Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 9984666, 12 pages
https://doi.org/10.1155/2021/9984666

Research Article

BMC-SDN: Blockchain-Based Multicontroller Architecture for Secure Software-Defined Networks

Abdelouahid Derhab,[1] **Mohamed Guerroumi**,[2] **Mohamed Belaoued**,[3] **and Omar Cheikhrouhou**[4]

[1]*Center of Excellence in Information Assurance (CoEIA), King Saud University, Saudi Arabia*
[2]*Faculty of Electronic and Computer Science, USTHB University, Algiers, Algeria*
[3]*LICUS Lab., Department of Computer Science, University of 20 August 1955, Skikda, Algeria*
[4]*College of Computers and Information Technology, Taif University, P.O. Box 11099, Taif 21944, Saudi Arabia*

Correspondence should be addressed to Abdelouahid Derhab; abderhab@ksu.edu.sa

Received 8 March 2021; Revised 21 March 2021; Accepted 31 March 2021; Published 22 April 2021

Academic Editor: Muhammad Shafiq

Multicontroller software-defined networks have been widely adopted to enable management of large-scale networks. However, they are vulnerable to several attacks including false data injection, which creates topology inconsistency among controllers. To deal with this issue, we propose BMC-SDN, a security architecture that integrates blockchain and multicontroller SDN and divides the network into several domains. Each SDN domain is managed by one master controller that communicates through blockchain with the masters of the other domains. The master controller creates blocks of network flow updates, and its redundant controllers validate the new block based on a proposed reputation mechanism. The reputation mechanism rates the controllers, i.e., block creator and voters, after each voting operation using constant and combined adaptive fading reputation strategies. The evaluation results demonstrate a fast and optimal detection of fraudulent flow rule injection.

1. Introduction

Software-defined networks (SDNs) [1] have been widely deployed in many application fields, as they replace the conventional TCP/IP architecture with another one that decouples the networking devices from their control management. This is achieved by moving the network functions that are performed by the network devices to a central entity, which allows an easy and low-cost network management.

To manage large-scale and multidomain networks, multicontroller SDN is proposed [2], where each controller is responsible for one domain. In multicontroller SDN, there are two types of communication: vertical and horizontal. In vertical communication, OpenFlow protocol [3] manages the southbound interface between the controller and the forwarding devices, e.g., switches, by telling switches where to send data flows. The northbound interface manages the communication between the controller and the applications. In horizontal communication, controllers exchange information about network topology between them via their east-west interfaces.

The horizontal communication allows controllers to share any update about network topology like the link state, network devices, changes in the flow table, list of network hosts, and the association between each switch and its controller. Thus, it is important for controllers to maintain the same global network view. To this end, an intercontroller traffic must be exchanged between the controllers, through their east-west interfaces. So, a logical centralized view of the network state must be guaranteed for SDN controllers to facilitate the development of advanced network applications.

The main concern for the network administrator and network programmer is to keep the SDN controllers synchronized and having similar network topology information

to make the correct routing decisions. However, multicontroller SDN could be targeted by several attacks [4–6], including false data injection, where a compromised controller sends fraudulent flow information to other controllers. This could cause routing malfunctions, routing loops, and incorrect functionality of firewalls.

To deal with this issue, we propose a security architecture that integrates blockchain technology with multicontroller SDN. The main idea of the architecture is to associate a set of controllers to each domain. Differently from [7] that deploys many controllers for fault-tolerance purposes, our architecture is aimed at ensuring a secure and trustworthy intercontroller communication. To this end, the proposed architecture considers a master controller and redundant controllers for each network domain. Each controller can be master in one domain and redundant in other domains. The master controller creates blocks of network flow updates, and the redundant controllers decide whether to validate the created blocks or not. The architecture also integrates a reputation mechanism that rates the controllers after each voting operations using constant and adaptive fading reputation strategies. In this way, malicious master controllers and redundant controllers that provide incorrect voting will be detected. More specifically, the main contributions of the paper are as follows:

(i) We propose BMC-SDN, a security architecture that integrates SDN and blockchain technologies to secure intercontroller communication. BMC-SDN assigns a single master controller and multiple redundant controllers to each domain. The controllers are members of the blockchain; the master controller creates blocks, and its behavior is monitored by the redundant ones.

(ii) We integrate a reputation mechanism to BMC-SDN, which rates controllers according to two strategies: (1) *constant fading reputation* that allows forgetting past operations of the controller at a constant rate and (2) *combined adaptive fading reputation* that rates the controller using different constants according to the controller's reputation. More precisely, the more the controller misbehaves, the faster positive histories are forgotten. On the other hand, the more the controller well-behaves, the faster negative histories are forgotten.

(iii) We implement the proposed BMC-SDN architecture using different tools such as ONOS, MultiChain, and Mininet software platforms. The evaluation results show that BMC-SDN can detect all injected flows with a low detection delay. In addition, the reputation mechanism allows adaptive detection time of malicious controllers according to the requirements of the network administrator.

The rest of this paper is organized as follows: Section 2 presents related work. The system and threat model is given in Section 3. Section 4 provides a detailed description of BMC-SDN. The implementation and performance evaluation of BMC-SDN are presented in Sections 5 and 6, respectively. Finally, Section 7 concludes the paper.

2. Related Work

Security of SDN has attracted much attention from researchers [6, 8]. Tayfour and Marsono [9] proposed a collaborative technique to detect and mitigate distributed denial-of-service (DDoS) flooding attacks on software-defined network (SDN) across multicontroller domains, using sflow-RT [10] and Snort rules. Halder et al. [11] proposed a mechanism that detects conflicts in distributed SDN controller environment. Each controller generates a directed graph from the forwarding rules. The different graphs are merged to generate a global network state, which allows detecting any kind of flow rule violation and forwarding loops. Das et al. [7] proposed an SDN architecture that replaces the single controller with multiple independent controllers, which allows tolerance to controller failure. Varadharajan et al. [12] proposed a policy-based security architecture for a multidomain SDN network. The packet flows are analyzed to identify an eventual unauthorized flow and dynamically update security policies.

Blockchain has been adopted as an option to secure one-controller SDN. Derhab et al. [13] proposed BICS (Blockchain-based Integrity Checking System), which sends the traffic flow rules of the vSwitch to the blockchain. The fraudulent flow rules are detected if the rules sent by the SDN controller are different from the ones in the blockchain. In [14], the blockchain is also used to secure the communication between the SDN controller and the other network elements against false flow rule injection. In [15], a blockchain mechanism is proposed to protect against unauthorized access and DDoS attack. In this mechanism, the switches are registered and verified using zero proof of knowledge. They are also validated in the blockchain using a voting-based consensus mechanism. Also, a Boltzmann deep learning machine is applied to identify anomalous flow traffic. DistBlockNet [16] verifies and validates the version of the flow rule table using a blockchain and downloads the latest flow rules for the IoT forwarding devices. TrustBlock [17] computes the trust values of SDN nodes based on blockchain. The consensus mechanism of the blockchain is used to filter out dishonest nodes that provide unfair recommendations.

Blockchain is also used to secure multicontroller SDN. Fernando and Wei [18] proposed an infrastructure comprising two layers: (1) multicontroller SDN networking layer and (2) blockchain-based layer. The control/management commands of the SDN controller are hashed and recorded in a smart contract of the blockchain and sent to the targeted SDN controller. The latter verifies the integrity of the command by checking the smart contract. Yang et al. [19] proposed BlockTC, a distributed blockchain-based trusted multidomain collaboration for mobile edge computing, wherein all SDN controllers use the blockchain to obtain topology information of other domains and verify the legitimacy of the routing. Azab et al. [20] proposed a system, where multiple controllers manage the same set of switches.

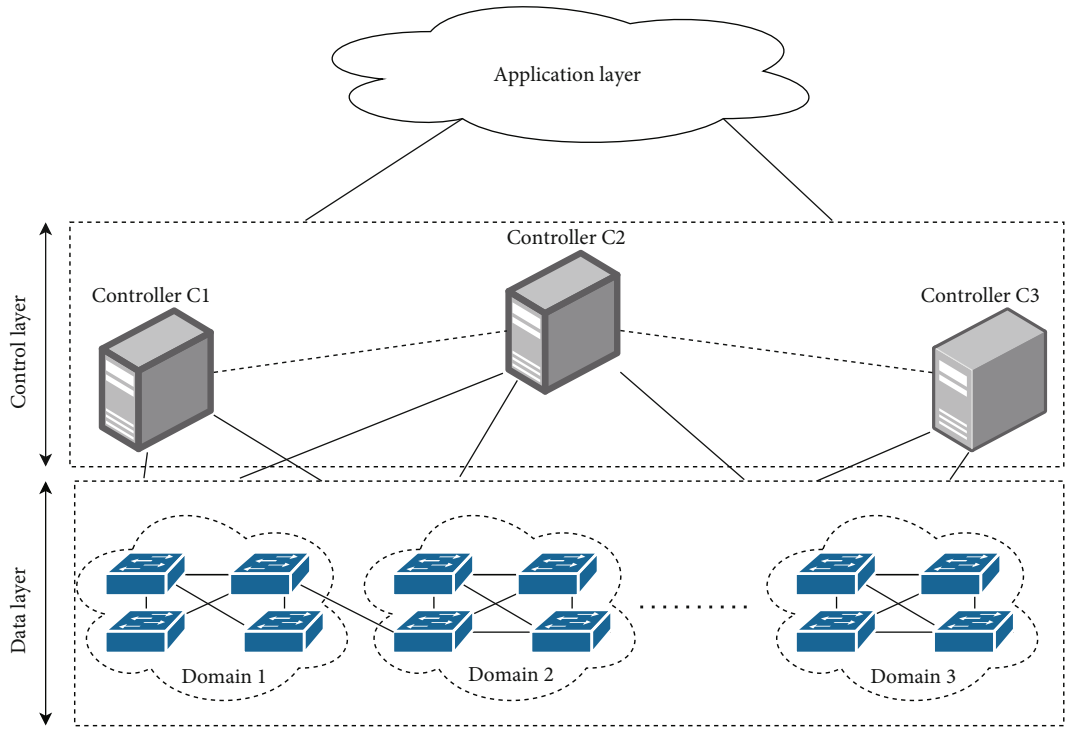


Figure 1: General architecture of multicontroller SDN.

The blockchain is used to ensure consistency between SDN controllers. The switches and the controllers are considered nodes in the blockchain. When the switch broadcasts its request to all controllers, each node in the blockchain can verify and validate the request. Kataoka et al. [21] proposed a trust list that is distributed among IoT devices using blockchain and SDN. The IoT devices can only trust other devices and services that are whitelisted in the trust list.

Blockchain is also proved to be a promising technology to mitigate false data injection in IoT networks. For example, Cheikhrouhou and Koubaa in [22] leveraged blockchain technology to secure localization in IoT networks and guarantee the correctness of the shared information. The proposed solution permits detection of false data injection and therefore reduces the localization error.

3. System and Threat Model

As shown in Figure 1, we consider an SDN network with multiple and distributed controllers. The general SDN architecture contains three layers: application, control, and data layers. The application layer contains programs that explicitly communicate their network requirements and desired network rules to the controllers. The control layer contains N controllers deployed in a distributed manner. The data layer contains devices that are configured in N different domains. Each domain is controlled by one master controller, and each master controller has more than one slave or redundant controllers. In addition to its primary role, the master controller is configured as a redundant controller for more than one domain. The controllers in a distributed multicontroller SDN keep the same network global view (i.e., same topology and same link status). Any change in the state of each controller such as new flow configurations and link failures must be quickly sent to other controllers. If the controllers have an inconsistent view, the network strategies may not be executed correctly, which could cause network misconfigurations such as routing loops, packet losses, and firewall leaks.

Figure 2 shows the possible threats and attacks that could occur in a multicontroller SDN architecture. If the communication between controller C1 and the other controllers fails, then the network could be partitioned. In this case, if the network topology changes in the domain of controller C1, the other controllers cannot be informed and vice versa. Therefore, the controllers could make routing decisions based on their own old view of the network topology, which could lead to unforeseen events.

Moreover, an attacker can intercept the communication between controllers and injects false data. This man-in-the-middle attack (Figure 2: arrow 1) can lead to an inconsistent network view, which leads to routing errors such as routing loops (Figure 2: arrow 2), firewall leaks, poor and false routing decision, and congestion of critical links when the majority of the false flow rules go through the same link, which ultimately affects the performance of the network.

In addition, it is possible for an attacker in this communication model to spoof controllers (Figure 2: arrow 3) and send false information about the topology, the state of the links, and the hosts of each domain. This false information can cause several problems such as traffic congestion, device overloading, and wrong routing information.

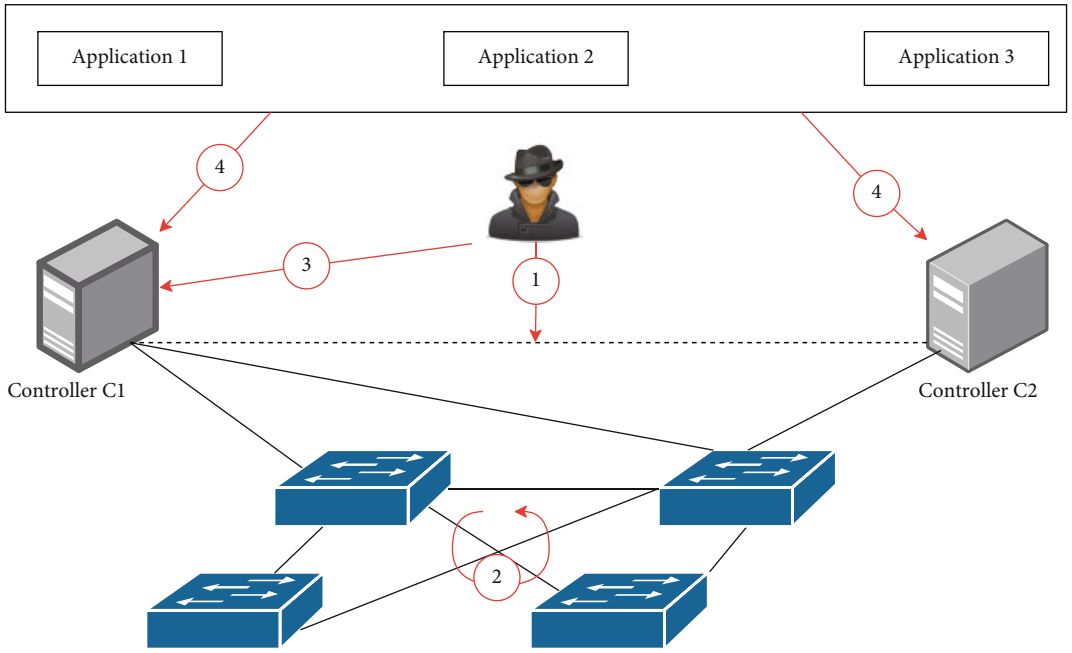


Figure 2: Threat model for multicontroller SDN architecture.

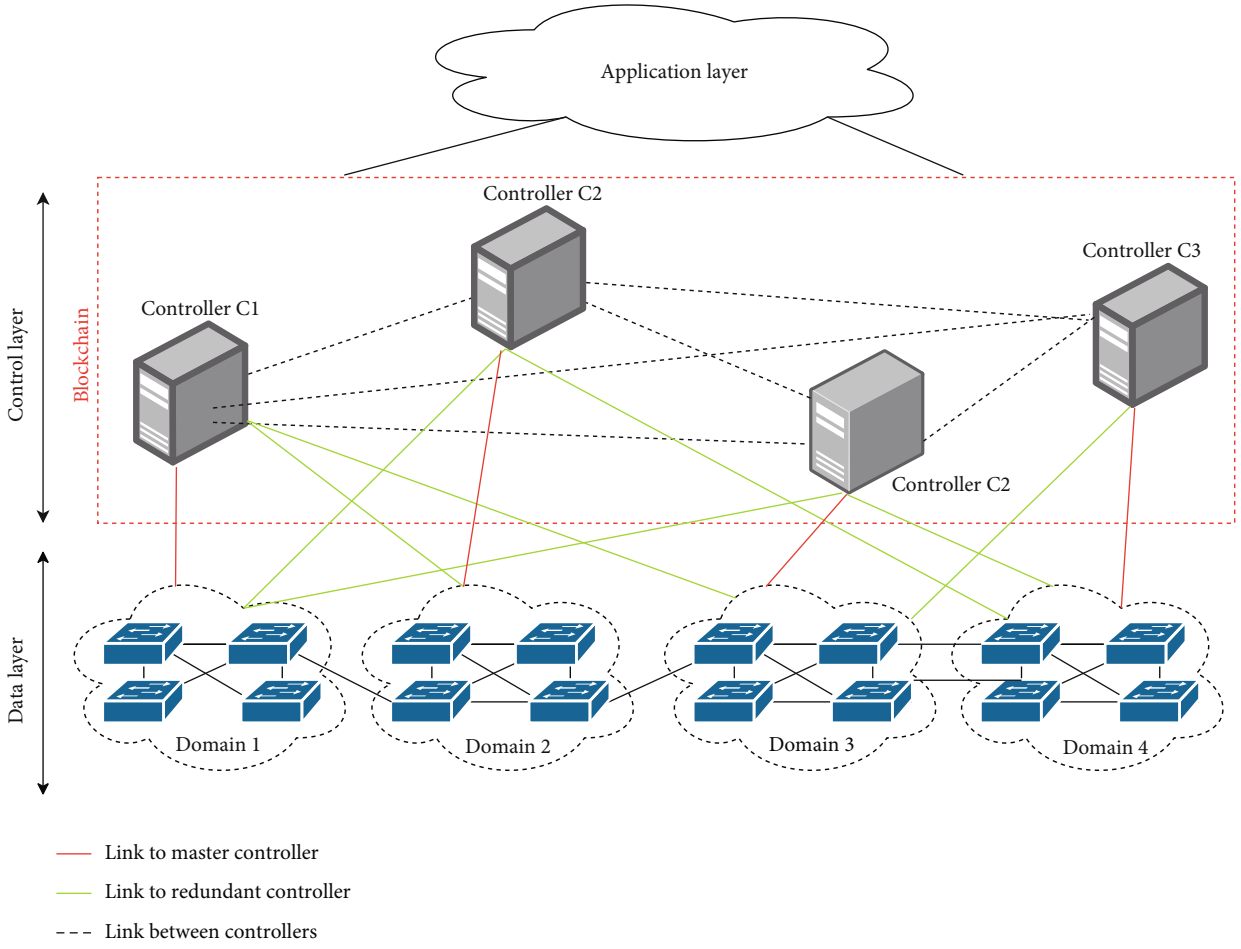


Figure 3: The proposed BMC-SDN architecture.

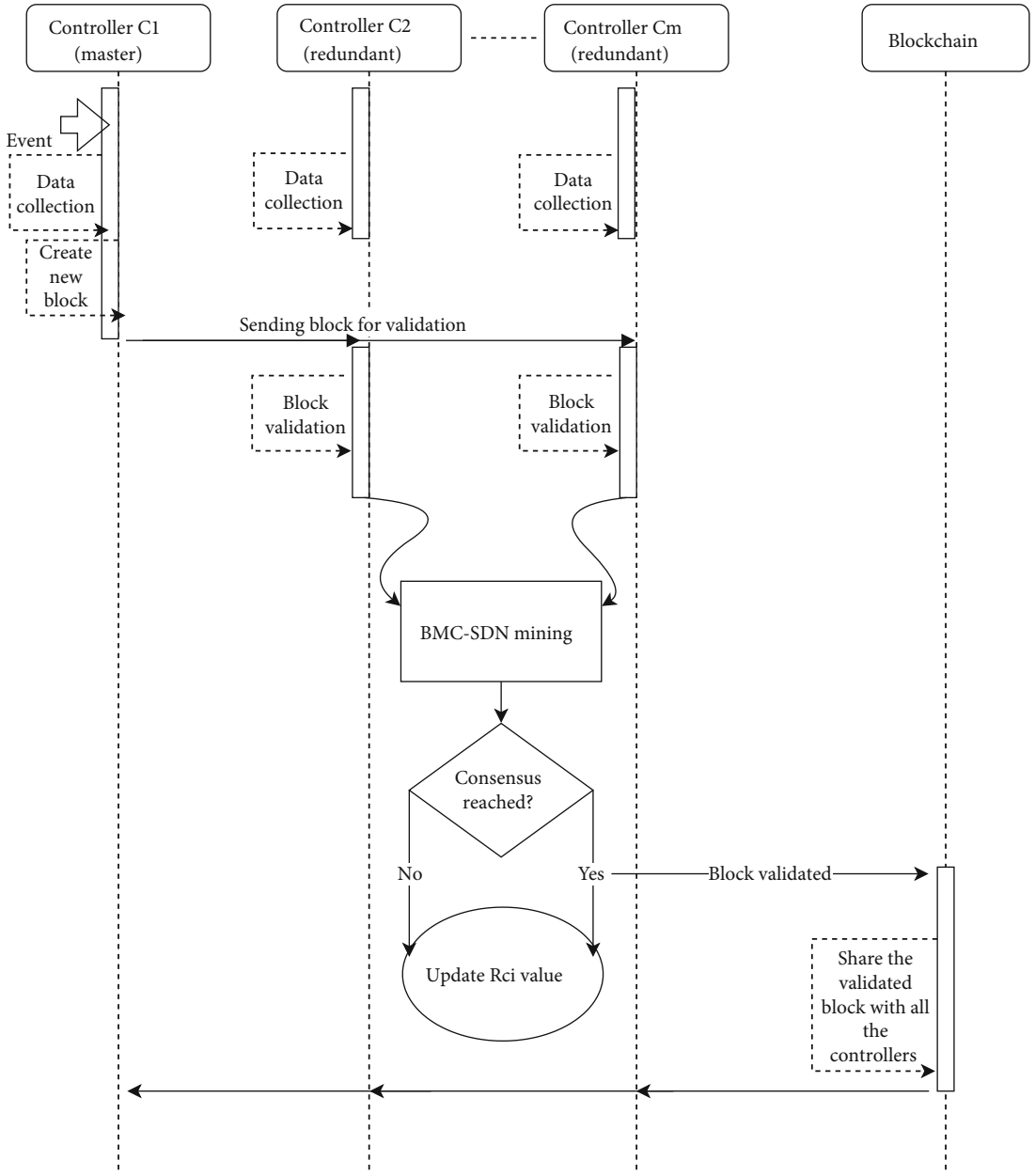


Figure 4: BMC-SDN sequence diagram.

Another critical attack could be launched by exploiting some APIs that are offered by the SDN controller [23]. These APIs allow developers to implement new network control applications. Indeed, if an application is compromised (Figure 2: arrow 4), the security of the entire network can be compromised. Such an attack can have dramatic consequences such as the modification of the behavior of the network and the intercepting of the network traffic, which can trigger large distributed denial-of-service attacks [24].

4. Blockchain-Based Multicontroller Software-Defined Network (BMC-SDN)

In this section, we provide a description of the proposed BMC-SDN architecture. The aim of BMC-SDN is to protect the control layer of the SDN architecture, previously presented in Figure 1, against the different attacks previously discussed in Section 3. To this end, BMC-SDN uses blockchain to protect the communication among controllers. Figure 3 gives a general overview of our proposed BMC-SDN solution. The control layer is protected thanks to blockchain. More precisely, all controllers are members of a blockchain network, and they communicate with each other via this blockchain. In BMC-SDN, we focus on the security of the control layer and the east-west communication of this layer. For the security of the communication between the controllers and the data layer devices, we consider our work in [13, 14]. Let N be the number of controllers in the network. For each domain, we select a single master controller and M redundant controllers, $2 \leq M < N$. The redundant

controllers replace the master controller in case of failure. A redundant controller cannot replace several master controllers unless it is the only available redundant one. The selection of the redundant controller that will replace the master controller is made according to its identity. More precisely, the redundant controller with the smallest ID is selected. In addition, the M redundant controllers of the same domain monitor the behavior of their master controller and participate in the consensus of validating the data blocks created by the master controller.

Figure 4 shows the sequence diagram that explains the main steps and operations in BMC-SDN. The master controller is responsible for orchestrating the traffic of its domain, and its redundant controllers monitor the domain, receive the same events, and apply the same updates as the master controller but have no influence on the domain. A redundant controller can control the domain in case the master controller fails as already explained. The controller has a local structure that contains OpenFlow commands [3] and local information such as network topology, list of hosts, and link state. This information could be changed due to several events such as topology change, link failure, and device failure. In case of any change, the master controller receives the corresponding event, performs the required update, and informs the other controllers. So, the master controller creates a block containing these updates and sends it through the blockchain to its redundant controllers. The latter, which have already received the same event as the master, validate this block according to the following consensus protocol: If the number of redundant controllers validating the block exceeds a threshold S, then the consensus is reached. In this case, the block is considered valid and will be added to the blockchain. This allows all controllers to have a global view of the entire network. If the consensus is not reached, the block is considered invalid. In this case, the master controller and the redundant controllers who validated this block will be negatively rated, as explained in Section 4.2.

4.1. BMC-SDN Trusted Node. The trusted node in BMC-SDN has the read and write permissions on the blockchain. All the master controllers are considered trusted nodes. They can read from blockchain and create new blocks. The creation of a new block is triggered by receiving a new external event that is sent from the data layer. When a master controller receives new information from the data layer devices of its domain, like flow rule request, it creates a new block containing adequate information and shares this new block with the redundant controllers for validation. The validated block is shared with all controllers in the network. Therefore, each controller can build the same global network view.

4.2. BMC-SDN Consensus and Reputation Mechanism. The consensus process is performed by the redundant controllers. These controllers are considered miners. They are responsible for validating newly created blocks. After creating a new block by the master controller, the new invalid block is shared with the miners. The miners start the validation process by comparing the similarity between the information contained in the invalid block and their local information.

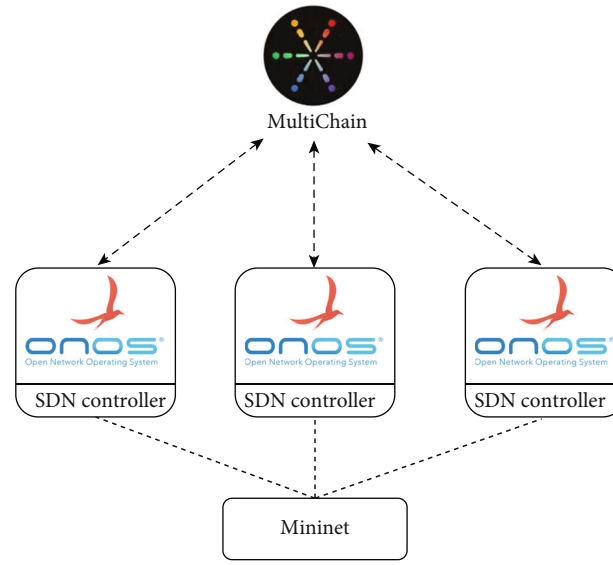


Figure 5: Implementation of BMC-SDN architecture.

Table 1: SDN implementation parameters.

Parameter	Value
Number of SDN domains	3
Number of redundant controllers	2
Number of switches	[10-100]
Number of hosts	[10-450]

The miners receive the same request as the master controller and then generate the needed reply. For example, they generate the same flow rule for the same flow rule request. So, the miner can check the similarity and accordingly validate the new block. Upon validation, the new block will be added to the chain. Miners who have a different opinion than the majority and the master controller whose block has not been validated could be considered a malicious controller. The detection of the malicious controller will be calculated based on the below reputation mechanism.

This reputation mechanism is seen as an additional layer of security to protect the control layer and therefore the entire network. This mechanism is based on the reputation management of controllers. Each controller C_i has a reputation value R_i shared between miners in the same blockchain. The value of R_i is between 0 and 1 ($0 \le R_i \le 1$). In this mechanism, each controller can be in one of the following three states, depending on its reputation value R_i:

(i) Trustworthy controller, if $R_i \ge 0.8$. In this case, the information sent by the controller is evaluated and taken into account by the miners.

(ii) Suspicious controller, if $0.5 \le R_i < 0.8$. In this case, the information sent by the controller is evaluated but are not taken into account by the miners.

Table 2: Description of the data structures.

Data stores	Description
Mastership store	Associates between each switch and its master.
Network topology store	Describes the network topology in terms of links and switches.
Flow store	Saves the flows of each switch from the master controller to the slave controller, when a change in the flow table is detected.
Host store	Manages the list of network hosts.
Application store	Manages the application inventory.
Intent store	Manages the inventory of intentions. Intentions are part of the ONOS intention framework used by applications to define the network policy, without specifying in detail how the data plan should actually be programmed.
Component configuration store	Stores system-wide configurations for various software components in ONOS.
Network configuration store	Stores the inserted network configurations in ONOS.
Security mode store	Manages the authorizations granted to applications using the RAFT protocol [33]. Instead, security violations are handled using antientropy protocol.

```
yassine@ubuntu:~$ python3 attack.py  10 1 100
flow rules injection at : 27/06/2019      12:48:00
flow rules injection at : 27/06/2019      12:48:01
flow rules injection at : 27/06/2019      12:48:02
flow rules injection at : 27/06/2019      12:48:03
flow rules injection at : 27/06/2019      12:48:04
flow rules injection at : 27/06/2019      12:48:05
flow rules injection at : 27/06/2019      12:48:06
flow rules injection at : 27/06/2019      12:48:07
flow rules injection at : 27/06/2019      12:48:08
flow rules injection at : 27/06/2019      12:48:09
```

Figure 6: Attack experiment.

(iii) Malicious controller, if $R_i < 0.5$. The communication traffic of this controller is ignored by others, until intervention of the network administrator. When $R_i < 0.5$, the reputation of C_i is continuously updated and switches to the suspicious and the trustworthy states when $R_i \geq 0.5$ and $R_i \geq 0.8$, respectively.

The miner controllers evaluate the controller C_i according to the consensus result:

(i) If the consensus is reached, the block sent by the master controller will be validated and the value of its reputation increases.

(ii) If the consensus is not reached, the block sent by the master controller will not be validated, and thus, the value of its reputation decreases.

(iii) For miners who have the same opinion as the majority, their reputation will increase.

(iv) For miners who have a different opinion than the majority, their reputation will decrease.

More precisely, the value of R_i is computed as follows:

(i) We compute the reputation of controller C_i, denoted by RP_i, during each time period (or observation interval). Formally, $RP_i = P_i/TP_i$, where P_i is the number of positive participation and TP_i is the total number of positive participation made by controller C_i in blockchain operations (creation and validation of blocks).

(ii) If $RP_i < 0.5$, the reputation of C_i during the current time period RT_i is set to 0. Otherwise, it is set to 1.

(iii) $R_i = \omega R_i + (1 - \omega)RT_i$, where $\omega \in [0, 1]$ is a constant fading (or discount) factor for past participations.

By using the constant fading factor, both positive and negative histories are forgotten at the same rate. Let us consider the following scenario:

(i) If ω is low, we have two scenarios:

(a) If the controller is trustworthy and starts behaving maliciously, then the positive history will be forgotten slowly, and hence, detection time of the controller will be high.

(b) If the controller is malicious and starts well-behaving, then the negative history will be forgotten slowly, and hence, redemption time of the controller will be high.

(ii) If ω is high, we have two scenarios:

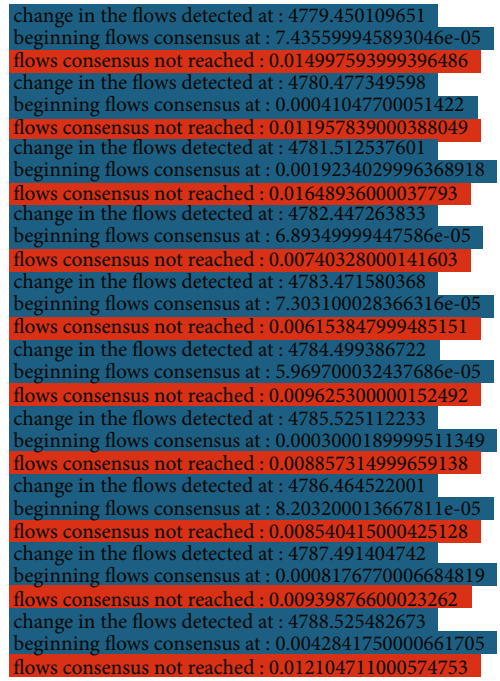


Figure 7: Detection experiment.

(a) If the controller is trustworthy and starts behaving maliciously, then the positive history will be forgotten quickly, and hence, detection time of the controller will be low.

(b) If the controller is malicious and starts well-behaving, then the negative history will be forgotten quickly, and hence, redemption time of the controller will be low. In this scenario, the controller could take advantage of the reputation system and act maliciously most of the time, and when its state becomes suspicious or malicious, it just needs to perform few positive operations to return back to the trustworthy state.

Based on the above scenarios, we can observe that using a constant fading factor has some disadvantages. To deal with this issue, we propose using different fading factors according to the reputation of the controller, as follows:

$$R_i = \begin{cases} \omega_3 R_i + (1-\omega_3) RT_i, \text{ when } R_i \geq 0.8, \\ \omega_2 R_i + (1-\omega_2) RT_i, \text{ when } 0.5 \leq R_i < 0.8, \\ \omega_1 R_i + (1-\omega_1) RT_i, \text{ when } R_i < 0.5, \end{cases} \quad (1)$$

where $\omega_3, \omega_2, \omega_1 \in [0, 1]$ and $\omega_3 > \omega_2 > \omega_1$. The above equation uses combined fading factors, i.e., the more the controller misbehaves, the faster (resp., the slower) positive histories (resp., negative histories) are forgotten. On the other hand, the more the controller well-behaves, the faster (resp., the slower) negative histories (resp., positive histories) are forgotten.

Table 3: Detection rate vs. number of attacks.

Number of attacks	Detection rate (%)
10	100
20	100
30	100
40	100
50	100
60	100
70	100
80	100
90	100
100	100

5. Implementation

5.1. Implementation Environment. In this section, we implement the blockchain-based secure multicontroller architecture, as shown in Figure 5, using the following components:

(1) *SDN Controller*. We use the Open Network Operating System (ONOS) [25] to implement the SDN controller. It provides the control plane that supports the deployment of several controllers as a domain. The implementation parameters of the SDN controller are shown in Table 1.

(2) *Blockchain*. We use MultiChain [26], an open source platform to implement a private blockchain. It can assign privileges to nodes and control who can connect, send, and receive transactions and who can create flows and blocks. Each MultiChain node is accessible through the MultiChain Web Demo [27], which is a simple web interface for multiChain blockchains.

(3) *Mininet* [28]. It creates a virtual network supporting OpenFlow [3] and consisting of switches and real applications that are deployed on a single machine (virtual or real machine or cloud).

The code that we used to implement BMC-SDN can be found in [29]. In addition, we use other tools to implement our solution, such as Postman [30], an application that allows sending http requests and manage authentication. We also use SecureCRT [31], which is software for network administration and end-user access. Our solution is implemented using Python programming language, and some libraries like HTTPBasicAuth, and Requests that authenticate and interact with REST APIs of the ONOS SDN controllers, respectively. We also use JSON (i.e., JavaScript Object Notation) [32] in order to represent data that are processed by the controller data.

5.2. Data Structures. We handle data stores, which are the real distributed data structures in ONOS controllers; the main ONOS Stores are presented in Table 2.

Among the distributed stores presented in Table 2, there are those that are related to the behavior of the data plane such as the network topology store, the flow store, and the host store. The other distributed stores are considered application-specific.

6. Performance Evaluation

In this section, we evaluate the performance of BMC-SDN using the following metrics:

(1) *Total Execution Time*. It represents the required time to transfer a flow through the blockchain, denoted by (T_{Total}). It is the sum of three elements, relating to the number of switches and the number of hosts in the network, namely, (1) the consensus time, (2) the time to send a block, and (3) the time to update data:

$$T_{\text{Total}} = T_{\text{Consensus}} + T_{\text{Sent}} + T_{\text{Update}}. \quad (2)$$

(2) *Detection Rate (DR)*. It represents the ratio of the number of detected attacks to the total number of injected attacks.

(3) *Detection Time (DT)*. It records the required time to detect malicious controllers.

To assess the robustness of our BMC-SDN solution, we inject fraudulent flows at the controller, as depicted in Figure 6. In Figure 7, we can see that these flows are detected as malicious and reported to the administrator by adding an entry to the log file, containing details of the detected anomaly.

Table 4: Execution times vs. number of switches.

Number of switches	CTC (s)	TTBC (s)	UTBC (s)	TT (s)
10	0.019	0.007	0.018	0.055
20	0.037	0.012	0.013	0.062
30	0.054	0.013	0.02	0.087
40	0.044	0.016	0.035	0.095
50	0.059	0.019	0.025	0.103
60	0.108	0.035	0.049	0.192
70	0.089	0.053	0.059	0.201
80	0.109	0.093	0.079	0.281
90	0.15	0.074	0.099	0.323
100	0.193	0.053	0.087	0.333

CTC: consensus time; TTBC: transfer time within the blockchain; UTBC: update time from the blockchain; TT: total time.

Table 5: Execution times vs. number of hosts.

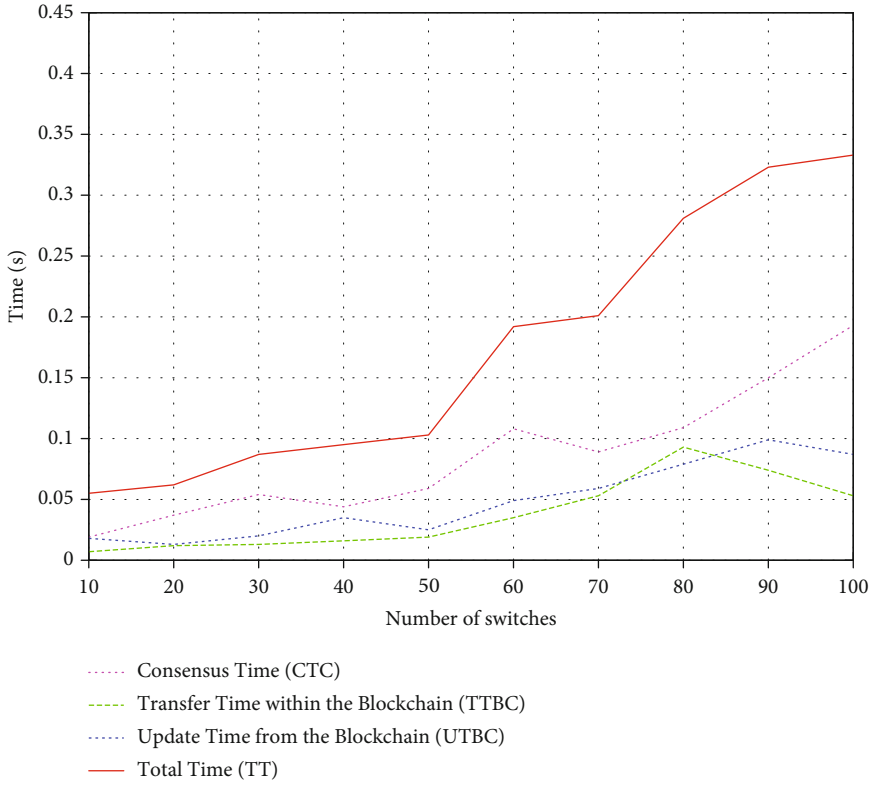
Number of hosts	CTC (s)	TTBC (s)	UTBC (s)	TT (s)
10	0.017	0.007	0.002	0.026
50	0.018	0.007	0.012	0.037
100	0.014	0.008	0.014	0.036
150	0.026	0.028	0.018	0.072
200	0.037	0.026	0.017	0.08
250	0.043	0.036	0.032	0.111
300	0.039	0.036	0.029	0.104
350	0.047	0.038	0.047	0.132
400	0.045	0.04	0.056	0.141
450	0.051	0.055	0.049	0.155

CTC: consensus time; TTBC: transfer time within the blockchain; UTBC: update time from the blockchain; TT: total time.

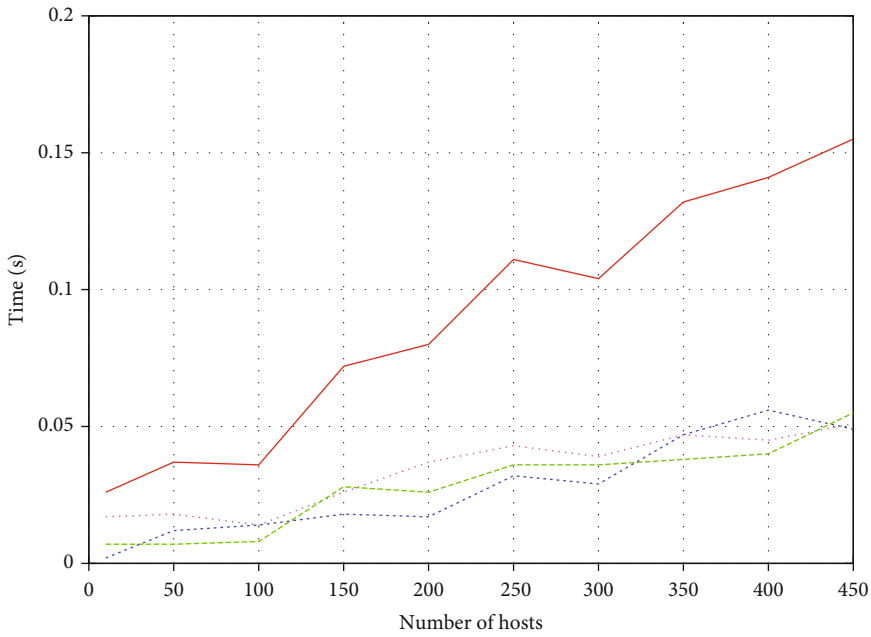
Table 3 represents the detection rate according to the number of injected attacks. As shown in the table, BMC-SDN ensures the detection rate of 100%, which indicates success in detecting all the injected attacks in the network. As the redundant controllers have the same network view as the master controller, any fraudulent flow that is injected by the master is detected by the redundant ones during the block validation.

As shown in Tables 4 and 5 and Figure 8, we can observe that the total execution time increases as the number of switches increases. We can also observe that the execution time of the consensus increases when the number of switches and hosts is increased. However, the recorded values of execution time are very low.

Figure 9 shows the detection time of the reputation mechanism when the controller behaves maliciously under three values of constant fading factors $\omega = 0.2,0.5,0.8$ and under the combined fading factor such that $\omega_3 = 0.8$, $\omega_2 = 0.5$, and $\omega_1 = 0.2$. We can observe that the reputation of the



(a) Execution times vs. number of switches



(b) Execution times vs. number of hosts

Figure 8: Execution time.

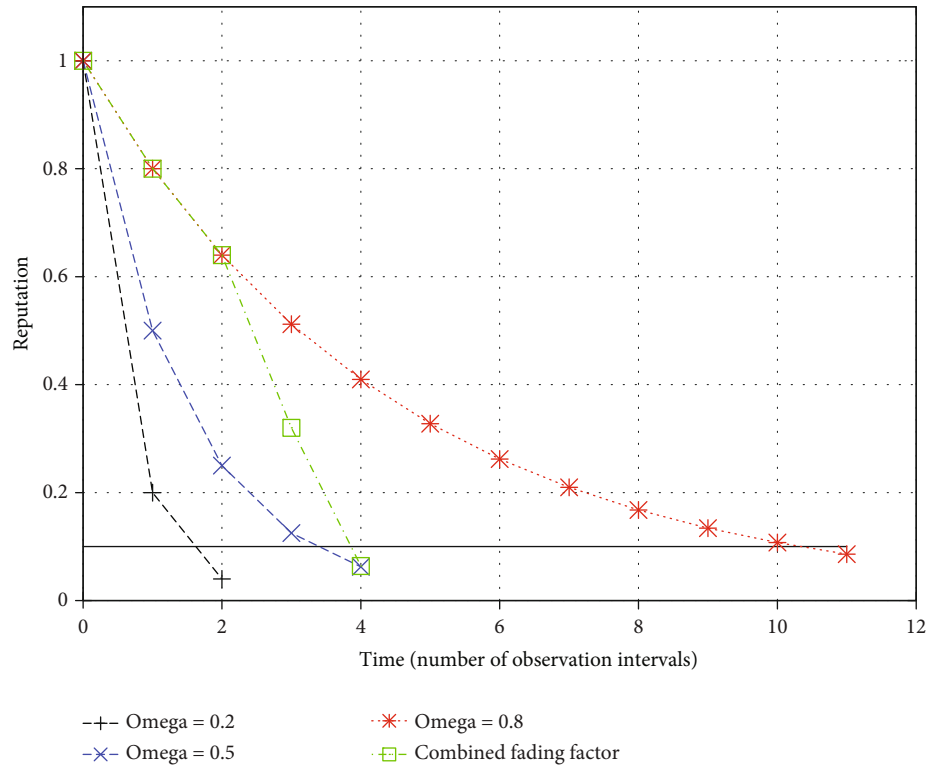


Figure 9: Detection time of reputation mechanism.

controller decreases slowly under high constant fading factor, and hence, high detection time is achieved (i.e., $\omega = 0.8$), and decreases quickly under low constant fading factor, and hence, low detection time is achieved (i.e., $\omega = 0.2$). We can also observe that the combined fading factor employs different fading factors depending on the reputation of the controller. When $R_i \geq 0.8$, it slowly decreases when the fading factor is high (i.e., $\omega = 0.8$). When $R_i < 0.8$, it decreases at a higher speed, and hence, lower detection time is achieved.

7. Conclusion

In this paper, we have proposed BMC-SDN, a blockchain-based multicontroller architecture for secure software-defined networks. In this architecture, we cluster network devices into SDN domains. One master controller and multiple redundant controllers are assigned to each SDN domain. We have used a blockchain where the master controller creates blocks of network flow updates, and redundant controllers validate the blocks. After each voting operation, a reputation mechanism is invoked to rate the controllers, i.e., block creator and voters. The reputation mechanism employs constant and adaptive combined fading reputation strategies to manage and customize the detection time of malicious controllers. The proposed security architecture has been implemented and tested using ONOS, MultiChain, and Mininet software platforms. The evaluation results have reached 100% detection of flow rule injections in a short time. In addition, the proposed combined-fading reputation mechanism has allowed adaptive configuration of fading parameters to reach desired detection time. As BMC-SDN only considers the security of east-west interfaces, we plan as future work to cover the rest of the security planes of SDN architecture, especially the southbound interfaces.

Data Availability

We used data generated from attack scripts and simulators.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

Dr. Omar Cheikhrouhou thanks Taif University for its support under the project Taif University Researchers supporting project number (TURSP-2020/55), Taif University, Taif, Saudi Arabia.

References

[1] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: from concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[2] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: a survey," *IEEE Access*, vol. 6, pp. 15980–15996, 2018.

[3] 2021, https://www.sdxcentral.com/sdn/definitions/what-is-openflow/.

[4] M. Imran, M. H. Durad, F. A. Khan, and A. Derhab, "Toward an optimal solution against denial of service attacks in software

defined networks," *Future Generation Computer Systems*, vol. 92, pp. 444–453, 2019.

[5] W. LI, W. MENG, Z. LIU, and M. H. AU, "Towards blockchain-based software-defined networking: security challenges and solutions," *IEICE Transactions on Information and Systems*, vol. E103.D, no. 2, pp. 196–203, 2020.

[6] Z. Shah and S. Cosgrove, "Mitigating ARP cache poisoning attack in software-defined networking (SDN): a survey," *Electronics*, vol. 8, no. 10, p. 1095, 2019.

[7] R. K. Das, F. H. Pohrmen, A. K. Maji, and G. Saha, "FT-SDN: a fault-tolerant distributed architecture for software defined network," *Wireless Personal Communications*, vol. 114, no. 2, pp. 1045–1066, 2020.

[8] R. Swami, M. Dave, and V. Ranga, "Software-defined networking-based DDoS defense mechanisms," *ACM Computing Surveys*, vol. 52, no. 2, pp. 1–36, 2019.

[9] O. E. Tayfour and M. N. Marsono, "Collaborative detection and mitigation of distributed denial-of-service attacks on software-defined network," *Mobile Networks and Applications*, vol. 25, no. 4, pp. 1338–1347, 2020.

[10] M. Afaq, S. Rehman, and W.-C. Song, "Large flows detection, marking, and mitigation based on sFlow standard in SDN," *Journal of Korea Multimedia Society*, vol. 18, no. 2, pp. 189–198, 2015.

[11] B. Halder, M. S. Barik, and C. Mazumdar, "Detection of flow violation in distributed SDN controller," in *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 1–6, Kolkata, 2018.

[12] V. Varadharajan, K. Karmakar, U. Tupakula, and M. Hitchens, "A policy-based security architecture for software-defined networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 897–912, 2019.

[13] A. Derhab, M. Guerroumi, A. Gumaei et al., "Blockchain and random subspace learning-based IDS for SDN-enabled industrial IoT security," *Sensors*, vol. 19, no. 14, p. 3119, 2019.

[14] S. Boukria, M. Guerroumi, and I. Romdhani, "BCFR: blockchain-based controller against false flow rule injection in SDN," in *2019 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1034–1039, Barcelona, Spain, 2019.

[15] M. Singh, G. S. S. Aujla, A. Singh, N. Kumar, and S. Garg, "Deep-learning-based blockchain framework for secure software-defined industrial networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 606–616, 2021.

[16] P. K. Sharma, S. Singh, Y.-S. Jeong, and J. H. Park, "DistBlockNet: a distributed blockchains-based secure SDN architecture for IoT networks," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 78–85, 2017.

[17] B. Zhao, Y. Liu, X. Li, J. Li, and J. Zou, "TrustBlock: an adaptive trust evaluation of SDN network nodes based on double-layer blockchain," *PloS one*, vol. 15, no. 3, article e0228844, 2020.

[18] P. Fernando and J. Wei, "Blockchain-powered software defined network-enabled networking infrastructure for cloud management," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–6, Las Vegas, NV, USA, 2020.

[19] H. Yang, Y. Liang, J. Yuan, Q. Yao, A. Yu, and J. Zhang, "Distributed blockchain-based trusted multidomain collaboration for mobile edge computing in 5G and beyond," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 11, pp. 7094–7104, 2020.

[20] M. Azab, R. R. Ergawy, E. M. Ghourab, A. Mokhtar, and M. Rizk, "Towards blockchain-based multi-controller managed switching for trustworthy SDN operation," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 0991–0998, Vancouver, BC, Canada, 2019.

[21] K. Kataoka, S. Gangwar, and P. Podili, "Trust list: Internet-wide and distributed IoT traffic management using blockchain and SDN," in *2018 IEEE 4th world forum on internet of things (WF-IoT)*, pp. 296–301, Singapore, 2018.

[22] O. Cheikhrouhou and A. Koubaa, "BlockLoc: secure localization in the internet of things using blockchain," in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 629–634, Tangier, Morocco, 2019.

[23] O. Cheikhrouhou, "Secure group communication in wireless sensor networks: a survey," *Journal of Network and Computer Applications*, vol. 61, pp. 115–132, 2016.

[24] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: a survey, some research issues, and challenges," *IEEE communications surveys & tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

[25] 2021, https://www.opennetworking.org/onos/.

[26] 2021, https://www.multichain.com/.

[27] 2021, https://github.com/MultiChain/multichain-web-demo.

[28] 2021, http://mininet.org/.

[29] Bmc-sdn code. https://github.com/medguerroumi/BMC-SDN/blob/35a61658700b1fe2d791b8d4f75d2edaa872839f/README.md.

[30] 2021, https://www.postman.com/.

[31] 2021, https://www.vandyke.com/products/securecrt/.

[32] 2021, https://www.jsonrpc.org/specification.

[33] The raft consensus algorithm. https://raft.github.io/.