Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6662175, 9 pages
https://doi.org/10.1155/2021/6662175

Research Article

Reaching Consensus with Byzantine Faulty Controllers in Software-Defined Networks

Chien-Fu Cheng,[1] **Jerry Chun-Wei Lin**,[2] **Gautam Srivastava**,[3,4] **and Chu-Chiao Hsu**[5]

[1]*Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung City 202301, Taiwan*
[2]*Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen 5063, Norway*
[3]*Department of Mathematics & Computer Science, Brandon University, Brandon, Canada*
[4]*Research Centre for Interneural Computing, China Medical University, Taichung, Taiwan*
[5]*Hon Lin Technology Company Limited, Taipei 11492, Taiwan*

Correspondence should be addressed to Jerry Chun-Wei Lin; jerrylin@ieee.org

Received 12 December 2020; Revised 4 March 2021; Accepted 17 March 2021; Published 12 April 2021

Academic Editor: Keping Yu

The combination of the Internet of Things (IoT) and Cloud computing are both popular applications that are based on the Internet. However, the traditional networking structure can no longer support the transmission of the huge number of packets required by cloud computing and IoT. Therefore, a new-generation architecture, software-defined networking (SDN), came into being. The new-generation SDN can control routing through software, making flexible and convenient management a key feature of SDN. However, attacks and threats are prevalent in today's networking environment. When an SDN controller encounters a hacker attack or virus infection, it will not function properly. Hence, we need to design a fault-tolerant mechanism for the above environment. In this paper, a fault-tolerant consensus protocol is proposed to improve the fault tolerance of SDNs with multiple controllers.

1. Introduction

With the flourishing enhancement of cloud computing and the Internet of Things (IoT), the scale of the Internet has grown at a very rapid rate [1–5]. This also makes the current Internet Protocol (IP) network architecture gradually unable to forward such a large amount of network traffic. This is because, in the current IP network infrastructure, the packets are forwarded through routers. The routing table of these routers is constructed by traditional routing protocols, which are preconfigured and embedded in the hardware by the vendor and run by a specific model. Network administrators do not have control over packets forwarding, thus resulting in poor utilization of network bandwidth [6]. Take multitenancy technology in data centers as an example, this cloud service is very convenient to users, but it is a heavy burden for the traditional IP network infrastructure [7]. The reason is that the routes to forwarding packets are calculated by dynamic routing protocols, and it is hard for the network administrator to identify which route the packets of a specific application have taken. Moreover, when customized adjustments of the network configurations are needed, the network administrator has to log in to each router or switch to manually change the settings via the Command-Line Interface (CLI). This method has a major drawback, that is, manual configuration of routers one by one involves a high risk of mistyping or giving inconsistent commands, which may cause failure of the entire network [8].

1.1. Software-Defined Networks (SDN). When attempting to solve issues of increased utilization of network bandwidth using the classical IP network infrastructure, the software-defined networking- (SDN-) based architecture is proposed. SDN divides a network into a control plane and a data plane. In this network, the controller in the control plane is responsible for the control of the network, while switches in the data

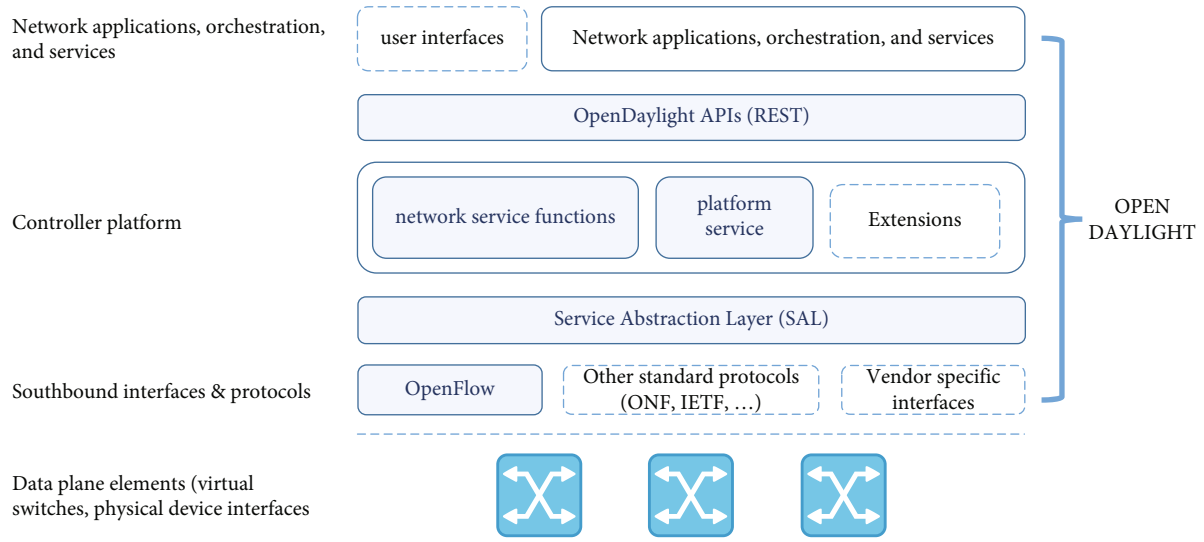

Figure 1: OpenDaylight platform [16].

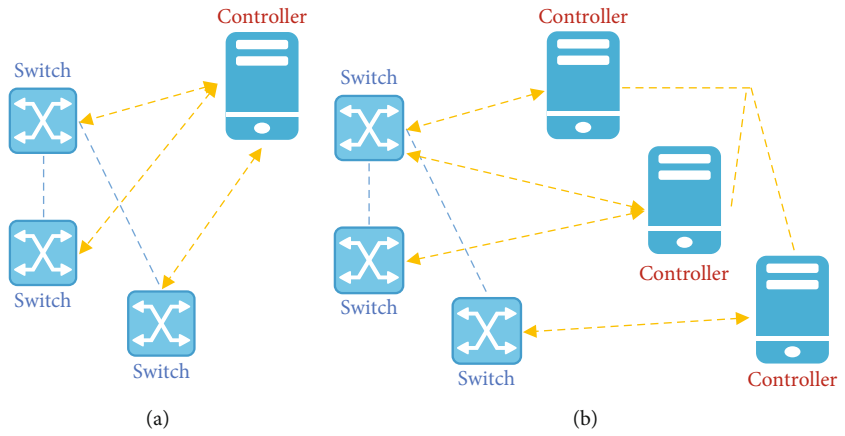

Figure 2: Illustrations of control architecture. (a) The centralized architecture. (b) The distributed architecture.

plane are responsible for packets forwarding. The controller can communicate with the switches in the data plane via the southbound interface. The southbound interface in SDN is better standardized. In the present, the OpenFlow interface is the most influential southbound interface [9]. On the other hand, applications can communicate with the controller via the northbound interface, but so far, this interface has not been standardized. Common northbound interfaces include REST [10] and SNMP [11].

Simply put, SDN is a concept of networking architecture. It does not specify the use of any specific technology. The core ideas are to separate control from forwarding and to use software programs to control the network and flexibly manage the network [12]. OpenFlow is a concrete protocol that can be implemented as the southbound interface in SDN. SDN has the following characteristics: Applications can flexibly control network traffic, optimize load balancing according to user needs, and make the use of bandwidth more adequate. Moreover, SDN considers time and energy saving to network management according to user needs. On the other hand, SDN combined with the virtualization technology can be used to enable different services in data center, thus sharing network equipment and reducing equipment investment.

Nowadays, Open Networking Foundation (ONF) is the most influential organization for SDN. It was founded in 2011 by Google, Facebook, and Microsoft [13]. Unlike other standards organizations formed by manufacturers of network equipment, ONF was created by "users" of network equipment. The mission of ONF is to promote OpenFlow protocol as the only standard for the southbound interface [14]. Established in 2013, OpenDaylight (ODL) is another well-known organization for SDN. It was initiated by 18 well-known IT companies [15]. The objective of ODL is to create an open SDN platform. As shown in Figure 1, this platform comprises three major layers. The top layer consists of the northbound interface and built-in applications and services. The middle layer is formed by network services and platform services. The bottom layer is the southbound interface. The northbound interface supports REST APIs. The southbound interface supports OpenFlow protocols and many other protocols, such as SNMP, LISP, XMPP,

Table 1: The requirements of the consensus protocol.

Consensus	*Each* correct processor should compute a common value.
Validity	If the initial value of each processors is v_i, then *each* correct processor should obtain the value v_i.

PCEP, OF-Config, Net-Config, BGP-LS, and even private interfaces defined by vendors [16].

However, attacks and threats are prevalent in today's networking environment [17–19]. As for security, a number of scholars have investigated the security mechanisms of SDN in recent years. The security issues that have been discussed in the SDN literature include access control [20], authentication [21], and nonrepudiation [22]. For access control, Nayak et al. [20] constructed a system called Resonance in an OpenFlow-based network. This system executes dynamic access controls based on flow-level messages and real-time alerts. For authentication, Porras et al. [21] designed for the NOXZ OpenFlow controller a software program called FortNOX to support role-based authorization as well as detection and negotiation of conflicting flow-entries. For nonrepudiation, Yao et al. [22] proposed a source address validation mechanism, called the Virtual Source Address Validation Edge (VAVE), for the OpenFlow/NOX network architecture. VAVE has three major features, including supporting on-demand filtering, rapidly reacting to route changes, and avoiding unnecessary checks to reduce the load on the router.

The Virtualizing Network Function (NFV) instead uses software such as firewall, router, load balancer, and customer's premise equipment (CME) to implement the network functions of physical devices. The NFV undermines the idea that the network features should exist on a particular hardware device. This is because, after virtualization of the network function, it has excellent elastic configuration features, allowing the deployment effectiveness of network services to be speeded up and costs of buying dedicated hardware reduced. Many great personalized services can be introduced easily with the complementarity of SDN and NFV.

1.2. Distributed Control Architecture. The controller of SDN is responsible for managing network resources and planning flow-entries according to the demand of upper-layer applications for the bottom-layer switches. Figure 2 illustrates a centralized control architecture and a distributed control architecture. The earlier SDN architecture is based on centralized control, that is, the entire network is managed by only one controller [23]. However, the single point of failure (SPOF) is likely to happen when the SDN is managed by a single controller. Moreover, the scalability problem is also an important issue if the network is huge. It would be hard to ensure the stability and efficiency of the network. Therefore, subsequent scholars proposed to use multiple controllers to collaboratively deal with the control tasks [24].

For example, Kyung et al. [25] propose the Load Distribution (LD) algorithm to avoid the overload problem of the controllers. The basic concept of the LD algorithm is that when the loading of the default controller reaches the threshold, the tasks it has received will be transferred to other controllers. Wang et al. [26] suggested to segment a large network into several domains and then use multiple controllers to, respectively, manage the domains to reduce the loading of the controllers. Wang et al. pointed out that finding an optimal number of controllers for a multidomain SDN environment is an NP-hard problem. For this problem, they proposed an approximation algorithm called the Greedy Subgraph Cover Problem (GSGCP), which is capable of placing a smaller number of controllers to manage a multidomain SDN. In addition, Yannan et al. [27] suggested that considering the expected percentage of control path loss in the problem of placing controllers can help effectively increase the reliability of SDN. They call the problem as the Reliability-aware Controller Placement (RCP) problem. In their study, the authors also show that the RCP problem is also an NP-hard problem.

1.3. Fault-Tolerant Mechanism. As mentioned above, a distributed control structure involving the collaboration of multiple controllers can effectively enhance the performance and stability of SDNs. However, attacks and threats are prevalent in today's networking environment. SDN controllers may not work normally when it is encountered by hackers or virus infection. Generally, faults of SDN controllers can be classified into crash fault [28], omission fault [29], and Byzantine fault [30–32]. A crash fault means that the controller will stop functioning properly. The omission fault of a controller occurs when messages are fully or partially omitted by the controller. Unlike the crash fault or omission fault, a Byzantine fault may cause a controller to send wrong messages or conspire with other faulty devices to interfere with the operation of nonfaulty controllers, thereby delivering incorrect computing results. These malicious behaviours include, for example, sending an incorrect flow-entry to the switch to prevent packets from being delivered to the correct destination and sending a specific flow-entry to the switch to enable the attacker to receive a copy of all the packets delivered to or sent from a particular host (i.e., eavesdropping packets).

To provide a reliable SDN with multiple-controller architecture, we need a mechanism that ensures the correctness of the computing results even if any controller has a fault or is under attack. That is, we have to design a protocol that can take advantage of the distributed system to let controllers work together to resist attacks from Byzantine controllers. Therefore, even in the presence of a faulty controller, the system will still deliver correct computing results. In a distributed system, we can establish the system's fault tolerance capability by solving the consensus problem. Common applications of the consensus protocol include the leader election problem in the P2P network [33], duplicated files storage coordination problem [34], cruise control problem in the car platoon [35], and clock synchronization [36, 37]. The requirements of the consensus protocol are shown in

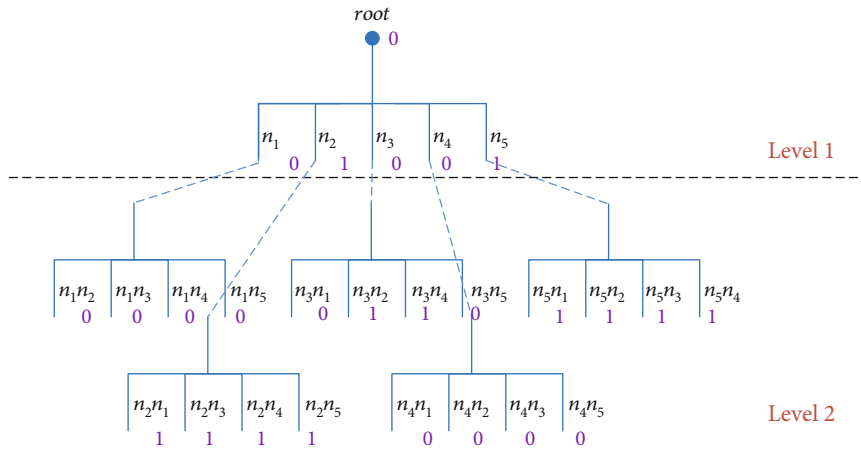

Figure 3: An example of SDN-tree.

Table 1 [38]. The main contributions of this paper are summarized as follows: (1) We design a consensus protocol for SDN with multiple-controller architecture. (2) The proposed consensus protocol can tolerate the most damaging type of faults (i.e., Byzantine fault). (3) By solving the consensus problem in SDN with multiple-controller architecture, we can create a highly reliable networking environment.

The remainder of this paper is organized as follows. "Preliminaries" describes the preliminary. "Concept and Approach" describes the protocol for solving the consensus problem. "An Execution Example" shows an example of solving the consensus problem. Finally, conclusions are drawn in "Conclusion."

2. Preliminaries

In this study, we assume that the underlying SDN is synchronous, and the failure type of faulty controllers includes Byzantine fault and dormant fault. Besides, we must reasonably limit the number of faulty controllers to ensure the system can work properly. The number of faulty controllers that can be tolerated by the system is subject to the number of controllers in the network. The following are environmental assumptions:

(i) The underlying network is synchronous

(ii) Let N be the set of controllers in the network, where $N = \{n_i \mid 1 \le i \le n, n = |N|\}$

(iii) Each controller has an identifier and can be identified uniquely in the network

(iv) The failure types of the fallible controllers are crash, omission, and Byzantine fault

(v) Each controller has an initial value, and the domain range $D = \{0, 1\}$

(vi) The maximum number of faulty controllers allowed is $n \ge 3fb + fd + 1$, where f_b is the number of Byzantine controllers and f_d is the number of dormant (i.e., crash fault and omission fault) controllers

(vii) A nonfaulty controller does not know the fault status of other controllers.

In this paper, a consensus protocol is designed for SDNs. The proposed consensus protocol can take advantage of the distributed system to let controllers work together to resist attacks from Byzantine controllers. That is, all of the nonfaulty controllers within the networks are able to compute an identical consensus value by using the proposed consensus protocol. We will present the problem to address using the following equations: We can clearly express the objective function as in Equation (1), where we say that $C(n_i)$ is the consensus value computed using n_i. Equation (2) restricts each controller's initial value and says that it comes from the range {0, 1}. Equation (3) constrains the number of allowed faulty controllers in the SDN (we will explain how this quantity limit is derived in "Concept and Approach").

$$C(n_i) = C(n_j), \forall \text{non-faulty controller } n_i, n_j \in N \text{ and } i \ne j, \tag{1}$$

subject to

$$D(n_i) \in \{0, 1\}, \forall n_i \in N, \tag{2}$$

$$n \ge 3f_b + f_d + 1. \tag{3}$$

3. Concept and Approach

Here, we identify our concepts as well as approach for Consensus Protocol for SDN (CP_{SDN}). First, all controllers are able to select the initial value using domain range $D = \{0, 1\}$. As the next step, each controller is examined and allowed to exchange their corresponding initial value with all of the other controllers. After message exchange finishes, all of the controllers compute their own consensus value making use of the collected messages. In other words, there are exactly 2 phases in (CP_{SDN}) which are known as *Msg_Exchanging* as well as *Cons_Making* phase, respectively. Next, we give descriptions for the two phases introduced as follows.

```
vote_SDN Function/ // for each controller n_i ∈ N
1. Begin
2.   if vertex σ is a leaf then — (1)
3.     output val(σ)
4.   else
5.     if #Ω^0(σ) ≥ 3 · (f_b − i + 1) + ⌈(n − 1) mod 3⌉ —— (2)
6.       output the value of σ
7.     if maj_child(σ) = Ω^i, where 1 ≤ i ≤ ⌊(n − 1)/3⌋ − 1 —— (3)
8.       output Ω^(i−1)
9.     if maj_child(σ) = v ——— (4)
10.      output v
11.    if maj_child(σ) = null ———(5)
12.      output ⊥
13.  End
#. val(σ) is the value of vertex σ. Function maj_child(σ) is used to find out the majority value in vertex σ's child nodes.
```

Algorithm 1: The $Vote_{SDN}$ function.

```
/* phase 1: Msg_Exchanging */
1.  crt(root,v_i);
2.  for i = 1 to fb + 1 do
3.    pack(i − 1, msg);
4.    for n_j ∈ N do
5.      send(⟨MSG, n_i, msg⟩, n_j);
7.    wait until (time-out interval)
6.    end
8.      for n_j ∈ N do
9.        if (dormt(n_j) = true) then
10.         package(i − 1, m);
11.         for α ∈ m do
12.           v = Ω^0;
13.           crt(αn_j, v);
14.         end
15.       else if receive ⟨MSG, n_j, msg⟩ from n_j then
16.         unpack(i − 1, msg);
17.         for α ∈ msg do
18.           if v = Ω^i then
19.             v = Ω^(i+1);
20.           else
21.             v = val(α);
22.           crt(αn_j, v);
23.         end
24.       del_rpt(SDN − tree);
25.     end
/* Phase 2: Cons_Making */
26.   if i = fb + 1 do
27.     consensus value = vote_SDN(SDN − tree);
28.   return consensus value;
```

Algorithm 2: The proposed CP_{SDN} protocol (for each controller).

3.1. Msg_Exchanging Phase. To start, each and every controller sends its initial value to every other controller in the SDN. Later, the controller will rely on adequate collaboration (i.e., exchanging the collected initial values from other controllers) to overcome the faults and attacks from a few controllers. Hence, the first work is to compute the number of rounds required in the message exchanging. According to Siu et al. [39], if a message is unauthenticated in the network with m Byzantine processors and b dormant processors, there must be at least $\lfloor (n-1)/3 \rfloor + 2m + b + 1$ processors, and the minimum number of rounds of message exchange is $\lfloor (n-1)/3 \rfloor + 1$. In the proposed protocol, only the controller is involved. Therefore, the constraint of our system model is $n > \lfloor (n-1)/3 \rfloor + 2f_b + f_d$, and the number of rounds of message exchange is at most $f_b + 1$ ($f_b = \lfloor (n-1)/3 \rfloor, f_b$ is the maximum number of Byzantine controllers $n = |N|$).

Next, we will explain how the controller stores the collected messages. As mentioned above, the number of rounds of message exchanging is $f_b + 1$. Besides, in each and every round, the controller will transmit the messages collected in the previous round to all the other controllers in the network. This means the number of messages will grow at a very rapid speed (exponentially). Therefore, using an appropriate data structure to store the collected messages is very important. Bar-Noy et al. [40] pointed out that the tree structure is very suitable for storing data collected by this exchange method. In this paper, we will also store messages collected in the message exchange process in a tree structure, called the SDN-tree. Figure 3 shows a clear example of what is known as a 2-level SDN-tree. The SDN-tree as given can be organized and labelled in the following manner: Each and every vertex of a given SDN-tree is first labelled to be a non-repeating sequence α of the controller identifies. To avoid being strongly influenced by well-known Byzantine controllers, no vertices are kept with repeating names within an SDN-tree. The SDN-tree root is then labelled clearly as "*root*" as well as every parent of a given vertex is then labelled clearly as the sequence αn_i (i.e., α concatenates n_i, where n_i is a single controller name) is labelled α, and the value of vertex of SDN-tree is a value. Figure 3 shows an example of that the parent of the vertex n_2n_1 (i.e., n_2 concatenates n_1) is the vertex n_2. The value stored in the vertex n_2n_1 is "1."

The checksum, as well as the time-out mechanism, may be able to detect messages that are contaminated by any dormant controllers. As such, in our proposed protocol, we

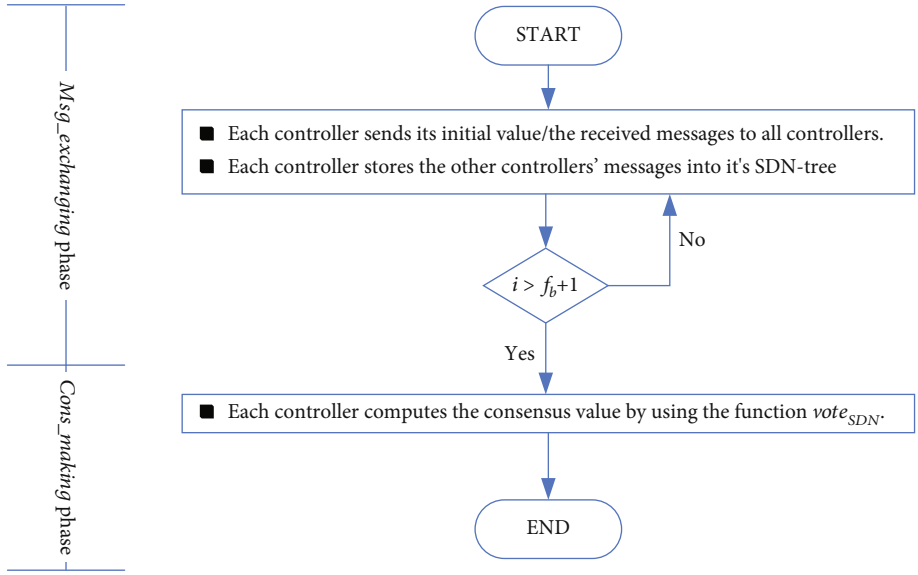

Figure 4: The flow chart of the proposed CP_{SDN} protocol.

Table 2: The initial value of each nonfaulty controller.

	n_1	n_2	n_4
$v(i)$	0	1	0

show that those contaminated messages are clearly marked by "Ω^0." Here, the value "Ω^0" is to be used only for the marking of contaminated messages and only from dormant controllers. So, in the upcoming rounds of the message exchange as defined earlier, any and all messages collected from previous rounds will automatically be exchanged. Then, if and only if the message that is received is one of Ω^j, Ω^{j+1} instead then Ω^j will then be clearly stored so as to represent the contaminated value which came from the previous round. We note here that the main purpose of +1 noted earlier is to be able to show the number of rounds for message exchange that have already passed. This very simple procedure can then aid us to be able to determine if and only if the message exchange has a contaminated value of the preceding controller or even a contaminated value for the current controller.

3.2. Cons_Making Phase. Here, we describe what is known as the *Cons_Making* phase. This is known as where a consensus value can be computed. So, to describe the *Cons_Making* phase, we start by noting that the function called $vote_{SDN}$ is first used for the computation of consensus value starting at the root in SDN-tree. We work here from the furthest leaves all the way to root. The function $vote_{SDN}$ contains 5 conditions that can be presented in the following manner: If and only if vertex α can be defined as a leaf, then and only then there will be only 1 value within vertex α. Therefore, majority of the value then can be defined as value of vertex α (this is Condition 1); Condition 2 can be defined as when we used to be able to remove any influence from the well-known Byzantine controllers; Condition 3 then will be used for the removal of any influence that may arise from the dormant controllers; Condition 4 then will only be used if and only if we get a majority value. If and only if majority value exists, then and only then the output for default value $\perp$ exists, where we can say that $\perp \notin V$ (Condition 5). Conditions 1, 4, and 5 bear similarities to the well-known majority voting convention [32]. The function $vote_{SDN}$ can be clearly seen in Algorithm 1.

3.3. The Pseudo Code of CP_{SDN}. The CP_{SDN} protocol pseudocode is clearly given in Algorithm 2. We define the involved functions in the protocol as follows:

(i) $crt(\alpha p, v)$: create vertex αp, set $val(\alpha p) = v$

(ii) $dormt(n_j)$: check controller n_j is dormant controller

(iii) $send(\langle \text{MSG}, n_j, msg\rangle, n_k)$: send MSG message with value msg as proposed by controller n_j to n_k

(iv) $pack(i, msg)$: structure of level i of SDN-tree, pack level i of SDN-tree with message msg

(v) $unpack(i, msg)$: structure of level i of SDN-tree, unpack message msg

(vi) $del_{rpt}(\text{SDN} - \text{tree})$: delete vertices with duplicate signs in SDN-tree

(vii) $vote_{SDN}(\text{SDN} - \text{tree})$: input leaves of SDN-tree and compute consensus value with accordance with tree structure of SDN-tree.

4. An Execution Example

The flowchart of the proposed CP_{SDN} is shown in Figure 4. An example in this section will be presented that is able to show via demonstrating the manner in which CP_{SDN} helps controllers achieve consensus. We assume here that the

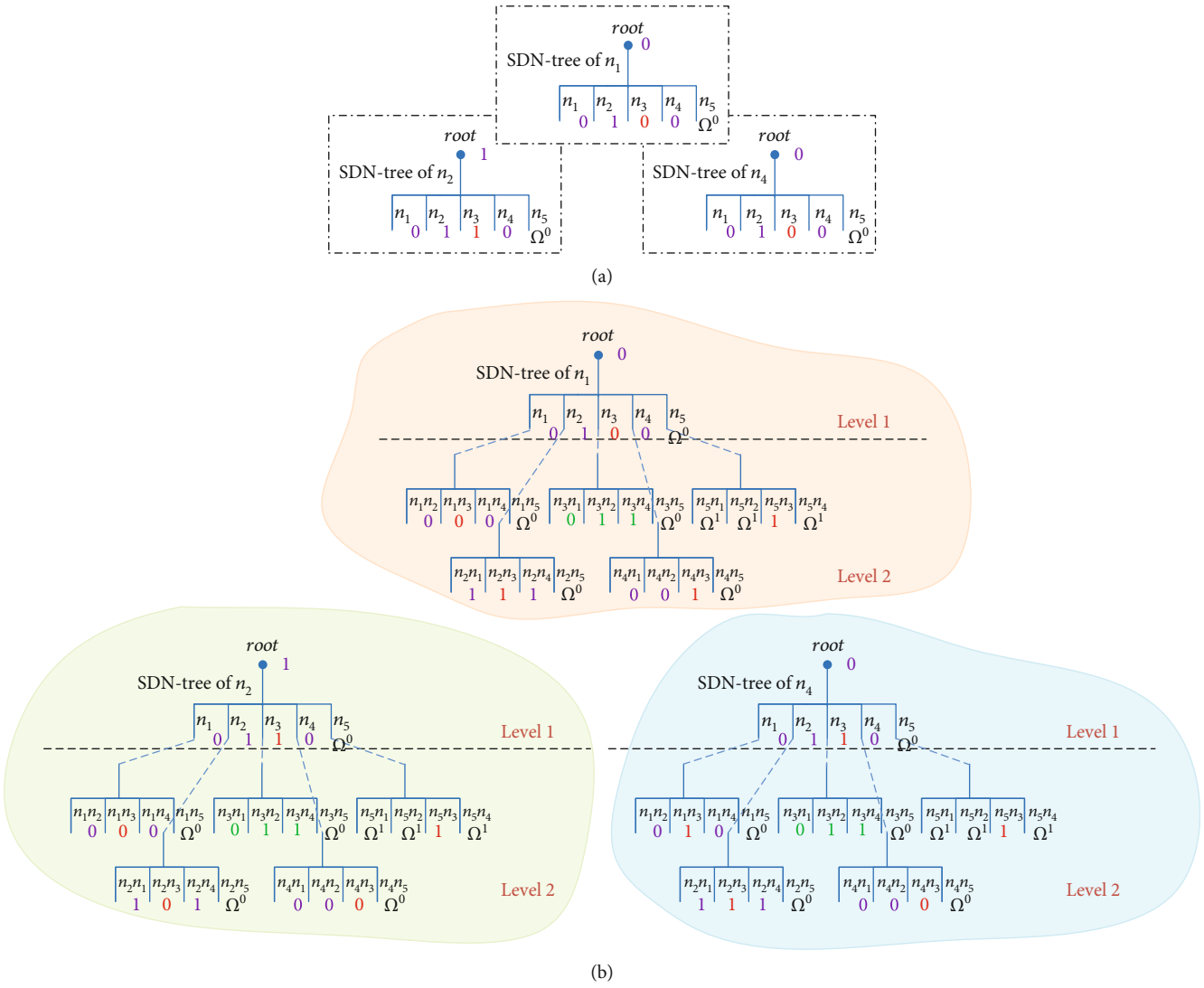

Figure 5: An example of execution CP_{SDN}. (a) The SDN-trees of nonfaulty controllers n_1, n_2, and n_4 after the 1st of *Msg_Exchanging* phase. (b) The SDN-trees of nonfaulty controllers n_1, n_2, and n_4 after the 2nd of *Msg_Exchanging* phase.

Table 3: The process of computing the consensus value by $vote_{SDN}$ *function.*

$vote_{SDN}(n_1) = vote_{SDN}(n_1n_2, n_1n_3, n_1n_4, n_1n_5)$
(i) $vote_{SDN}(n_1) = vote_{SDN}(0,0,0,\Omega^0) = 0$
✓ $vote_{SDN}(n_2) = vote_{SDN}(n_2n_1, n_2n_3, n_2n_4, n_2n_5)$
(ii) $vote_{SDN}(n_2) = vote_{SDN}(1,1,1,\Omega^0) = 1$
✓ $vote_{SDN}(n_3) = vote_{SDN}(n_3n_1, n_3n_2, n_3n_4, n_3n_5)$
(iii) $vote_{SDN}(n_3) = vote_{SDN}(0,1,1,\Omega^0) = 1$
✓ $vote_{SDN}(n_4) = vote_{SDN}(n_4n_1, n_4n_2, n_4n_3, n_4n_5)$
(iv) $vote_{SDN}(n_4) = vote_{SDN}(0,0,1,\Omega^0) = 0$
✓ $vote_{SDN}(n_5) = vote_{SDN}(n_5n_1, n_5n_2, n_5n_3, n_5n_4)$
(v) $vote_{SDN}(n_5) = vote_{SDN}(\Omega^1, \Omega^1, 1, \Omega^1) = \Omega^0$

network consists of five controllers, which includes $N = \{n_1, n_2, n_3, n_4, n_5\}$. Among these controllers, controller n_3 is a Byzantine controller, and controller n_5 is a dormant controller. Table 2 shows the initial value of each nonfaulty controller.

In the first round of the *Msg_Exchanging* phase, each controller will send its initial value to all other controllers in the network. When nonfaulty controller receives the initial values from other controllers, it will store the received initial values in level 1 of its SDN-tree. Controller n_5 is a dormant controller. Under the operation of error checking codes and the time-out mechanism, it will be detected (as shown in Figure 5(a)). On the other hand, due to the controller n_3 is a Byzantine faulty controller, it may send inconsistent initial value to other controllers in the network. As shown in Figure 5(a), some controllers receive 0 as the initial value of Controller n_3, and some receive 1.

Next, the controllers enter the second round of the *Msg_Exchanging* phase. In the second round, controllers will

exchange the messages collected in the first round with each other. Byzantine controller n_3 is likely to continue interfering (i.e., sends arbitrary values to nonfaulty controllers). On the other hand, when forwarding the value received from a dormant controller, nonfaulty controllers will mark the value by Ω^i. The SDN-trees of nonfaulty controllers n_1, n_2, and n_4 after the 2nd of *Msg_Exchanging* phase are shown in Figure 5(b).

In this example, the number of rounds required in the *Msg_Exchanging* phase is 2 ($f_b + 1 = \lfloor (n-1)/3 \rfloor + 1 = 2$). Hence, after two rounds of message exchange, each nonfaulty controller will enter the *Cons_Making* phase. In the *Cons_Making* phase, each nonfaulty controller will use the $vote_{SDN}$ function to compute the consensus value. In this example, the consensus value computed by the nonfaulty controllers is "$\perp$" ($vote_{SDN}(root) = vote_{SDN}(0,1,1,0,\Omega^0) = \perp$). The process of computing the consensus value by $vote_{SDN}$ function is shown in Table 3.

5. Conclusion

A distributed control structure involving the collaboration of multiple controllers can effectively enhance the performance and stability of SDNs. However, attacks and threats are prevalent in today's networking environment. SDN controllers may not work normally when it is encountered by hackers or virus infection. To provide a reliable SDN with a distributed control architecture, we need a mechanism that ensures the correctness of the computing results even if any controller has a fault or is under attack. Hence, we discussed the fault-tolerant consensus problem in the SDN with distributed control architecture. With the proposed CP_{SDN} protocol, if the number of controllers n is greater than $\lfloor (n-1)/3 \rfloor + 2f_b + f_d$, we can ensure that the nonfaulty controller can reach the common consensus value after $f_b + 1$ rounds of message exchange. For some applications that require very high reliability, reaching a consensus is not enough. Hence, we must consider another related problem, called the fault diagnosis problem. This will be the direction of our future research in SDNs.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] W. Fang, F. Xue, Y. Ding, N. Xiong, and V. Leung, "EdgeKE: an on-demand deep learning IoT system for cognitive big data on industrial edge devices," *IEEE Transactions on Industrial Informatics*, 2020.

[2] M. Gheisari, Q. Pham, M. Alazab, X. Zhang, C. Fernández-Campusano, and G. Srivastava, "ECA: an edge computing architecture for privacy-preserving in IoT-based smart city," *IEEE Access*, vol. 7, pp. 155779–155786, 2019.

[3] H. Cheng, Y. Shi, L. Wu, Y. Guo, and N. Xion, "An intelligent scheme for big data recovery in Internet of Things based on multi-attribute assistance and extremely randomized trees," *Information Sciences*, vol. 557, pp. 66–83, 2021.

[4] J. C. W. Lin, G. Srivastava, Y. Zhang, Y. Djenouri, and M. Aloqaily, "Privacy preserving multi-objective sanitization model in 6G IoT environments," *IEEE Internet of Things Journal*, 2020.

[5] F. Naeem, G. Srivastava, and M. Tariq, "A software defined network based fuzzy normalized neural adaptive multipath congestion control for the internet of things," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2155–2164, 2020.

[6] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an SDN platform for cloud network services," *IEEE Communications and Magazine*, vol. 51, no. 2, pp. 120–127, 2013.

[7] S. Peng, B. Guo, C. Jackson et al., "Multi-tenant software-defined hybrid optical switched data centre," *Journal of Lightwave Technology*, vol. 33, no. 15, pp. 3224–3233, 2015.

[8] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.

[9] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: a survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[10] W. Zhou, L. Li, and W. Chou, "SDN northbound REST API with efficient caches," in *Proceeding of the 2014 IEEE International Conference on Web Services*, pp. 257–264, Anchorage, USA, 2014.

[11] Y. Zhang, X. Gong, Y. Hu, W. Wang, and X. Que, "SDNMP: enabling SDN management using traditional NMS," in *Proceeding of the 2015 IEEE International Conference on Communication Workshop*, pp. 357–362, London, UK, 2015.

[12] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, G. Srivastava, S. Mohan, and A. M. Rababah, "Cost optimization of secure routing with untrusted devices in software defined networking," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 36–46, 2020.

[13] *Open Networking Foundation*, 2021, https://www.opennetworking.org/.

[14] Open Networking Foundation, *Software-Defined Networking: the new norm for networks*, white paper, 2012, 2021, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[15] *OpenDaylight*, 2021, https://www.opendaylight.org/.

[16] *OpenDaylight Platform Architecture*, 2021, https://www.opendaylight.org/lithium.

[17] Z. Guo, Y. Shen, A. K. Bashir et al., "Robust spammer detection using collaborative neural network in internet of thing applications," *IEEE Internet of Things Journal*, 2020.

[18] N. Shi, L. Tan, W. Li, X. Qi, and K. Yu, "A blockchain-empowered AAA scheme in the large-scale HetNet," *Digital Communications and Networks*, 2020.

[19] K. Yu, L. Tan, X. Shang, J. Huang, G. Srivastava, and P. Chatterjee, "Efficient and privacy-preserving medical research support platform against COVID-19: a blockchain-based approach," *IEEE Consumer Electronics Magazine*, vol. 10, no. 2, pp. 111–120, 2021.

[20] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in

Proceeding of the 2009 ACM workshop on Research on enterprise networking, pp. 11–18, Barcelona, Spain, 2009.

[21] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceeding of the 2012 ACM International Workshop on Hot Topics in Software Defined Networks*, pp. 121–126, Helsinki, Finland, 2012.

[22] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with OpenFlow/NOX architecture," in *Proceeding of the 2011 IEEE International Conference on Network Protocols*, pp. 7–12, Vancouver, Canada, 2011.

[23] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[24] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "A load-balancing mechanism for distributed SDN control plane using response time," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1197–1206, 2018.

[25] Y. Kyung, K. Hong, T. M. Nguyen, S. Park, and J. Park, "A load distribution scheme over multiple controllers for scalable SDN," in *Proceeding of the 2015 International Conference on Ubiquitous and Future Networks*, pp. 808–810, Sapporo, Japan, 2015.

[26] G. Wang, Z. Zhao, J. Peng, R. Li, and H. Zhang, "An approximate algorithm of controller configuration in multi-domain SDN architecture," in *Proceeding of the 2014 International Conference on Communications and Networking in China*, pp. 601–605, Maoming, China, 2014.

[27] H. Yannan, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *China Communications*, vol. 11, no. 2, pp. 38–54, 2014.

[28] Y. C. Chan, K. Wang, and Y. H. Hsu, "Fast controller failover for multi-domain software-defined networks," in *Proceeding of the 2015 European Conference on Networks and Communications*, pp. 370–374, Paris, France, 2015.

[29] S. Jafar, A. Krings, and T. Gautier, "Flexible rollback recovery in dynamic heterogeneous grid computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 1, pp. 32–44, 2009.

[30] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[31] J. Lembke, S. Ravi, P. Eugster, and S. Schmid, "RoSCo: robust updates for software-defined networks," *The IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1352–1365, 2020.

[32] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[33] V. King, J. Saia, V. Sanwalani, and E. Vee, "Towards secure and scalable computation in peer-to-peer networks," in *Proceeding of the 2006 Annual IEEE Symposium on Foundations of Computer Science*, pp. 87–98, Berkeley, CA, USA, 2006.

[34] C. F. Cheng and H. C. Liao, "The k-set consensus problem with weight consideration," *The Journal of Supercomputing*, vol. 71, no. 1, pp. 144–161, 2015.

[35] M. Correia, A. N. Bessani, and P. Verissimo, "On Byzantine generals with alternative plans," *The Journal of Parallel and Distributed Computing*, vol. 68, no. 9, pp. 1291–1296, 2008.

[36] F. Shi, X. Tuo, L. Ran, Z. Ren, and S. X. Yang, "Fast convergence time synchronization in wireless sensor networks based on average consensus," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1120–1129, 2020.

[37] H. Wang, L. Chen, M. Li, and P. Gong, "Consensus-based clock synchronization in wireless sensor networks with truncated exponential delays," *IEEE Transactions on Signal Processing*, vol. 68, pp. 1425–1438, 2020.

[38] C. F. Cheng and K. T. Tsai, "Eventual strong consensus with fault detection in the presence of dual failure mode on processors under dynamic networks," *The Journal of Network and Computer Applications*, vol. 35, no. 4, pp. 1260–1276, 2012.

[39] H. S. Siu, Y. H. Chin, and W. P. Yang, "A note on consensus on dual failure modes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, pp. 225–230, 1996.

[40] A. Bar-Noy, D. Dolev, C. Dwork, and H. Raymond Strong, "Shifting gears: changing algorithms on the fly to expedite Byzantine agreement," *Information and Computation*, vol. 97, no. 2, pp. 205–233, 1992.