Hindawi
Security and Communication Networks
Volume 2023, Article ID 6013270, 13 pages
https://doi.org/10.1155/2023/6013270

Research Article

FACSC: Fine-Grained Access Control Based on Smart Contract for Terminals in Software-Defined Network

Bingcheng Jiang,[1] **Qian He**,[1] **Mingliu He**,[1] **Zhongyi Zhai**,[1] **and Baokang Zhao**[2]

[1]*College of Computers and Information Security, Guilin University of Electronic Technology, Guilin, China*
[2]*College of Computers, National University of Defense Technology, Changsha, Hunan, China*

Correspondence should be addressed to Qian He; heqian@guet.edu.cn

Received 14 October 2022; Revised 3 March 2023; Accepted 13 April 2023; Published 15 May 2023

Academic Editor: Zhe-Li Liu

Physical terminals provide network services to upper-layer applications, but their limited memory and processing power make it challenging to perform security updates and patches, leaving them vulnerable to known security threats. Attackers can exploit these weaknesses to control the terminals and attack the network. To restrict unauthorized access to the network and its resources, appropriate access control mechanisms are necessary. In this paper, we propose a fine-grained access control method based on smart contracts (FACSC) for terminals in software-defined networking (SDN). FACSC utilizes the attribute-based access control (ABAC) model to achieve fine-grained control over terminal access networks. To ensure the security and reliability of access control policies and terminal-related attribute information, we utilize smart contract technology to implement the ABAC model. Furthermore, we leverage the programming protocol-independent packet processor (P4) to filter and forward packets in the data plane based on the packet option field, enabling rapid terminal access. Experimental results show that our proposed method achieves fine-grained secure authentication of terminals in SDN networks with a low authentication processing overhead.

1. Introduction

With the increasing adoption of emerging technologies in various fields, such as the Internet of Things, social networks, and mobile Internet, there is a growing need for proper management of large-scale dynamic networks [1]. Fortunately, software-defined networking (SDN) offers a viable solution to this pressing problem. SDN innovatively changes the existing network structure by dividing it into a data plane and a control plane [2], making it possible to optimize network resource allocation and improve network quality. However, the open and untrustworthy network environment of SDN leaves it vulnerable to attackers who may use forged user identities or malicious terminals to attack the network [3, 4].

Access control is a standard approach to safeguard valuable resources from illegal access by unauthorized users or improper use by authorized ones. However, the native SDN controller lacks access control mechanisms for terminal access and cannot perform authentication functions for terminals. As a result, malicious terminals can gain access to the SDN and launch various attacks, leaving the entire network vulnerable to known attacks, such as denial-of-service (DoS) attacks.

The identifier network [5] presents a new possibility for access control of terminals in SDN. By utilizing identifier network technology, all terminals can be uniformly bound with attributes, which makes each terminal unique by its set of attributes, providing support for developing access control policies for terminals. The attribute-based access control model (ABAC) has made a significant breakthrough in addressing complex access control policies, access control granularity, and dynamic scaling of terminal access [6, 7]. ABAC introduces the idea of entity attributes, which describe subject, object, operation, and environment attributes in a unified manner. This makes ABAC an appropriate solution for addressing the problem of secure and controllable network access for many terminals. The ABAC-based scheme [8–11] implements policy-based access

control, which combines various types of attributes (subject, object, operation, and environment attributes). These schemes grant access rights to subjects by defining a set of rules.

It is worth noting that in the aforementioned schemes, authentication of the subject's access rights is usually performed by a centralized entity, which is vulnerable to single points of failure. To avoid the aforementioned security problems, some attempts have been made in recent literature to solve the distributed authentication problem using blockchain technology [7, 12].

Blockchain can be technically understood as a distributed database without the problem of a centralized single point of failure. And the tamper-evident and traceable nature of blockchain can strongly endorse the data on the chain. Thanks to the invention of smart contracts (executable code residing in the blockchain), the blockchain has now evolved into a promising platform for developing distributed and trusted applications. It has attracted much attention from researchers in the SDN community [13, 14]. Predictably, blockchain technology is emerging as a key enabler for achieving distributed and trusted access control.

In this paper, the ABAC model is implemented as smart contracts with the help of blockchain technology, which makes the access control policy free from malicious tampering and enables secure and controlled access to the SDN network for terminals.

The main work of this paper consists of the following.

(1) This paper proposes a fine-grained access control mechanism based on smart contracts for terminals in SDN-based networks (FACSC). It leverages attributes to identify terminals uniquely, and network administrators combine multiple attributes to formulate access control policies based on the ABAC model.

(2) In FACSC, we improve token-based authorization by introducing blockchain and ABAC. Dedicated smart contracts are designed to encapsulate, distribute and verify tokens to satisfy terminals' decentralized, reliable, and flexible access control requirements.

(3) We introduce the P4 forwarding device to realize SDN data plane programmability, which enables fast packet filtering by parsing data streams in the P4 control plane.

The remainder of this paper is organized as follows: Section 2 reviews the related work in recent years. Section 3 introduces the preliminary P4, smart contracts, and ABAC situation. Section 4 introduces the system model. Section 5 presents the proposed ABAC access control scheme based on smart contracts. Section 6 presents the experiments and performance evaluation. Section 7 concludes this article.

2. Related Work

As SDN technology becomes increasingly mature, it has been widely used in production environments, such as Google Cloud Data Center, Huawei Cloud Data Center, etc. In addition, it has also been used in some higher education institutions, such as Stanford University and Tsinghua University, which have implemented SDN as the basic network architecture. However, SDN-based networks also have security threats, such as the lack of access control mechanisms for terminal access.

Duy et al. [15] construct an access control scheme for SDN northbound, introducing the B-DAC framework for decentralized authentication and fine-grained access control for northbound interfaces, which assists administrators in managing and protecting critical resources, indirectly enabling terminals access functionality. Kammoun et al. [16] propose a new SDN architecture based on IoT trust management and access control, where a predefined trust management algorithm calculates the terminal's trust value. Based on trust value, malicious devices are prevented from accessing the network. However, this scheme does not consider the security of the access control policy and is prone to problems such as policy leakage. Awasthi et al. [17] design a scalable, efficient, and cost-effective network architecture that not only meets the changing needs of users but also increases the number of accessing IoT devices, which embed network elements in software rather than dedicated hardware, making it easy to rent from the pool of available devices, enabling rapid device access. Matias et al. [18] propose FlowNAC, an access control scheme for SDNs, which grants users access to the network based on the user's requested target and implements fine-grained access control functionality. However, this scheme is time-consuming and relies on third-party authorization for data flow access, which is likely to be insecure. Benzekki et al. [19] propose an access authentication model based on an SDN network by improving the 802.1X protocol, in which a switch supporting the 802.1X protocol must exist and DHCP and RADIUS servers are connected to this switch to reduce the communication latency with the controller, but the disadvantage of this model is the lack of flexibility in deployment. Fathima and Vennila [20] propose a new algorithm for building IEEE 802.1X-based port authentication schemes, which extends the implementation of EAP from 802.1X to the application and control layers in IPv6, thus improving network throughput and terminals authentication efficiency. However, this scheme is highly targeted and cannot flexibly provide authentication services for more terminals. Ferrazani and Duarte [21] propose an access control model that combines information about users with OpenFlow flow tables, which solves the problem of fine-grained user access control. Still, the model has a single authentication method and is less scalable when the number of terminals is large. Hesham et al. [22] propose a simple authentication model for M2M, which provides different levels of access rights and bandwidth for users through controllers. Still, this model has a single authentication method and cannot be applied to large data centre networks. Yakasai and Guy [23] propose a virtualized network access control scheme based on SDN architecture, which provides a virtualized network access control scheme by combining a stateful role-based firewall with an authorization process to provide a solution for endpoint access control in enterprise networks; however,

this work is applied to a single domain and is not very pervasive.

Although access control policies are considered in some of the previous literature, they are often inflexible and not robust enough regarding security. Furthermore, these schemes do not address the issue of reducing access time overhead. As a result, problems such as high authentication overhead and tampering with control policies may arise.

3. Preliminaries

3.1. Attribute-Based Access Control. In response to the challenge of dynamic and fine-grained access control, which cannot be effectively addressed by traditional models, researchers have proposed the attribute-based access control model (ABAC) [6]. Unlike other models, the ABAC model determines a user's access control privileges based on their entity attributes rather than solely on their identity, eliminating the need for the explicit privilege granted to a subject. The structure of the ABAC model is shown in Figure 1. The core elements of the ABAC model include subject, object, environment, and operational constraints, all of which are described using attributes and attribute values. The generation of access control policies is composed of entity attributes in a flexible way, which improves the representability of access control policies and the model's flexibility. In addition, the ABAC model can also represent the permissions used to control roles and security in other access control models in the form of attributes.

Therefore, the ABAC model is suitable for controlling massive data access. In the terminals access control designed in this paper, attributes are used to identify terminals, making the terminal access flexible and controllable.

3.2. Programming Protocol-Independent Packet Processor (P4). SDN divides the traditional network architecture into the control plane and the data plane, which becomes more flexible than the traditional network but also has some drawbacks. OpenFlow was designed to control only the forwarding behaviour of switches and routers, which limits its ability to manage network traffic and resources. As networks grow larger and more complex, OpenFlow may be unable to handle the increased traffic and routing demands. Because OpenFlow allows for remote control of network devices, there are concerns about security vulnerabilities and potential attacks. OpenFlow is not a standardized protocol, which means that there may be interoperability issues between different vendors' devices and software [24]. To solve the problem of poor scalability caused by OpenFlow's own design, Bosshart et al. [25] proposed the Programming Protocol-Independent Packet Processors (P4) language and the corresponding forwarding model [26, 27]. With the data plane programming capability brought by P4, administrators can not only implement existing network device functions and network protocols such as bridges, routers, and firewalls but also easily support new protocols including VxLAN and RCP [28].

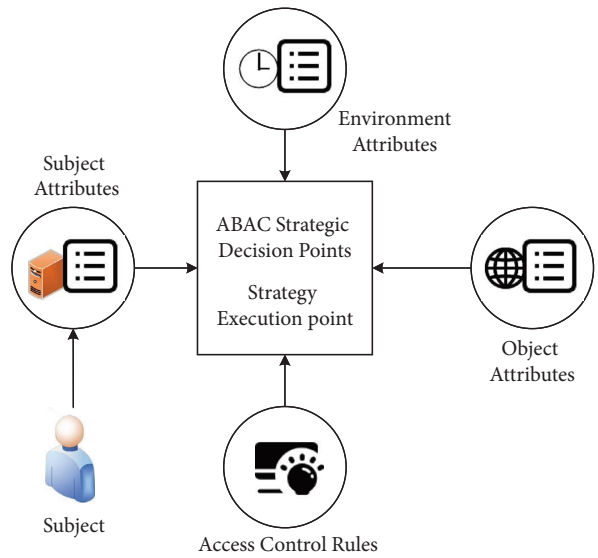

Figure 1: Model structure.

P4 has the language properties of reconfigurability, protocol independence, and platform independence. To this end, P4 defines a set of abstract forwarding models to support the above three language properties. The abstract forwarding model consists of three main components.

(1) **Header parser:** P4 enables developers to customize the packet header structure and parsing process and to configure the debugged P4 code into programmable hardware devices that support P4. This allows for flexible parsing of various packet formats. Upon receiving a message from a terminal, the P4 programmable device follows the message processing logic to separate the packet header from the payload. The information within the packet header is then stored in a self-defined field according to the state transfer rules of the parsing graph, allowing for matching with the flow table in the subsequent pipeline.

(2) **Multilevel pipeline:** This includes the ingress pipeline and the egress pipeline. The ingress pipeline is responsible for modifying the data grouping and determining the port from which the data is forwarded. The egress pipeline only has the function of modifying the attributes associated with the messages. If the researchers want to fulfil the custom business requirements, they have to customize the information in the P4 code such as the matching header field, the executive action and parameters, the number of flow tables in each match action table (MAT), and decide the execution order of the MAT [29].

(3) **Control program:** The written P4 program can be compiled by the P4 compiler to generate a control interface for data parsing or matching. Through this interface, data flow forwarding rules can be installed for the data plane, and hardware facilities such as counters and registers can also be configured

through this interface, as well as statistical information on the status of P4 forwarding devices during operation.

In this paper, we use P4 forwarding devices to implement packet processing in the data plane.

3.3. Blockchain and Smart Contracts. Blockchain is a decentralized digital ledger technology that allows data to be recorded and stored in a secure, transparent, and tamper-proof manner. It was originally developed for the cryptocurrency Bitcoin but has since been applied to a wide range of industries and use cases [7, 30–32]. Transaction information is stored in blocks containing timestamps and references to the previous block and grows as a chain, which is maintained by all participants, and the consistency of the ledger is ensured by consensus algorithms [33]. According to the access rules, blockchains can be divided into public blockchains and consortium blockchains. For public blockchains, participants are free to join and withdraw, and the number of participants is not fixed, as in the case of Bitcoin [34]. For consortium blockchain, only authorized users can join, and the set of participants is usually predefined, such as IBM's hyperledger fabric. With its transparent, traceable, and robust features, blockchain can establish reliable trust between unknown parties and is an effective solution to replace vulnerable central servers in insecure environments. As a blockchain with access control, the consortium blockchain is suitable for access control scenarios that require prevetted users and a relatively stable set of participants.

A smart contract is a concept introduced by cryptographer Nick Szabo in the 1990s. However, due to the lack of a trusted execution environment at that time, smart contracts were not widely applied or developed until the emergence of Ethereum. With the introduction of Ethereum, smart contracts were revitalized and began to gain more attention and use. Smart contracts are designed to eliminate reliance on traditional trusted third parties and are deployed on physical hardware to generate a variety of flexible and controllable smart assets. The life cycle of a smart contract consists of six phases: negotiation, development, deployment, operation and maintenance, learning, and self-destruction. Among them, the development phase includes functional testing of the contract to ensure the correctness of its results, and the learning phase includes operational feedback and updates to the smart contract. In the fabric network, the debugged contract is wrapped in the form of a Docker image, installed in the form of a Docker container in each peer node, and the Init method in the contract is executed after the installation. The installed contract will wait to be invoked by the related business.

In this paper, the ABAC model is implemented as a smart contract, and the contract interface is encapsulated as a Restful service using Fabric-Java-SDK and SpringBoot technology. The encapsulated Restful service is used to realize the functions of terminal access verification and data storage.

4. System Model

We propose a fine-grained terminal access control method based on ABAC and smart contracts to address the lack of effective terminal access control mechanisms in SDN-based networks. The system model, shown in Figure 2, comprises terminals/devices, P4 forwarding devices (P4FD), a blockchain platform, attribute management center (AMC), controllers, and OpenFlow Switches (OFS).

(i) **Terminal/Device:** A terminal is a client used by a user to access the SDN-based network. A device is the carrier of the network resource that the terminal wants to access. When the terminal tries to access the SDN-based network, it will put its own attributes into the *Options* field of the IP packet and send it to the P4FD.

(ii) **P4FD:** The P4FD is responsible for packet processing, including parsing IP packets sent by terminals, filtering out packets without *Options*, and forwarding packets with *Options*. Additionally, it can mirror the packets to the P4 control plane, enabling fast access for the terminals.

(iii) **Blockchain:** The blockchain is the core component of the access control model, and all nodes are required to be authenticated by the Certificate Authority when they join the blockchain system. In our scheme, the blockchain has the following two functions.

(1) The ABAC model is implemented through smart contracts, which mainly include three kinds of contracts, namely, policy contract (PC), device contract (DC), and access contract (AC). The PC formulates access control policies for terminal access to the SDN network based on terminal attributes, device attributes, operation attributes, and environment attributes and stores the policies on the blockchain. The DC stores the set of attributes submitted by the AMC in the blockchain state database and provides attribute support for the PC. The AC adjudicates whether the terminal has the authority to access SDN network resources according to the access control policy.

(2) Provide RestFul service for network administrators to implement smart contract addition, modification, deletion, and query operations.

(iv) **AMC:** AMC is divided into subject AMC (SAMC) and object AMC (OAMC), with two main functions.

(1) The SAMC manages the attribute sets of terminals and submits the attribute sets to the blockchain in batches to prevent terminals from interacting with the blockchain directly and improve the performance of the blockchain.

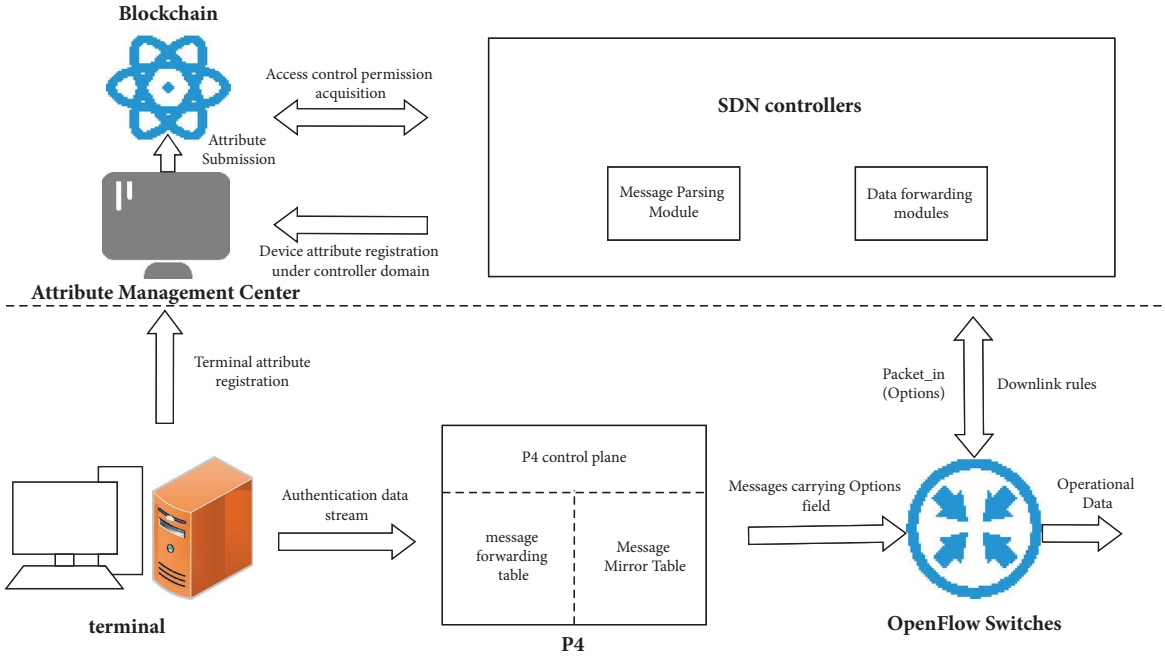

Figure 2: System model.

(2) The OAMC manages the attribute sets of devices under the SDN-controllers domain and submits them to the blockchain in batches.

(v) **Controller:** The main modules developed in the controller are the message parsing module and the data forwarding module.

The main function of the message parsing module is to get the IP packets carrying the *Options* field after filtering by the P4FD. The controller parses the value of the *Options* field, constructs the access request for the current terminal based on the parsing result, and submits the request to the AC through the encapsulated RestFul service. The AC verifies whether the terminal has permission and returns the response status code to the controller.

The main function of the data forwarding module is that the controller determines whether to issue a flow table to instruct the OpenFlow switch to forward messages based on the status code.

(vi) **OFS:** In this paper, we use OpenvSwitch (OvS) as OpenFlow switches, whose main function is to encapsulate packets sent by P4FD into Packet_in messages for forwarding to the controller and to forward the messages normally according to the flow table issued by the controller.

5. ABAC and Smart Contract-Based Access Control

To address the lack of effective access control for terminal access in software-defined networks, we propose an ABAC and smart contract-based access control method for terminal access. First, the ABAC model is formally defined, followed by a detailed description of the access control process.

5.1. ABAC Model. The attribute is the core concept of the ABAC model, which is described by a four-element set $\langle S_i, O_j, P_k, E_n \rangle$. The meaning of each element is explained as follows:

S_i represents the *i-th* subject attribute, which is the terminals attribute, and uses $S_i = [A_{S_i}: V_{S_i}]$ to denote any one attribute item and attribute value in the subject, where A_{S_i} denotes the subject attribute name, such as terminal Mac, IP, etc., and V_{S_i} denotes the attribute value corresponding to the subject attribute name. O_j represents the *j-th* object attribute, which is the devices attribute, and $O_j = [A_{O_i}: V_{O_i}]$ is used to denote any one attribute item and attribute value in the object. $P_k = [A_{P_k}: V_{P_k}]$ represents the operation of the subject on the object, such as read, add, execute, etc. Here, there are two values of V_{P_k}, when V_{P_k} is 1, it means that the terminal is allowed to access the SDN network, and when V_{P_k} is 0 or other values, the terminal is denied access to the SDN network. $E_n = [A_{E_n}: V_{E_n}]$ represents the environment attribute, which indicates the environment attribute required by the access control policy when the current subject accesses the object. E_n represents environment attributes, indicating the environment properties required by the access control policy when the current subject accesses an object. $E_n = [A_{E_n}: V_{E_n}]$ represents any attribute item and value in the environment attributes. A_{E_n} represents any attribute name in the environment attributes, such as the policy effective time, allowed terminal MAC, and IP information. V_{E_n} represents the attribute value corresponding to the attribute name in the environment attributes.

Definition 1. An attribute group $AG_m = \{[A_{m_1}: V_{m_1}] \wedge [A_{m_2}: V_{m_3}] \wedge \cdots \wedge [A_{m_p}: V_{m_p}]\}$ represents a collection of attribute items of the same type, where $m \in \{S_i, O_j, P_k, E_n\}$.

Definition 2. An attribute access request $AAR = \{AG_{S_i} \wedge AG_{O_j} \wedge AG_{P_k} \wedge AG_{E_n}\}$ is a collection of subject attribute groups AG_{S_i}, object attribute groups AG_{O_j}, action attribute groups AG_{P_k}, and environment attribute groups AG_{E_n}. It indicates that the terminal with attribute group AG_{S_i} is requesting operation AG_{P_k} for device attribute group AG_{O_j} under environment attribute group AG_{E_n}.

Definition 3. Attributed-based access control policy (ACP) is a collection of subject attributes, object attributes, operational attributes, and environment attributes formed by means of merging or parsing. $ACP = \{AG_{S_i} \wedge or \vee AG_{O_j} \wedge or \vee AG_{P_k} \wedge or \vee AG_{E_n}\}$ represents the access control rules of the subject to the object and represents the set of attributes required to access the protected object resources.

5.2. Terminal Access Control. Terminal access control has three main parts. In this section, the implementation steps of each part will be explained in detail.

(i) **Registration phase:** Terminals and devices register attributes in the AMC. Administrators generate access policies based on attribute sets (see Algorithm 1).

(1) The terminals and devices submit attributes to the attribute management centre, which uses a RestFul service to execute device contracts with the submitted attribute set and store them on the blockchain.
(2) The administrator obtains the set of attributes submitted by the terminal or the device under the controller domain through the device contract. Then, the access control policy is formulated for the terminal access to the SDN network based on $AG_{S_i}, AG_{O_j}, AG_{P_k}, AG_{E_n}$.
(3) During the formulation of the access control policy, a unique *Token* is generated for the terminal. The *Token* is created by encrypting the relevant attributes in the access control policy. All of these operations are performed in the chain to ensure the *Token* is not tampered with. Finally, the administrator signs the access control policy to ensure its validity.
(4) Once the administrator has defined the ACP, the RestFul service of the operation policy contract is used to add, delete, modify, and verify the access control policy.

(ii) **Authentication phase:** The controller calls the access contract, verifies that the terminal *AAR* requests, and returns the response to the SDN controller (see Algorithm 2).

(1) First, the terminal encapsulates the attribute set in the IP packet's *Options* field. When the P4 forwarding device receives the message, it quickly filters out messages without the *Options* field based on the *IHL* field value. If the packet carries the *Options* field, it's forwarded to the connected switch, which encapsulates it into a Packet_in message and sends it to the controller. The controller then parses the Packet_in message, retrieves the *Options* value, and uses it to construct the AAR request for the access contract for the terminal.
(2) The blockchain verifies whether the AAR request constructed by the controller satisfies the ACP and, if so, generates a response status code and returns it to the controller. At the same time, the controller sends the flow table to OVS, allows the terminal traffic to be forwarded, and caches the *Token* generated by the terminal corresponding to the access control policy with the key as the terminal ID and value as the *Token* value in the cache database and the blockchain. The Token value in the cache database is consistent with the *Token* on the blockchain, and when the *Token* on the chain changes, it will be synchronized to the cache database in real time. If the AAR request does not satisfy the access control policy, the blockchain returns an error message to the controller.

(iii) **Access phase:** If the terminal's connection is interrupted for external reasons, two situations will occur when it is accessed again: first-time access and nonfirst-time access (see Algorithm 3).

(1) For first-time access, the terminal needs to do the same operations as in the authentication phase.
(2) For nonfirst-time access, the terminal adds the Token value obtained for the first time to the *Options* field of the IP packet and initiates an access request to the SDN network. First, when the packet carrying the *Options* field arrives at the P4 forwarding device, the P4 forwarding device parses the packet and filters out the packets without the *Options* field in the IP packet using the *IHL* field. Then, the packet is mirrored to the P4 control plane through the *to_cpu* action, and the P4 control plane parses the *Options* field value and queries the corresponding *Token* to the cache database through the RestFul service. If the corresponding *Token* is queried and is within the validity period, a function similar to the *Packet_in* message is implemented in the P4 control plane, and the flow table is distributed in the control plane to allow the terminal to join the SDN network. Otherwise, IP packets carrying *Options* are

Require: Attribute set
(1) Terminal and device submit attributes to the AMC
(2) AMC submit all attributes to the device contract and store attributes in the blockchain
(3) Administrators generate an access control policy Policy = gen(AG_{S_i}, AG_{O_j}, AG_{P_k}, AG_{E_n})
(4) compute *Token* = *MakeToken* (*Policy.AE.AllowedMAC*, *Policy.AO.DeviceId*, *Policy.AS.TerminalId*)

Algorithm 1: Attributes registration phase.

(1) Terminal sends packets with Identification to P4FD
(2) **if** Option field is none **then**
(3) discards the packets
(4) **end if**
(5) Forwarding the packets to the controller
(6) Controller sends a AAR request to Access Contract
(7) Access Contract verifies the ACP and return a statuscode
(8) **if** statuscode = 200 **then**
(9) Controller sends flow rules to OFS
(10) **else**
(11) **return** Authentication failed
(12) **end if**

Algorithm 2: Authentication phase.

(1) Terminal add *Token* to the *Options* field of the IP packet
(2) Initiate access requests
(3) **if** The terminal is nonfirst-time access **then**
(4) P4FD parses the IP header of packets
(5) **if** *IHL* = *0x05* **then**
(6) Discard the packets
(7) **end if**
(8) The packet is mirrored to the P4 control plane
(9) The P4 parses the *Options* and get the *Token*
(10) Query the *Token* from the cache database
(11) **if** The *Token* exists and has not expired **then**
(12) Distribute the flow table in the p4 control plane
(13) **else**
(14) Resend the IP packet with *Options* to the controller
(15) **end if**
(16) **else**
(17) Perform the same operation as in the authentication phase
(18) **end if**

Algorithm 3: Access phase.

resent to the controller, which realizes terminal access to the SDN network after the operation of the authentication stage.

5.3. Smart Contracts of ABAC. This section provides details on the structure and interface of the ABAC smart contract, which is implemented using smart contracts and can be accessed by the application through the RestFul service.

(1) **Policy Contract:** The responsibility of the PC includes generating, updating, finding, and deleting access control policies. The *Policy* struct is used for this purpose, which consists of AS, AO, AP, and AE substructs. AS, AO, and AE represent the attributes of the terminal, device, and environment, respectively, while *AP* represents access permissions. The main functions of the PC are as follows.

(1) AddPolicy(): This method primarily generates an access control policy using ⟨AS, AO, AP, AE⟩ as the input parameters. The algorithm first validates the legitimacy of the input parameters and then uses the *parsePolicy* method to parse and match the policy JSON string with the policy structure, ensuring the type and number of attributes are correct. Following this, the

CheckPolicy method is called to verify whether the access control policy set by the network administrator satisfies the policy requirements. If it meets the requirements, the *MakeToken* method is called to create a *Token* using $SHA256(AAS.ID, AO.ID)$, which is then assigned to the Token field in the Policy structure. The formulated policy is stored in the blockchain state database as a key-value pair, where the key is generated using $SHA256(AS.ID, AO.ID)$ and the value is policy. Finally, *Policy.ID* is returned.

(2) QueryPolicy(): First, the method verifies the legitimacy of the input parameters and then queries the policy details in the blockchain state database based on *Policy.ID*.

(3) DeletePolicy(): This method executes the DelState method to remove the access control policy corresponding to *Policy.ID* from the blockchain state database.

(4) UpdatePolicy(): This method will override the original access control policy.

(5) QueryToken(): This method queries the corresponding *Token* based on the input *Policy.ID*.

(ii) **Device Contract:** The DC is responsible for adding and finding attributes related to terminals or devices, and it performs the following main functions.

(1) AddAS(): This method saves the registered attributes of the terminal in the blockchain state database. Initially, it validates the input parameters' accuracy and uses the parserAS function to parse the terminal's registered attributes to the AS struct. If the attribute value complies with the defined data type, the terminal's ID is obtained and used as the key, and the attributes are saved as the value. Finally, the blockchain state database stores the key-value pair $\langle ID, Attributes \rangle$.

(2) AddAO(): Similar to AddAS(), this method receives the device's attributes under the management domain of a particular SDN controller and stores them in the blockchain state database.

(3) GetAS(): This method queries the attributes from the blockchain based on the terminal ID and returns the details of the terminal attributes.

(4) GetAO(): Similar to GetAS(), this method queries the device's attributes.

(iii) **Access Contract:** The main function of the AC is to verify whether the terminal has the right to access the SDN network.

(1) AuthACP(): This method verifies the correctness of the struct for the input AAR.

(2) CheckAccess(): This method validates the access privileges of the terminal by examining the AAR request received from the controller. Initially, it validates the legitimacy of the passed parameters and uses the AuthACP() method to verify the AAR struct. Subsequently, the *GetAttrs* method is called to retrieve AO.ID, AS.ID, and AS.MAC from the verified *AAR*. The *QueryPolicy()* method is then used to retrieve the access control policy. If the value of *Policy.AP* is 1, indicating that the terminal has access rights, and access is granted. Otherwise, it is denied. Using the four AE parameters (*CreatedTime*, *EndTime*, *AllowedIP*, and *AllowedMAC*), the AC checks the current access time's validity and the legality of MAC and IP. Finally, the method returns the outcome of the access verification to the controller.

6. Evaluation

To evaluate the feasibility and performance of our scheme, we realized a prototype of its proof-of-concept using Mininet [35] and Hyperledger Fabric [36]. Mininet is a network simulation tool that rapidly creates large-scale SDN prototype systems on ordinary computers with limited resources. Hyperledger Fabric is an open-source consortium blockchain platform widely used in various domains.

6.1. Simulation Setup. As illustrated in Figure 3, we simulate an SDN network using Mininet with Floodlight as the SDN controller. We modify the message parsing module in each controller to parse the Options field of Packet in messages and the data forwarding module to implement the flow table for postauthentication distribution. To enable blockchain functionality, we combine each controller with a Fabric node. Additionally, we leverage a pastry-based dynamic load balancing algorithm [37] to ensure load balancing among controllers. The experiments are conducted on an Ubuntu-20.04 system running on VMware ESXi 6.5 with an Intel(R) Xeon(R) Silver 4114 CPU @2.20 GHz and 16 GB of memory.

6.2. Comparative Summary. In the comparative summary, we focus on four key features: decentralization, fine-grained access control, dynamic access control, and a programmable data plane. It should be noted that among all the schemes compared in Table 1, only our scheme meets all these features. The detailed explanations of the comparative summary are presented below.

6.2.1. Decentralization. Decentralization requires that the entire solution not rely on a central server. For example, in SDN, a single controller may not be able to handle the service requests from a large number of terminals. With distributed edge controllers, service requests from terminals are dispersed to closer controllers, effectively avoiding the vulnerability of a single point of failure.

6.2.2. Fine-Grained Access Control. Fine-grained access control in SDN environments allows administrators to control who can access the network, what they can access, and how they can access it. This helps prevent unauthorized

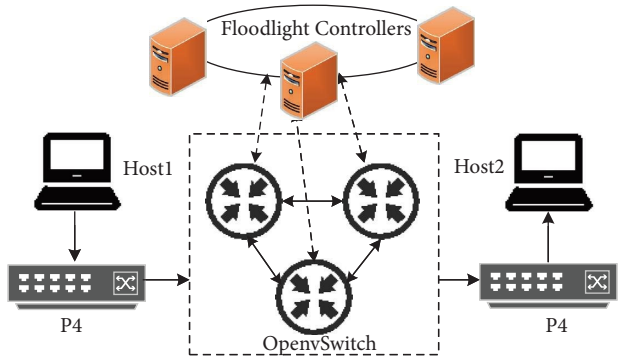

Figure 3: The proposed prototype topology.

Table 1: Comparative summary features.

Schemes	Decentralization	Fine-grained access control	Dynamic access control	Programmable data plane
B-DAC [15]	√	√	×	×
FlowNAC [18]	×	√	√	×
SILedger [7]	√	√	√	×
FGAC [38]	√	√	×	×
FACSC	√	√	√	√

access to the network, which can help protect against cyber threats such as network intrusions, data breaches, and denial-of-service attacks.

6.2.3. Dynamic Access Control. Access policies can be updated and enforced in real time based on changes in the environment. This means that access control decisions can be made on the fly, which can help improve security and reduce risk. ABAC is a scalable access control model that can be easily applied to large, complex environments. This means that organizations can easily manage access control policies for a large number of users and resources.

6.2.4. Programmable Data Plane. Programmable data planes enable greater flexibility and control over how packets are processed and forwarded through the network. This can lead to improvements in network performance, security, and reliability, as well as enable the development of new network applications and services. With programmable data planes, network engineers can define how packets should be processed and forwarded through the network using a high-level programming language such as P4.

6.3. Performance of ABAC Smart Contracts. In this subsection, we conduct tests to fully assess the performance of the ABAC model. Specifically, we measure the average completion time of the three smart contracts in the ABAC model under varying concurrency levels of 10, 50, 100, 150, and 200.

6.3.1. Policy Contracts. Figure 4 shows the average completion time for add, delete, query, and update operations in the policy contract under varying concurrent requests. The figure indicates that the AddPolicy(), QueryPolicy(), DeletePolicy(), UpdatePolicy(), and QueryToken() functions have average response times of 139.4 ms, 36.6 ms, 100.2 ms, 197 ms, and 39 ms, respectively. We also conducted tests for a single operation of each function and found that the completion time for a single add or update operation was consistently between 80–140 ms, a single query operation was consistently between 25–50 ms, and a single delete operation was consistently between 55–90 ms. These results demonstrate that the performance of the policy contract can effectively meet the daily requirements of network administrators for policy add, delete, query, and update operations.

6.3.2. Device Contracts. The performance test results for the device contract are presented in Figure 5, which includes interfaces for adding and querying *AS* and *AO* attributes. As shown in the figure, the average completion times for AddAS(), AddAO(), GetAS(), and GetAO() are 129.6 ms, 141.4 ms, 36.1 ms, and 48 ms, respectively. Additionally, we conducted tests on individual add or query operations, with completion times for the add interface ranging from 70–125 ms and for the query interface ranging from 30–50 ms. These results indicate that the device contract's performance is sufficient for registering and querying device attributes.

6.3.3. Access Contracts. The performance test results of the access contract interface are presented in Figure 6. The average completion time for verifying terminal access rights is approximately 175.4 ms under different concurrent requests, while the completion time of the policy verification function interface remains stable at 82–150 ms during a single verification operation. This takes more time as it

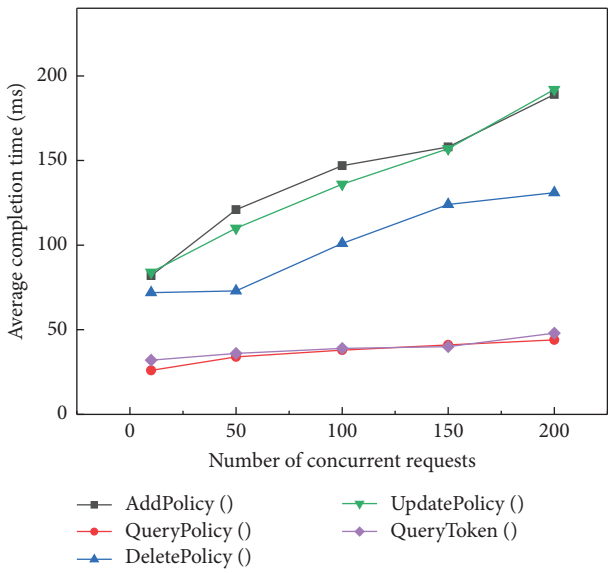

Figure 4: Average completion time of policy contract calls under concurrency.

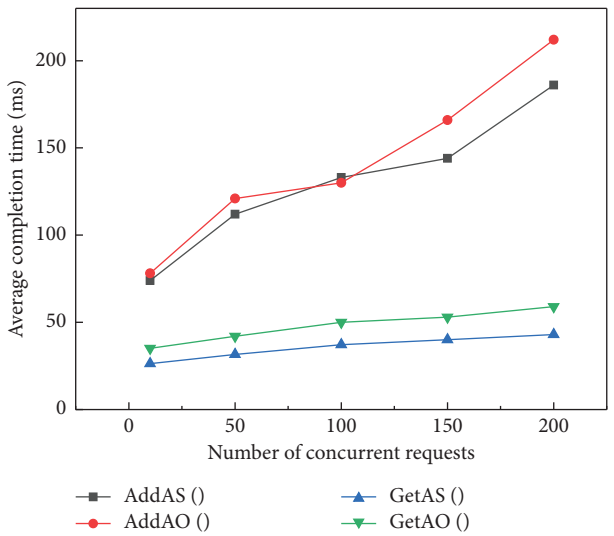

Figure 5: Average completion time of device contract calls under concurrency.

requires interchain code calls in the AC, such as calling the *QueryPolicy* function in the PC.

Based on the performance tests conducted on the policy contracts, device contracts, and access contracts, we can draw the following conclusions: (1) query operations have a minimal time overhead since they do not require consensus and do not need to be recorded on the blockchain. (2) Add and update operations have a significant time overhead because consensus is required among the blockchain nodes before data can be saved.

6.4. Time Overhead for Terminal Access. In our system, there will be two cases of terminal access to the network: first access and nonfirst access.

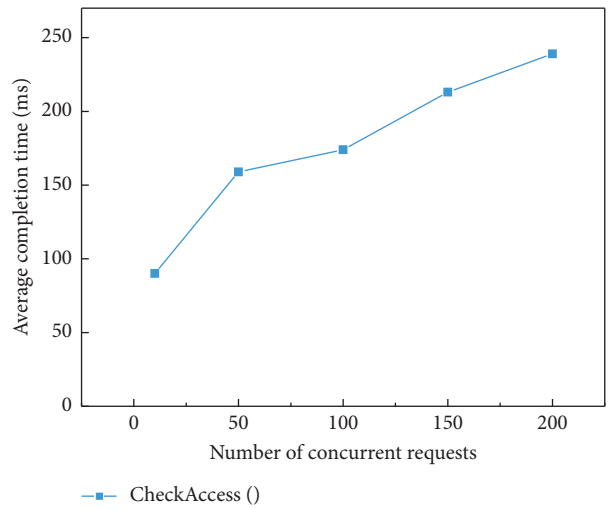

Figure 6: Average completion time of access contract calls under concurrency.

(1) First-time access: For the client-server experiment, we designated $Host_1$ as the client and $Host_2$ as the target server. We ran the client and server codes on their respective hosts. The client $Host_1$ encapsulated the IP packet with the *Options* field into a UDP packet and sent it to P4FD, which filtered the UDP packet and forwarded it to the OpenvSwitch switch. OvS then encapsulated the UDP packet into a *Packet_in* message and transmitted it to the Floodlight controller.

The floodlight controller parsed the message and constructed an *AAR* request. Upon receiving the *AAR* request, the blockchain called AC and returned the response status code to the controller. Finally, the controller sent the flow table to OvS according to the status code, thereby achieving the first access to the terminal.

(2) Nonfirst-time access: When the access is not the first time, $Host_1$ retrieves the previously obtained *Token* and inserts it into the *Verification_Token* field in *Options*, then initiates the *ping* operation. As shown in Figure 7, the terminal successfully passes the *Token* verification in the P4 control plane, and subsequently, the P4 control plane issues the flow table, enabling the successful execution of the *ping* command.

We compared the time overhead of first-time and nonfirst-time requests for terminal access to the SDN network. As shown in Figure 8, for first-time access, the average authentication completion time for different numbers of packets is approximately 197 ms. For nonfirst-time access, message parsing in the P4 control plane and verification of the *Token* are simulated, and the average completion time for verifying each packet authentication is approximately 35.6 ms for different numbers of packets. From the comparison results, it is evident that the authentication overhead for nonfirst-time access is much lower than that for first-time authentication. Therefore, the nonfirst-time terminal

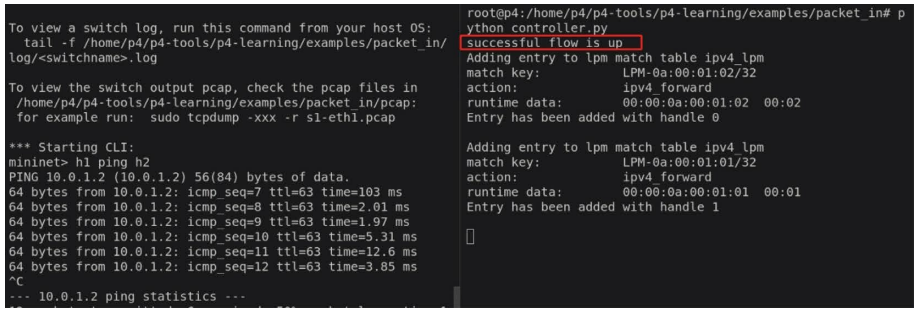

Figure 7: Flow table issued by P4 control plane.

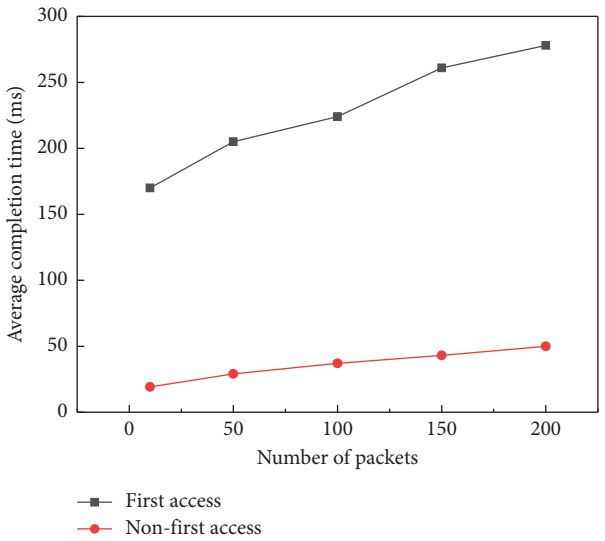

Figure 8: Comparison of time overhead for first-time and nonfirst-time access.

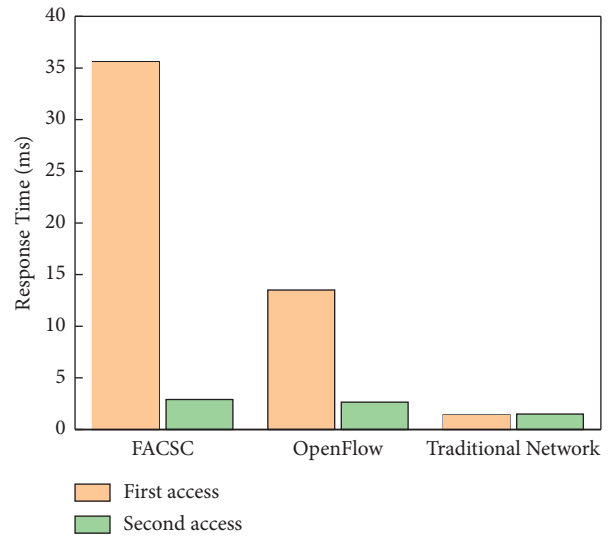

Figure 9: Comparison of forwarding latency between different schemes.

access method in this scheme can compensate for the time-consuming nature of first-time access.

6.5. Data Forwarding Delay. Considering that first-time access to the terminal requires permission verification from the blockchain, which consumes more time, we only compare the latency of nonfirst-time access for the following comparison. We compare the latency of performing two *ping* operations in the traditional network, the OpenFlow network, and FACSC.

In FACSC, when P4FD receives the first *ping* packet, the control plane of P4FD does not issue any flow rules, so it cannot forward the data. At this point, the P4 control plane calls the RestFul service to find the corresponding *Token* of the terminal from the cached database according to the terminal ID. If the *Token* is the same, the P4 control plane issues the flow table, and the traffic will be transmitted. Otherwise, the P4 control plane will refuse to issue the flow rule. The results of the latency evaluation for different schemes are shown in Figure 9.

Based on the comparison of time overhead for the first *ping* in the traditional and OpenFlow networks, it can be concluded that the Floodlight controller takes around 12.1 ms to process data forwarding. In our proposed solution, the time overhead for the first *ping* is 35.62 ms. This delay is higher because the terminal must retrieve and verify the *Token* from the cache database before accessing the network. However, the time overhead for the second *ping* in FACSC is similar to that of the traditional and OpenFlow networks since it only involves normal packet flow between terminals without complex authentication. Therefore, FACSC provides secure terminal access to the SDN network while meeting normal usage requirements for authentication delay.

7. Conclusion

Securing terminal access in SDN networks is crucial for ensuring network security. However, most SDN architectures lack effective access control methods, leaving the network vulnerable to malicious terminal attacks. To address this issue, we propose the Fine-Grained Access Control System for SDN (FACSC), which uses blockchain technology and the ABAC model to implement smart contracts that provide strong security and flexible control policies for terminal access. Additionally, we utilize the programmability characteristics of SDN networks and P4 forwarding devices to offer convenient, efficient, and secure

terminal access, further enhancing the network's security. Our experimental simulations demonstrate that FACSC enables secure, controllable, and traceable terminal access to SDN networks. In future work, we will focus on reducing the authentication time and cost for initial access and using P4 to directly transmit filtered packets to the controller. We also plan to deploy the ABAC model on multiple physical nodes in a real environment for performance testing.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 62162018 and 61861013, in part by the Innovation Research Team Project of Guangxi Natural Science Foundation 2019GXNSFGA245004.

References

[1] D. Chattaraj, S. Saha, B. Bera, and A. K. Das, "On the design of blockchain-based access control scheme for software defined networks," in *Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 237–242, IEEE, Toronto, ON, Canada, July 2020.

[2] O. I. Abdullaziz, L.-C. Wang, and Y.-J. Chen, "Hiauth: hidden authentication for protecting software defined networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 618–631, 2019.

[3] M. Bonola, G. Bianchi, G. Picierro, S. Pontarelli, and M. Monaci, "Streamon: a data-plane programming abstraction for software-defined stream monitoring," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 6, pp. 664–678, 2017.

[4] P. Krishnan, K. Jain, K. Achuthan, and R. Buyya, "Software-defined security-by-contract for blockchain-enabled mud-aware industrial iot edge networks," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 10, pp. 7068–7076, 2022.

[5] H. Zhang, W. Quan, H. C. Chao, and C. Qiao, "Smart identifier network: a collaborative architecture for the future internet," *IEEE network*, vol. 30, no. 3, pp. 46–51, 2016.

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

[7] W. Ren, Y. Sun, H. Luo, and M. Guizani, "Siledger: a blockchain and abe-based access control for applications in sdn-iot networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4406–4419, 2021.

[8] N. Ye, Y. Zhu, R. C. Wang, R. Malekian, and Q. M. Lin, "An efficient authentication and access control scheme for perception layer of internet of things," *Applied Mathematics & Information Sciences*, vol. 8, no. 4, 2014.

[9] S. Bhatt, F. Patwa, and R. Sandhu, "Access control model for aws internet of things," in *International Conference on Network and System Security*Springer, Berlin, Germany, 2017.

[10] R. Zhang, G. Liu, S. Li, Y. Wei, and Q. Wang, "Absac: attribute-based access control model supporting anonymous access for smart cities," *Security and Communication Networks*, vol. 2021, Article ID 5531369, 11 pages, 2021.

[11] Y. Xu, W. Gao, Q. Zeng, G. Wang, J. Ren, and Y. Zhang, "A feasible fuzzy-extended attribute-based access control technique," *Security and Communication Networks*, vol. 2018, Article ID 6476315, 11 pages, 2018.

[12] 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, 2019.

[13] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, Q. Zhang, and K.-K. R. Choo, "An energy-efficient sdn controller architecture for iot networks with blockchain-based security," *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 625–638, 2020.

[14] A. Rahman, M. J. Islam, A. Montieri et al., "Smartblock-sdn: an optimized blockchain-sdn framework for resource management in iot," *IEEE Access*, vol. 9, p. 28361, 2021.

[15] P. T. Duy, H. D. Hoang, D. T. T. Hien, A. G. T. Nguyen, and V. H. Pham, "B-dac: a decentralized access control framework on northbound interface for securing sdn using blockchain," *Journal of Information Security and Applications*, vol. 64, Article ID 103080, 2022.

[16] N. Kammoun, R. Abassi, S. Guemara El Fatmi, and M. Mosbah, "A new sdn architecture based on trust management and access control for iot," in *Proceedings of the Workshops of the International Conference on Advanced Information Networking and Applications*, pp. 245–254, Springer, Sydney, Australia, April 2020.

[17] C. Awasthi, I. Sehgal, P. K. Pal, and P. K. Mishra, "Software-defined network (sdn) for cloud-based internet of things," in *Transforming Management with AI, Big-Data, and IoT*, pp. 185–213, Springer, Berlin, Germany, 2022.

[18] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "Flownac: flow-based network access control," in *Proceedings of the 2014 third European workshop on software defined networks*, pp. 79–84, IEEE, Budapest, Hungary, September 2014.

[19] K. Benzekki, A. El Fergougui, and A. El Belrhiti El Alaoui, "Devolving ieee 802 Devolving IEEE 802.1X authentication capability to data plane in software-defined networking (SDN) architecture: d," *Security and Communication Networks*, vol. 9, no. 17, pp. 4369–4377, 2016.

[20] T. Fathima and S. M. Vennila, "Emphasizing a productive and protective access control to improve authentication using 802.1 x with software-defined networks," in *Proceedings of the International Conference on Computing, Communication, Electrical and Biomedical Systems,*Springer, Berlin, Germany, 2022.

[21] D. M. Ferrazani Mattos and O. C. M. B. Duarte, "Authflow: authentication and access control mechanism for software defined networking," *Annals of Telecommunications*, vol. 71, no. 11-12, pp. 607–615, 2016.

[22] A. Hesham, F. Sardis, S. Wong, T. Mahmoodi, and M. Tatipamula, "A simplified network access control design and implementation for m2m communication using sdn," in *Proceedings of the 2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 1–5, IEEE, Seoul, South Korea, May 2017.

[23] S. T. Yakasai and C. G. Guy, "Flowidentity: software-defined network access control," in *Proceedings of the 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 115–120, IEEE, San Francisco, CA, USA, November 2015.

[24] R. Bifulco and G. Rétvári, "A survey on the programmable data plane: abstractions, architectures, and open problems," in *Proceedings of the 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pp. 1–7, IEEE, Bucharest, Romania, June 2018.

[25] P. Bosshart, D. Daly, G. Gibb et al., "P4: programming protocol-independent packet processors," *ACM SIGCOMM - Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[26] P. Bosshart, G. Gibb, H.-S. Kim et al., "Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn," *ACM SIGCOMM - Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[27] S. Chole, A. Fingerhut, S. Ma et al., "drmt: disaggregated programmable switching," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 1–14, Beijing China, August 2017.

[28] N. Dukkipati, "Rate Control Protocol (RCP): congestion control to make flows complete quickly," *Cités*, vol. 12, no. 2, pp. 45–56, 2008.

[29] S. Kaur, K. Kumar, and N. Aggarwal, "A review on p4-programmable data planes: architecture, research efforts, and future directions," *Computer Communications*, vol. 170, pp. 109–129, 2021.

[30] S. Jiang, J. Cao, J. A. McCann et al., "Privacy-preserving and efficient multi-keyword search over encrypted data on blockchain," in *Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain)*, pp. 405–410, Atlanta, GA, USA, July 2019.

[31] M. Zhang, J. Cao, Y. Sahni, Q. Chen, S. Jiang, and L. Yang, "Blockchain-based collaborative edge intelligence for trustworthy and real-time video surveillance," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1623–1633, 2023.

[32] T. Wang, C. Zhao, Q. Yang, S. Zhang, and S. C. Liew, "Ethna: analyzing the underlying peer-to-peer network of ethereum blockchain," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2131–2146, 2021.

[33] D. Huang, X. Ma, and S. Zhang, "Performance analysis of the raft consensus algorithm for private blockchains," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 1, pp. 172–181, 2020.

[34] S. Nakamoto and A. Bitcoin, "A peer-to-peer electronic cash system," *Bitcoin*, vol. 4, p. 2, 2008.

[35] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Proceedings of the 2014 IEEE Colombian conference on communications and computing (COLCOM)*, pp. 1–6, IEEE, Bogota, Colombia, June 2014.

[36] E. Androulaki, A. Barger, V. Bortnikov et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*Springer, Berlin, Germany, 2018.

[37] B. Jiang, Q. He, X. Li, and H. Huang, "Qos control method based on sdn for mobile cloud service," in *Proceedings of the 2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pp. 275–283, Honolulu, HI, USA, September 2020.

[38] Y. Zhu, X. Wu, and Z. Hu, "Fine grained access control based on smart contract for edge computing," *Electronics*, vol. 11, no. 1, p. 167, 2022.