electronics

MDPI

Article

BCNBI: A Blockchain-Based Security Framework for Northbound Interface in Software-Defined Networking

Sultan Algarni [1,*], Fathy Eassa [2], Khalid Almarhabi [3], Abdullah Algarni [2] and Aiiad Albeshri [2]

[1] Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia
[2] Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia; feassa@kau.edu.sa (F.E.); amsalgarni@kau.edu.sa (A.A.); aaalbeshri@kau.edu.sa (A.A.)
[3] Department of Computer Science, College of Computing at Alqunfudah, Umm Al-Qura University, Makkah 21514, Saudi Arabia; kamarhabi@uqu.edu.sa
* Correspondence: saalgarni@kau.edu.sa

Abstract: Software-defined networking (SDN) has emerged as a flexible and programmable network architecture that takes advantage of the benefits of global visibility and centralized control over a network. One of the main properties of the SDN architecture is the ability to offer a northbound interface (NBI), which enables network applications to access the SDN controller resources. However, the NBI can be compromised by a malicious application due to the lack of standardization and security aspects in the most current NBI designs. Therefore, in this paper, we propose a novel comprehensive security solution for securing the application–controller interface, named BCNBI. We propose a controller-independent lightweight blockchain architecture and exploit the security features of blockchain while limiting the blockchain's computational overhead. BCNBI automatically verifies application and SDN controller credentials through token-based authentication. The proposed solution enforces fine-grained access control for each application's API request and classifies the permission set into strict and normal policies, in order to add an extra level of security. In addition, the trustworthiness of applications is evaluated in order to prevent malicious activities. We implemented our blockchain-based solution to analyze its security, based on the confidentiality–integrity–availability model criteria, and evaluated the introduced overhead in terms of processing time and packet overhead. The experimental results demonstrate that the BCNBI can effectively secure the NBI, based on the fundamental security goals, while introducing insignificant overhead.

Keywords: software-defined networking (SDN); northbound interface security; lightweight blockchain; policy enforcement; trust evaluation

Citation: Algarni, S.; Eassa, F.; Almarhabi, K.; Algarni, A.; Albeshri, A. BCNBI: A Blockchain-Based Security Framework for Northbound Interface in Software-Defined Networking. *Electronics* **2022**, *11*, 996. https://doi.org/10.3390/electronics11070996

Academic Editor: Vijayakumar Varadarajan

Received: 27 February 2022
Accepted: 20 March 2022
Published: 23 March 2022

1. Introduction

The SDN is a revolutionized network architecture that enables the programming of network devices from a central controller. The SDN motivates the idea of decoupling the data plane from the control plane, which removes many complexities and adds flexibility to the overall network infrastructure [1]. The Open Networking Foundation (ONF) promotes the SDN architecture and enables users to define how the data and control planes communicate with each other [2]. The separation of these two planes has given SDN applications the ability to program and control the underlying network infrastructure through the SDN northbound interface. The SDN architecture can be broadly classified into the following planes [3]: an application plane, a control plane, and a data plane, as shown in Figure 1.

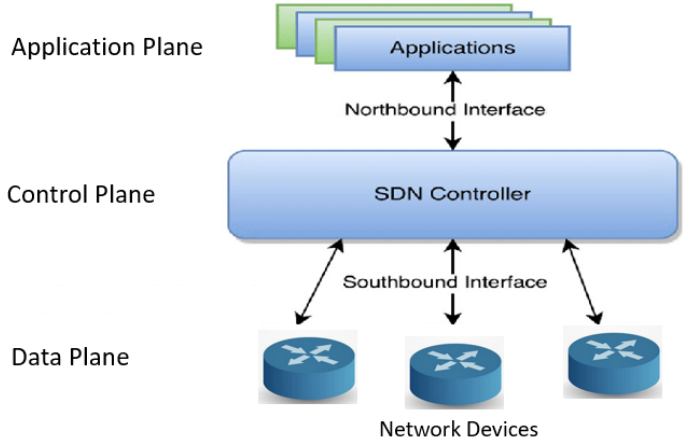

Figure 1. The three planes in the SDN architecture.

The application plane plays a great role in manipulating, configuring, and monitoring the network behavior through the SDN controller. Network policies can be implemented by the SDN applications that communicate with the SDN controller through northbound interfaces. REST APIs are widely used by most SDN controllers in the northbound interface [1,2,4–7]. Accordingly, we use a REST API as a means of communication between the application plane and the SDN controller. The control plane is centralized in a software decision-making entity named the SDN controller [8]. Therefore, the centralized SDN controller enables network administrators to have a global view of the entire network and to configure all network devices from single management. Therefore, the SDN controller can be considered as the brain of the SDN, and is responsible for tasks such as setting up routing, routing packets from one device to another, and ensuring quality of service [1]. The control plane works as a network operating system (NOS) and a middle layer between SDN applications and data plane devices. The southbound interface enables interactions between the SDN controller and its lower layer—namely, the data plane—through an OpenFlow protocol. Moreover, the northbound interface enables interaction between the SDN controller and its upper plane—namely, the application plane—through a REST API. The data-forwarding plane comprises data-forwarding devices, such as switches and routers, that are responsible for forwarding network traffic once paths are determined by the SDN controller. The data plane communicates with its upper layer—namely, the control plane—through the southbound interface.

Despite the advantages brought by the SDN, it introduces new security issues. According to [9–11], some threat vectors have been identified in the SDN architecture. However, the northbound interface (NBI), which enables communication between the application plane and the SDN controller, has been identified as one of the most vulnerable points in the SDN architecture. Current northbound interface implementations may encounter several weaknesses [2,6,12,13]. First, there is a lack of security mechanisms to authenticate and authorize the interactions between the application and the SDN controller at the northbound interface. The absence of such a security mechanism at the NBI enables any application to have full access to read or write the SDN infrastructure, which allows for potentially malicious applications. Secondly, there are no mechanisms to detect the malicious behavior of applications in order to protect the SDN controller from a rogue or compromised application. Unlike the southbound interface, which uses a well-known OpenFlow standard by the Open Networking Foundation (ONF), a standard for the northbound interface is currently lacking [5]. However, many SDN controllers implement REST APIs in the NBI. REST APIs are also used by Facebook and Google, as they offer simple integration and minimal communication overhead. However, most of the SDN controllers implement a REST API without considering the security aspects which could enable malicious applications to access the controller REST API without any constraints [12,14]. Hence, there is a need to secure the application–controller interface in the SDN.

Several studies have attempted to enhance the security of the northbound interface. These studies can be classified into centralized-based solutions and blockchain-based solutions. However, a comprehensive solution that addresses the confidentiality, integrity, and availability threats between the application and the SDN controller, while maintaining cost-effectiveness aspects, is still absent. Most of the centralized-based solutions focus on AAA (authentication, authorization, and accounting) mechanisms; however, their approaches are based on the centralization model, which could lead to a single point of failure. In addition, previous solutions are lacking in terms of ensuring that the database integrity is protected.

Blockchain-based solutions seem to be promising; however, there is a need to take advantage of blockchains while avoiding their high computational overhead. In addition, existing solutions rely on manual operations and ignore SDN behavior-monitoring applications. Due to the prominent features of blockchains, such as decentralization, immutability, auditability, and persistency, the attention of researchers has been attracted by blockchain technology, who have aimed to exploit its security aspects in their solutions since it was introduced in 2008 [15,16]. A blockchain is based on a peer-to-peer (P2P) network, in which each peer stores a replicate of the same data. Accordingly, the system availability and reliability are improved and the possibility of a single point of failure is reduced. Moreover, transactions committed into the blockchain are immutable and cannot be modified, due to the block structure of blockchain technology, as each block is chained to its prior block through the hash of the predecessor block [17].

In this paper, we propose a lightweight permissioned blockchain to leverage the security characteristics of blockchain technology while maintaining low overhead and delay. Unlike public or permissionless blockchains, which allow anyone to join and participate in the consensus procedure, our solution utilizes private or permissioned blockchains, which restrict access to approved participants.

The primary contribution of this paper is to propose a novel lightweight blockchain-based architecture for securing an SDN application–controller interface, named BCNBI. The architecture provides a comprehensive solution that includes the main security features while eliminating the overhead associated with blockchain technology. Its security properties comprise automated token-based authentication, fine-grained authorization for REST APIs, accountability, hashing, encrypted communication, and application trust monitoring. In addition, our solution is controller independent and can be applied to various SDN controllers. We use a web-based REST API as a means for communication among applications, our proposed solution, and the SDN controller. Unlike other blockchain-based solutions, which involve high computational overhead and delays, our proposed solution applies a lightweight blockchain architecture. Finally, we evaluate the security of our solution based on the confidentiality–integrity–availability (CIA) model, which provides fundamental security goals for any secure system [18]. We implemented our solution and evaluated its performance by comparing the performance of the Floodlight controller with and without our blockchain-based solution, in order to measure the additional overhead introduced by our system. The results show that the proposed solution defended the northbound interface against threats to the CIA model, while generating insignificant overhead.

In summary, our major contributions are as follows:

- We propose a novel blockchain-based lightweight architecture for securing the application–controller interface. The solution provides a comprehensive and decentralized security feature to address security threats to the NBI based on the CIA model.
- We propose a lightweight architecture to maintain the security benefits of blockchains while mitigating their high computation overhead by avoiding computationally expensive consensus mechanisms.
- We provide a controller-independent solution to secure the SDN controller against third-party applications, as well as enabling the framework to be applied to other SDN controllers.

- We provide a mechanism to monitor the behavior of SDN applications and automatically evaluate the trustworthiness of an application based on its operations.
- We critically analyze the main centralized and blockchain-based solutions for securing the northbound interface of the SDN controller.

The remainder of this paper is structured as follows: Section 2 reviews the related works and provides a comparison of existing approaches. Section 3 elaborates the design goals, as well as the proposed system design and architecture of BCNBI. Section 4 presents the security analysis and performance evaluation of BCNBI, and discusses the experimental results. Finally, we conclude the paper and introduce future research directions in Section 5.

2. Related Work

Several studies have been conducted to enhance the security of the northbound interface and ensure secure communications between the application and SDN controller. Most of the proposed solutions can be classified as centralized-based solutions [4,6,7,12,19–26] or blockchain-based solutions [14,27–32].

2.1. Centralized-Based Solutions

Centralized-based solutions focus on applying security techniques based on a centralized security model, a basic authentication approach, and simple database storage.

Tseng et al. [4] have proposed a lightweight plug-in solution called ControllerSEPA to secure the interface between SDN applications and the controller. ControllerSEPA is controller independent and provides centralized AAA mechanisms. This approach was evaluated and the authors claimed that the security of the controller had been enhanced with small latency; however, the paper was lacking in terms of protecting the integrity of the data. Oktian et al. [6] have designed a framework to secure the REST NBI for controllers using OAuth 2.0; however, the proposed scheme was not evaluated.

Phan The Duy et al. [7] have introduced a Trust Trident authentication framework for securing NBIs. The framework provides services of AAA mechanisms based on the centralization model in order to secure the communications between the application and SDN controller against malicious applications. Each application is assigned a trust value, which is used to measure the trustworthiness of the application based on its behavior. The application is disabled from further operations when it reaches the deactivation policy threshold. They evaluated their solution based on the STRIDE methodology; however, they did not protect the NBI against availability threats, which could lead to a single point of failure.

Banse et al. [12] have proposed a web-based secure northbound interface that supports external applications and is controller independent. Their framework adheres to encrypted communication between SDN applications and the controller. They offered security services, such as a trust manager, to ensure SDN application authenticity based on application certificates. In addition, the authors introduced a permission system to ensure the appropriate permission is granted to the proper SDN application based on its required network resources or operations; however, the paper was lacking in terms of protecting the integrity of the data.

Toshniwal et al. [19] have proposed a dynamic access control called BEAM, which grants permissions based on the analysis of SDN application behavior. BEAM modifies the assigned access permissions to SDN applications during runtime, for a specified short duration, by analyzing and verifying the application's behavior against its granted permissions. The authors argue that analyzing the application log can enable abnormal application behavior to be detected while decreasing the controller's overhead. BEAM examines the behavior of an application based on its log file, which has a small size. However, BEAM does not ensure that the consistency of policy updates are protected and preserved during runtime by multiple applications at the same time.

Cui et al. [20] have proposed a mutual authentication mechanism between applications and controllers. They addressed the issue of conflicts between unauthenticated

SDN applications and requests. Their authentication system has good portability, as it is controller independent; furthermore, it does not result in significantly high overhead. The authentication system has four main modules: the system log, the security manager, the resource manager, and the permission manager. The authors claimed that, based on their test results, the system worked effectively, defending against illegal application access. However, the solution does not map the application type to its permission sets; furthermore, it has limitations in detecting the behavior of malicious SDN applications effectively.

Aliyu et al. [21] have proposed a trust management framework for enabling secure communication between the SDN application and the controller. It aims to grant request-required permission to perform its required task and not to exceed the permitted operation. Every SDN application has attributes that describe the behavior of the SDN application. These attributes control the primary function of SDN applications based on four main permissions; namely, Read, Write, Notify, and System calls. However, the access control information is stored in a normal database, which could be illegally modified or tampered with. This framework has not been evaluated on a testbed.

Wen et al. [22] have proposed a fine-grained permission system with an isolation mechanism, called PermOF, for protecting the controller by applying the least privilege on the SDN application. PermOF specifies a set of 18 permissions, which can be categorized into four types; namely, Read, Write, Notification, and System permissions. PermOF aims to enforce minimum permissions on third-party SDN applications in order to ensure a secure controller. However, the proposed solution has not been evaluated.

Tseng et al. [23] have proposed a controller-independent dynamic access control system called Controller DAC. This framework enhances the security of the controller against malicious SDN applications. In addition, it addresses the issues of API abuse through use of the following four permission types: READ, ADD, UPDATE, and REMOVE. Controller DAC contains three modules, which are a northbound security extension, a controller-specific IDS, and a high-level policy engine. Their results showed that Controller DAC has low deployment complexity, as code modification is generally not required. Additionally, it can protect the controller against API abuse with a negligible performance overhead. However, the paper does not address the issue of protecting the integrity of data against illegal modification.

Tseng et al. [24] have presented a novel SDN architecture called SENAD, which enables the deployment of trusted SDN applications. They stated that their solution protects NBIs against malicious applications. The proposed architecture splits the SDN controller into an application plane controller (APC) and a data plane controller (DPC). They secure the APC by providing the following security services: Access control, monitoring applications, and authentication. Based on their evaluation, the latency remained low.

Additionally, Al-Alaj et al. [25] have proposed a role-based access control model, called SDN-RBAC, which enables the principle of least privilege at the application level. The framework can detect application sessions and discard malicious operations at runtime.

Moreover, Aliyu et al. [26] have proposed a trust framework to secure the communications between SDN applications and the controller. Their framework has three main modules: the trust module, which evaluates the reliability and trustworthiness of SDN applications by Subjective Logic Reasoning; the authentication module, which verifies and validates SDN applications through a token-based method; and authorization, which authorizes SDN applications using a Boolean Access Matrix. The results demonstrated the scalability and efficiency of the proposed framework.

2.2. Blockchain-Based Solutions

Although the utilization of blockchain-based security solutions has received widespread attention in both academia and industry, only a few papers have utilized blockchains to secure the SDN architecture, and even fewer have considered protecting the northbound interface [27].

To the best of our knowledge, there are only three surveys [27–29] that have discussed the integration of blockchain and SDN architectures. Wenjuan et al. [28] have reviewed blockchain-based SDN frameworks and discussed the opportunities and threats when combining blockchain and SDN technologies. They pointed out that the combination of these technologies enhanced the security architectures in various scenarios. The authors analyzed the security applications that use blockchain technology in SDN, and observed that most of the proposed studies have focused on IoT solutions [27]. In [29], the implementation of blockchains in SDNs was reviewed based on security and non-security perspectives. In addition, the challenges of integrating these two technologies were discussed.

Several studies have leveraged the benefits of blockchain technology to enhance the security of the application–controller interface [14,30–32]. In [30], ChainGuard was used as a firewall for blockchain applications, interacting with blockchain nodes to filter network traffic. The proposed framework filters traffic by recognizing the origin of legal traffic and blocks illegal packets to protect the blockchain nodes. ChainGuard applies an access control mechanism, and its primary goal is to ensure that the nodes are secure against flooding attacks and unauthorized accesses.

Hoang et al. [31] have applied blockchain technology to their proposed solution to enhance the security of NBIs. Their framework uses blockchain technology to ensure the integrity of the database against unauthorized access and protect the SDN controller from the single point of failure. In addition, their framework provides AAA mechanisms for enhancing the security of the NBI. However, they did not analyze the trustworthiness of application API requests to detect the malicious behavior. Their solution was also not fully automated, as it involves human intervention in issuing and verifying the authentication token.

In [32], the authors proposed a blockchain-based trust establishment between the application and the SDN controller. The objective of the system is to control the application's authentication and behavior, as well as network resource management. The system uses a smart contract to store application information. The authors indicated that the use of blockchain enables secured authentication between vehicular applications and the SDN controller. However, their proposed solution has not been evaluated.

In [14], the authors proposed an STHM architecture which utilizes SDN with blockchains to establish a trust management framework. The STHM is used to secure the northbound interface of the SDN controller in the context of healthcare monitoring applications. Their framework ensures trust verification of the IoT healthcare devices and provides AAA mechanisms to establish trust between network applications and the SDN controller. They use permissioned blockchains and apply the Proof-of-Work (PoW) consensus algorithm, which is computationally expensive and may be not suitable for Internet of Things (IoT) devices. Jiang et al. [33] proposed BlocHIE, a Blockchain-based platform for healthcare information exchange. They introduced two fairness-based transaction packing algorithms named FAIR-FIRST and TPFAIR. These algorithms improve the throughput of the system and the fairness among users. The packing algorithms are evaluated in terms of fairness and throughput. The result indicates that FAIR-FIRST improves fairness, and TPFAIR enhances the system throughput. Moreover, in [34], the authors proposed a blockchain-based data management system that enables privacy-preserving and efficient multi-keyword search protocol. They developed a bloom filter-enabled multi-keyword search protocol which allows searching over encrypted data on the blockchain with minimum delay and low financial cost. The experimental results demonstrate the advantages of the proposed protocol over prior traditional approaches. Jiang et al. [35] presented a fairness-based transaction packing algorithm for private blockchain-based Industrial IoT, namely FAIR-PACK. The authors proposed a heuristic and a min-heap-based optimal algorithm inside FAIR-PACK for diverse parameter settings. The results articulate that FAIR-PACK outperforms prior packing algorithms in terms of fairness and average response time.

3. System Design

3.1. BCNBI: Design Objectives

This subsection discusses the design objectives of our proposed solution BCNBI, as detailed in the following.

- Tamper-proof data storage: We applied blockchains to our framework, which ensure data immutability and the achievement of integrity checks by design against data modification. The blockchain applies a Merkle tree which is a hash-based data structure.
  Data stored in the BCNBI cannot be changed or deleted, which protects our system against tampering, as each block in the blockchain is connected by a cryptographic hash link. Each block has a cryptographic hash of the predecessor block and each block generates its hash value based on all previous block elements, along with its hash value. Furthermore, the blockchain storage is duplicated at each BC Peer. Hence, it is hard to tamper with the data stored in all BC Peers and compromise the cryptographic hash of all connected blocks.
- Decentralization architecture: As a centralized SDN controller could result in a single point of failure, there is a need to defend the SDN controller against such a failure. Taking advantage of the decentralization aspect of blockchain technology significantly improved the availability and reliability of our framework and eliminated the need for an authentic third party. The blockchain is based on a peer-to-peer network, in which a group of peers manages the network and each peer has a replicate of the blockchain data.
- Lightweight architecture: The BCNBI framework provides a lightweight solution based on a customized private blockchain. Our framework eliminates the blockchain's overhead and delay while maintaining the security and privacy benefits of blockchain technology. Consensus mechanisms, such as Proof of Stake (PoS) or Proof of Work (PoW)—which are computationally expensive—were eliminated. In addition, we maintained the security of blockchains to meet the strict requirements of industries and governments in keeping their sensitive data private. The BCNBI framework is centrally managed by its owner, and restricts access to only selected nodes based on the blockchain owner.
- Controller independence: Our solution is based on a REST-based northbound API, which enables communication through a web-based interface to achieve flexibility, is controller independent, and supports external SDN applications. REST uses HTTP methods to access resources, and the data are transmitted in JSON format. Our BCNBI framework takes place outside of the SDN controller.
- Authentication: The proposed framework authenticates and verifies the network application and SDN controller before interaction. As the communication channel between the network application and the SDN controller is encrypted with TLS, when the user's credentials are successfully authenticated, the BCNBI delivers a JSON Web Token (JWT) to the authenticated user; this token needs to be used by the user with each request between a network application and SDN controller. As a result, user authenticity is ensured based on the user's credentials and authenticated token.
- Fine-grained authorization: We aim to achieve the concept of a secure method by design. Therefore, the permissions of applications are stored in BC Peers, in order to ensure the security and integrity of the access control policies. The goal of authorization in our framework is to validate whether a request from the application is approved or disapproved based on certain conditions. If the application API request is approved, then the application only accesses its predefined API permission on the controller resources. Otherwise, the application request is rejected.
- Confidentiality: To prevent eavesdropping and tampering attackers, the application–controller plane interface (A–CPI) is encrypted using the Transport Layer Security (TLS) protocol. Therefore, the data transmitted between the application and SDN controller are kept secret and are only read by the authorized user.

- Application trust evaluation: Our system prevents the abnormal behavior of API requests by monitoring SDN application–controller communication. Each application is assigned with a trust attribute, which measures the application's trustworthiness based on its operations. This module can be used to decide whether to disable an application when a threshold, which considers the abused application as malicious, is reached.

3.2. BCNBI: Design Overview

Based on the design objectives stated above, we propose a novel lightweight blockchain system, named BCNBI, for efficiently securing the SDN northbound interface. The BCNBI framework adheres to a Security-by-Design principle and comprises security features such as encrypted communication, token-based authentication, fine-grained authorization, and application trust monitoring. These restrict access to trusted applications and ensure data confidentiality. In addition, inheriting the securely distributed architecture of the blockchain technology while limiting the blockchain's computational overheads is one of the essential design principles of the BCNBI framework. The data integrity is ensured by the blockchain immutability feature, and the availability and reliability of our framework are improved by the decentralized nature of the blockchain. We introduced a security layer between the application plane and the control plane in order to enable secure communications between the application and the SDN controller. This security layer is implemented based on the REST-based northbound interface, which uses HTTP methods for communication. REST APIs are flexible as they support various SDN controllers such as OpenDaylight, ONOS, Floodlight, HP VAN SDN, DISCO, Ryu, and POX; hence, our solution uses REST APIs [1].

Our solution uses private blockchains, which meet the requirements of industries and businesses in providing a controlled and permissioned environment to protect their sensitive data while taking the advantage of a secure blockchain system.Unlike Bitcoin, which uses PoW, our framework applies lightweight consensus algorithms to eliminate the overhead and delay associated with traditional blockchains. Our system not only stores the block header and transactions in blockchains, but also stores the user and policy header in each BC Peer, which significantly enhances the level of security and ensures the immutability and availability of our system.

Our system protects the SDN controller's APIs by intercepting and evaluating the security of application requests before they reach the northbound interface of the SDN controller.

3.3. System Architecture

The high-level architecture of our solution is given in Figure 2, consisting of the following main components: BCNBI APIs, the BCNBI Blockchain Manager, and the Controller API Handler. Figure 3 demonstrates the main scenario and high-level interactions among the SDN application, the BCNBI system, and the SDN controller. In the following subsections, we explain each component in detail.

3.3.1. BCNBI APIs

These REST APIs represent the interface and front end of our proposed solution. They enable fixable and efficient interactions among the applications, the BCNBI system, and the SDN controller. The BCNBI REST APIs enable controller independence, as the proposed solution is independent of the underlying SDN controller. The BCNBI APIs include the entities detailed below.

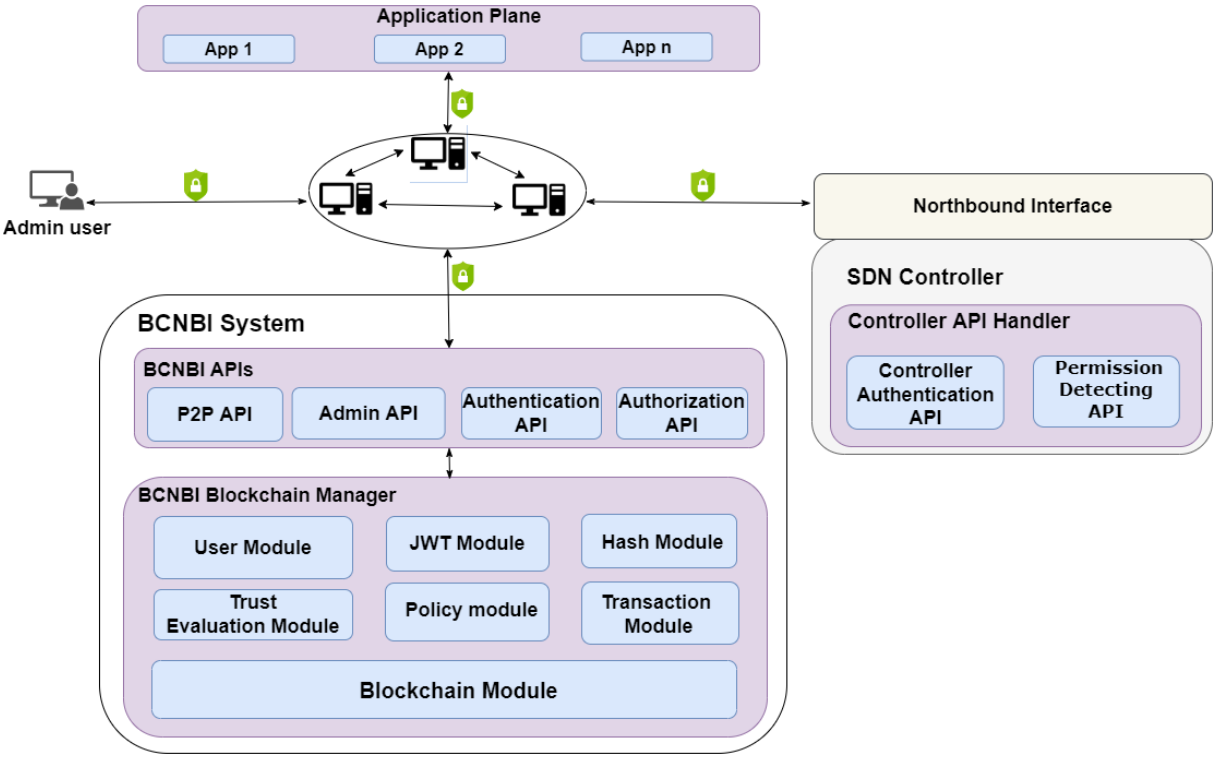

Figure 2. The high-level architecture of the proposed solution.

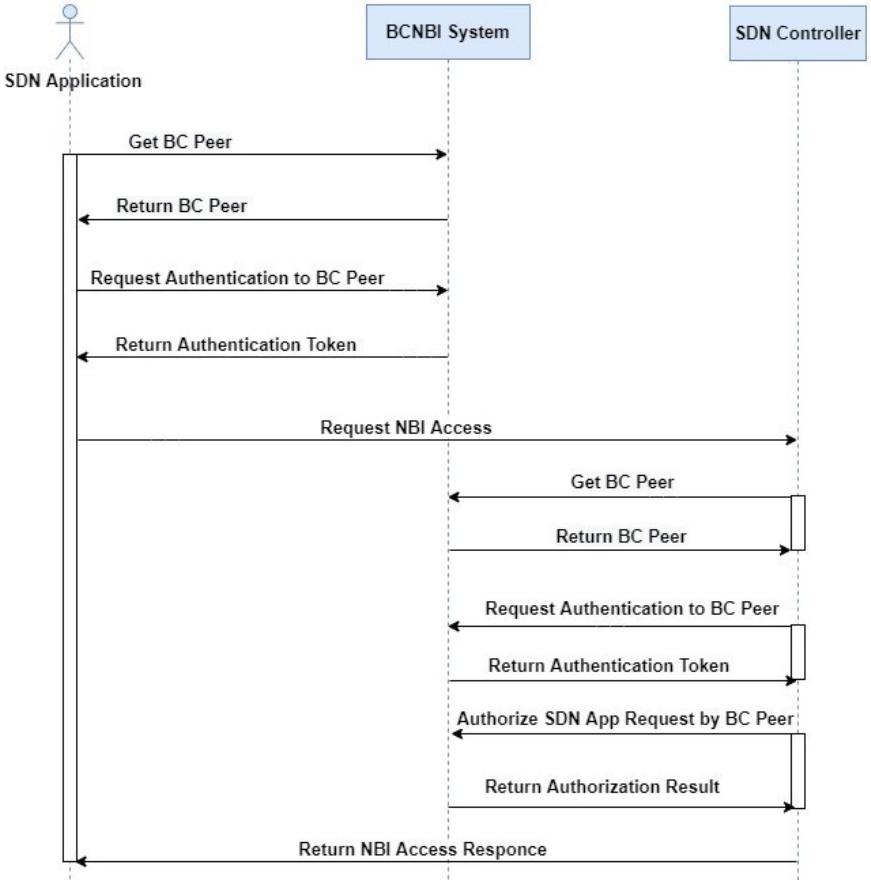

Figure 3. Main scenario and high-level interaction of the proposed solution.

Authentication API: This is a REST API that is responsible for authenticating all users, such as admin users, application users, and SDN controller users. Figure 4 shows the workflow for authenticating the SDN application through the Authentication API. As we use a P2P network, our system uses a round-robin algorithm to balance the requests among the BC Peers. Therefore, the user authentication request is assigned to one of the BC Peers based on the round-robin algorithm. The user undergoes the following authentication process:

- Ensure the BC Peer that sends the request is joined in our blockchain network;
- Authenticate the user's credentials from the latest block's user header;
- Check that the user trust level is not malicious based on the latest block's user header of BC Peer.

If all of the above conditions are met, then the user is successfully authenticated and calls the JWT module to generate an authentication session token, which needs to be used by the user with each request during the session period.

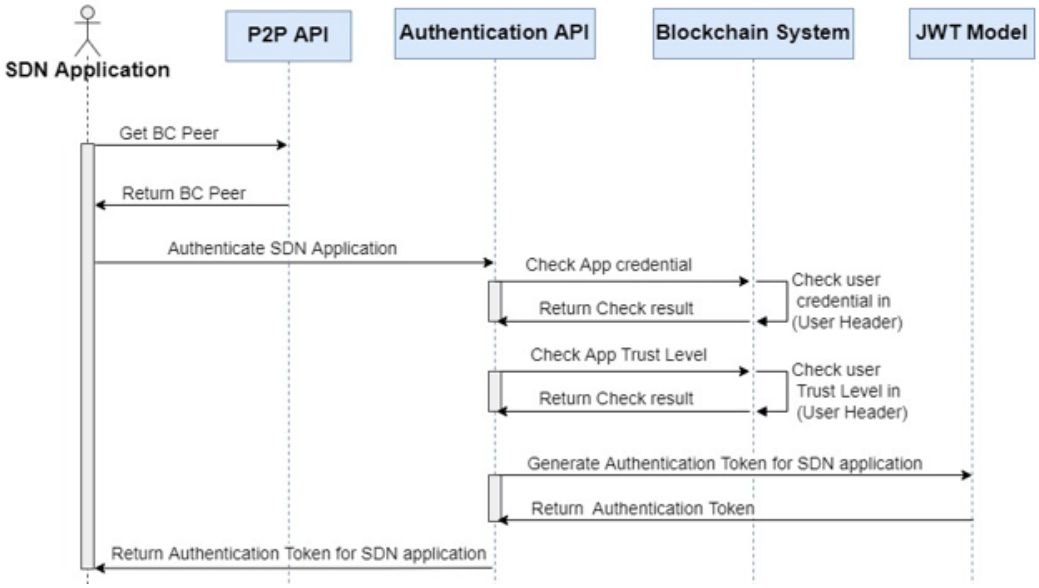

Figure 4. Authentication API workflow for SDN application.

Authorization API: This REST API is responsible for authorizing the application API request that comes through the SDN controller. Therefore, this API is only accessed by the SDN controller for validating application API requests, as illustrated in Figure 5. When the SDN controller receives an API request from an application to access its resources, it communicates with our BCNBI system and obtains one of the BC Peers based on the round-robin algorithm. Then, it validates the application API request based on the following authorization process:

- Ensure the BC Peer is joined in our blockchain network;
- Authenticate the SDN controller's credentials based on the latest block's user header, by invoking the Authentication API;
- Check the user trust level is not malicious, based on the latest block's user header of BC Peer;
- When the SDN controller is successfully authenticated, then the JWT module generates an authentication session token that needs to be used by the SDN controller with each request within the session period;
- Ensure the requested user is a controller from the latest block's user header, as this API is only permitted for the SDN controller user;

- Validate the application API request which is encapsulated within the SDN controller authorization request that is sent to our BCNBI system, taking into account that the SDN controller request must contain two authentication tokens (in particular, for the application and for the SDN controller);
- Validate the authentication tokens by the JWT module;
- Check whether the application user has permission to request the API of the SDN controller based on the latest block's policy header;
- Ensure the user's trust level is not malicious, based on the latest block's user header; and
- Validate the transaction based on the agreement of other BC Peers, using validate-BlockchainApi.

If all of these conditions are satisfied then the application request is successfully authorized and the SDN controller allows the application to access the requested API of the SDN controller; otherwise, the application request is discarded. Our system maintains a value for the last blockchain update, named the DateTime parameter, which is used to determine the last updated blockchain. This value is continuously updated with each transaction added to the blockchain. The system adds a transaction to the local BC Peer and broadcasts the updated local blockchain to the other peers using the P2P API, as illustrated in Algorithm 1.

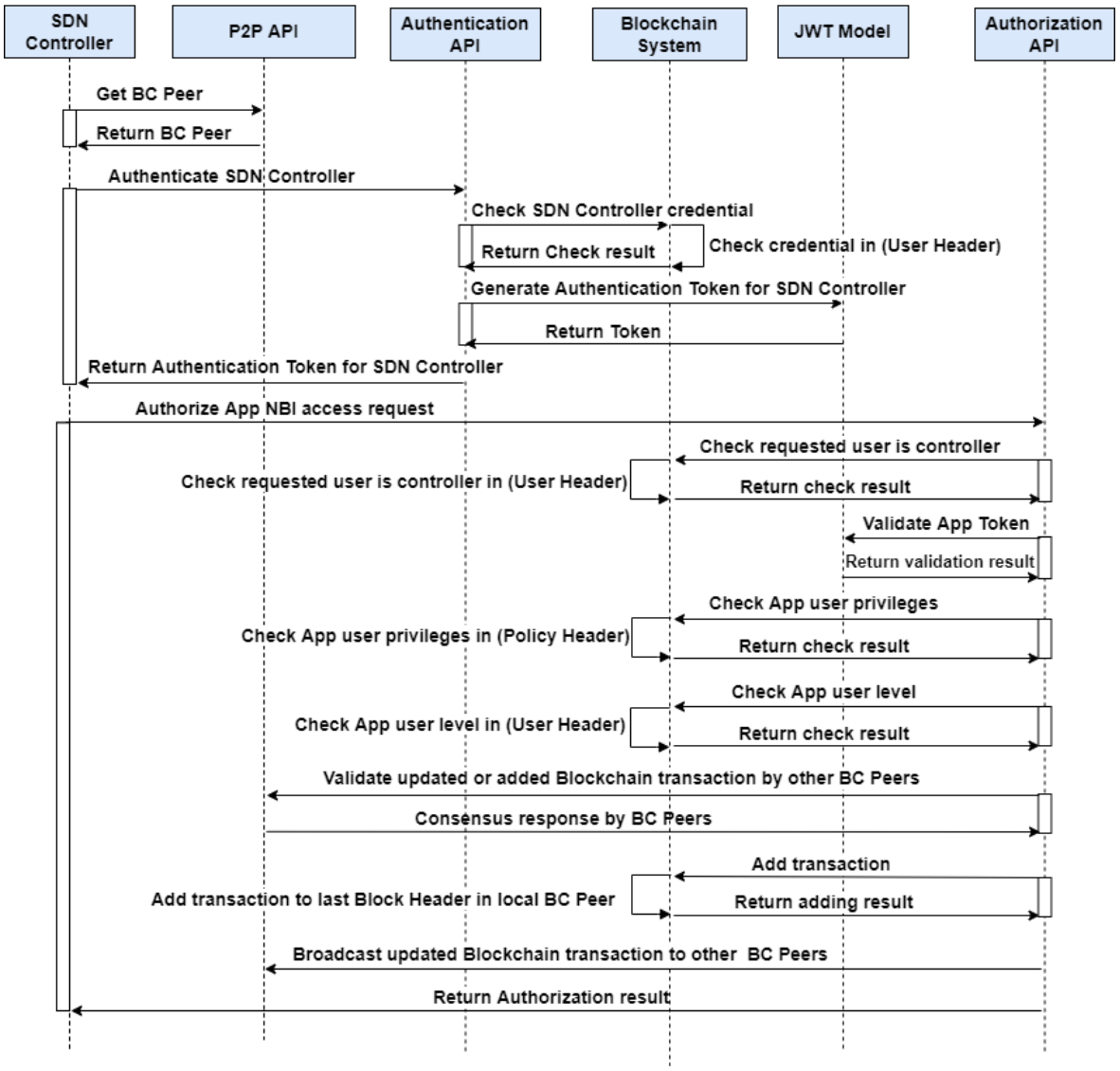

Figure 5. Authorization API workflow.

Algorithm 1 Add transaction to blockchain of BC peer.

```
Input:
    newTransaction: new transaction needs to be added to BC peer blockchain.
    localPeer: the BC peer that will add the newTransaction into its blockchain.
    p2pNetwork: the P2P network of all BC peers.
Steps:
Set tempBlockchain = localPeer.blockchain
Add newTransaction to tempBlockchain
Set isTempBlockchainAccepted = true
Set allPeers = p2pNetwork.getAllPeers()
for each currentPeer in allPeers do
    Set validationResponse = invoke(currentPeer.validateBlockchainApi(tempBlockchain))
    if valicationResponse == "blockchain invalid" then
        Set isTempBlockchainAccepted = false
    end if
end for
if isTempBlockchainAccepted = true then
    Set localPeer.blockchain = tempBlockchain
    for each currentPeer in allPeers do
        Invoke(currentPeer.setBlockchainApi(tempBlockchain))
    end for
    return "transaction accepted"
else
    return "transaction rejected"
end if
```

Admin API: This REST API is responsible for configuring the users and policies of the proposed solution. As we apply permissioned blockchains, it is important to have a central authority that limits the participants to known and authorized users. Private or permissioned blockchains satisfy the requirements of businesses and governments in protecting their privacy. The Admin API is responsible for adding, deleting, and modifying the latest block of user and policy header in the BC Peer. The only authorized user who can access this API is the admin user, created by default in the Genesis Block. As an example, the following explains the steps required for the Admin API to add a user:

- Ensure the BC Peer is joined in our blockchain network;
- The admin user needs to authenticate themself by Authentication API;
- If the admin user is successfully authenticated, they receive an authentication token that needs to be used with each request by the admin user;
- When the admin user sends an API request to add a user to the Admin API, it validates the authentication token of the admin user using the JWT module, and ensures the requested user type is admin through the latest block's user header of BC Peer;
- Validate the transaction based on the agreement of other BC Peers, using validate-BlockchainApi.

If all of the above conditions are fulfilled, the Admin API adds the user to the user header of the last block and adds the transaction to the local BC Peer. To ensure each BC Peer obtains the last updated blockchain, we update the DateTime parameter, which is updated with each update in the blockchain. Finally, the updated blockchain broadcasts to other BC Peers.

P2P API: This API is responsible for managing our P2P network. The P2P API comprises the entities listed below.

Set Blockchain API: This API enables the broadcasting of blockchain updates to other Peers. The Broadcast Blockchain To P2P function uses this API to update other BC Peers.

Get Blockchain API: This API enables the acquiring of the blockchain by other joined Peers in the network. This API is used by the Get updated blockchain from the P2P function to obtain the blockchains of other Peers, in order to determine the most updated blockchain to be stored in a newly joined BC Peer.

Broadcast Blockchain To P2P: This function is executed after any modification or update in blockchain data, such as after any POST requests by the Admin API and after successful or unsuccessful authorization requests from applications. Broadcast Blockchain To P2P is executed as follows:

- After successful validation by validateBlockchainApi;
- Get all joined BC Peers;
- Execute Set Blockchain API, which is a REST API request to update all BC Peers.

Therefore, this function plays a major role in updating other BC Peers when there is any update or change in the blockchain. This ensures identical blockchain storage in all joined BC Peers.

Get updated blockchain from P2P: This function, as the name indicates, obtains the most updated blockchain. Even though we believe that all BC peers have the same copy, we execute this function when a new BC Peer joins our network in order to ensure that we maintain the most updated blockchain replica. The function is executed as follows:

- Get all joined BC Peers from BC storage;
- Send Get Blockchain API to all BC Peers to obtain the most updated blockchain database;
- Compare the blockchain database of Peers in terms of number of blocks, number of transactions in the last block, and last blockchain modification based on the DateTime parameter.

As a result, we obtain the most updated blockchain and store it in the newly joined BC Peer.

ValidateBlockchainApi: This API validates the blockchain update by other BC Peers, as follows:

- Ensure the requested BC Peer is joined in our blockchain network;
- Get all joined BC Peers from the BC storage;
- Each BC Peer validates the updated blockchain using the isValid function, in order to ensure data integrity;
- Each BC Peer compares the number of blocks of the updated blockchain with the number of blocks of its BC;
- Each BC Peer compares the number of transactions in the last block of the updated blockchain with the number of transactions in the last block of its blockchain;
- Each BC Peer compares the last timestamp of the updated blockchain with the last timestamp of its BC.

If all BC Peers reach an agreement on the blockchain update, as demonstrated in Algorithm 2, then the blockchain update is considered to be a valid transaction.

IsValid: This function validates any modification or updates in the blockchain, in order to protect the integrity of data against any tampering attack, as demonstrated in Algorithm 3. In addition, this function is used by validateBlockchainApi to validate blockchain updates by other BC Peers.

As each block in the blockchain has its own hash value and the hash value of the previous block, isValid is executed as follows:

- Check if the received hash of each block is equal to its calculated hash of the block's received data;
- Check if the previous hash value of each block is equal to the actual hash of the previous block.

Algorithm 2 ValidateBlockchainApi.

Input:
peerBC: the blockchain of another peer needs to be validated by local peer.
localBC: the blockchain of local peer.
Steps:
if peerBC.isValid() **then**
if *peerBC.blocks.count* > *localBC.blocks.count* **then**
return "Blockchain Valid"
end if
if *peerBC.blocks.count* == *localBC.blocks.count* **then**
if *peerBC.latestBlock.trans* > *localBC.latestBlock.trans* **then**
return "Blockchain Valid"
end if
end if
if *peerBC.blocks.count* == *localBC.blocks.count* **then**
if *peerBC.lastUpdate* > *localBC.lastUpdate* **then**
return "Blockchain Valid"
end if
end if
end if return "Blockchain Invalid"

If both conditions are satisfied and all hashes match, we can ensure the data integrity of the blockchain.

Algorithm 3 IsValid.

Input:
blockchain: the blockchain which needs to be validated .
Steps:
Set index = 1
while *index* < *blockchain.blocks.count* **do**
Set currentBlock = blockchain.blocks[index]
Set prevBlock = blockchain.blocks[index − 1]
if currnetBlock.hash != currentBlock. calculateHash() **then return** false
end if
if currentBlock.prevHash != prevBlock.hash **then return** false
end if
index = index + 1
end while return true

3.3.2. BCNBI Blockchain Manager

The BCNBI Blockchain Manager represents the back end of our proposed solution, which is accountable for building and managing our blockchain-based architecture. The BCNBI Blockchain Manager comprises the entities listed below.

Blockchain Module: This is a core component in our framework, which is responsible for constructing and handling the blockchain structure. It includes a Block Header, which is responsible for building and chaining the entire blockchain. Each block in a BC Peer has transactions and three headers, namely, a block header, a user header, and a policy header, as shown in Figure 6. However, the last updated user and policy headers are placed in the last block of all BC Peers, which are used for assuring and checking API permissions and users. Each block contains the hash of its previous block to ensure immutability of the BC. The block header has all information related to the block, such as the block index, hash, previous hash, and timestamp. To ensure secure and immutable access control of our system, we store our access-control privileges in the policy header, which assigns each application to its allowed permissions. When the system creates the first block, it generates

a genesis block and an admin user to control and manage our blockchain in terms of adding, deleting, and modifying users or policies.

Block 1

Block Header	
Index	1
Hash	sJU96ntT654LkYqBCvwnMOT9ZlOzcSwkJzWaz/lEPWA=
Previous Hash	
Time Stamp	1/25/2022 9:02:43 AM

Users Header					
User ID	User Type	User System	Normal Policy Threshold	Strict Policy Threshold	User Level
bc_admin	Admin	Blockchain App	100	100	Benign

Policies Header	
Application User ID	Permission

Transactions										
App User	Controller Name	Controller User	API Uri	API Method	API Permission	Is Authorized	Policy Threshold	App User Level	Date	Creator

Figure 6. Block structure example.

User Module: The responsibility of this module is to store all information related to users or participants of our system, such as admin users, application users, and SDN controller users. Each participant has seven parameters: userID, userPassword, userType, userSystem, NormalPolicyThreshold, StrictPolicyThreshold, and UserLevel. The userID and userPassword are used by the Authentication API, while the userType, userSystem, NormalPolicyThreshold, StrictPolicyThreshold, and UserLevel are used by the Authorization API for the authorization process and for evaluating an application's behavior. Although each block in the blockchain contains a user header, the most updated user header is positioned in the last block's header, which is used for validating and modifying users.

Policy Module: This module stores the access control policy of applications and has two parameters: ApplicationUserID and API Permission. API permissions are assigned to each ApplicationUserID, based on application type and roles. This module is used by the Authorization API for comparing the requested API permission of the application and its predefined API permission in the policy header of the last block. If there is a match, it allows the request to continue to the authorization process; otherwise, it discards the request. Though each block in the blockchain contains a policy header, the most updated policy header is positioned in the last block's header, which is used for validating and modifying policies.

Transaction Module: Any communication between an application and the SDN controller is considered a transaction, which includes an ApplicationUserID, ControllerName ControllerUserID, ApiUri, ApiMethod, ApiPermission, IsAuthorized, PolicyThreshold, ApplicationUserLevel, Date, and Creator, as shown in Figure 7.

Transactions										
App User	Controller Name	Controller User	API Uri	API Method	API Permission	Is Authorized	Policy Threshold	App User Level	Date	Creator
app_user2	FloodLight	cont_user1	/wm/device/	GET	LIST_ALL_CONNECTED_DEVICES	True	Normal:100	Benign	1/26/2022 11:39:08 AM	BC Peer 8091
app_user2	FloodLight	cont_user1	/wm/core /controller /switches/json	GET	LIST_ALL_CONNECTED_SWITCHES	False	Normal:95	Benign	1/26/2022 11:39:56 AM	BC Peer 8091
app_user2	FloodLight	cont_user1	/wm/firewall /rules/json	POST	ADD_FIREWALL_RULE	True	Strict:100	Benign	1/26/2022 11:40:32 AM	BC Peer 8091
app_user2	FloodLight	cont_user1	/wm/firewall /module /enable/json	PUT	ENABLE_FIREWALL	False	Strict:95	Benign	1/26/2022 11:41:58 AM	BC Peer 8091
app_user2	FloodLight	cont_user1	/wm/firewall /module /enable/json	PUT	ENABLE_FIREWALL	False	Strict:90	Benign	1/26/2022 11:42:55 AM	BC Peer 8091

Figure 7. Transaction structure.

Hash Module: A vital component of blockchain technology is to apply the hash function to the content of each block, including the previous block's hash of the current block, which is used to ensure that a tamper-proof and immutable database is maintained. We use the lightweight hash algorithm SHA256 to calculate the hash value for each block. This module produces the hash value for each block based on BlockIndex, BlockTimeStamp, PreviousHash, PoliciesHeader, UsersHeader, and Transactions. The hash module is invoked by the isValid function after any update by the Admin API or Authorization API.

Trust evaluation Module: This module monitors the behavior of applications when communicating with the northbound interface of the SDN controller, as illustrated in Algorithm 4. The interactions between an application and SDN controller can be classified into reading network status and writing network permissions [13].

Algorithm 4 Application trust evaluation.

Input:
transaction: the transaction that represents the request from application.
to controller with its authorization status.
appUser: the user of SDN application.
Steps:
if transaction.appRequestMethod == "GET" **then**
  if transaction.isAppAuthorized == false **then**
    Decrease appUser.normalPolicyThreshold by 5
    if *appUser.normalPolicyThreshold* < 50 **then**
      Set appUser.level = "Malicious"
    end if
  end if
else
  if transaction.isAppAuthorized == false **then**
    Decrease appUser.strictPolicyThreshold by 5
    if appUser. strictPolicyThreshold == 85 **then**
      Set appUser.level = "Malicious"
    end if
  end if
end if

Read permission: Read permissions allow applications to send HTTP requests with the GET method to the SDN controller. This permission enables the application to obtain information about the state of controllers, switches, devices, topology, routing, and statistics of events. Therefore, Read permissions allow applications to gather the network's statistical information and learn about the structure of the SDN. Should a malicious application have this permission, it could discover vulnerabilities of the system to be exploited at a later time or to launch a reconnaissance attack. However, Read permissions can only be used to read information about the network without modifying it, making them less risky than Write permissions.

Write permission: Write permissions allow applications to send HTTP requests including POST, PUT, and DELETE to the SDN controller. These permissions plays a significant role in setting, modifying, and deleting the configuration of the SDN; for example, an application with this permission can perform very dangerous operations, such as adding or deleting static flow on flow entries, creating or dropping firewall rules, modifying the firewall status, and creating or deleting virtual networks. This permission enables an attacker to generate multiple attacks, such as flow rule tampering, redirection of network traffic, and controller spoofing and, finally, to compromise the entire SDN. Therefore, we classify the policy in our system as a strict policy or a normal policy based on the type of the requested HTTP method.

Strict policy: This policy is associated with Write permissions, and aims to protect our system from unauthorized modification which could result in direct damage to the SDN

controller. In this policy, the application is blocked and its trust level is considered to be malicious if the number of unauthorized HTTP requests exceeds three times. This threshold was set based on common security practices in most strict security systems, in order to defend our SDN from being compromised or attacked.

Normal policy: This policy is linked with Read permissions, and aims to protect our system from unauthorized reads which could result in understanding and exploiting system weaknesses. In this policy, the application is blocked and its trust level is considered to be malicious if the number of unauthorized HTTP requests exceeds eleven times. Set according to [7], the threshold can be tolerated to some extent, as some over-privileged requests caused by mistaken and illegitimate Read permissions cannot amend the state of the network, compared to Write permissions.

Trust attribute: This attribute is stored in the latest block's user header of a BC Peer, and is used to measure the application's trustworthiness based on its behavior. Each application is assigned with a Trust attribute, which is periodically evaluated and updated based on the application's operations. The Trust attribute can be represented by a trust level, such as benign or malicious, or by a trust value (0–100). At the beginning, by default, the trust level is set to normal and the trust value is set to 100. The trust value drops by five points per unauthorized request, while remaining at the same value for permitted requests. There are two types of thresholds, based on the predefined policies, as shown in Figure 6. The strict policy uses a strict threshold, blocking the application when its trust value reaches 85 and turning its trust level to malicious. Regarding the normal policy, an application is deactivated once its trust value is less than 50, and its trust level becomes malicious. The Trust attribute is assigned with each application request and checked by the Authentication API and the Authorization API, in order to establish a trust relationship between the application and the SDN controller.

JWT module: This module is responsible for the generation and validation of authentication tokens. The GenerateToken function encrypts the user's identity based on the HMAC (Hash-Based Message Authentication Code), which uses a secret cryptographic key in conjunction with a hash function to protect the integrity of the data and adds an extra layer of user authentication. The ValidateToken function checks the validity of received authentication tokens. In addition, it validates the token expiry duration, as each token has an expiration date (called expireMinutes). The Authentication API uses this module to generate tokens, while the Authorization and Admin APIs use this module to validate tokens.

3.3.3. Controller API Handler

The Controller API Handler is used as an SDN controller plugin to facilitate secure interactions with our proposed system, and to intercept and analyze the application requests for validation purposes. The Controller API Handler encompasses the entities detailed below.

Controller Authentication API: To achieve mutual authentication, we optimized the security of the Floodlight controller by building this API to validate the identity of the Floodlight controller when communicating with our framework using a username, password, and authentication token. This API interacts with one of the BC Peers, based on the round-robin algorithm, and sends an API request to the Authentication API to authenticate the credentials of the Floodlight controller. After successful authentication, the Authentication API of the BC Peer generates an authentication token for the Floodlight controller, which needs to be used with each request.

Permission-Detecting API: This API facilitates communications among the application, Floodlight controller, and our proposed solution. Permission Detecting can be divided into two main elements: Check API permission and Authorize application request.

Check API permission: This API maintains policies and permissions for Floodlight REST APIs and their associated HTTP methods. The Floodlight controller offers several REST APIs, such as Controller APIs, Topology and Routing APIs, Device APIs, and Fire-

wall APIs. These APIs could have URIs with various HTTP methods, such as GET, POST, DELETE, and PUT; for instance, Firewall APIs, which are used for listing, creating, and deleting firewall rules, have one URI */wm/firewall/rules/json* with various HTTP methods. To illustrate that the following URI */wm/firewall/rules/json* with the GET method is used for listing firewall rules, the same URI with the POST method is used to create a new firewall rule. Therefore, this module compares the requested API permission of the application with the predefined permissions of Floodlight APIs. If there is a match, it forwards the result to the Authorize Application Request API for further analysis; otherwise, it discards the request.

Authorize Application Request API: This API is an SDN controller plugin, used to authorize and verify the API requests of applications based on our framework. This API communicates with one of the BC Peers and sends an authorization request to the Authorization API, taking into account that the request should include certain parameters; namely, a controller token, controller name, application token, API permission, API URI, and API HTTP method. The Authorization API validates the identity of the application and the SDN controller, ensures the authorization request is initiated by the SDN controller, checks the application's permissions based on the policy header in the BC Peer, and evaluates the application's trust level. Based on the above mentioned conditions, the Authorization API responds to the SDN controller to approve or deny application requests.

4. BCNBI Evaluation

The proposed solution, BCNBI, was evaluated based on a qualitative security analysis and its quantitative performance overhead. The security analysis demonstrated that our architecture can achieve the fundamental security objectives of confidentiality, integrity, and availability, and examines how primary security threats were mitigated. In addition, we evaluated the performance overhead of the proposed solution.

4.1. Experimental Setup

We implemented a proof-of-concept prototype of our BCNBI solution in the ASP.NET framework as a permissioned blockchain, in order to achieve the security benefits of blockchain technology while maintaining privacy and control requirements. Unlike Proof-of-Work and other highly computationally intensive consensus processes, we implemented a lightweight blockchain framework similar to our previous work in [36] and similar to that of [37]. In our previous work, we demonstrated a lightweight instantiation of blockchain for IoT systems by mitigating the heavy computational cost of a consensus algorithm.

We executed the BCNBI solution on a Windows 10 operating system with 12 GB RAM and an Intel Core i7-4510U CPU at 2.60 GHz. In addition, the Mininet simulator was used to emulate the SDN on Ubuntu 14.04 LTS in a VMware Workstation with an Intel Core i7-4510U CPU at 2.60 GHz and 8 GB memory. The Floodlight controller was used to control and manage the SDN. The SDN topology comprised one Floodlight controller andthree Open vSwitch (OVS) switches with one connected host per switch in a linear-type model. We simulated the Blockchain network in our solution with three connected BC peers. In addition, we simulated the SDN application in our framework using JMeter [38] to test the northbound interface through consuming the Floodlight REST APIs.

4.2. Security Analysis

We evaluated our solution based on the confidentiality–integrity–availability (CIA) triad, which is a well-known information security model. We chose the CIA triad to evaluate our system, as it is commonly used as the key objective in computer security design. Each security objective guards the system against different types of threats. For example, confidentiality ensures that information is disclosed and accessed only by authorized participants, while integrity ensures that information is accurate and cannot be modified; finally, availability ensures that the information is available to trusted users. Next, we analyzed the effectiveness of our solution to prevent threats related to the CIA triad.

4.2.1. Confidentiality Threats

The aim of protecting confidentiality is to protect information against unauthorized access or disclosure. This protection includes protecting data while being transmitted from an application to a SDN controller and data that are stored in BC Peers. Our system applies encryption, authentication, and authorization to achieve confidentiality. In addition, we utilize a private blockchain to protect privacy, as the flow of data is controlled and the consensus process is limited to predefined participants.

Encryption technique: This method aims to make stored and transmitted data unreadable to illegal users. While the data in the storage phase are stored in the BC Peers, which encrypt the stored data by hashing, the data cannot be interpreted in the transfer phase, as we use Hyper Text Transfer Protocol Secure (HTTPS) over Transport Layer Security (TLS) to encrypt data during transmission and enable a secure channel between the application and the SDN controller. This technique guards our system against eavesdropping and sniffing threats, such as man-in-the-middle attacks.

Authentication technique: This technique verifies the application user's identity to prevent unauthorized users from accessing the SDN controller. As the first line of defense to protect the SDN controller from malicious applications, the application needs to be authenticated by our system before it can communicate with the SDN controller. The authentication process involves three steps that take place between the application and the Authentication API in the BC Peer. The first step involves validating whether the requested application's credentials match the actual application's predefined credentials that are stored in our BC Peer. If there is a match, then it successfully passes the first line of defense. Then, if the application trust level is not malicious, the BCNBI generates an authentication session token that must be used by the application with each request within the session period. Finally, the authentication technique maintains the application trust level, which can be updated and altered based on the activities of the application.

Example 1. *Let us assume that an adversary bypasses the HTTPS protocol and manages to acquire the credentials of a legal application user.*

The attacker would be unable to authenticate itself, as it does not have an authentication session token that is generated and validated by our BC Peer. The worst-case scenario consists of the attacker capturing the authentication session token. Then, in this case, there are two situations: First, if the token is expired or modified the PC Peer would detect this as a malicious request. On the other hand, if the intruder acquires a valid token then this countermeasure would be passed; however, our system would check the application's trust level and update it based on the application's behavior. Accordingly, if the application exceeded the trust-value threshold it would result in blocking of the application and it would be considered as a malicious application. Therefore, this technique validates the identity of the application before communicating with the SDN controller and mitigates the threat of spoofing based on multiple security levels.

Authorization technique: The next step, after successfully identifying and verifying the identity of the application, involves permitting the application to access the resources of the SDN controller APIs based on particular conditions. The SDN controller is the only entity that can communicate with the Authorization API in the BC Peer. The Authorization API is implemented in each BC Peer, which is responsible for validating the application request and granting permission to the application to access APIs of the SDN controller based on several verification steps. These verification steps include authenticating the application and the SDN controller, checking the application operation based on the predefined access policies in the BC Peer, monitoring the application's trustworthiness based on a predefined threshold and, finally, validating the transaction by other BC Peers.

Example 2. *Let us assume that one of the authorized applications attempts to elevate its allowed privileges and access unauthorized REST APIs of the Floodlight controller.*

These requests would be discarded by the Authorization API in the BC Peers, and the application's trust value would automatically drop by five points per illegal request as a punishment. Our system classifies the threshold into strict and normal, based on the type of HTTP method. On one hand, the strict threshold is activated when the application requests three unauthorized controller API resources with a Write permission, such as POST, PUT, and DELETE. Then, the trust value of this application is declined to 85 as a penalty for this behavior, and the application's trust level is set to malicious. On the other hand, the normal threshold is activated when the application sends eleven over-privileged HTTP requests with a Read permission, such as GET. Then, the trust value of this application is declined to 45 and the application's trust level is set to malicious as a penalty for this behavior, which disables the application from sending further requests. Thus, our design has hierarchical security layers, mitigates attacks based on the severity of the launched attack, and secures the APIs of the SDN controller against the threat of the elevation of privileges.

4.2.2. Integrity Threats

Integrity involves protecting data from being modified or tampered with by an illegitimate party. Data integrity includes data that are stored on BC Peers and data that are transmitted between the application and SDN controller. Centralized approaches store the data and policy in a normal database, which can be exploited and modified by hackers. Therefore, we utilize blockchain technology to ensure the integrity and tamper resistance of the BCNBI database. We enhance the security of the access control information by storing them in the header of each BC Peer, as shown in Figure 6; however, the most updated access control information is stored in the header of the last BC Peer. The proposed solution maintains the immutability of data stored in the BC Peers by utilizing the blockchain feature, as each block contains a hash value of its content and a hash value of the previous block, which forms secure interconnections between the blocks. Our system uses REST APIs, which operate based on the HTTPS protocol, which encrypts the communication channel between application and SDN controller, in order to guard the data integrity during transmission.

Example 3. *Let us assume that an adversary successfully attacks one of the BC Peers and updates the trust level of an application from malicious to normal.*

This attack can be detected by other BC Peers as any modification or update in one of the BC Peer needs to be validated by other blockchain Peers before confirmation. In addition, the proposed solution inherits the properties of blockchain which runs on a P2P network, and each BC peer has a duplicate copy of the blockchain data. Therefore, even if the intruder succeeds in hacking one BC Peer, other BC Peers will detect this modification. This attack can be detected by validateBlockchainApi, as other BC Peers calculate each block's hash and the link to the previous block's hash to detect changes between the received block's hash of the tampered block and the calculated block's hash. In addition, each BC Peer compares the number of blocks of the updated blockchain with the number of blocks of its BC. Moreover, each BC Peer compares the number of transactions in the last block of the updated blockchain with the number of transactions in the last block of its blockchain. Finally, each BC Peer compares the last timestamp of the updated blockchain with the last timestamp of its BC. Thus, the validation not only examines the hash value of the block and the link to the previous block's hash, but also examines the number of blocks, the number of transactions in the last block, and the last timestamp of the updated blockchain. Furthermore, our solution enables the recording and tracking of all activities performed by a specific user, which can be used for auditing and ensuring the accountability of operations. Accordingly, our design protects the data stored in the BC Peers and data on the transmissions from tampering threats.

4.2.3. Availability

The goal of availability is to ensure that the system is available and responsible for a valid request. This security goal assures that the system is reliable and available for authorized users in a timely manner. Our solution can mitigate the availability threats to the NBI of the SDN controller by applying various techniques, namely redundancy and

fault tolerance. We employ blockchain technology, which enables redundancy and fault tolerance as it runs on a P2P network. Leveraging the decentralized nature of blockchain networks, which include numerous peers, each joined peer has a duplicate copy of the blockchain ledger, which improves the network resilience.

Example 4. *Let us consider the case in which one of the BC Peers has been attacked or overwhelmed. Our system would continue executing, as other BC Peers would handle further application requests. Moreover, to enhance the network performance and minimize response time, it is important to equally distribute an application request across a group of BC Peers which have identical storage and computing resources. Hence, we apply round-robin load balancing, which forwards the application request to each BC Peer on a cyclical basis. Thus, our design can overcome the threat of a single point of failure, which is one of the most common attacks against a centralized SDN controller.*

4.3. Performance Evaluation

The introduced performance overhead of the proposed architecture was evaluated by measuring the time overhead of Write and Read transactions between applications and the SDN controller. We evaluated the performance overhead introduced by our solution by comparing the time and packet overheads with and without the use of our blockchain-based method.

In the first scenario, we employed our blockchain-based solution when an application invoked the REST API of the Floodlight controller [39].

In the second scenario, an application invoked the REST API of the Floodlight controller without a blockchain-based solution. While we refer to the former scenario as a block-based method, we refer to the latter scenario as a baseline method. On one hand, the block-based method has several security features securing the application–controller interface or northbound interface, such as encryption, authentication, authorization, hashing, P2P architecture, logging, and application trust level. On the other hand, the baseline method does not have any security features to protect the REST API of the Floodlight controller from malicious applications; for example, in the baseline method, the application can send HTTP requests in plaintext and access the REST APIs of the Floodlight controller without authentication, authorization, or behavior inspection. This means that the application can modify the network state and perform malicious operations without any defensive mechanisms to protect the REST API of the Floodlight controller. Consequently, this could lead to a severely negative impact on the SDN.

We simulated the scenarios using the JMeter tool as an application that sends requests to the Floodlight controller, in order to measure the time and packet overheads introduced with and without our proposed solution. The JMeter tool was used as an application, which sent an HTTP GET request to one of the Floodlight REST APIs; namely, */wm/device/* (see Figure 8). This API is an example of a Read transaction that retrieves all devices tracked by the controller. For Write transactions, the application would send an HTTP POST request to */wm/firewall/rules/json*, which is one of the Floodlight REST APIs that is responsible for creating new firewall rules. The Read and Write transactions were repeated 10 times, and the averaged overheads are presented in Figures 9 and 10, as well as Table 1. Our evaluation was based on the following metrics.

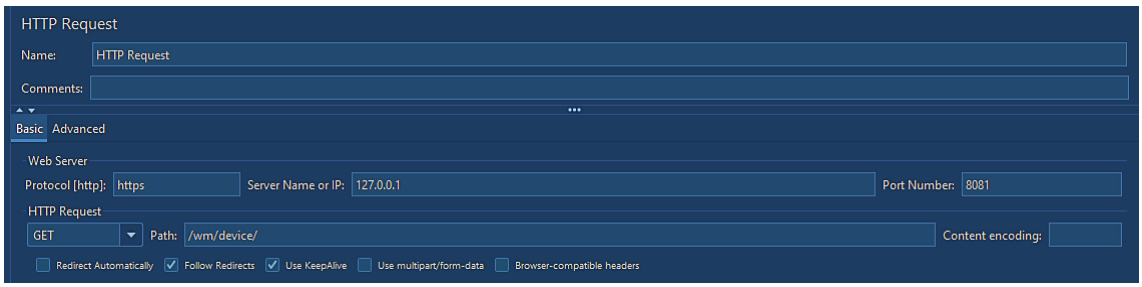

Figure 8. JMeter request to the Floodlight REST API.

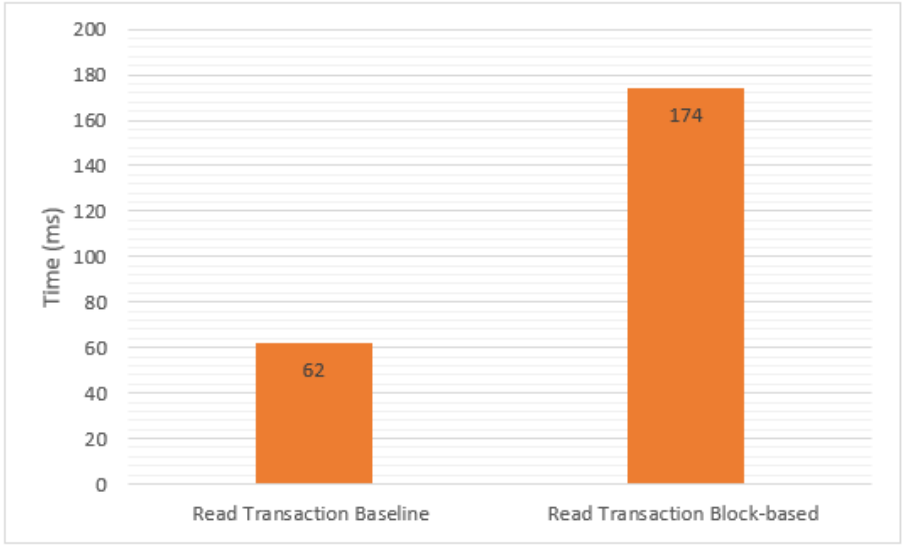

Figure 9. Evaluation of Read time overhead.

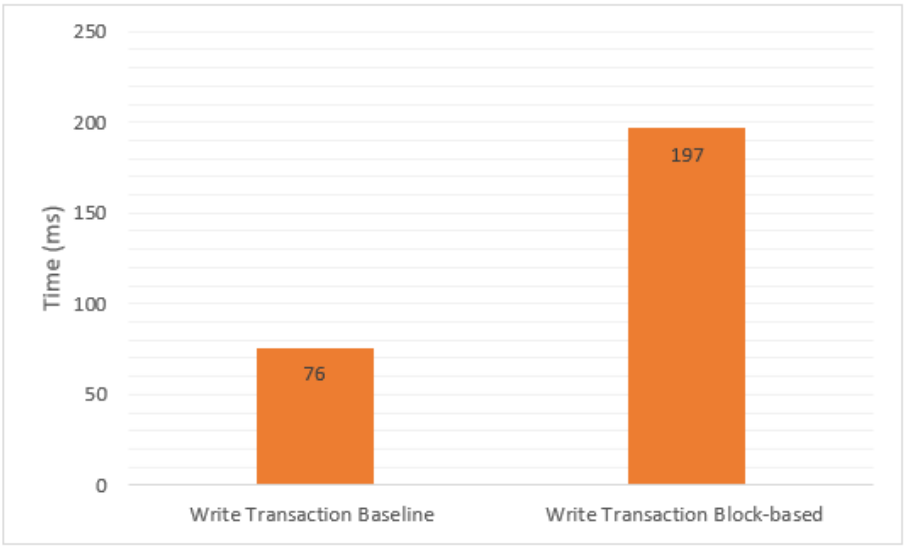

Figure 10. Evaluation of Write time overhead.

Table 1. Packet overhead evaluation (in bytes).

Packet Route	Block-Based Solution	Baseline
Read transaction	338	155
Write transaction	417	212

4.3.1. Performance Analysis Based on Time Overhead

- **Read transaction:** This metric refers to the time consumption from when the application requests to read the resources of the Floodlight controller until the response is sent to the application. The Read transaction involves the time when the application retrieves the data regarding the state of the Floodlight controller, the duration of the handling of the request, and the time taken to respond to the application. Figure 9 shows the simulation results for the time overhead of the Read transaction. The block-based method took a longer time to process packets than the baseline method due to additional security operations, such as hashing, enforcing access control and encryption. However, the average increase in overhead was approximately 0.174 s, which is not significant.

- **Write transaction:** This metric refers to the time consumption from when the application requests to modify the resources of the Floodlight controller until the response is sent to the application. The processing time for Write transactions involves the time taken for the application to send a request to the SDN controller, that for the processing of the request, and that for responding to the application. The most time-consuming part is the time required to handle the request by the Floodlight controller, which communicates with the BCNBI framework to verify the application's identity, permissions, and trust before giving the response to the application. The process of the Write transaction is more time consuming than that of the Read transaction, as Read transactions simply retrieve data and do not change the network configuration. Figure 10 illustrates the results for the time consumption of the Write transaction. It is obvious, from the figure, that applying blockchain technology and security mechanisms increased the packet processing time in our solution compared to the baseline method. Nevertheless, the block-based method increased the overhead by only about 0.121 s, imposing a relatively small effect on the Floodlight controller.

4.3.2. Performance Analysis Based on Packet Overhead

This metric refers to the amount of additional packets introduced by the proposed solution. Table 1 demonstrates the results for packet overhead in bytes. As stated above, our blockchain-based solution applies encryption and hashing, which increased the size of the packets; particularly, it uses the JSON Web Token, which needs to be embedded in the HTTP Header with each request. However, the additional packet overhead which was generated by the Write transaction was less than 0.25 kilobytes, which can be considered negligible. In summary, the security and protection advantages of our solution significantly outweigh the relatively low overheads introduced.

4.3.3. Comparative Analysis

In this section we critically analyze the related work and classify the previous studies into centralized and blockchain-based solutions. Centralized-based solutions lack adequate security properties, specifically those protecting the northbound interface against a single point of failure, as well as ensuring data immutability and controller-independent platforms. Most of the current blockchain-based solutions are also lacking, in terms of minimizing the high computational overhead associated with blockchains, monitoring the SDN application's behavior, and supporting automated security solutions. The public blockchain-based solution uses Proof of Work (POW) or Proof of Stake (POS) consensus algorithms which demand high computational resources to solve a cryptography puzzle. For instance, the transaction confirmation in Bitcoin is about 10 min whereas it is about 15 s in Ethereum [40,41]. In Table 2, we compare the limitations of existing approaches in securing the northbound interface of the SDN controller with our proposed solution; the comparison is performed in terms of AAA mechanisms, Decentralization, App Trust Evaluation, Immutability, Security Automation, and Lightweight.

In addition, we demonstrate the efficiency of the proposed solution by comparing the overheads of the BCNBI framework with another private blockchain-based solution that aims to secure the northbound interface of the SDN controller with the same environmental simulation. The proposed BCNBI framework is compared to BlockAS [31] which uses blockchain to secure the northbound interface of the SDN controller. They implemented a CFT (crash-fault-tolerant) consensus algorithm in their proposed BlockAS system. The SDN topology used in the simulation comprised one Floodlight controller, three switches, and three connected hosts per switch in a linear-type model. As shown in Figure 11, the proposed BCNBI framework achieves better performance while the BlockAS system took a longer time handling the Read and Write transaction compared to our proposed solution. Thus, to fill the above gaps, we propose a novel lightweight blockchain-based architecture, called BCNBI, to secure the northbound interface of the SDN controller.

Table 2. Comparison of existing approaches for securing the northbound interface of the SDN controller. ✓, fully achieved; ✳, partially achieved; ✗, not achieved.

References	AAA	Decentralization	App Trust Evaluation	Immutability	Security Automation	Lightweight
[12]	✓	✗	✓	✗	✗	✓
[19]	✓	✗	✓	✗	✗	✓
[7]	✓	✗	✓	✗	✗	✓
[20]	✓	✗	✗	✗	✗	✓
[21]	✓	✗	✓	✗	✗	✓
[22]	✳	✗	✗	✗	✗	✓
[23]	✓	✗	✗	✗	✓	✓
[4]	✓	✗	✗	✗	✓	✓
[24]	✳	✗	✗	✗	✗	✓
[6]	✳	✗	✗	✗	✓	✓
[25]	✳	✗	✗	✗	✗	✓
[26]	✳	✗	✓	✗	✓	✓
[30]	✳	✓	✗	✓	✗	✳
[31]	✓	✓	✗	✓	✗	✳
[14]	✓	✓	✓	✓	✗	✗
Proposed framework	✓	✓	✓	✓	✓	✓

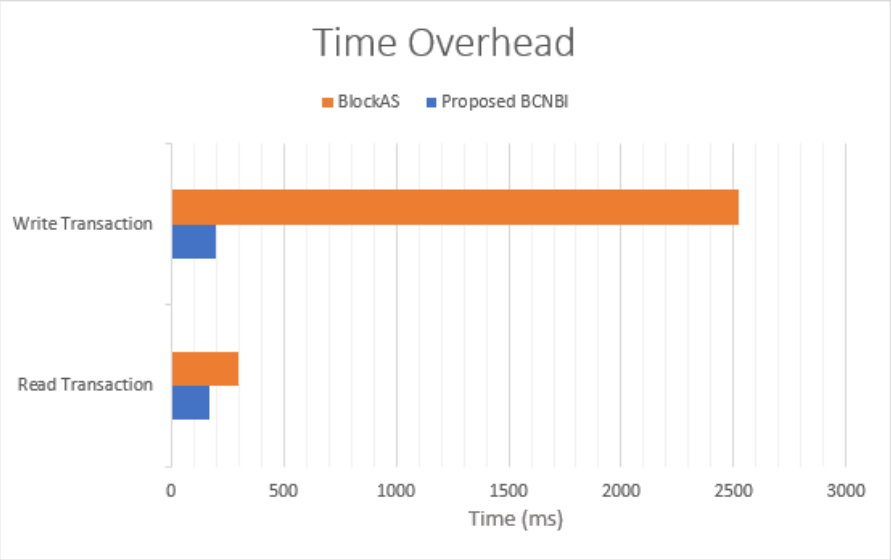

Figure 11. Performance evaluation comparison between proposed framework and BlockAS [31].

5. Conclusions and Future Work

In this paper, we proposed a comprehensive decentralized security framework based on a lightweight blockchain-based architecture for securing the SDN application–controller interface. We proposed a novel security solution, named BCNBI, which addresses the security threats of the northbound interface based on the well-known security goals of confidentiality, integrity, and availability. Our framework aims to utilize blockchain technology to provide effective and automated security features while eliminating the associated high computational overhead and delay. The framework is controller independent and authenticates applications and the SDN controller through token-based authentication, which is issued and verified by the BCNBI. The system enforces the access control policy for each application REST API request to the SDN controller. The application behavior is monitored based on an application trust model, which enables or disables applications based on their operations. The proposed solution was tested on the Floodlight controller, and the results illustrated that the framework provides effective protection against CIA

triad threats with negligible overhead. For future work, we intend to extend the solution to a distributed controller environment and focus on addressing security and data consistency issues [42,43]. In addition, we plan to enhance the precision of our system by applying machine learning methods.

Author Contributions: Conceptualization, S.A., F.E. and K.A.; funding acquisition, A.A. (Abdullah Algarni) and A.A. (Aiiad Albeshri); investigation, S.A. and F.E.; methodology, S.A. and F.E.; project administration, F.E.; supervision, F.E.; validation, S.A., F.E. and K.A.; visualization, S.A., F.E., A.A. (Abdullah Algarni) and A.A. (Aiiad Albeshri); writing—original draft, S.A.; writing—review and editing, S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This study was funded by King Abdulaziz University.

Acknowledgments: This project was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, under grant no. (RG-21-611-38). The authors, therefore, acknowledge with thanks the DSR technical and financial support.

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

References

1. Kreutz, D.; Ramos, F.M.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proc. IEEE* **2014**, *103*, 14–76. [CrossRef]
2. Jimenez, M.B.; Fernandez, D.; Rivadeneira, J.E.; Bellido, L.; Cardenas, A. A Survey of the Main Security Issues and Solutions for the SDN Architecture. *IEEE Access* **2021**, *9*, 122016–122038. [CrossRef]
3. Latif, Z.; Sharif, K.; Li, F.; Karim, M.M.; Biswas, S.; Wang, Y. A comprehensive survey of interface protocols for software defined networks. *J. Netw. Comput. Appl.* **2020**, *156*, 102563. [CrossRef]
4. Tseng, Y.; Zhang, Z.; Naït-Abdesselam, F. ControllerSEPA: A security-enhancing SDN controller plug-in for OpenFlow applications. In Proceedings of the 2016 17th IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Guangzhou, China, 16–18 December 2016; pp. 268–273.
5. Rauf, B.; Abbas, H.; Usman, M.; Zia, T.A.; Iqbal, W.; Abbas, Y.; Afzal, H. Application Threats to Exploit Northbound Interface Vulnerabilities in Software Defined Networks. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–36. [CrossRef]
6. Oktian, Y.E.; Lee, S.; Lee, H.; Lam, J. Secure your northbound SDN API. In Proceedings of the 2015 IEEE 7th International Conference on Ubiquitous and Future Networks, Nanchang, China, 13–14 June 2015; pp. 919–920.
7. Au, N.N.H.; Pham, V.H. Toward a Trust-Based Authentication Framework of Northbound Interface in Software Defined Networking. In *Industrial Networks and Intelligent Systems, Proceedings of the 5th EAI International Conference, INISCOM 2019, Ho Chi Minh City, Vietnam, 19 August 2019*; Springer: New York, NY, USA, 2019; Volume 293, p. 269.
8. Ahmad, I.; Namal, S.; Ylianttila, M.; Gurtov, A. Security in software defined networks: A survey. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2317–2346. [CrossRef]
9. Kreutz, D.; Ramos, F.M.; Verissimo, P. Towards secure and dependable software-defined networks. In Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013; pp. 55–60.
10. Scott-Hayward, S.; O'Callaghan, G.; Sezer, S. SDN security: A survey. In Proceedings of the 2013 IEEE SDN For Future Networks and Services (SDN4FNS), Trento, Italy, 11–13 November 2013; pp. 1–7.
11. Alhaj, A.N.; Dutta, N. Analysis of Security Attacks in SDN Network: A Comprehensive Survey. In *Contemporary Issues in Communication, Cloud and Big Data Analytics*; Springer: New York, NY, USA, 2022; pp. 27–37.
12. Banse, C.; Rangarajan, S. A secure northbound interface for sdn applications. In Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA, Washington, DC, USA, 20–22 August 2015; Volume 1, pp. 834–839.
13. Scott-Hayward, S.; Kane, C.; Sezer, S. Operationcheckpoint: Sdn application control. In Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols, Raleigh, NC, USA, 21–24 October 2014; pp. 618–623.
14. Barka, E.; Dahmane, S.; Kerrache, C.A.; Khayat, M.; Sallabi, F. STHM: A secured and trusted healthcare monitoring architecture using SDN and Blockchain. *Electronics* **2021**, *10*, 1787. [CrossRef]
15. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. *Decentralized Bus. Rev.* 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 27 February 2022).
16. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. In Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress), Boston, MA, USA, 11–14 December 2017; pp. 557–564.
17. Guru, D.; Perumal, S.; Varadarajan, V. Approaches towards Blockchain Innovation: A Survey and Future Directions. *Electronics* **2021**, *10*, 1219. [CrossRef]
18. Swami, R.; Dave, M.; Ranga, V. Software-defined networking-based DDoS defense mechanisms. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–36. [CrossRef]

19. Toshniwal, B.; Joshi, K.D.; Shrivastava, P.; Kataoka, K. BEAM: Behavior-based access control mechanism for SDN applications. In Proceedings of the 2019 28th IEEE International Conference on Computer Communication and Networks (ICCCN), Valencia, Spain, 29 July–1 August 2019; pp. 1–2.
20. Cui, H.; Chen, Z.; Yu, L.; Xie, K.; Xia, Z. Authentication mechanism for network applications in SDN environments. In Proceedings of the 2017 20th IEEE International Symposium on Wireless Personal Multimedia Communications (WPMC), Bali, Indonesia, 17–20 December 2017; pp. 1–5.
21. Aliyu, A.L.; Bull, P.; Abdallah, A. A trust management framework for network applications within an SDN environment. In Proceedings of the 2017 31st IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA), Taipei, Taiwan, 27–29 March 2017; pp. 93–98.
22. Wen, X.; Chen, Y.; Hu, C.; Shi, C.; Wang, Y. Towards a secure controller platform for openflow applications. In Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013; pp. 171–172.
23. Tseng, Y.; Pattaranantakul, M.; He, R.; Zhang, Z.; Naït-Abdesselam, F. Controller DAC: Securing SDN controller with dynamic access control. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
24. Tseng, Y.; Nait-Abdesselam, F.; Khokhar, A. SENAD: Securing Network Application Deployment in Software Defined Networks. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.
25. Al-Alaj, A.; Krishnan, R.; Sandhu, R. Sdn-rbac: An access control model for sdn controller applications. In Proceedings of the 2019 4th IEEE International Conference on Computing, Communications and Security (ICCCS), Rome, Italy, 10–12 October 2019; pp. 1–8.
26. Aliyu, A.L.; Aneiba, A.; Patwary, M. Secure Communication between Network Applications and Controller in Software Defined Network. In Proceedings of the 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 26–28 September 2019; pp. 1–8.
27. Alharbi, T. Deployment of blockchain technology in software defined networks: A survey. *IEEE Access* **2020**, *8*, 9146–9156. [CrossRef]
28. Li, W.; Meng, W.; Liu, Z.; Au, M.H. Towards blockchain-based software-defined networking: Security challenges and solutions. *IEICE Trans. Inf. Syst.* **2020**, *103*, 196–203. [CrossRef]
29. Nguyen, H.N.; Tran, H.A.; Fowler, S.; Souihi, S. A survey of Blockchain technologies applied to software-defined networking: Research challenges and solutions. *IET Wirel. Sens. Syst.* **2021**, *11*, 233–247. [CrossRef]
30. Steichen, M.; Hommes, S.; State, R. ChainGuard—A firewall for blockchain applications using SDN with OpenFlow. In Proceedings of the 2017 IEEE Principles, Systems and Applications of IP Telecommunications (IPTComm), Chicago, IL, USA, 25–28 September 2017; pp. 1–8.
31. Hoang, H.D.; Duy, P.T.; Pham, V.H. A Security-Enhanced Monitoring System for Northbound Interface in SDN using Blockchain. In Proceedings of the 10th International Symposium on Information and Communication Technology, Hanoi, Vietnam, 4–6 December 2019; pp. 197–204.
32. Mendiboure, L.; Chalouf, M.A.; Krief, F. Towards a blockchain-based SD-IoV for applications authentication and trust management. In Proceedings of the International Conference on Internet of Vehicles, Paris, France, 20–22 November 2018; Springer: New York, NY, USA, 2018; pp. 265–277.
33. Jiang, S.; Cao, J.; Wu, H.; Yang, Y.; Ma, M.; He, J. Blochie: A blockchain-based platform for healthcare information exchange. In Proceedings of the 2018 IEEE International Conference on Smart Computing (Smartcomp), Sicily, Italy, 18–20 June 2018; pp. 49–56.
34. Jiang, S.; Cao, J.; McCann, J.A.; Yang, Y.; Liu, Y.; Wang, X.; Deng, Y. Privacy-preserving and efficient multi-keyword search over encrypted data on blockchain. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 405–410.
35. Jiang, S.; Cao, J.; Wu, H.; Yang, Y. Fairness-based packing of industrial IoT data in permissioned blockchains. *IEEE Trans. Ind. Inform.* **2020**, *17*, 7639–7649. [CrossRef]
36. Algarni, S.; Eassa, F.; Almarhabi, K.; Almalaise, A.; Albassam, E.; Alsubhi, K.; Yamin, M. Blockchain-based secured access control in an IoT system. *Appl. Sci.* **2021**, *11*, 1772. [CrossRef]
37. Dorri, A.; Kanhere, S.S.; Jurdak, R.; Gauravaram, P. Blockchain for IoT security and privacy: The case study of a smart home. In Proceedings of the 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kona, HI, USA, 13–17 March 2017; pp. 618–623.
38. Latah, M.; Toker, L. Load and stress testing for SDN's northbound API. *SN Appl. Sci.* **2020**, *2*, 1–8. [CrossRef]
39. Floodlight Rest Api. 2022. Available online: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343539/Floodlight+REST+API (accessed on 10 February 2022).
40. Dorri, A.; Kanhere, S.S.; Jurdak, R.; Gauravaram, P. LSB: A Lightweight Scalable Blockchain for IoT security and anonymity. *J. Parallel Distrib. Comput.* **2019**, *134*, 180–197. [CrossRef]
41. Singh, H.J.; Hafid, A.S. Transaction confirmation time prediction in ethereum blockchain using machine learning. *arXiv* **2019**, arXiv:1911.11592.

42. Phemius, K.; Bouet, M.; Leguay, J. Disco: Distributed multi-domain sdn controllers. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–4.
43. Benamrane, F.; Ben Mamoun, M.; Benaini, R. An East-West interface for distributed SDN control plane: Implementation and evaluation. *Comput. Electr. Eng.* **2017**, *57*, 162–175. [CrossRef]