Byzantine-Resilient Secure Software-Defined Networks with Multiple Controllers in Cloud

He Li, *Student Member, IEEE,* Peng Li, *Member, IEEE,*
Song Guo, *Senior Member, IEEE,* and Amiya Nayak, *Senior Member, IEEE*

Abstract—Software-defined network (SDN) is the next generation of networking architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. In SDN, network management is facilitated through software rather than low-level device configurations. However, the centralized control plane introduced by SDN imposes a great challenge for the network security. In this paper, we present a secure SDN structure, in which each device is managed by multiple controllers, not just a single as in a traditional manner, with the dynamic and isolated instance provided by the cloud. It can resist Byzantine attacks on controllers and the communication links between controllers and SDN switches. Furthermore, we study a controller minimization problem with security requirement and propose a cost-efficient controller assignment algorithm with a constant approximation ratio. From the experiment result, the secure SDN structure has little impact on the network latency, provide better security than general distributed controller, and the proposed algorithm performs higher efficiency than random assignment.

Index Terms—Software-Defined Network, Byzantine Attack, Cloud Computing, Approximation Algorithm

1 Introduction

Software-defined network (SDN) is a promising network paradigm that separates the control plane and data plane in networks such that switches become simple data forwarding devices and network management is controlled by logically centralized servers [1], [2], [3], [4]. It has shown great advantages in simplifying network management such that network administrators have central programmable control of network traffic via controllers, and new functions can be easily supported without physical access to the network switches [5], [6], [7].

Although SDN can significantly improve network applicability and efficiency, it is exposed to new threats that are more serious than those in traditional networks, where an attack is made in a cooperative manner to multiple switches if they are distributed in different regions, and probably protected by different organizations [8]. However, the attack becomes much easier in SDN since a malicious controller could compromise the entire network.

An intuitive method to enhance security of SDN is to employ multiple controllers for each switch [9]. When a controller is disabled because of attacks, a backup one can be immediately activated to take over the controlling function. Unfortunately, such a method can only deal with attacks that halt controllers. In practice, there are other attacks, e.g., to forge or temper the commands issued by controllers, which imposes the requirement for a more secure mechanism to protect controllers in SDN.

Byzantine fault tolerant (BFT) technology [10] has been proposed to defend against Byzantine failures, in which components of a system fail in arbitrary ways. While the existing BFT solutions are suited for the file system environment, there are some problems to apply BFT to the SDN network. Unlike the file system, security requirement of each switch is different because many switches may not forward important packets. For example, in data center networks, since the core switches forward network packets from the entire network, they are much more important than the leaf switches which forward packets for less nodes. With different routing, the security requirements for sensitive data forwarding at each switch are also different. As a result, controller assignment should be dynamic and on demand. Meanwhile, since the BFT mechanism usually requires the independence of failures among replicas, it needs isolated environments with different operation systems and applications. A cloud environment can easily provide a dynamic environment and various instances to deploy Byzantine mechanism [11], [12].

In this paper, we design a novel SDN architecture to resist the attack on the control plane by BFT mechanism in Cloud. Different from the traditional SDN architecture that each switch is controlled by a single controller [13], we propose to use multiple controllers to confirm the update of flow tables in each switch. Specifically, we apply the BFT mechanism to guarantee that each switch can correctly update its forwarding tables even some compromised controllers issue false instructions.

Based on the proposed architecture, we then study how to assign controllers to a set of switches such that their security requirements are satisfied. This assignment problem has two challenges. First, each switch may

- *H. Li, P. Li and S. Guo are with the School of Computer Science and Engineering, The University of Aizu, Japan. E-mail: {d8141105, pengli, sguo}@u-aizu.ac.jp*
- *A. Nayak is with the School of Information Technology and Engineering, University of Ottawa, Canada. E-mail: anayak@site.uottawa.ca*

require a different number of controllers, and each controller can provide services to multiple switches. Considering the cost for deploying controllers to commercial servers, we need to minimize the number of controllers. Second, as the Byzantine mechanism would incur frequent message exchanges among the set of controllers associated with the same switch, its performance highly depends on the link latency among these controllers. When we assign controllers for a switch, the maximum latency among these controllers must be guaranteed within a threshold.

To evaluate the performance of our proposal, we implement a prototype on the emulator and the cloud instances to test the network latency brought by the BFT mechanism, which is the most concerned issue to the network performance. The results show that the overhead brought by BFT is affordable. To evaluate the security, we also design an experiment to simulate the network connection when controllers under attacks. From this simulation, we find that the Byzantine-resilient secure SDN network shows much better stability than normal distributed controller network.

The main contributions of this paper are summarized as follows.

- First, we propose a secure SDN architecture, in which each switch is controlled by multiple controllers in cloud using Byzantine mechanism.
- Second, we study a controller assignment problem to minimize the number of employed controllers while satisfying the security requirement of each switch, in terms of the required number of associated controllers and the maximum latency among them. We propose an efficient algorithm to solve the controller assignment with a good result ratio with the optimal assignment.
- Finally, we evaluate our work with experiments on a prototype and simulations. The results of network latency on the prototype show that the BFT mechanism does not impact the network performance seriously. Extensive simulations are conducted and the results show that the security of the proposed structure is better than ordinary distributed structure and the performance of the proposed algorithm can significantly reduce the number of controllers.

The rest of this paper is summarized as follows. Section II reviews the related work. Our network model and threat model are introduced in Section III. Section IV presents the system design and problem formulation. An efficient algorithm is proposed in Section V. Section VI gives the simulation results. Finally, Section VII concludes this paper.

2 Related Work

2.1 Software defined network

The concept of SDN stems from the research of the network operating system that provides a uniform and centralized programmatic interface to the entire network. As the first attempt of building a network operating system at a large scale, NOX [14] achieves a simple programming model for control function based on OpenFlow. Later, Maestro [15] exploits parallelism with additional throughput optimization techniques while keeping the simple programming model for programmers.

FlowVisor [16] is the first testbed for SDN, which slices the network hardware by placing a layer between control plane and the data plane. Its basic idea is that if unmodified hardware supports some basic primitives, then a worldwide testbed can ride on the coat-tails of deployments without an extra expense.

Casado et al. [17] introduce an idea of designing a new network-wide software layer that exposes one or more logical forwarding elements. Instead of interfacing directly to the networking hardware, the control software reads and writes to these logical forwarding elements. This approach allows the network state to be largely decoupled from the underlying hardware, paving the way for migration, failure resilience, and more complex state management.

Recently, a slice abstraction that can easily isolate network programs from each other is proposed in [18]. Efficient algorithms are designed for compiling slices to OpenFlow switches, and they are evaluated on a prototype implementation.

A structure with distributed multi-domain SDN controller is the same with the structure with multiple SDN controllers[19]. Even though it introduces a communication model with multiple controllers, this structure is used for cooperation of controllers in different network domains rather than improving the security of the control plane in a single network.

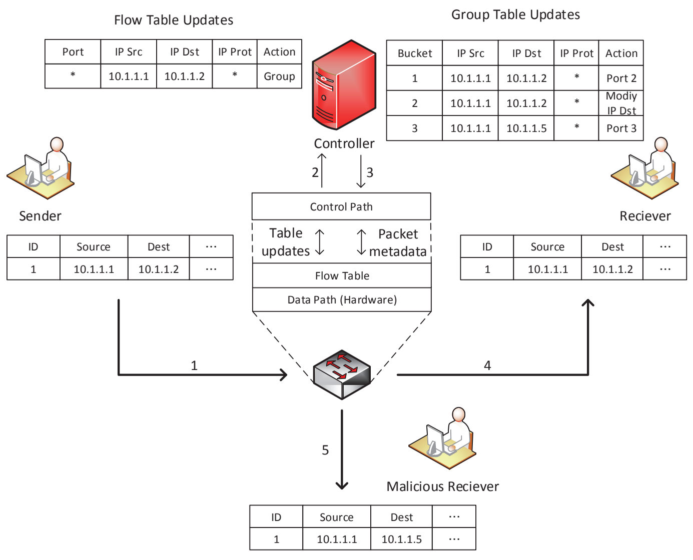
2.2 Security issues in SDN

The security issues in SDN also attract lots of attentions.

FortNOX [20] is a software extension that provides role-based authorization and security constraint enforcement for the NOX OpenFlow controller. FortNOX enables NOX to check flow rule contradictions in real time, and implements a novel analysis algorithm that is robust even in cases when an adversarial OpenFlow application attempts to strategically insert flow rules that would otherwise circumvent flow rules imposed by OpenFlow security applications.

FRESCO [21] is proposed to address several key issues that can accelerate the composition of new OpenFlow-enabled security services. It exports a scripting API that enables security practitioners to code security monitoring and threat detection logic as modular libraries. CORONET [22] is an SDN fault-tolerant system that recovers from multiple link failures in the data plane. However, it cannot deal with the attacks on the control plane.

Reiblatt et al. [23] present a new language, called FatTire, for writing fault-tolerant network programs. The central feature of this language is a programming



Flow Table Updates

Port	IP Src	IP Dst	IP Prot	Action
*	10.1.1.1	10.1.1.2	*	Group

Group Table Updates

Bucket	IP Src	IP Dst	IP Prot	Action
1	10.1.1.1	10.1.1.2	*	Port 2
2	10.1.1.1	10.1.1.2	*	Modiy IP Dst
3	10.1.1.1	10.1.1.5	*	Port 3

ID	Source	Dest	···
1	10.1.1.1	10.1.1.2	···

ID	Source	Dest	···
1	10.1.1.1	10.1.1.2	···

ID	Source	Dest	···
1	10.1.1.1	10.1.1.5	···

Fig. 1. SDN operations managed by a single controller

construct based on regular expressions, that allows developers to specify the set of paths that packets may take through the network as well as the degree of fault tolerance required.

Sezer et al. [24] give a brief summary of the challenges in SDN, including several security issues. Kreutz et al. [8] list six threat vectors that may enable the exploit of SDN vulnerabilities, and sketch the design of a secure and dependable SDN control platform. However, they do not provide any detailed solutions for specific security problem in SDN.

Mailik et al. [25] proposed a framework that delegates the SDN network controls to the cloud. With their work, users can define their packet forwarding path with the controllers in a Infrastructure as a Servic (IaaS) cloud. Even though they illustrate the tradeoffs bettwen security and the level of network abstractions provided to users, the security issue of the controller is not solved.

Among the main existing solutions focus on the security issue, few of them consider the potential security threat in centralized structure [3], in which the logically centralized methodology is hard to ensure the security of the data plane. In the our previous work [26], this challenge has bee briefly tackled and a simple solution, with no performance guarantee, been proposed to the minimum controller assignment problem.

3 Background and Motivation

3.1 Software defined network

In a typical network architecture, each network device (e.g., router or switch) consists of a control plane and a data plane, where the control plane is responsible for device configuration and routing management, and the data plane forwards data traffic according to the rules defined by the control plane. Traditionally, both control and data planes are integrated in the firmware of network devices. Once the device configuration or the routing strategy needs to be changed, we have to modify the firmware of all involved devices, which would incur a high labor cost and take a long time, especially when there are lots of devices distributed in a large region.

Software defined network (SDN) decouples the control and data planes, and implements the control plane in software instead, which enables programmatic access to make network administration much more flexible. In SDN, the data plane in each device forwards data traffic according to a set of rules specified by the control plane that is implemented in a remote server called the controller. As an example shown in Fig. 1, we consider a network flow through a switch from a source with address 10.1.1.1 to the destination 10.1.1.2. When a packet of this flow arrives the switch, the switch searches a forwarding rule among the ones stored in its local cache for this packet according to its source and destination addresses. If a rule is found successfully, the data plane forwards the packet according to the action defined by the rule such that the packet can eventually arrive the destination. Otherwise, the packet is forwarded to the controller that executes the routing algorithm, and adds a new forwarding entry to the flow table in the switch.

3.2 Threats in the control plane

The controllers are not always safe when they provide services to the network applications. It is needed to allow users some access to these controllers for deploying new applications, monitoring the network status, and so on. Therefore, it is possible that the controller is accessed by malicious users who may insert some unsafe network applications or operate the rule space directly.

As shown in Fig. 1, the control planes of multiple devices are moved into a centralized controller residing in general servers in SDN. Although such a novel network paradigm shows great advantages in simplifying network management, it is exposed to new threats targeted on the control plane that a malicious adversary can compromise the controller to dominate the entire network by modifying the flow tables on network devices. For example, the compromised controller in Fig. 1 can set up rules to modify the destination address of the packets from the source 10.1.1.1 such that all data are forwarded to a malicious receiver 10.1.1.5. To protect SDN from such a kind of threats to the control plane, we need to deal with the challenge of letting the controllers continue operating correctly, even if some of them exhibit arbitrary, possibly malicious behavior. Obviously, simply setting up backup controller is not enough to eliminate such threats because they are activated only when the crash of primary controllers is detected.

3.3 Control plane on cloud with BFT

Byzantine Fault-Tolerance (BFT) provides a powerful state machine replication approach for providing highly reliable and consistent services in spite of the presence of

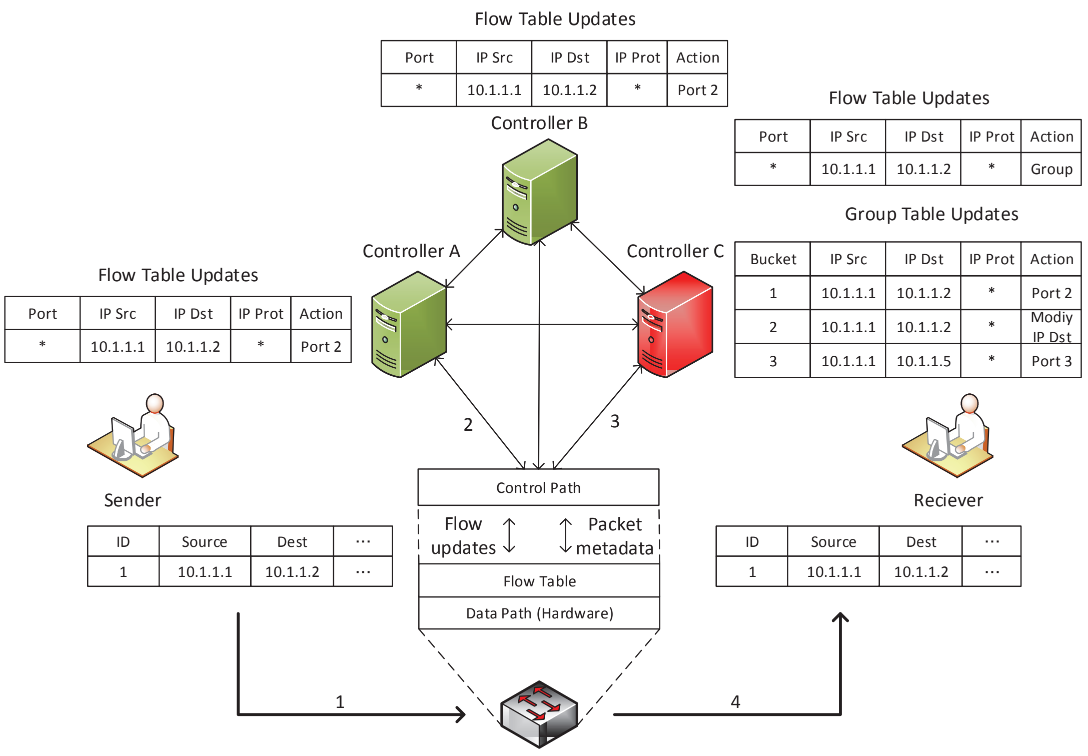

Flow Table Updates
Port | IP Src | IP Dst | IP Prot | Action
* | 10.1.1.1 | 10.1.1.2 | * | Port 2
Controller B
Flow Table Updates
Port | IP Src | IP Dst | IP Prot | Action
* | 10.1.1.1 | 10.1.1.2 | * | Group
Group Table Updates
Bucket | IP Src | IP Dst | IP Prot | Action
1 | 10.1.1.1 | 10.1.1.2 | * | Port 2
2 | 10.1.1.1 | 10.1.1.2 | * | Modiy IP Dst
3 | 10.1.1.1 | 10.1.1.5 | * | Port 3
Controller A
Controller C
Flow Table Updates
Port | IP Src | IP Dst | IP Prot | Action
* | 10.1.1.1 | 10.1.1.2 | * | Port 2
2
3
Sender
Reciever
ID | Source | Dest | ...
1 | 10.1.1.1 | 10.1.1.2 | ...
Control Path
Flow updates
Packet metadata
Flow Table
Data Path (Hardware)
1
4

Fig. 2. SDN operations managed by multiple controllers

failures. When we apply BFT to resist the attacks on the control plane, each switch is connected to n controllers that run the BFT protocol, such as those in [27], [28], to form a "big" fault-free controller that can tolerate up to f faulty controllers. The relationship between n and f will be different under different BFT protocols. For example, $n \geq 3f + 1$ is guaranteed by the PBFT protocol [27], and MinZyzzyva [28] improves the performance with $n \geq 2f + 1$.

We use the example in Fig. 2 to show the benefit of BFT, in which a switch is connected to three controllers A, B, and C. Even though the controller C is compromised to issue faulty commands to modify the flow tables, this event can be detected by other two controllers that guarantee correct flow table update at the switch.

For a better understanding of BFT, we briefly describe the PBFT that is the first practical BFT algorithm. It is a replication algorithm in asynchronous environments based on a Byzantine failure model with independent node failures. It assumes that the adversary cannot delay correct nodes indefinitely or subvert the cryptographic techniques.

Sevices are replicated and executed on independent replicas. The replicas move through a succession of configurations called views and one replica is the primary in a view. When the primary fails, the view will change. The algorithm works briefly as follows.

1 The client sends a request to the primary to require a service operation.
2 The primary sends the request to other replicas.
3 Replicas execute the request and send a reply to the client.
4 The client waits for $f + 1$ replies from different replicas with same result.

When we apply BFT to a general SDN with multiple switches, we note that cloud is very suitable for BFT mechanism as follows. First, a large number of replicas are needed to satisfy the redundant requirement of BFT. For example, m identical switches under PBFT with $f = 1$ need at least $4m$ controllers. To ensure the validity of BFT mechanism, replicas should run isolated environments with different operation systems and controller implementations. IaaS cloud environment can easily meet such requirement, because controllers are isolated into different instances. Second, the number of controllers needed by each switch varies because of different fault-tolerant requirements. These requirements may also change over time. With flexible resource provision, cloud becomes a perfect platform for controller assignment under such a dynamic environment. Finally, cloud can provide isolation among controllers belonging to different users by accommodating them into virtual machines.

Above facts motivate us to move controllers into the cloud that is a perfect platform for a cost-efficient and flexible resource provision. Further, using the cloud service to provide a secure and on-demand SDN controller service is an emerging approach [30].

4 The CAFTS problem

In this section, we study the statement of the problem of controller assignment in fault-tolerant SDN (CAFTS).

TABLE 1
Notations in the CAFTS problem

Notation	Description
S	Set of all switches
s_i	Switch i
C	Set of all controllers
c_i	Controller i
B_i	Required controllers of s_i
Φ_i	Assigned controllers for s_i
L_{jk}	Latency between c_j and c_k
δ_i	Threshold of the communication latency between any two controllers associated with s_i
Ψ_j	Set of switches associated with c_j
$\mathcal{N}$	Total number of controllers employed to satisfy the requirements of all switches

After that, we analyze the hardness of the CAFTS problem. The notations used in the statement of the CAFTS problem are listed in Table 1.

4.1 Problem statement

We consider a two-layer SDN as shown in Fig. 3. The lower layer is the data plane consisting of a set $S = \{s_1, s_2, ..., s_m\}$ of switches, each of which has a cache with limited capacity to store flow tables. The upper layer is the control plane residing in a data center with a set $C = \{c_1, c_2, ..., c_n\}$ of identical severs referred to as controllers in this paper. The communication latency between any two controllers c_i and c_j is denoted by L_{ij}.

To deal with the threats in the control plane, each switch $s_i \in S$ is assigned a set Φ_i of controllers that run BFT protocol to achieve consistency about flow table updating. The BFT protocol imposes two requirements for the controller assignment. First, the number of controllers in Φ_i should be at least B_i, i.e.

$$|\Phi_i| \geq B_i, \forall s_i \in S. \tag{1}$$

The value of B_i is determined by c_i's tolerance of faulty controllers and the BFT algorithm. For example, if a switch can tolerance at most 2 faulty controllers and the PBFT algorithm [27] with $n \geq 3f + 1$ is applied, we need to assign at least 7 controllers to this switch. Second, the controllers associated with a switch need to frequently exchange messages in the BFT protocol. In order to guarantee a certain level of performance, the communication latency between any two controllers associated with switch s_i should not exceed a threshold δ_i, i.e.,

$$L_{jk} \leq \delta_i, \forall s_i \in S, \forall c_j \in \Phi_i, c_k \in \Phi_i. \tag{2}$$

We let Ψ_j denote the set of switches associated with controller c_j. Since each controller can support at most K switches due to resource limit (such as CPU and memory), we have:

$$|\Psi_j| \leq K, \forall c_j \in C. \tag{3}$$

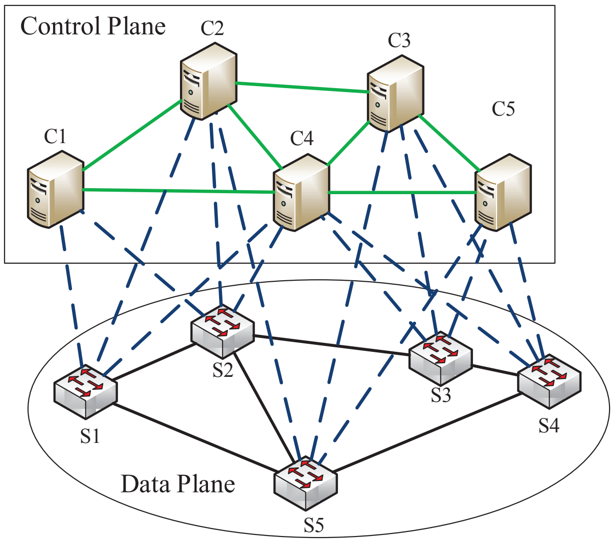


Fig. 3. System model

The total number of controllers employed to satisfy the requirements of all switches is denoted by $\mathcal{N}$, which can be calculated by:

$$\mathcal{N} = |\cup_{i=1}^{m} \Phi_i|. \tag{4}$$

The problem of controller assignment in fault-tolerant SDN (CAFTS): given a set of controllers and a set of switches, the CAFTS problem attempts to assign the minimum number of controllers to these switches such that BFT protocol can be applied to eliminate the threats in the control plane.

4.2 Hardness analysis

Theorem 1: The controller assignment problem is NP-hard.

Proof: We prove the NP-hardness of the controller assignment problem by reducing the well-known bin-packing problem defined as follows.

The bin-packing problem: given a set of items $\{a_1, a_2, ...a_m\}$, each item a_i with a size b_i, and a number of n bins of size W, is there a bin-packing scheme such that all items can be accommodated into n bins?

As shown in Fig. 4, for each item a_i of size b_i, we create a switch s_i that requires $B_i = b_i$ controllers, among which the maximum latency should be no greater than δ, to achieve a certain level of security. For each bin, we create a group of W controllers with capacity 1, i.e., each controller can provide service for at most one switch. The latency between any two controllers within the same group is no greater δ, and the links across groups have latency that is greater than δ.

We first suppose a solution that the bin-packing problem that all items can be accommodated into n bins. In the corresponding solution of the controller assignment problem, we choose B_i controllers of the same group for each switch s_i, and the total number of assigned controller in each group is less than W.

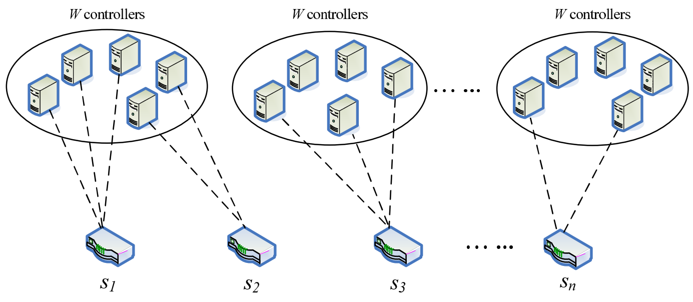


Fig. 4. An instance of the CAFTS problem

We then suppose that the controller assignment problem has a solution that each switch s_i is assigned a set of B_i controllers. Since the latency between different groups of controllers is greater than δ, the set of controllers assigned to each switch should belong to the same group, which forms a solution of the bin-packing problem.

It is easy to see that the controller assignment problem is in NP class as the objective function associated with a given solution can be evaluated in a polynomial time. Thus, we conclude that the controller assignment problem is NP-hard. □

5 Solving the CAFTS problem

5.1 Algorithm design

In this section, we propose an algorithm, called requirement first assignment (RQFA) algorithm, to solve the CAFTS problem. Its basic idea is to iteratively assign controllers to switches that are sorted according to their required number of controllers in a descending order. We always keep a set of controllers as candidates for assignment. For each switch s_i, if it can be accommodated by these candidates, i.e., B_i and δ_i can be satisfied, we assign the candidates with minimum residual capacity to it. Otherwise, we find a set of new candidates that can accommodate this switch.

The pseudo code of the proposed algorithm is shown in Algorithm 1. Without loss of generality, we assume that the switches in set $S = \{s_1, s_2, ..., s_m\}$ are sorted such that $B_1 \geq B_2 \geq ... \geq B_m$. The set of controllers as candidates are maintained in set C' that is initialized to be empty. For each switch s_i, if there are not enough controllers with non-zero residual capacity in current candidate set, we update set C' by finding a new set of B_i candidates using function FIND_NEXT_SET whose code is shown in Algorithm 2. Note that the communication latency between any two controllers in this new candidate set should be no greater than δ_{min} as calculated in line 4. In the following, we assign controllers in the candidate set to switch s_i. Specifically, we always give priority to the controllers with less residual capacity in the assignment, which is achieved by sorting the controllers in C' according to their residual capacity maintained in k_{Π_j}. Finally, we update the value of k_{Π_j} as well as the sets C, C' and S.

Algorithm 1 The main procedure of requirement first assignment

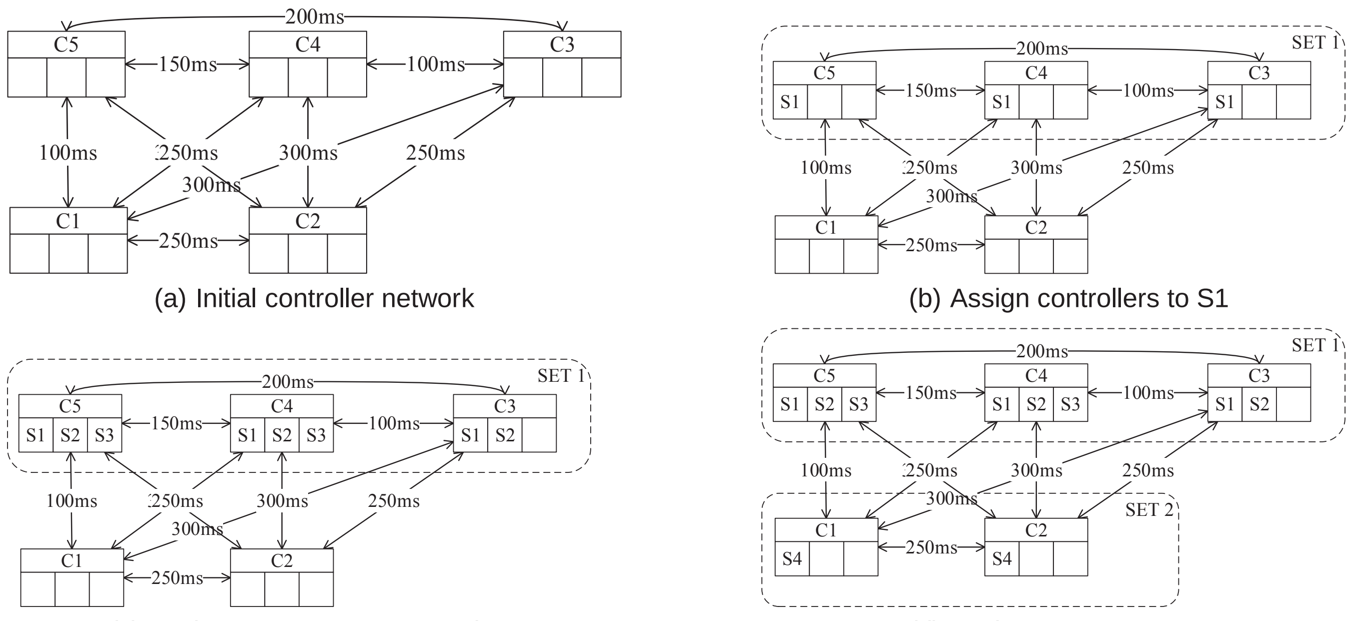
```
1:  $C' \leftarrow \emptyset$;
2:  for $i = 1$ to $m$ do
3:      if the number of controllers with non-zero residual capacity in $C'$ is less than $B_i$ then
4:          $\delta_{min} = \arg\min_{s_i \in S_i} \delta_i$;
5:          $C' \leftarrow$ FIND_NEXT_SET$(C, \emptyset, \delta_{min}, B_i)$;
6:      end if
7:      sort the controllers in $C' = \{c_{\Pi_1}, c_{\Pi_2}, .., c_{\Pi_{|C'|}}\}$ such that $k_{\Pi_1} \leq k_{\Pi_2} \leq ... \leq k_{\Pi_{|C'|}}$;
8:      for $j = 1$ to $|C'|$ do
9:          if $k_{\Pi_j} > 0$ then
10:             $\Phi_j \leftarrow c_{\Pi_j}$;
11:             $k_{\Pi_j} = k_{\Pi_j} - 1$;
12:             $C' = C' - \{c_{\Pi_j}\}$; $C = C - \{c_{\Pi_j}\}$;
13:             $S = S - \{s_i\}$;
14:         end if
15:     end for
16: end for
```

Algorithm 2 The procedure of Find_Next_Set

```
1:  function FIND_NEXT_SET$(C, C', \delta_{min}, B)$
2:      if $B = 0$ then
3:          return $C'$;
4:      end if
5:      for $c_i \in C = \{c_1, c_2, ..., c_{|C|}\}$ do
6:          if $L(c_i, C') \leq \delta_{min}$ and $k_i = K$ then
7:              $T \leftarrow$ FIND_NEXT_SET$(C - \{c\}, C' + \{c\}, \delta_{min}, B - 1)$
8:              if $|T| = B$ then
9:                  return $T$;
10:             end if
11:         end if
12:     end for
13:     return $C'$;
14: end function
```

The function FIND_NEXT_SET, as shown in Algorithm 2, finds a set of B controllers that satisfy the minimum latency δ_{min} in a recursive way. If the input parameter B is 0, the function end with returning the set of controllers C' in line 2 to 4. Otherwise, it continue to check the other controllers in set C. For each controller $c \in C$ that satisfies the latency requirement, we begin a new recursion by invoking the FIND_NEXT_SET function. If the size of allocated controller set C' return by the recursion is equal to B, the function returns T. Otherwise, the function continues to try the next controller in C.

In a worst case, it is needed to find a new controller set after one of the existed assigning set controller is full. As K denotes the capacity of each controller, after assign K switches, the algorithm searches another set for assignments. Therefore, the total running time of this assignment algorithm is $\mathcal{O}(\frac{nC^B}{K})$, where n denotes the number of switches, C denotes the number of controllers



(a) Initial controller network

(b) Assign controllers to S1

(c) Assign controllers to S2 and S3

(d) Assign controllers to S4

Fig. 5. The steps of controller allocation with RQFA algorithm

and B denotes the maximum required number of controllers of switches. Considering the required number of switches is no more than 20, the time complexity is acceptable in general case.

5.2 Example

For better understanding, we use an example to show the execution process of the proposed RQFA algorithm. We consider a control plane consisting of five controllers as shown in Figure 5, where the number on each link indicates the communication latency. There are four switches S1, S2, S3 and S4 that requires 3, 3, 2 and 2 controllers, respectively, with maximum latency 300ms, 200ms and 250ms.

According to the RQFA algorithm, we first sort the switches with their required number of controllers and find the minimum latency is 200ms. Next, we find a controller set SET 1 including C3, C4 and C5 with capacity is 3 and maximum latency is 200ms, then we assign this set to switch S1 as shown in Fig. 5(b). For the next two switches S2 and S3, with enough capacity in each controllers, we can assign the previous set to switch them again. The assignment of S1, S2 and S3 is shown in Fig. 5(c). After that, as the capacity of C3 and C4 become 0, SET 1 can not fit the requirement of next switch S4. From the design of RQFA algorithm, with the maximum required number is 2 and minimum latency is 250ms, we find another controller set SET 2 including C1 and C2 and assign controllers in SET 2 to S4. After all assignment, the result of the network is shown in Fig. 5(d).

5.3 Performance analysis

We first define some notations that will be used in the following analysis. We let C_i' denote the i-th candidate set found in our algorithm, and its cardinality is N_i, i.e., $N_i = |C_i'|$. When we attempt to find a new candidate set, the number of controllers without residual capacity, which is also referred to as full controllers, in current candidate set C_i' is denoted by N_i^f. The total number of candidate sets is H.

Lemma 1: In the candidate set $C_i', 1 \leq i \leq H-1$, the number of full controllers is greater than $\frac{N_i}{2}$, i.e., $N_i^f > \frac{N_i}{2}, \forall 1 \leq i \leq H-1$.

Proof: We finish the prove by contradiction. Suppose $N_i^f \leq \frac{N_i}{2}$. The number of controllers of the last switch accommodated by C_i', which is denoted by B_j, must be no greater than N_i^f, which leads to

$$B_j \leq N_i^f \leq \frac{N_i}{2}. \tag{5}$$

When we consider the next switch s_{j+1}, we will find a new candidate set according to our algorithm. Since the switches are sorted according to the number of required controllers in a descending order, we have $B_{j+1} \leq B_j$. Combined with (5), it is easy to see that the number of controllers with non-zero residual capacity in set C_i' is greater than B_{j+1}, and it is not necessary to find a new candidate set. □

Theorem 2: The number of controllers employed by the CAFTS algorithm is no more than two times of that in an optimal solution.

Proof: We let $\mathcal{N}^*$ denote the number of controllers employed by the optimal solution. It is easy to see that its lower bound is $\frac{\sum_{i=1}^{m} B_i}{K}$. As we have shown in Lemma

1, over a half of controllers in each candidate set $C'_i, 1 \leq i \leq H$, are full, which leads to

$$\frac{K}{2}\sum_{j=1}^{H-1} N_j < \sum_{i=1}^{m} B_i. \quad (6)$$

The performance ratio between CAFTS and the optimal solution can be calculated by:

$$\begin{aligned} \frac{\mathcal{N}}{\mathcal{N}^*} &\leq \frac{\sum_{j=1}^{H} N_i}{\frac{\sum_{i=1}^{m} B_i}{K}} \leq \frac{\sum_{j=1}^{H-1} N_i + N_H}{\frac{\sum_{i=1}^{m} B_i}{K}} \\ &\leq \frac{\frac{2\sum_{j=1}^{m} B_i}{K} + N_H}{\frac{\sum_{i=1}^{m} B_i}{K}} \\ &= 2 + \frac{KN_H}{\sum_{i=1}^{m} B_i} \end{aligned} \quad (7)$$

Since $KN_H \leq KN_1 \leq \sum_{i=1}^{m} B_i$, we finally get $\frac{\mathcal{N}}{\mathcal{N}^*} \leq 2$. $\square$

6 Performance Evaluation

We conduct both simulation and emulation based experiments to evaluate the performance of the proposed algorithms. In the simulation, a case study is given first and then simulation results under different network parameters are presented.

6.1 Case Study

For a better understanding of how our algorithm performs compared to the optimal solution, we elaborate the results of controller assignment in a small-scale SDN network with 20 switches that can be controlled by 100 servers in the cloud, each with a capacity of $K = 10$. To simulate the communication latency between controllers that are distributed in different groups, we randomly generate latency in exponential distribution with mean values of 100, 200, and 300ms respectively. The required number of controllers and maximum latency of each SDN switch are listed in Table 2.

By the exhaustive search, the optimal solution is found as shown in Table 3, in which only 13 controllers are chosen and each of them fully exploits their capacity of 10. The result of RQFA given in Table 4 shows that 17 controllers, 4 more compared to the optimal assignment, are required. The controller 15, 16, 18, 19, 36, 39 and 45 exploit their capacity of 7, 6, 2, 1, 9, 8, 4 and 3, while other 9 controllers are fully utilized. From the RQFA algorithm, the assignment first selects the set of controller 3, 4, 6, 9, 10, 12, 13, 15, 16, 18 and 19 with the maximum requirement from switch 6 and assigns these controllers to the switches. After assigning controllers to 10 switches, this set cannot afford the switch 1 who has a requirement of 6 controllers, which is larger than half size of first set. Therefore, RQFA algorithm selects the second set of 6 controllers for assignment. Since the remaining switches are no more than 10, there is no assignment after assigning controllers in the second set.

TABLE 3
Result of the optimal algorithm

Controller ID	Controlled switches	Used
10	0, 2, 3, 6, 7, 8, 10, 11, 12, 13	10
13	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
18	0, 6, 10, 11, 12, 13, 14, 15, 16, 17	10
22	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
35	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
39	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
52	6, 10, 11, 12, 13, 14, 15, 16, 17, 18	10
53	0, 2, 3, 5, 6, 7, 8, 9, 10, 11	10
56	0, 2, 3, 6, 7, 8, 9, 10, 11, 12	10
57	10, 11, 12, 13, 14, 15, 16, 17, 18, 19	10
68	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
78	10, 11, 12, 13, 14, 15, 16, 17, 18, 19	10
96	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10

TABLE 4
Result of the RQFA algorithm

Controller ID	Controlled switches	Used
3	0, 2, 3, 5, 6, 7, 8, 9, 10, 11	10
4	0, 2, 3, 5, 6, 7, 8, 9, 10, 11	10
6	0, 2, 3, 5, 6, 7, 8, 9, 10, 11	10
9	0, 2, 3, 5, 6, 7, 8, 9, 10, 11	10
10	0, 2, 3, 5, 6, 7, 8, 9, 10, 11	10
12	0, 2, 3, 5, 6, 7, 8, 9, 10, 11	10
13	0, 2, 3, 5, 6, 7, 8, 9, 10, 11	10
15	0, 2, 3, 6, 7, 8, 9	7
16	0, 2, 3, 6, 7, 8, 10, 11, 12	6
18	0, 6	2
19	6	1
21	1, 4, 12, 13, 14, 15, 16, 17, 18, 19	10
30	1, 4, 12, 13, 14, 15, 16, 17, 18, 19	10
33	1, 4, 12, 13, 14, 15, 16, 17, 18, 19	9
36	1, 4, 12, 13, 14, 15, 16, 17	8
39	1, 4, 12, 13,	4
45	1, 4, 12	3

Finally, the assignment cost 11 controllers for the first set and 6 controllers for the second set. In summary, our RQFA algorithm performs close to the optimum with a ratio of 1.2.

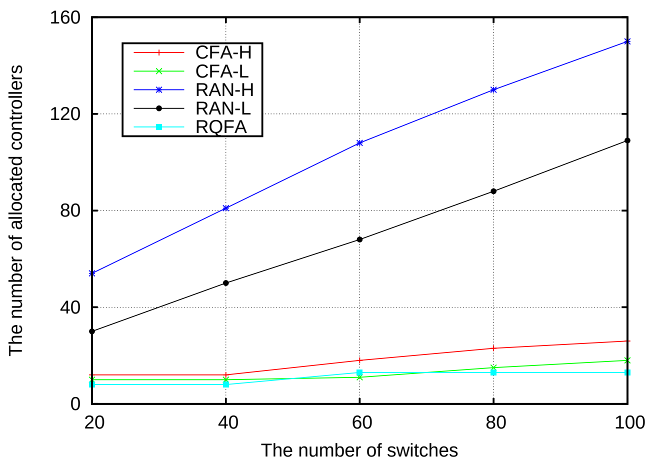
6.2 General cases

To evaluate the performance in general cases, we generate random networks and compare the cost of different controller assignment algorithms. In our simulation, we run a python 2.7 script with the networkx library 1.6 on a desktop computer. Parameters L_{jk} are uniformly distributed within the ranges $[100ms, 200ms]$ and $[100ms, 2900ms]$ representing network settings with low latency and high latency among controllers, respectively. The former range is obtained from the ping latency collected in the same data center of google compute engine or amazone EC2, while the latter range is from different data centers and different cloud services. The maximum latency required by each switch is also set following a uniform distribution in the range $[100ms, 1900ms]$.

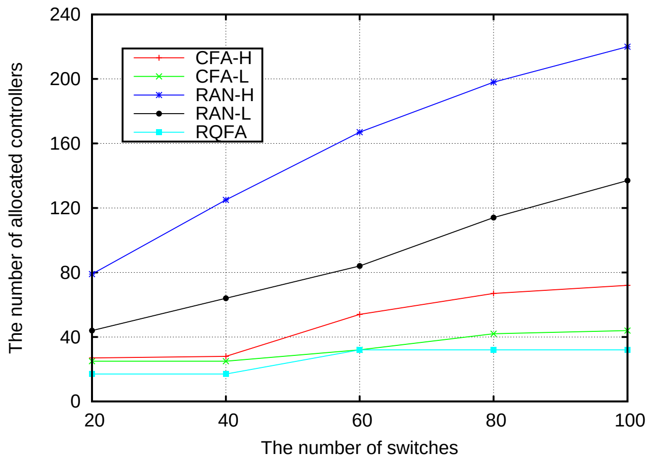
For the purpose of comparison, the following algorithms are also considered: (1) a random assignment algorithm denoted as RAN, and (2) the capacity first assignment (CFA) algorithm. The basic idea of CFA is to

TABLE 2
The requirement of SDN switches in case study

Switch ID	0	1	2	3	4	5	6	7	8	9
Required number of controllers	10	6	9	9	6	7	11	9	9	8
Maximum latency (ms)	330	320	330	330	320	330	330	330	330	330
Switch ID	10	11	12	13	14	15	16	17	18	19
Required number of controllers	7	7	6	5	4	4	4	4	3	2
Maximum latency (ms)	310	310	310	210	210	200	200	200	110	100

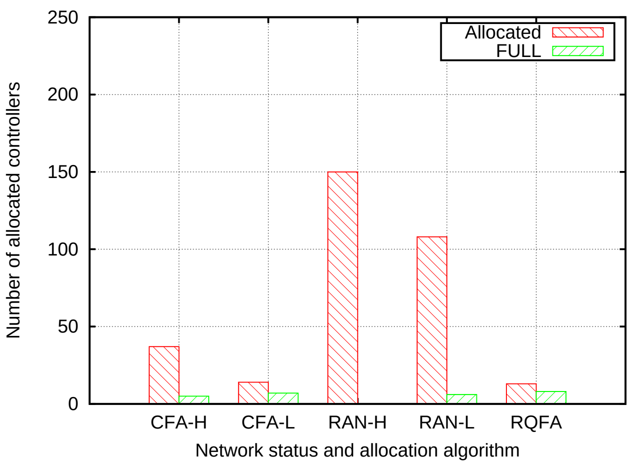


(a) B_i is uniformly distributed within {3, 8}

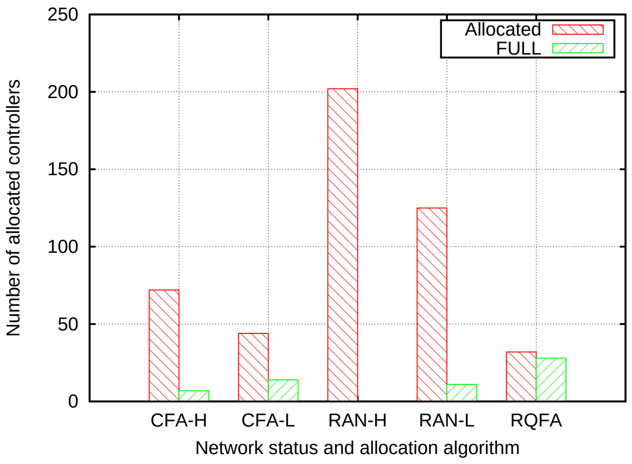


(b) B_i is uniformly distributed within {13, 17}

Fig. 6. Algorithm performance under different number of switches



(a) B_i is uniformly distributed within {3, 8}



(b) B_i is uniformly distributed within {13, 17}

Fig. 7. Number of full controllers under different algorithm.

iteratively assign controllers with the minimum residual capacity to each switch. For any algorithm X, we use notations X-L and X-H to denote the results obtained in networks with low and high latency, respectively. All simulation results are averaged over 20 network instances.

We study the performance of the proposed algorithm under different number of switches. The number of controllers required by each switch is randomly specified in a uniform distribution in [3, 8] and [13, 17] as shown in Fig. 6. We observe that the RQFA algorithm performs the same in different network latencies.

When B_i is within [3, 8] as shown in Fig. 6(a), the number of employed controllers increases as the size of data plane grows for all algorithms. When the number of switches is small, all algorithms have the similar performance. As the number of switches increases, the result outperforms RAN and CFA under both low- and high-latency networks. For example, to cover 100 switches, RQFA only needs 13 controllers, while CFA-H and CFA-L need 23 and 19 controllers, respectively. This number obtained by RAN-H and RAN-L goes even higher to 140 and 105, respectively. Similar observations are made in Fig. 6(b), but the performance gap among these algorithms becomes larger.

We show the number of controllers whose resources are fully utilized after assignment in Fig. 7. Similarly, we also show the performance when B_i is uniformly distributed within $[3, 8]$ and $[13, 17]$, respectively. We observe that only a small portion of controllers is full in random algorithm. Although CFA-L and RQFA have similar performance in total number of assigned controllers, more full controllers are fully utilized under RQFA. This phenomenon is very obvious in Fig. 7(b), where over 80% controllers are full under RQFA while the percentage is less than 40% under CFA-L.

Finally, we investigate the influence of controller capacity to the performance of the proposed algorithm. The capacity of controller nodes is set to uniform distribution with five ranges shown in Fig. 8(a) and 8(b). The number of switches is fixed to 10. When each switch requires 5 controllers, i.e., $K = 5$, as shown in Fig. 8(a), RQFA always outperforms other algorithms. The capacity of controllers has little effect to the performance under random algorithms. However, the performance of CFA-H and CFA-L decreases by 46% when capacity range changes from [15, 25] to [55, 65].

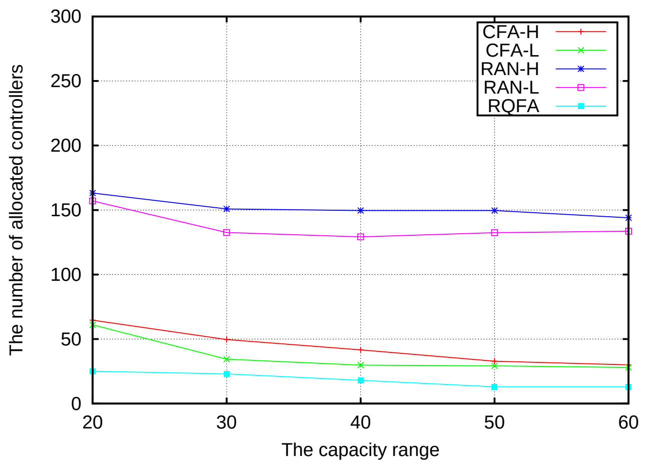
The results under different capacity ranges when $K = 15$ is shown in Fig. 8(b). The performance of all algorithms shows obviously as decreasing functions of controller capacity. For example, as the capacity range changes from [15, 25] to [55, 65], the performance of RQ-FA decreases by 74%, while CFA based algorithms and the random allocation has about 50% 10% decreasing, respectively.

The simulation results show that the RQFA performs well in several different network settings. Compared to the random assignment, both RQFA and CFA have much better efficiency on the CAFTS problem. These two algorithms allocate almost controllers to meet the assignment requirement when the switches are less than 40. While the switches increased, the RQFA needs fewer controllers than CFA assignment. It is because, with the switches scaled up, since CFA considers the capacity, firstly, it will happen during the assignment that the existed available controller network cannot meet the latency requirement even though the controllers have enough space for the assignment. While the RQFA considers the latency requirement to find the available network, the assignment ends only when there are not enough controllers with capacity.

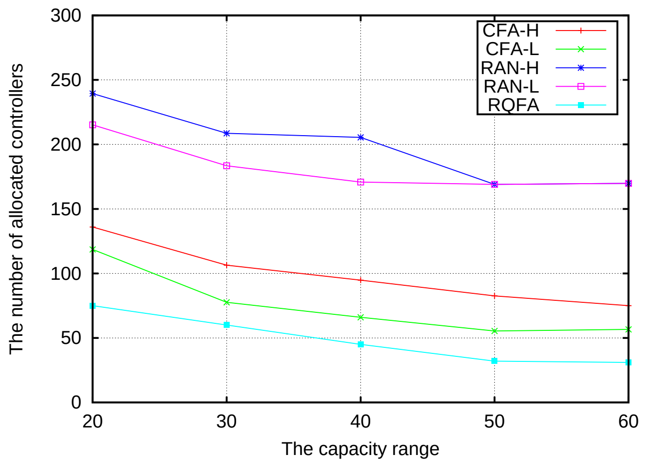
6.3 Security level

In the security level test, the data plane consists of 100 switches, connected by a regular random network, each of which is associated by a single host. We consdier two settings of control plane with 20 controllers and 50 controllers.

For comparison, the following two schemes are considered: (1) Each switch is managed by 4 controllers using the proposed BFT-based scheme. (2) Each switch is managed by a single controller using the traditional scheme.



(a) Each switch requires 5 controllers



(b) Each switch requires 15 controllers

Fig. 8. Algorithm performance under different controller capacity.

With this network setting, we study the security level by measuring connectivity losses of the data plane when the control plane is under attack. The attack to the control plane will intrude some controllers and send malicious rules to all switches connected to the compromised controllers. After each round of attack, we randomly select the compromised controllers in the control plane. The number of selected compromised controllers is increased from 2 to 7.

A connectivity loss means a pair of nodes lose connectivity in the data plane. In our simulation, when a controller in the traditional scheme is compromised by the attack, we remove all connected switches and their associated links in the topology. Under the BFT scheme, we only remove the switches and links when their related BFT algorithms fail. The connectivity loss is measured by the function shortest_path provided by networkx. We invoke this function to test each pair of nodes in the data plane. If this function can not return

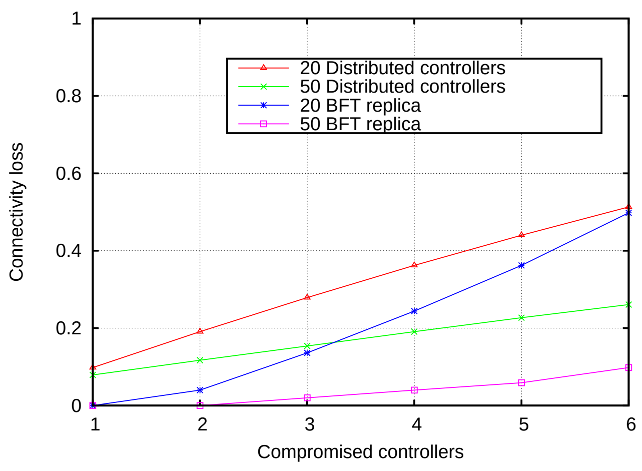


Fig. 9. The network connectivity loss of data plane during the control plan under attack

a link, it means a connectivity loss.

As shown in Fig. 9, the proposed BFT-based scheme incurs much less connectivity losses, i.e., achieves higher security level, compared to the traditional one under all cases. For example, the control plane still works under our scheme when one controller is compromised, while the traditional scheme loses quite connectivity. Another observation is that more controllers can contribute to enhancing the security level siganificantly. Under our scheme, when six controllers are simultaneously compromised, about 45% connectivity will be lost in 20-controller network, while this number drops to 10% in 50-controller network. Even though the replicas manage more switches than the controller in the traditional scheme, the possibility that more than two failed replicas manage the same switch in the BFT-based scheme is less than the possibility that a switch is managed by a failed controller.

6.4 Emulation results

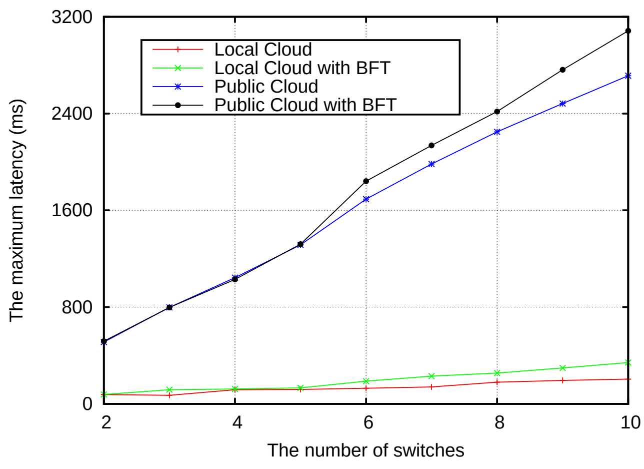


Fig. 10. The maximum latency with different switch scale.

The main purpose of this set of experiments into evaluate the network latency when rule update happens. We create a 10-switch SDN, each connecting to a host, using the well-known mininet emulator [31] under two scenarios as described below.

1) Local Cloud: We use two desktop computers: one is used to create a light weight local cloud environment, and the other for the mininet emulator [31]. These two computers are both equipped with Intel Core i7 4770k 3.4GHz processors, 12GB of RAM and Gigabyte NICs and connected by a Gigabyte switch. For the local cloud environment, we install the VMWare Workstation System to provide four virtual machines (VM) as cloud instances with Ubuntu 12.04 LTS, CentOS 6.3, Fedora 17 and Debian 6.0.8 as the operation systems, separately.
2) Public Cloud: To evaluate the performance in public cloud environment, we use four instances provided by google compute engine service. Each instance is assigned a vCPU and 3.8 GB of RAM. We use same operation systems on these instances. Since our prototype is based on the POX, a Python implementation of NOX [14], we also installed original POX as the comparison.

We test the ping latency from the first host to other hosts at 2-10 hops away. We test each ping latency 20 times and record the average value. To create a link for the ping command between any two ports of each switch, it takes 50ms of the network latency to update the rule to this switch in the emulator. As shown in Fig. 10, the latency incurred by applying BFT to controllers in cloud is close to that under the traditional scheme with single controller for both scenarios, especially when the hop number is less than 4. Even in the case of 10-hop ping distance, the accumulated latency is only 13% higher in the public cloud scenario.

The overhead of BFT is not obvious because the latency for rule updating is nearly 50ms in the mininet. Since the number of hops between any two nodes in a typical data center network is less then 6 (leaf switches, aggregation switches and core switches) and rule updating or forwarding only happens in the beginning of each network flow, the BFT is a practical and efficient mechanism to resist malicious access.

7 Conclusion and Future Work

In this paper, we propose a secure SDN architecture that uses multiple controllers to confirm the update of flow tables in each switch. We apply the Byzantine mechanism to guarantee that each switch can correctly update its flow tables even some compromised controllers issue false instructions. To meet the requirement the requirements of dynamic and isolation form the BFT mechanism, we shift the SDN control plan to the cloud and deploy the BFT replicas in different instances. Based on this architecture, we study a controller assignment problem to minimize the number of employed controllers while satisfying the security requirements of a given set of switches. Finally, extensive simulations are conducted to show that the proposed algorithm can significantly reduce the number of controllers.

In the future, we plan to implement a complete SDN solution with modified openflow protocol to support multiple controllers. Meanwhile, it is signification to find appropriate BFT algorithm to suit the SDN control plane. A deeper experiment with the real word testbed is also needed to evaluate the efficiency of the new SDN solution.

References

[1] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," *in Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[2] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 254–265.

[3] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 1–6.

[4] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 1–14.

[5] S. Shenker *et al.*, "The future of networking, and the past of protocols," *Open Networking Summit*, 2011.

[6] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!" in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 7:1–7:6.

[7] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying sdn programming using algorithmic policies," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 87–98.

[8] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 55–60.

[9] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "Devoflow: Cost-effective flow management for high performance enterprise networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 1:1–1:6.

[10] D. Mazières and D. Shasha, "Building secure file systems out of byzantine storage," in *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*, ser. PODC '02. New York, NY, USA: ACM, 2002, pp. 108–117.

[11] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec. 2008.

[12] C. Cachin, I. Keidar, and A. Shraer, "Trusting the cloud," *SIGACT News*, vol. 40, no. 2, pp. 81–86, Jun. 2009.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[14] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[15] Z. Cai, A. L. Cox, and T. E. N. Maestro, "A system for scalable openflow control," Technical Report TR10-08, Rice University, Tech. Rep., 2010.

[16] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?" in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[17] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, ser. PRESTO '10. New York, NY, USA: ACM, 2010, pp. 8:1–8:6.

[18] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: a slice abstraction for software-defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 79–84.

[19] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," *CoRR*, vol. abs/1308.6138, 2013.

[20] P. Porras, S. Seungwon, Y. Vinod, F. Martin, and G. G. Tyson-Mabry, "A security enforcement kernel for openflow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 121–126.

[21] S. Seugwon, P. Phillip, Y. Vinod, F. Martin, G. Guofei, and T. Mabry, "Fresco: Modular composable security services for software-defined networks," in *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, ser. NDSS Symposium 2013. San Diego, CA United States: Internet Society, February 2013.

[22] H. Kim, J. Santos, Y. Turner, M. Schlansker, J. Tourrilhes, and N. Feamster, "Coronet: Fault tolerance for software defined networks," in *Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP 2012)*, 2012, pp. 1–2.

[23] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "Fattire: declarative fault tolerance for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 109–114.

[24] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. –, 2013.

[25] M. Malik, M. Montanari, J. Huh, R. Bobba, and R. Campbell, "Towards sdn enabled network control delegation in clouds," in *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013)*, June 2013, pp. 1–6.

[26] H. Li, P. Li, S. Guo, and S. Yu, "Byzantine-resilient secure software-dened networks with multiple controllers," in *Proceedings of IEEE International Conference on Communications*, ser. ICC '14. Sydney, Australia: IEEE, 2014, pp. 695–700.

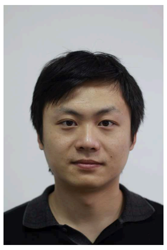
[27] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[28] G. Veronese, M. Correia, A. Bessani, L. C. Lung, and P. Verissimo, "Efficient byzantine fault-tolerance," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 16–30, 2013.

[29] H. Takabi, J. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security Privacy*, vol. 8, no. 6, pp. 24–31, Nov 2010.

[30] Y. Zhang, Z. Zheng, and M. Lyu, "Bftcloud: A byzantine fault tolerance framework for voluntary-resource cloud computing," in *Proceedings of IEEE International Conference on Cloud Computing (CLOUD 2011)*, July 2011, pp. 444–451.

[31] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6.



He Li received the BS and MS degrees from Huazhong University of Science and Technology in 2007 and 2009, respectively. Currently, he is an PhD student in Graduate school of Computer Science and Engineering, University of Aizu, Japan. His research interests include cloud computing and software defined networking. He is a student member of the IEEE and the IEEE Communication Society.

Peng Li received his BS degree from Huazhong University of Science and Technology, China, in 2007, the MS and PhD degrees from the University of Aizu, Japan, in 2009 and 2012, respectively. He is currently an Associate Professor at School of Computer Science and Engineering, the University of Aizu, Japan. His research interests include networking modeling, cross-layer optimization, wireless sensor networks, cloud computing, smart grid, performance evaluation of wireless and mobile networks for reliable, energy-efficient, and cost-effective communications. He is a member of the IEEE.

Song Guo (M'02-SM'11) received the PhD degree in computer science from University of Ottawa, Canada. He is currently a Full Professor at School of Computer Science and Engineering, the University of Aizu, Japan. His research interests are mainly in computer networks, parallel and distributed computing, cyber-physical systems, and cloud computing. He has published over 250 papers in referred journals and conferences in these areas and received three IEEEACM best paper awards. Dr. Guo currently serves as Associate Editor of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Emerging Topics in Computing with duties on emerging paradigms in computational communication systems, and on editorial boards of many others. He has also been in organizing and technical committees of numerous international conferences. Dr. Guo is a senior member of the IEEE and the ACM.

Amiya Nayak received his B.Math. degree in Computer Science and Combinatorics & Optimization from University of Waterloo in 1981, and Ph.D. in Systems and Computer Engineering from Carleton University in 1991. He has over 17 years of industrial experience in software engineering, avionics and navigation systems, simulation and system level performance analysis. Currently, he is a Full Professor at the School of Electrical Engineering and Computer Science at the University of Ottawa. His research interests are in the area of sensor networks, cloud computing, and distributed systems with over 200 publications in refereed journals and conference proceedings. He has co-edited Handbook of Applied Algorithms: Solving Scientific, Engineering, and Practical Problems, John Wiley & Sons (2008), and co-authored Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication, John Wiley & Sons (2010).