement of the Internet and computer technology, many devices are linked through wireless networks, expanding the scope of the Internet of Things (IoT). IoT consists of many decentralized sensors and gateways, enabling many sensor-enabled embedded devices to gather and exchange data without human intervention. IoT devices such as home appliances, vehicles, and physical and smart devices constantly interact with their surroundings, creating various raw data resources such as digital signals, images, text, audio, videos etc. We live in an era in which everything is interconnected to effectively exchange resources and information of all IoT devices and applications via the Internet

[1]. According to Gartner's Ericsson Mobility Report, as of June 2022, the number of connected IoT devices has reached around 14.6 billion [2]. Due to the scattered deployment and massive number of IoT devices, resource access management faces significant issues, new security concerns, and several attacks on IoT devices and data in transit. The resources created by IoT devices often include sensitive data; obtaining them unlawfully can have significant implications [3]. However, traditional access control mechanisms cannot be implemented in IoT networks due to their decentralized nature, low power, and low bandwidth. Therefore, IoT needs real-time lightweight, comprehensive, and scalable access control mechanisms [4,5].

Access control technology has been extensively used to secure IoT resources in various systems and situations. Access control is a technique that restricts the operations or activities of an authorized user. Many traditional access control approaches exist, such as Discretionary Access Control (DAC), Identity-Based Access Control (IBAC), and Mandatory Access Control (MAC). However, all of these approaches are centralized, with the drawbacks of having a single point of failure, difficulty expanding, limited reliability, and low throughput. Due to its performance limitations and mobility, centralized access control presents challenges in meeting access control requirements in an IoT context. An access control paradigm known as Attributed-based access control (ABAC) uses entries, actions, and associated environments to determine who has access to what and how things are used [6,7]. To begin, ABAC gathers information about a person, an object, permission, and a context. Finally, these traits are linked together flexibly. Turning permission management into attribute management provides a fine-grained and dynamic approach to controlling access.

Blockchain (BC) [8] is another new type of data management technology that ensures data is reliable by storing it in different places. Transactions records in the blocks that cannot be reversed or deleted. A hash algorithm links the blocks together in a chain to verify that the data is accurate. The consensus algorithm ensures that all participants in the BC network agree on the same item, which ensures that the information is always the same across all nodes. In this paper, the authors design and implement an Ether-IoT cache-enabled access control system on IoT using BC technology based on Ethereum and ABAC. Ether-IoT can trace records, give dynamic access control management, and address access control issues in IoT by using a distributed architecture.

The following are the paper's key contributions:

1. Create a model for sharing device resources based on the data produced in real-time. The model creates a unique device ID (DID) for each device's data resource. It makes the system simpler to share and save the device's resources.
2. Present Ether-IoT, an access control solution for IoT cache-enabled built on BC technology, and describe how it will function. Users and devices are kept apart by a distributed design that dynamically regulates access rights.
3. On the Ethereum platform, we create four smart contracts (SCs) based on the ABAC model, resource request and action flow, ABAC policy management and device resource management, all based on Ethereum SCs.
4. Explain how to set up the network, install solidity, and call an SC in Ether-IoT.
5. Provide a case study to illustrate the proposed framework using a real-time environment.
6. Design two sets of comparative experiments to evaluate the consensus speed and the system performance using simulators.

The following sections of this paper are structured as follows. Section 2 presents the related work; Section 3 describes the construction and operation of Ethereum and the ABAC paradigm. Managing the security of IoT resources, configuring the ether-IoT system structures, resource and policy models, workflows, and SC implementations are covered in Section 4. The outcomes of two sets of comparison experiments and analysis are covered in Section 5. The last Section, will conclude this report and provide a glance at future research.

2 Related Work

The fast growth of IoT networks, requires a decentralized, secure, and cost-efficient solution, which can accomplish using BC. BC has four main advantages for access control systems: data encryption, decentralization, security, and immutability. BC technology has grown to version 3.0, which has enhanced its core functionality. SCs provide a secure and reliable auto-generated working environment for decentralized applications (dApp). As a result, researchers from several sectors have proposed IoT access control systems that combine conventional methodologies with BC and SCs.

Access control architecture for 5G-enabled IIoT focused on consortium BC was presented in [9] to create efficient and reliable access control. Using the Hyperledger-Fabric framework's three forms of chaincodes, the authors have implemented them on the industrial edge gateways. Compared to Practical Byzantine fault tolerance (PBFT), the suggested framework keeps a lower consensus period as nodes rise and deliver four to five times throughput with reduced hardware resource use and communication consumption. However, the emphasis of this research is not on optimizing BC node data storage for IIoT. The authors in [10] proposed and implemented a BZBAC (BC and Zero-knowledge Token-Based Access Control) paradigm based on BC technology to protect IoT access using zero-knowledge proof and SCs. The traditional access model's single point of failure and credibility has been solved by using an SC to distribute attribute information and access policy. Using zero-knowledge proof technology to realize anonymous access has resolved the issue that sensitive data cannot be kept on a BC. The authors in [11] introduced Attribute-Based Searchable Encryption (ABSE) that enables multiple keyword searches and a BC-based reward and punishment mechanism to establish a fair and reliable searchable encryption system. In the proposed method, the data owner transmits the ciphertext of indexes to searchable encryption service providers (SESP) and uploads the supplementary data to the BC. In [12] authors proposed a BC-based IoT access control system using the ABAC paradigm and Hyperledger Fabric. Each IoT domain has a local BC ledger with channels to control cross-domain access. Each edge device utilizes the requestor's unique ID to block unlawful requests. A policy decision selection method was developed to make off-chain policy real-time judgments. The last step is to develop and analyze the suggested system to show its usefulness.

In [13] BC technology was used to create a lightweight distributed access control system that is more usable and scalable. A Markov chain-based trust management approach has been developed to improve the security of IoT devices. An access control prototype using Ethereum and SCs is also offered. It has been shown that the framework may be used to offer distributed trustworthy access control in the IoT and that it can survive attacks on the network from immoral behavior. According to the authors of [14], the typical centralized publish-subscribe system's security and privacy issues may be addressed with an access control mechanism based on BC and a fully homomorphic encryption algorithm. Experimental evidence supports the theoretical analysis and shows that the plan is technically feasible and efficacious in certain respects. An IoT Cross-Domain Delegation Access Control (CDDAC) method in [15] based on the BC provides a better structure for lighter devices. The adopted trajectory-on-BC strategy makes the system much easier to scale up and easier to change policies. The authors suggested a multidomain delegation trajectory aggregation mechanism to help forensic analysis of intra or cross-domain delegation. It helps confirm suspicions and determine who is responsible for bad behavior after it has happened. However, the research ignored protecting privacy during cross-domain delegation.

Reference [16] for IoT private and sensitive data, a BC-based auditable access control system was suggested to integrate ABAC with BC. To combat external security concerns, zero-knowledge proof is implemented. The private data was saved on the BC network, while the public data was stored beneath the BC network, based on the actual data supplied by IoT devices. An auditable access control model for IoT-based private data access was presented based on ABAC's definition. Hyperledger Fabric was used to develop an open-source BC-based auditable access control solution for IoT data privacy management. Reference [6] proposed a BC-Based Access Control System in IoT called Fabric-IoT. The authors used Hyperledger Fabric to form an SC that enables their framework to represent a fine-grained dynamic access control system in IoT. However, the Hyperledger Fabric is a permissioned BC, preventing full transparency in the system. In reference [17], a Lightweight Scalable BC (LSB) IoT security and anonymity was proposed, which eliminates Proof-of-Work (PoW) in a smart home environment and uses a lightweight consensus algorithm using the SHA256 encryption method. However, SHA256 computationally is very slow, affecting the performance of IoT networks, especially those with many devices. LSB also uses shared keys made by a centralized system manager, which has problems with security, privacy, and speed. Such as a single point of failure, sensitive data being leaked, and too much processing can make it hard to scale. Reference [18] proposed a secure Fabric-based data transmission technology and introduced a power trading center in industrial IoT, which addressed the issues of poor security, high administration costs, and supervision difficulties. This work did not implement on large-scale applications yet, and due to data capacity and storage limitations, the block cannot be stored on the BC.

Reference [19] proposed an SC-Based Access Control (SCAC) architecture made up of numerous Access Control Contracts (ACCs), a Judge Contract (JC), and a Register Contract (RC) to accomplish centralized and trustworthy access control for IoT applications. ACC offers a subject-object pair access control system that validates static access rights based on specified rules and dynamic access rights by observing the subject's behavior. By gathering incorrect information from the ACCs, the JC adopts a misconduct assessment technique to make it easier for the ACCs to validate automatically. The RC stores access control information, misbehavior-judging algorithms, and SCs, as well as maintenance tools to maintain these approaches (register, upgrade, and delete). However, each ACC provides only one client resource pair, because each new pair of apps requires a new contract, which also creates a bottleneck of resources. It is well acknowledged that the ACC has a severe problem with scalability, and this approach also increases the prices.

3 Smart Contract Platform

3.1 Ethereum Platform

In 2008, Nakamoto [20] introduced the concept of a peer-to-peer electronic cash system, called Bitcoin, based on BC technology. The success of bitcoin and BC technology has brought digital currencies to the world's attention. However, the following are some of the drawbacks of using a bitcoin: ***Throughput:*** It has a meagre throughput rate. A maximum of 7 transactions can process in one second. ***Latency:*** The final confirmation of each transaction typically takes around an hour to complete. ***Resource consumption:*** Proof of Work systems is inefficient in using energy, which is detrimental to the environment. When computers carry out more extraordinary computational work, an increased amount of power is used. Because of this, the total amount of additional energy used might end up being rather substantial. ***Privacy issues:*** The Bitcoin ledger is public, so there is no privacy in the transactions. ***Security issues:*** A sufficient level of security can only be provided by BC if there is an extensive network of miners participating for block rewards. Even if the network is relatively small, there is still the chance that an attacker may achieve a simple majority of the network's processing capacity and conduct what is known as a 51 percent assault. To solve all the above problems, **Vitalik Buterin**, a computer programmer, conceived **Ethereum** [21] in 2013 and launched it in July 2015. Ethereum is an open-source BC framework with SC and digital currency capabilities. The platform's native currency is Ether, the second-largest cryptocurrency after Bitcoin [22] in terms of market capitalization. Ethereum adds SCs with an efficient consensus mechanism and high throughputs, more than just a decentralized ledger and immutability. Ethereum introduced Proof-of-Stack in December 2020, with SC, consensus algorithm, more authenticated, encrypted, secured, scalable, and many other prominent features. All Ethereum nodes operate on the EVM in order to preserve BC consensus. Thus, Ethereum's BC offers not just a decentralized database but also a trustworthy computing environment.

3.2 Ethereum Components

The Ethereum SC platform serves as the foundation for the proposed framework, and the principal components of the platform are explained in more detail below. Please see [21] for a more in-depth explanation of the Ethereum platform. ***Block:*** Blocks are collections of transactions, each having a unique hash generated from the preceding block. All of the blocks added to the Ethereum network over its entire history are called BC. ***Mining Node:*** One sort of node in an Ethereum network is called a "Mining Node", responsible for storing all transactions in the Ethereum network's block. The other type of node is called a "Virtual Machine Node". ***Ether:*** Ether is a cryptocurrency similar to bitcoin utilized on the Ethereum network. Similar to Bitcoin, it is a peer-to-peer currency. ***EVM:*** Every user on the Ethereum network is responsible for storing and synchronizing their local copy of the state of the Ethereum Virtual Machine (EVM), which is a global virtual computer. ***Gas:*** The Ethereum network uses a kind of internal money known as gas.

In the same way that fuel is required to power a motor vehicle, Ethereum also needs fuel to run. ***Nonce:*** Nonce refers to the total number of transactions made using an account that the user does not control. In a contract account, nonce refers to the contract count. ***Smart Contract:*** An SC is comprised of variables serving as its states and functions, which store on the BC in a binary format (bytecodes) called Application Binary Interfaces (ABIs) under a particular address for viewing and modifying the states. ***Ethereum Accounts:*** Accounts for Ethereum may be divided into two sorts. *Account held by a third party:* Transactions are stored on these accounts. *Account under contract:* Accounts used to hold information about SCs, as the name implies. ***Transaction Execution:*** Formally, a "transaction request" refers to an EVM request for code execution, whereas a "transaction" refers to the EVM state being changed due to a completed transaction request. A transaction request may be broadcast from any node on the network. For a transaction request to affect the agreed-upon EVM state, another node must verify, execute, and "commit to the network" the transaction. When a piece of code is executed, the EVM's state changes and this new state is broadcast to all nodes in the network upon commitment.

3.3 ABAC Model

An access control system that considers the attributes of the subject, object, authorization, and environment is known as attribute-based access control (ABAC). Access requests are granted or denied based on whether or not they meet specific criteria. ABAC can effectively separate policy administration and access control because of the separation of subject and object properties. In order to make it more flexible, the policy's may be added or removed based on the current scenario. It is also possible to implement fine-grained access control by defining the properties of things from distinct viewpoints. There are four key attributes in ABAC, defined to a set as elements: $\mathbb{A} \in \{\mathbb{S}, \mathbb{O}, \mathbb{P}, \mathbb{E}\}$, where $\mathbb{A}$ represents the attribute, $\mathbb{A} = \{name\colon value\}$. A key name identifies an attribute's value. A user's "ID, name, age, and location, etc.", are all examples of subject ($\mathbb{S}$) attributes that may use to identify the individual initiating the access request. Attributes such as IP address, resource type, and network protocol are all represented by $\mathbb{O}$, which stands for object attribute. The element $\mathbb{P}$ represents permission attributes an action on an object by the subject (i.e., get, set, or execute). $\mathbb{E}$ stands for the "environment" attribute, which refers to the details of the current environment at the moment of the access request, i.e., time and location, Etc. *ABAC policy (ABACP)* is described as $ABACP = \{\mathbb{AS} \wedge or \vee \mathbb{AO} \wedge or \vee \mathbb{AP} \wedge or \vee \mathbb{AE}\}$, which describes the subject's access control rules for the item. It specifies the attribute set required for accessing protected resources. *ABAC request (ABACR)* is described four attributes as a set: $ABACR = \{\mathbb{AS} \wedge \mathbb{AO} \wedge \mathbb{AP} \wedge \mathbb{AE}\}$. $\mathbb{AP}$ denotes the Attributes of Permission taken from the $\mathbb{AS}$ (Attributes of Subject) on the Attributes of Object ($\mathbb{AO}$) in the Attributes of Environment ($\mathbb{AE}$).

4 System Design and Model

4.1 System Structure

The proposed architecture Ether-IoT, for IoT data access control system shown in Fig. 1 contains six layers: Application, Data Governance, Ethereum BC, Clouds and Cache, smart gateways, and the data source (IoT devices).

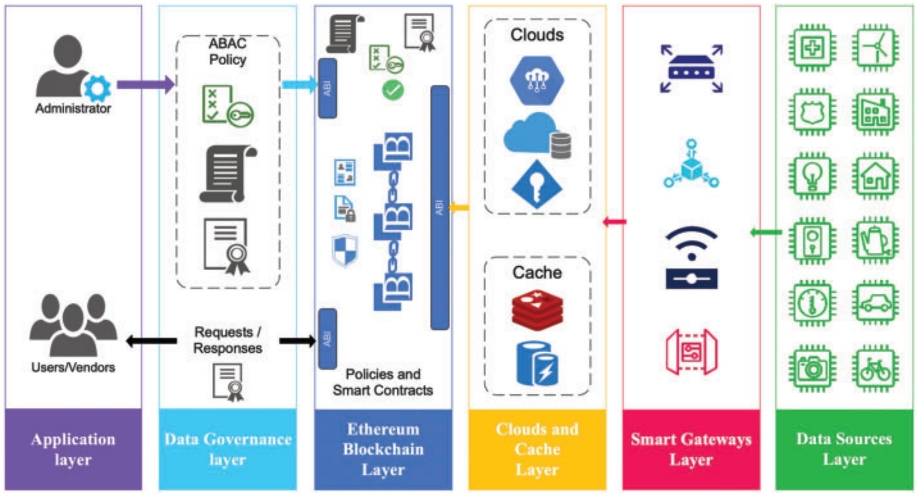


Figure 1: Proposed Ether-IoT architecture

Application and Data Governance Layer: The proposed system splits users into two categories: administrators and ordinary users/vendors (data consumers), where the administrator is accountable for managing the policies and maintaining the smart gateways. The administrator implements the policies at the data governance layer for users and IoT devices to get/set access to the data sources through Ethereum ABI. On the other hand, ordinary users obtain the resources by making an authorization request based on attributes to the BC system. **Blockchain Layer:** All nodes must be certified before joining the BC system. The system is built on the BC Ethereum platform, which executes access control via SCs. Users and smart gateways can access the resources through Ethereum ABI. It mostly takes care of three things: storing device resource data paths, managing user rights based on attributes, and authenticating user access to resources. **Clouds and Cache Layer:** This layer performs two functions: it stores IoT resources' data, maintains the logged-in users' attributes with requested data for a limited time in the cache, and responds to the results directly to the users without delay, which supports maintaining low latency and returns the requested data in real-time. **Smart Gateway Layer:** It's a bridge to make a link between devices and the database layers. It receives messages from devices layer, and sends them to the cloud server. **Data Sources Layer:** It is the most significant shareholder layer in the system, but IoT devices often lack computer power, storage capacity, and battery life. As a result, it is hard to install IoT devices directly as BC peer nodes. IoT devices have a unique device ID or MAC (Media Access Control) address that allows them to be recognized by other devices. In general, the devices may be associated with both users and vendors. When the device creates a new resource, the smart gateway receives a message with the resource's specific identity and transmits the messages using the MQTT (Message Queue Telemetry Transport) protocol to the system.

4.2 Resource and Policy Model

The vast majority of data generated by IoT devices is unstructured [23]. The resulting audio data may be used to create videos when using a camera or microphone to capture external sound. Physical IoT devices can send out signals like humidity, temperature, and light, which can be picked up by sensors and turned into digital data. This data cannot be stored in a relational database since it is mostly unstructured. Since they are real-time data, they must be sent to authorized users as soon as possible. It includes sound and video collected by the device, which is encrypted and transferred to the cloud server over WiFi or 5G. After checking the permissions, the server may send control signals to the device via MQTT or other protocols. To summarize, this study establishes a model for device ($\mathbb{D}$) resources ($\mathbb{R}$) and DID ($\mathbb{U}$): $\{\mathbb{D}\} \to \{\mathbb{R}\} \to \{\mathbb{U}\}$. When it comes to granting users access to resources, the access policy determines who has permission to do so. With permission verification, the BC system provides users with resource data through DID rather than by sending requests to devices directly. Fig. 2 illustrates the relationship between the resources of the devices and the users who utilize them. The step-by-step workflow is described here.

1. The IoT devices generate the raw data and transfer to the remote servers via smart gateways.
2. The remote server process and convert the raw data into helpful information and store into the cloud database and/or send to the users
3. One copy of the registered devices stores to the BC network for the first time only.
4. A chunk of frequently used resource data also stores in the local cache system.
5. To get authorization, users submit requests to access resources either from the cache or the remote server using attributes.
6. The data is sent to the permitted users via either local cache, from the cloud or BC system.

The following device access control policy (DACP) model combines the ABAC model with the characteristics of data produced by IoT devices:

$$\mathcal{P} = \{\mathbb{AS},\ \mathbb{AO},\ \mathbb{AP},\ \mathbb{AE}\},$$

$$\mathbb{AE} = \{createTime,\ endTime,\ allowedIP\}$$

$$\mathbb{AO} = \{deviceId,\ MAC\}$$

$$\mathbb{AS} = \{userId,\ role,\ group\}$$

$$\mathbb{AP} = \begin{cases} 0, & deney \\ 1, & allow \end{cases} \tag{1}$$

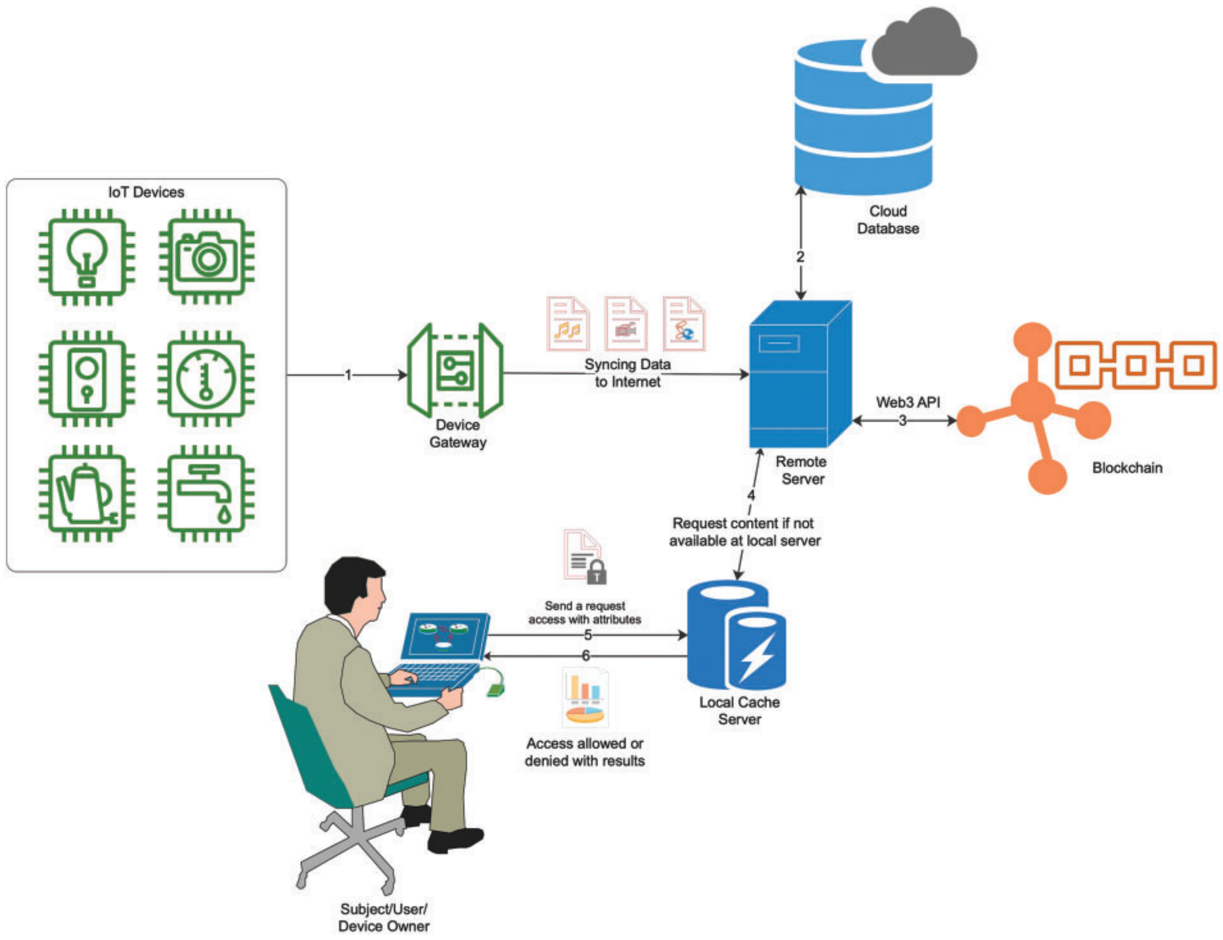

IoT Devices
1
Device Gateway
Syncing Data to Internet
Cloud Database
2
Remote Server
Web3 API
3
Blockchain
4
Request content if not available at local server
Send a request access with attributes
5
6
Local Cache Server
Access allowed or denied with results
Subject/User/ Device Owner

Figure 2: Communication between users and the resources of the IoT devices in the proposed architecture

$\mathcal{P}$ **stands for policy**: It symbolizes a policy for attributes, which contains $\mathbb{AS}$, $\mathbb{AO}$, $\mathbb{AP}$, *and* $\mathbb{AE}$ elements. **Attribute of Subject** denoted by $\mathbb{AS}$, which indicates the attributes of a user (subject) and contains three types: unique identification user (user ID), user role (role), and user group (group). **Attribute of Object** denoted by $\mathbb{AO}$, which indicates resource (object) attributes, consisting of MAC address and device ID. **Attribute of Permission** denoted by $\mathbb{AP}$, which indicates the attributes of the permission, whether the user has access to the resources or not. Allowed and disallowed are denoted by 1 and 0, respectively. The default setting for $\mathbb{AP}$ when it is activated is 1. The system administrator may adjust the value of $\mathbb{AP}$ may be adjusted to 0 to withdraw access authorization. **Attribute of Environment** denoted by $\mathbb{AE}$, which outlines the requirements for access control in the environment. There are three types of $\mathbb{AE}$ attributes: start time, end time, and authorized IP address. Time is a reference to the policy's inception date. The policy's expiry date is known as the end time. It will be null if the current time is later than the end time specified in the policy. Users will not be able to get access with unauthorized IP addresses permitted network access.

4.3 Ether-IoT Workflow

The whole system process is divided into four phases, as seen in Fig. 3. This Section describes the intermediate phases of each segment. Table 1 describes the symbols used in this paper.

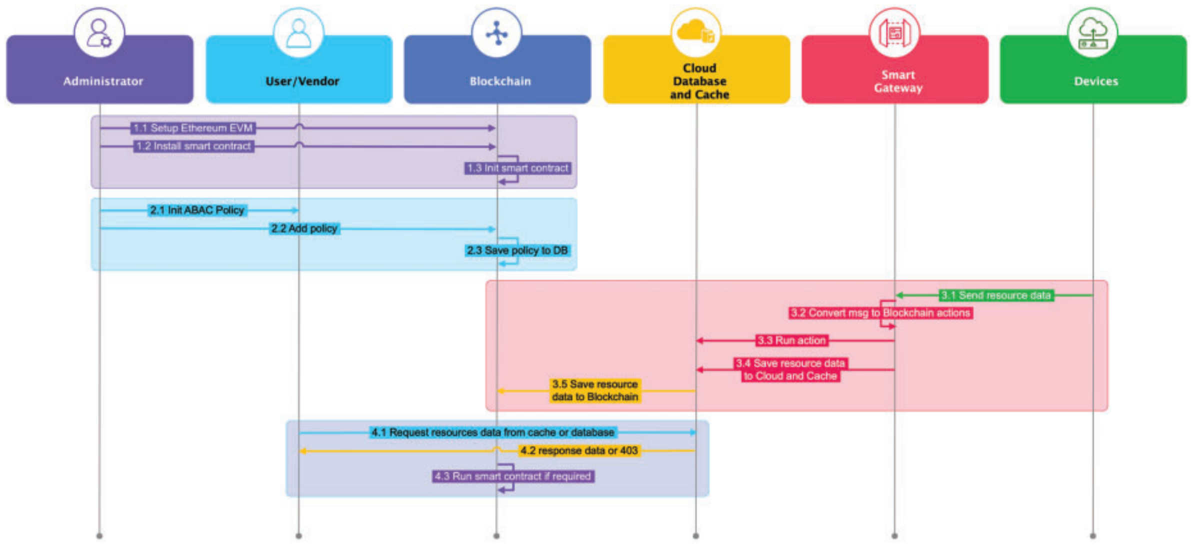


Figure 3: Proposed Ether-IoT workflow

Table 1: Explanation of the symbols used in the paper

Name	Description	Name	Description
$\mathbb{K}$	Secret key pair	$SCode$	Solidity code
AC	Access contract	SF	Solidity functions defined in the source code
CC	Cache contract	PF	Python functions defined in the source code
PC	Policy contract	$Ledger$	Ledger in Ethereum database (LevelDB)
DC	Device contract	$ABACP$	Attribute-based access control policy
$\mathbb{AE}$, $\mathbb{AO}$, $\mathbb{AP}$, $\mathbb{AS}$	Attributes of environment, Attributes of object, Attributes of permission, and Attributes of subject	SG	Smart gateway
SDK	Software development kit	BA	BC action

4.3.1 Phase 1

The system's foundational processes are the setup of the BC Ethereum network and the installation of EVM. These activities need the administrator's involvement in the network flow. This phase primarily consists of three steps.

Step 1: Before building an Ethereum network, one must establish secret key pairs for all its nodes. The $\mathbb{K}$ generates all secret key pairs as $\mathbb{K} \rightarrow \left\{\mathbb{K}_{peer},\ \mathbb{K}_{user}\right\}$

The peer node is hosted in an EVM. Before executing the code, the $\mathbb{K}$ must be moved into their associated node as $Move\,(\mathbb{K}) \overset{move}{\rightarrow} Peer \overset{execute}{\rightarrow} SC$.

Step 2: So far, a basic Ethereum network has been established. Now need to create SCs to build applications. SC source code is written in Solidity and Python $Code\,(\ SF\,(x)\,,\ PF\,(x) \ldots) \rightarrow SC$. To install SC, the admin uses the Ethereum SDK for Python. Every single SCode will be installed on the peer nodes: $Install\,(SC) \overset{SDK}{\rightarrow} Peer$.

Step 3: After the SCode has been installed, it has to go through being started. The SCode is first initialized using the invoke function as: $Invoke\,(Init) \overset{SDK}{\rightarrow} Peer$.

4.3.2 Phase 2

Create ABAC policies and store them into the EVM. In this phase, the administrator and users collaborate in advance to define and design the access policies, which are then uploaded to the BC network by the admin.

Step 1: Access rules (i.e., read, write) are created collaboratively by administrators and users and are based on the properties of the subject (device owner), object (resources), SC actions, and environment attributes as: $Decide\,(\mathbb{AS},\ \mathbb{AO}, \mathbb{AP}, \mathbb{AE}) \rightarrow ABACP$.

Step 2: The administrator then uploads the access policy to the BC $Upload\,(ABACP) \rightarrow contract$.

Step 3: The Admin connects to the BC and runs the PolicyContract to create, edit, or remove the policy. The policy's value is recorded in the ledger and is updated with the transaction's details $PolicyContract\,(ABACP) \rightarrow \{Ledger\}$.

4.3.3 Phase 3

The device sends the data to the smart gateway, which converts the raw data into meaningful formats, stores it in the cloud database and then into the BC as per the triggered actions.

Step 1: The devices' message (Msg), contains the device's identifier or MAC and raw data, sent to the IoT smart gateways through MQTT as: $\{deviceId/MAC, \text{Raw } data\} \rightarrow Msg \overset{MQTT}{\rightarrow} SG$.

Step 2: The smart gateway examines the messages and produces necessary actions for the cloud, cache and BC databases using BA as: $Interpret\,(Msg) \rightarrow BA$.

Step 3: The action is executed by connecting the smart gateway to cloud database and cache as: $Execute\,(\text{BA}) \overset{node}{\rightarrow} CC,\ DC$.

Step 4: Cloud and cache server saves the device ID or MAC address and data of the device resources by calling DC as: $BA\,(deviceId/MAC, Raw\ data, CC) \rightarrow \{Cloud,\ Cache\}$.

Step 5: BC saves the device ID or MAC address with data of the IoT resources by calling DC as: $BA\,(deviceId/MAC, Raw\ data, DC) \rightarrow \{Ledger\}$.

4.3.4 Phase 4

The main phase of the system is the selection of resources according to the attributes. It consists of three steps:

Step 1: User initiates attributed-based access requests to the cache or cloud database as: $userId \rightarrow Request\{\mathbb{AS} \wedge \mathbb{AO} \wedge \mathbb{AP}\} \rightarrow AC$.

Step 2: Invoking the functions of AC after receiving the requests, if the user attributes don't have access, the function will return "0", as: $AC\,(Request) \rightarrow \begin{cases} 0, & Forbidden \\ 1, & OK \end{cases}$.

Step 3: In the validation passes, the local cache or cloud database system queries data by invoking the function CC, if the results are not found in the local cache or cloud, the DC function will invoke and return the current data from IoT devices to the users. If it fails, 403 error (403 stands for forbidden in HTTP status codes) will be returned to the users as: $Result = \begin{cases} 0, & \rightarrow 403 \\ 1, & \overset{DC}{\rightarrow} URL \end{cases}$.

4.4 Smart Contract Design

The SCs serve as the foundation of the proposed solution. The SC structure includes four separate contracts: AC, CC, DC and PC.

4.4.1 Access Contract (AC)

AC function triggers, when the user's request fulfils the requirements of the ABACP. Similar to PC, the user's private key is used to sign the request data. After that, AC uses the user's public key to verify the signature and determine whether or not the user is whom they claim to be. The following is an explanation of each method: ***Auth()*** checks the user's request with the public key and validates its identity. ***GetAttrs()*** receive the attributes to parse the results to the users. ***CheckAccess()*** is the primary function that must be executed to handle access control successfully, as seen in Algorithm 1. First, it retrieves the attribute that was set by the ***GetAttrs***() function. If the requested policy is not found and the requested result is empty, it returns an error code 403.

Algorithm 1: AC.*CheckAccess():* To check the user's access.

```
Input: ABACR, Output: Result or Error
1:  $<\mathbb{A}_u\mathbb{E},\ \mathbb{A}_u\mathbb{O},\ \mathbb{A}_u\mathbb{P},\ \mathbb{A}_u\mathbb{S}> \leftarrow GetAttr\,(ABACR)$
2:  $\mathcal{P} = <\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \ldots \mathcal{P}_n> \leftarrow$ PC.$QueryPolicy(\mathbb{A}_u\mathbb{O},\ \mathbb{A}_u\mathbb{S})$
3:  $if\ \mathcal{P}\ != null\ than$
4:      for $\mathcal{P}\ in <\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \ldots \mathcal{P}_n>$ do
5:          $<\mathbb{A}_p\mathbb{E},\ \mathbb{A}_p\mathbb{P}> \leftarrow \mathcal{P}$
6:          if Value $(\mathbb{A}_p\mathbb{P})$ != 'allow' than
7:              Continue
8:          if $\mathbb{A}_u\mathbb{E} \cap \mathbb{A}_p\mathbb{E}$ than
9:              Continue
10:         Result $\leftarrow$ DC.$GetResult(\mathbb{A}_u\mathbb{O})$
11:     end for
12:     if Result != null then
13:         return Result
14:     end if
15:     return Error('403')
```

(Continued)

Algorithm 1: Continued

```
16: else
17:     return $Error$('403')
18: end if
```

4.4.2 Cache Contract (CC)

CC algorithm triggers, with all the requests by the users or IoT devices, it stores the copy of recently pulled IoT resources' data in the local cache for a limited time period (cache expiration time can vary as per the system settings, set by the admin or users). As seen in the Algorithm 2, ***DC.GetResult()*** function runs if requested results not found by the function ***CC.GetResult()***.

Algorithm 2: CC.*CacheAccess():* To store and check the cache.

Input: AC and DC, **Output:** Result or Error

```
1:  $<\mathbb{A}_u\mathbb{E},\ \mathbb{A}_u\mathbb{O},\ \mathbb{A}_u\mathbb{P},\ \mathbb{A}_u\mathbb{S}> \leftarrow GetAttr\ (ABACR)$
2:  $\mathcal{P} = <\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \ldots \mathcal{P}_n> \leftarrow$ PC.$QueryPolicy(\mathbb{A}_u\mathbb{O},\ \mathbb{A}_u\mathbb{S})$
3:  $if\ \mathcal{P}\ != null\ and\ AC.CheckAccess\ != null\ and\ PC.CheckAccess\ != null\ than$
4:      If CC.$GetResult(\mathbb{A}_u\mathbb{O})\ != null\ than$
5:          Result $\leftarrow$ CC.$GetResult(\mathbb{A}_u\mathbb{O})$
6:      else
7:          Result $\leftarrow$ DC.$GetResult(\mathbb{A}_u\mathbb{O})$
8:      end if
9:      if Result $!= null$ then
10:         return Result
11:     end if
12:     return $Error$('403')
13: else
14:     return $Error$('403')
15: end if
```

4.4.3 Device Contract (DC)

DC is primarily responsible for saving the device's resource parameters: {$deviceId/MAC$, Raw *data*} in the cloud or BC. ***AddResult()*** function is used to store DID as key and raw data as value in the database. *ie* : {*deviceId* : *Raw data*}. ***GetResult()*** returns device data against DID.

4.4.4 Policy Contract (PC)

In order to run ABACP, the PC performs the following functions: ***Auth()*** function same as ***AC.Auth()***. Admin creates an ABACP for users and submits a request to the BC system to add an attributed-based access control policy request (ABACP). The data is encrypted using the PC node's public key, and the request is subsequently signed using the private key. PC uses its public key to authenticate the admin's identity, then uses self's private key to decrypt the data. ***CheckPolicy()*** demonstrated in Algorithm 3, the PC must verify that ABACP is legitimate. The four properties, as mentioned earlier, must be present in a lawful ABACP, and the kind of each attribute must also fulfil the standards.

Algorithm 3: PC.CheckPolicy(): Double-check before inserting ABACP into the database.

```
Input: ABACP, Output: True/False
1:    $<\mathbb{AE}, \mathbb{AO}, \mathbb{AP}, \mathbb{AS}> \leftarrow ABACP$
2:    IsTrue = False
3:    for item in $<\mathbb{AO}, \mathbb{AP}, \mathbb{AS}>$ do
4:        if item $<\mathbb{AO}> \in <deviceId, MAC\ address><\mathbb{AS}> \in <userId, role, group>$ then or
if item  then or if item $<\mathbb{AP}> \in <createTime, endTime, allowedIP>$ then
5:            IsTrue = True
6:        end if
7:    end for
8:    If $\mathbb{AE} == 1$ or 0 then
9:            IsTrue = True
10:   end if
11:   return IsTrue
```

AddPolicy() demonstrated in the Algorithm 4. Once ***CheckPolicy()*** has determined that it is allowable for ABACP to be used, the PC calls the ***AddPolicy()*** function to add ABACP to the DB. Databases updates are handled by the method ***UpdatePolicy(),*** which can be used to alter the ABACP by administrators.

Algorithm 4: PC.*AddPolicy() or PC.UpdatePolicy()*: Add or Update ABACP to the Blockchain DB.

```
Input: ABACP, Output: Error/null
1:   @SmartContract interface implementation
2:   ABI ← Invoke()
3:   if CheckPolicy(ABACP) == True
4:       $Id \leftarrow$ KECCAK-256($ABACP.\mathbb{AO} + ABACP.\mathbb{AS}$)
5:       err ← ABI.PutState(Id, ABACP)
6:       if err != null then
7:           return Error(err.Text)
8:       end if
9:       return null
10:  else
11:      return Error("Wrong Policy")
12:  end if
```

DeletePolicy() can be used to cancel ABACP by admin or if the property "endTime" has expired ***PC.CheckAccess()*** can use this method to remove a linked policy, see Algorithm 5 to check how it works. ***QueryPolicy()*** provides a function to get ABACP by using AS or AO for other SC. Ethereum uses LevelDB, a key-value document-based database that supports complex queries similar to MongoDB.

Algorithm 5: PC.*DeletePolicy():* Delete ABACP from Blockchain DB.

```
Input: $\mathbb{AS}, \mathbb{AO}$, Output: Error/null
1:   ABI ← Invoke()
2:   $Id \leftarrow$ KECCAK-256($\mathbb{AS} + \mathbb{AO}$)
```

(Continued)

Algorithm 5: Continued

```
3:  err ← $ABI.GetState(Id)$
4:  if $err$ != $null$ then
5:      return $Error$(err.Text)
6:  end if
7:  err ← $ABI.DelState(Id)$
8:  if $err$ != $null$ then
9:      return $Error$(err.Text)
10: end if
11: return $null$
```

5 Experiment and Comparison

This Section will demonstrate the steps of the experiment and compare the results to show how well the Ether-IoT technology suggested in this study works in smart home environment. The first part will describe this study's hardware and software environments. The second part, demonstrated how to set up the system and implement access control. The last part, conducted a performance evaluation, a comparison experiment, and an analysis of the findings.

5.1 Hardware and Software

Two laptops and two Arduino microprocessors are used to simulate the experiment described in this work. For more details on the hardware and software configurations breakdown, see Fig. 4 and Table 2. IoT devices and laptops correspond to user devices and IoT gateways in the system. Each laptop device acts as a subject, while the IoT serves as the object, and looked at how to manage access between all devices. Installing the Geth client [24] (a command-line tool written in the Go programming language) on each laptop device turned it into a private Ethereum server. Geth is used to transmit and receive AC results from the SC through transactions.

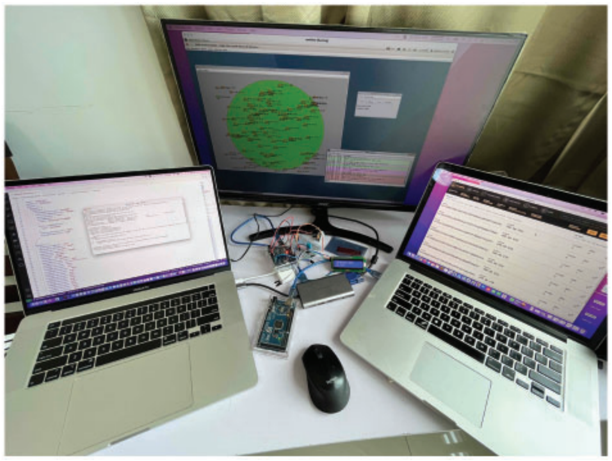


Figure 4: Hardware used in this case study

Table 2: Specifications of hardware and software used in this case study

Computer device	CPU	Memory	Hard disk	Operating system
MacBook Pro	2.5 GHz Quad-Core Intel Core	16 GB	512 GB	macOS monterey (v12.1)
MacBook Pro	2.3 GHz 8-Core Intel Core i9	16 GB	1 TB	macOS monterey (v12.4)
VirtualBox manager	2.3 GHz 8-Core Intel Core i9	8 GB	10 GB	Ubuntu (v12.04, 32-bit)
IoT device	Microcontroller	Flash Memory	RAM/SRAM	
Arduino Uno R3	ATmega328P, 16 MHz, 5 V	16 KB	1 KB	
Arduino mega 2560 R3	ATmega2560, 16 MHz, 5 V	256 KB	8 KB	
Software & tools	Version	Software & tools		Version
Geth (Go Ethereum) [24]	1.10.20	NodeJS [25]		15.15.0
Truffle [26]	5.5.20	NPM [27]		8.5.5
Ganache [28]	7.2.0	Arduino [29]		2.0
Solidity (solc-js) [30]	0.5.16	Visual studio code [31]		1.69.0
Web3.js [32]	1.7.4	ContikiOS cooja simulator [33]		2.7
Python [34]	3.10.2			

5.2 *Implementation*

5.2.1 Initialization of Ether-IoT

Ether-IoT comprises three primary tools, Ethereum Solidity, Python and Arduino. ***Step 1:*** Generate the Ethereum genesis block and secret key pairs for each peer node using the Puppeth tool on any test network. These key pairs are used for transactions, including the node configuration, and move the secret key pairs into each node's directory. The signatures of all nodes may be used to verify their identity on this private BC. ***Step 2:*** Each node's wallet address will be obtained by registering private accounts in the Keystore directory for each node and initializing and registering them.

5.2.2 Installation of Smart Contract

After the network is up and running, Truffle commands may be run in Powershell (Windows) or the terminal console (Linux, macOS) to install the SC. ***Step 1:*** Copy the SC source code to the client (Owner) node's directory. ***Step 2:*** Package and deploy each peer node using the truffle command. ***Step 3:*** In the last step, transfer each SC to other peer nodes and instantiate it. An endorsement copy of each SC is kept in a different node. The procedure of upgrading is comparable to the process of installation. Despite this, only the first node to install SC will be updated instantly; the remaining peer nodes will only be upgraded synchronously when a transaction is made.

5.2.3 Implementation of ABAC

The implementation of the ABAC will follow the models described in Section 4.4. SC may be invoked in a few different ways on the Ethereum platform. Ethereum clients or SDK (supporting .NET, Go, Java, JavaScript, Python, and Ruby, etc.) may be used to activate SC on the Ethereum platform by different nodes. Ethere-IoT uses Python SDK to call SC. ***Step 1:*** Client secret key pairs are generated by Puppeth and kept in a wallet belonging to each user. ***Step 2:*** The administrator starts a client that connects to the peer node to submit or analyze a transaction. ***Step 3:*** When a peer node requests or changes a database, it does so by agreeing with other peer nodes using the EVM node's service.

5.3 Result and Comparison

In order to evaluate how well the Ether-IoT system operates, the authors have prepared two sets of comparison experiments. These studies simulate concurrent access to the system via many threaded nodes. In the first set of studies, authors measure how long it takes AC, CC, DC, and PC to process requests when varying quantities of requests are being processed simultaneously. The virtual client count is configured to 100, 200, 500, 1000 and 5000. The statistical findings are shown in Fig. 5. As seen by the graphs, writing actions (such as "add" and "update") take longer than reading operations (such as "get" and "query"). The increase in the total number of requests leads to a corresponding rise in the system's throughput. Throughput tends to remain steady until it reaches a particular threshold amount. Even when the number of nodes grows, there is no indication that throughput decreases.

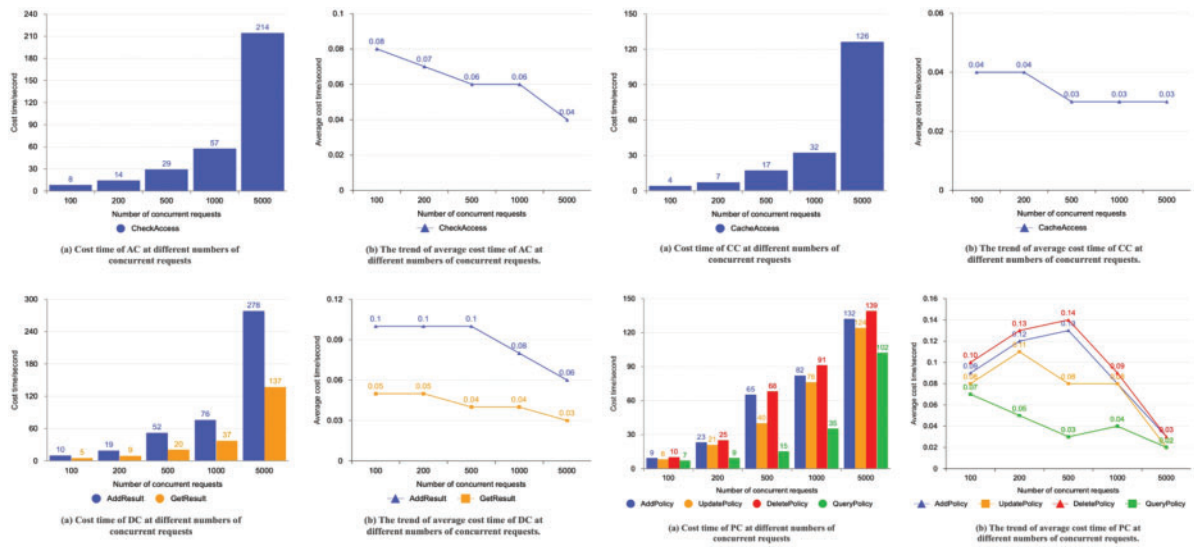
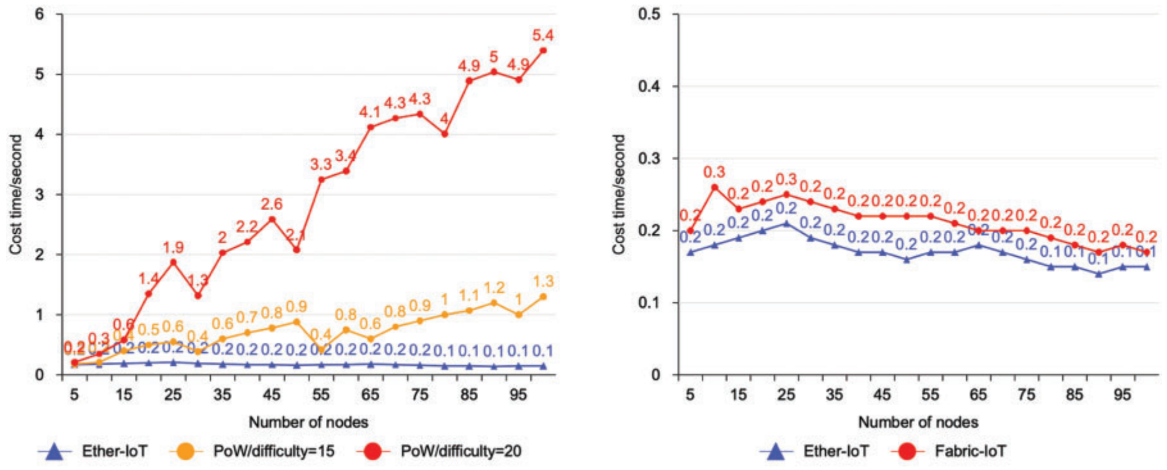


Figure 5: Statistical results of experiments

In the second set of tests, authors simulate the Ether-IoT consensus method by sorting transactions in an MQTT message queue in accordance with Ethereum's consensus mechanism. PoW consensus is implemented in Python, and the difficulty level is adjusted to match the experimental environment. Authors compare the cost time of Ether-IoT and PoW consensus mechanisms under various node counts to evaluate the efficiency of data consistency in a distributed system. The experiment uses a range of 5 to 100 nodes, as illustrated in Fig. 6, where Ether-IoT is compared with

Fabric-IoT [6]. The cost of consensus in Ether-IoT is considerably faster and cheaper than the Fabric-IoT and PoW. For data consistency, these two sets of tests show that Ether-IoT can achieve consensus in a distributed system and sustain good throughput even in large-scale request environments.



(a) Comparison of consensus speed between Ether-IoT and PoW

(b) Comparison of consensus speed between Ether-IoT and Fabric-IoT

Figure 6: Consensus speed comparison

6 Conclusion

This research investigated the IoT access control problems and presented a decentralized and trustworthy cache-enabled ABAC framework established on BC Ethereum named Ether-IoT, which solves traditional AC problems. Using the ABAC model, ABAC policy management and SC application are set up to ensure secure access to the device's resources. A real-time, lightweight, and scalable framework is implemented based on IoT resources and access control requirements to decrease latency and enhance throughput, which provides fine-grained and dynamic physical network access control. Finally, the methods of constructing a BC network, installing EVM, and executing an SC are discussed in detail, and the testing provides convincing results. This study primarily serves as a practical guide for other scholars to do relevant research. This work uses two PCs and two IoT devices to conduct the experiments. In the future, additional devices can be used to assess the system's dependability and performance. Future research might attempt to increase the scalability of Ether-IoT to integrate further IoT applications.

Funding Statement: This work was supported by Universiti Kebangsaan Malaysia under "Dana Pecutan Penerbitan FTSM 2022, Dana Softam 2022". The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] B. L. Nguyen, E. L. Lydia, M. Elhoseny, I. Pustokhina, D. A. Pustokhin *et al.,* "Privacy preserving Blockchain technique to achieve secure and reliable sharing of IoT data," *Computers, Materials & Continua*, vol. 65, no. 1, pp. 87–107, 2020.

[2] E. M. Report, *Ericsson Mobility report-June 2022 report edition*. Torshamnsgatan, Stockholm, Sweden, 2022. [Online]. Available: https://www.ericsson.com/49d3a0/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-june-2022.pdf

[3] U. Jafar, M. J. A. Aziz and Z. Shukur, "Blockchain for electronic voting system—Review and open research challenges," *Sensors*, vol. 21, no. 17, pp. 5874, 2021.

[4] H. A. Hussain, Z. Mansor and Z. Shukur, "Comprehensive survey and research directions on Blockchain IoT access control," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, pp. 239–244, 2021.

[5] U. Jafar, M. J. A. Aziz, Z. Shukur and H. A. Hussain, "A cost-efficient and scalable framework for e-voting system based on Ethereum Blockchain," in *International Conference on Cyber Resilience (ICCR)*, Dubai, United Arab Emirates, pp. 1–6, 2022.

[6] H. Liu, D. Han and D. Li, "Fabric-IoT: A Blockchain-based access control system in IoT," *IEEE Access*, vol. 8, pp. 18207–18218, 2020.

[7] U. Jafar, M. J. A. Aziz, Z. Shukur and H. A. Hussain, "A systematic literature review and meta-analysis on scalable Blockchain-based electronic voting systems," *Sensors*, vol. 22, no. 19, pp. 7585, 2022.

[8] U. Jafar and M. J. A. Aziz, "A state of the art survey and research directions on Blockchain based electronic voting system," in *Int. Conf. on Advances in Cyber Security*, Penang, MY, pp. 248–266, 2020.

[9] Y. Feng, W. Zhang, X. Luo and B. Zhang, "A consortium Blockchain-based access control framework with dynamic orderer node selection for 5G-enabled industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2840–2848, 2022.

[10] L. Song, X. Ju, Z. Zhu and M. Li, "An access control model for the Internet of Things based on zero-knowledge token and Blockchain," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–20, 2021.

[11] H. Gao, S. Luo, Z. Ma, X. Yan and Y. Xu, "BFR-SE: A Blockchain-based fair and reliable searchable encryption scheme for IoT with fine-grained access control in cloud environment," *Wireless Communications and Mobile Computing*, vol. 2021, pp. 1–21, 2021.

[12] S. Sun, R. Du, S. Chen and W. Li, "Blockchain-based IoT access control system: Towards security, lightweight, and cross-domain," *IEEE Access*, vol. 9, pp. 36868–36878, 2021.

[13] P. Wang, N. Xu, H. Zhang, W. Sun and A. Benslimane, "Dynamic access control and trust management for Blockchain empowered IoT," *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 12997–13009, 2021.

[14] H. Tian, X. Ge, J. Wang and C. Li, "Exploiting Blockchain and secure access control scheme to enhance privacy-preserving of IoT publish-subscribe system," *Research Square*, vol. 2021, pp. 1–12, 2021.

[15] C. Li, F. Li, L. Yin, T. Luo and B. Wang, "A Blockchain-based IoT cross-domain delegation access control method," *Security and Communication Networks*, vol. 2021, pp. 1–11, 2021.

[16] D. Han, Y. Zhu, D. Li, W. Liang, A. Souri *et al.,* "A Blockchain-based auditable access control system for private data in service-centric IoT environments," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3530–3540, 2022.

[17] A. Dorri, S. S. Kanhere, R. Jurdak and P. Gauravaram, "LSB: A lightweight scalable Blockchain for IoT security and anonymity," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 180–197, 2019.

[18] W. Liang, M. Tang, J. Long, X. Peng, J. Xu *et al.,* "A secure fabric Blockchain-based data transmission technique for industrial internet-of-things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3582–3592, 2019.

[19] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2018.

[20] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, pp. 1–9, 2008.

[21] Ethereum, *What is Ethereum*. Bern, Bern, Switzerland, 2022. [Online]. Available: https://ethereum.org/en/what-is-ethereum/

[22] Bitcoin, *Bitcoin*. San Jose, CA, USA, 2022. [Online]. Available: https://bitcoin.org/en/

[23] W. Liang, K. C. Li, J. Long, X. Kui and A. Y. Zomaya, "An industrial network intrusion detection algorithm based on multifeature data clustering optimization model," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 2063–2071, 2019.

[24] G. Ethereum, *Geth Client for building private blockchain networks*. Bern, Bern, Switzerland, 2022. [Online]. Available: https://geth.ethereum.org/docs/getting-started

[25] O. Foundation, *Node.js*. San Francisco, CA, USA, 2022. [Online]. Available: https://nodejs.org/en/

[26] E. Community, *Truffle Suite*. Brooklyn, NY, USA, 2022. [Online]. Available: https://trufflesuite.com/

[27] I. Z. Schlueter, *npm Inc. (a Subsidiary of GitHub, a Subsidiary of Microsoft)*, Oakland, CA, USA, 2022. [Online]. Available: https://www.npmjs.com/

[28] E. Community, *Ganache*. Brooklyn, NY, USA, 2022. [Online]. Available: https://trufflesuite.com/ganache/

[29] A. Community, *Arduino*. Somerville, MA, USA, 2022. [Online]. Available: https://www.arduino.cc/

[30] E. Community, *Solidity*. Bern, Bern, Switzerland, 2022. [Online]. Available: https://soliditylang.org/

[31] Microsoft. *VS Code*. *Redmond, WA, USA*, 2022. [Online]. Available: https://code.visualstudio.com/

[32] E. Community, *Web3.js*. Houston, TX, USA, 2022. [Online]. Available: https://github.com/ChainSafe/web3.js

[33] A. Dunkels, *ContikiOS*. 2022. Luleå, Sweden, [Online]. Available: https://github.com/contiki-os/contiki

[34] P. S. Foundation, *Python*. Wilmington, Delaware, USA, 2022. [Online]. Available: https://www.python.org