Defending against Flow Table Overloading Attack in Software-Defined Networks

Article *in* IEEE Transactions on Services Computing · August 2016

DOI: 10.1109/TSC.2016.2602861

CITATIONS
16

READS
237

6 authors, including:

Bin Yuan
Huazhong University of Science and Technology
13 PUBLICATIONS **27** CITATIONS
SEE PROFILE

Deqing Zou
Huazhong University of Science and Technology
137 PUBLICATIONS **796** CITATIONS
SEE PROFILE

Hai Jin
Huazhong University of Science and Technology
886 PUBLICATIONS **6,534** CITATIONS
SEE PROFILE

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

Defending against Flow Table Overloading Attack in Software-Defined Networks

Bin Yuan, Deqing Zou, Shui Yu, *Senior Member, IEEE,* Hai Jin, *Senior Member, IEEE,* Weizhong Qiang, and Jinan Shen

Abstract—The Software-Defined Network (SDN) is a new and promising network architecture. At the same time, SDN will surely become a new target of cyber attackers. In this paper, we point out one critical vulnerability in SDNs, the size of flow table, which is most likely to be attacked. Due to the expensive and power-hungry features of Ternary Content Addressable Memory (TCAM), a flow table usually has a limited size, which can be easily disabled by a flow table overloading attack (a transformed DDoS attack). To provide a security service in SDN, we proposed a QoS-aware mitigation strategy, namely, peer support strategy, which integrates the available idle flow table resource of the whole SDN system to mitigate such an attack on a single switch of the system. We established a practical mathematical model to represent the studied system, and conducted a thorough analysis for the system in various circumstances. Based on our analysis, we found that the proposed strategy can effectively defeat the flow table overloading attacks. Extensive simulations and testbed-based experiments solidly support our claims. Moreover, our work also shed light on the implementation of SDN networks against possible brute-force attacks.

Index Terms—SDN security, DDoS attacks, QoS, Security Service, Flow table.

1 Introduction

SDN (Software-Defined Network) is the key outcome of extensive research efforts over the past few decades towards transforming the Internet to a more open, programmable, and manageable infrastructure [1]. The main concept of the SDN is to decouple the control and data plane of a network. In the SDN paradigm, network devices become simpler and cheaper, since they only have to support general forwarding protocols, instead of complex logic of distributed routing protocols [2], [3].

Among all the implementations of southbound SDN protocol, OpenFlow is the most widely used and de facto standard protocol [4]. In the OpenFlow paradigm, a controller is connected to all switches. Each switch possesses a flow table. The controller remotely controls the switches' flow tables by installing, modifying, and deleting rules in them, and the switches perform packet processing according to the rules in their flow tables [3], [5].

Because of the growing demands for a fast and efficient data plane, flow tables are usually implemented using TCAM, which is very expensive, power-hungry, and therefore of limited size. As a result, most of the switches available on the market have relatively small flow table size, with up to 8000 entries [4].

Being aware of the limitation of flow table space, many studies have been done to optimize rule placement, minimize the number of rules, and reduce costs [6], [7], [8], [9]. However, these systems focused on improving performance of SDNs under general network conditions, while we treat the limited flow table resource as a potential weakness that can be used to destroy the services of SDNs. We attempt to investigate the vulnerabilities of SDN. To provide a security service in SDN, we direct our attention to mitigating the attacks that target the SDN (SDN devices). Therefore, these solutions [6], [7], [8], [9] cannot fundamentally solve the attacking problem that we try to solve.

Meanwhile, a few researchers [10], [11], [12] started to explore the security threats and possible attacks in SDNs. However, these works only analyzed the possible ways to attack an SDN and presented potential solutions in a general way. They did not propose executable and practical solutions to deal with the new threats in SDNs. As far as we know, Shin et al. [13] were the first to study the realistic attack cases to SDNs. In addition, they designed a new framework, AVANT-GUARD [14], that enables the control plane and the SDN network to be more resilient and scalable against control plane saturation attacks.

We carefully studied the vulnerability of SDN and pointed out one critical vulnerability in SDN, the size of flow table, which is most likely to be attacked and can be easily disabled by a *flow table overloading attack*. We believe a flow table overloading attack is a transformed DDoS attack in SDN. Traditional DDoS attacks usually target draining the hosts, servers, or network bandwidth resources, while a flow table overloading attack targets the SDN switch (NOT the host, server, or link), and aims at exhausting the flow table of the targeted switch, thus degrading or disabling the switch and eventually destroying the network services.

To mitigate the notorious DDoS attacks, many solutions have been proposed, such as exploiting the profound resources in the cloud, decreasing the SYN retransmission timer to defend against TCP SYN flood attacks, etc. [15], [16], [17], [18], [19]. With the state-of-art solutions, it is possible to successfully prevent the hosts or servers from being victimized by DDoS attacks.

However, these existing DDoS attack mitigation approaches are not effective when dealing with flow table overloading attacks, since flow table overloading attacks leverage different vulnerabilities and have different targets. Therefore, in this paper, we pay our attention to this specific attack in SDNs, investigating the effectiveness of such attacks and how to defeat such attacks.

- *B. Yuan, D. Zou, H. Jin, W. Qiang, and J. Shen are with the Cluster and Grid Computing Lab, Services Computing Technology and System Lab, Huazhong University of Science and Technology, Wuhan, 430074, China. Professor Jin is the corresponding author. E-mail: hjin@hust.edu.cn.*
- *S. Yu is with the School of IT, Deakin University, Victoria, 3125, Australia. E-mail: syu@deakin.edu.au.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

To demonstrate that SDNs are vulnerable to such attacks, we have conducted practical and feasible experiments to show how to launch such attacks. We sent a large number of new flows to a targeted switch, which forced the controller to install new rules into the flow table to process the flows. With the attack continuing, the flow table space was quickly consumed. At last, the flow table was overloaded and the switch was disabled. As a result, the later arriving packets were dropped.

In the real world, attackers can use compromised computers, such as a botnet, to start a flow table overloading attack by sending a huge number of spoofed "new" flows to the victim. Each new flow will result in installing one new rule in the targeted switch. Usually, the rules installed for such attacking flows (we call these rules *attacking rules*) are useless for legitimate processing. Nonetheless, rules usually have a finite life cycle. A rule will be removed due to timeout. Moreover, the controller can send messages to the switches to delete any rule at any time [5]. Thus, the flow table can be cleared up by deleting these attacking rules.

However, when under attack, the rate of new flows is so high that we cannot differentiate the attacking flows from legitimate flows in real time. Therefore, we cannot delete attacking rules in a timely way. What's worse, the legitimate rules will be deleted due to timeout and their space will be occupied by attacking rules. As a result, soon enough, the flow table will be almost fully occupied by attack rules, and the network services will be destroyed.

As aforementioned, a flow table overloading attack is a transformed DDoS attack in SDN. Recent researches have demonstrated that the essential issue of DDoS attack and defense is a competition for resources: the winner is the side who possesses more resources in the battle [20], [21], [22]. In addition, recent works [23], [24] have shown that the attack strength is also limited. Therefore, if we can invest more flow table resources in the battle, we can mitigate or even beat such attacks in SDNs.

Inspired by this, from the point of view of resource management, we propose a QoS-aware *peer support* strategy that integrates as many as possible idle flow table resources in the whole SDN system to mitigate flow table overloading attacks, and avoids violation of QoS as much as possible in the same time. With the peer support strategy, switches treat each other as peers. When a peer switch runs out of flow table resources, other peer switches support it by offering their idle flow table spaces to install these new rules, including both attacking rules and legitimate rules. Therefore, not only can we mitigate the attack, but also we can forward most of the (legitimate) traffic successfully, minimizing the QoS violation, when under attack.

With the peer support strategy, we established an executable and practical mathematic model to represent the SDN system. Testbed-based experiments and extensive simulations showed that our approach can remarkably enhance the capacity of SDN in defending against flow table overloading attacks. More importantly, we found that we can beat flow table overloading attacks by properly configuring the SDN with different flow table sizes and different SDN sizes (measured by the number of switches).

We summarize the contributions of this paper as follows:

- We pointed out that, in SDN, the limitation of flow table resources opens a door for *flow table overloading attack*, which is a potential threat to SDN.
- We proposed a QoS-aware strategy, *peer support*, to integrate as many as possible idle flow table resources in the whole SDN system to counter the attack.
- We established an executable and practical mathematical model and conducted a thorough analysis to estimate the capacity of SDN in defending against flow table overloading attacks.
- Based on quantified results, we provide theoretical guidance on how to arrange SDN flow table resources against different levels of brute-force attack investment, which is beneficial on system design of SDNs.

The remainder of this paper is structured as follows. Section 2 presents the related work. In Section 3, we briefly review the preliminary knowledge of SDN and the mathematical material that we use. We discuss the mitigation mechanism in Section 4. System modeling and analysis are presented in Section 5. Performance evaluations are conducted in Section 6. We discuss the limitations of our proposed approach in Section 7. Finally, we conclude this paper and discuss future work in Section 8.

2 Related Work

2.1 DDoS Attack Mitigation in Traditional Networks

DDoS attacks aim to exhaust the resources of victims, such as network bandwidth, computing power, and operating system data structures. Many well-known web sites, such as CNN, Amazon, and Yahoo, have been the targets of such attacks. Since the emergence of DDoS attacks, many solutions have been proposed to mitigate them [15].

Yau et al. [25] viewed DDoS attacks as a resource management issue in their DDoS defense proposal. They proposed the installation of a router throttle at selected upstream routers of a possible victim. The participant routers regulate the traffic flows to the protected server in a proactive way using a level-k max-min fairness strategy. Their target was to constrain the number of attack packets far away from the protected server.

Yu et al. [26] proposed a similarity based DDoS detection method to beat flash crowd mimicking attacks. Also, in [19], Yu et al. designed a strategy to dynamically allocate idle or reserved cloud resources to those cloud customers who are experiencing DDoS attacks in order to defeat the attacks, and at the same time guaranteeing the quality of service for benign users.

2.2 Attack Mitigation with SDN

As a new and promising network architecture, SDN brings many excellent functionalities to network management. As a result, many researchers attempted to leverage SDN to mitigate attacks.

Braga et al. [27] presented a lightweight method for DDoS attack detection based on traffic flow features. A NOX platform was used to provide a programmatic interface to facilitate the handling of switch information, in which the extraction of such information is made with a very low overhead.

Suh et al. [28] introduced a Content-oriented Networking Architecture (CONA) and implemented it on the NetFPGA-OpenFlow platform. In CONA, the hosts request contents and the agents deliver the requested contents. The agent keeps track of which contents are requested by which hosts. Due to the accountability and content-aware supervision, CONA can react to resource exhaustive attacks effectively.

Wang et al. [29] analyzed the impact of cloud computing and SDN on DDoS attack defense and proposed DaMask to defend DDoS attacks. They claimed that the SDN-based network monitoring and control mechanism allow companies to control and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

configure their defense mechanisms in clouds effectively without affecting other cloud users.

2.3 Flow Table Scaling Issues

The limitation of flow table space in SDN has gained a lot of attention in the research community.

Devoflow [9] proposed some modest design changes to keep flows in the data plane as much as possible while maintaining enough visibility for effective flow management, by pushing responsibility over most flows back to the switches and adding more efficient statistical information collection mechanisms. The results showed that the proposed solution had 10∼53 times fewer flow table entries.

Narayanan et al. [7] quantified the inherent limitations of merchant switching silicon and proposed SSDP, a new switch architecture that allows rapid network innovation by complementing a cost-effective, yet inflexible, front-end merchant silicon switch chip with a deeply programmable coprocessor subsystem.

Palette [8] and One Big Switch [30] attempted to address the problem of rule placement. Their goal was to minimize the number of rules in switches. The main idea of Palette was to partition end-to-end policies into subtables and then distribute them over the switches. One Big Switch, on the other hand, solved the problem separately for each path by choosing the paths based on network metrics, and then combining the results to reach a global solution.

2.4 Vulnerability of SDN

With the fast deployment of SDN, there are a few researchers starting to explore the security threats and possible attacks in SDNs, which are more related to our work.

Shin et al. [13] studied the realistic attack cases to SDNs. The authors showed the readers how to fingerprint SDNs and launch efficient resource consumption attacks. This work demonstrated that SDN brings new security issues that cannot be ignored.

Kreutz et al. [10] gave a thorough analysis of SDN. They presented several threats identified in SDN and argued for the need to build secure and dependable SDNs by design.

The security threats and vulnerabilities of OpenFlow were analyzed in [11], [12]. The authors also offered suggestions of attack mitigation measures that can potentially mitigate the security issues associated with OpenFlow networks.

However, these works only analyzed possible ways to attack an SDN and presented potential solutions in a general way. They did not propose executable and practical solutions to deal with the new threats in SDNs.

The work that is most similar to ours is AVANT-GUARD [14]. AVANT-GUARD is a new framework that tackles SDN security issues by adding intelligence to the OpenFlow data plane. It enables the control plane and the SDN network to be more resilient and scalable against CONTROL PLANE saturation attacks, while our work focuses on integrating idle resources to mitigate attacks against the DATA PLANE. In addition, by inspecting the TCP sessions at the data plane before notifying the controller, AVANT-GUARD is only capable of dealing with TCP-based attacks, while our solution is protocol agnostic.

3 Preliminary Knowledge

In this section, we briefly review the main concept of SDN and the mathematical material of power law distribution that we use when modeling the system.

3.1 SDN

SDN is a new approach to design, build, and manage networks that separates the network's control and data planes to better optimize each of them [31]. There are three layers in the SDN framework, as shown in Figure 1. The infrastructure layer is where the data plane lies. It contains the switches. The switches process the packets under the control of a remote controller. The controller sits in the control layer. It offers a centralized view of the overall network, and enables network administrators to direct the data plane on how to process network traffic. The networking applications lie on the application layer. Network administrators can provide different network services by implementing different applications and running them in the application layer.

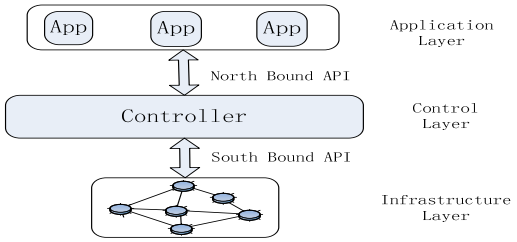


Fig. 1. The SDN Framework.

SDN uses northbound APIs to communicate with the applications. This helps network administrators to programmatically shape traffic and deploy services. On the other hand, southbound APIs are used to relay information to the switches in the underlying network topology. OpenFlow, considered as the first standard in SDN, was the original southbound API and remains as one of the most common protocols.

An OpenFlow-enabled switch has a flow table. A flow table consists of flow entries. The most important components of a flow entry are the *Match Fields* and the *Action Set*. When a packet arrives, the *Match Fields* are used to match against the packet, for example, IP address. If matched, the switch will process the packet according to the *Action Set* of the rule with the highest priority. If not matched, a *Packet-in* message will be sent to the controller. The controller then decides how to process the packet and sends a message to the switch to install a new rule in its flow table. Then, the packet is processed against the *Action Set* in the newly installed rule, as are the subsequent packets of the same flow.

It should be noted that the method of processing new flows described above is called the reactive method. Recently, to reduce the communication overheads between the control plane and the data plane when new flows arrive, researchers started to use proactive method to install forwarding rules, such as Devoflow [9]. In proactive model, the forwarding rules are pre-installed or generated locally, rather than invoking the controller to install the rule on each flow arrival.

3.2 Power Law Distribution

When the probability of measuring a particular value of some quantity varies inversely as a power of that value ($p_k = Ck^{\alpha}$), the quantity is said to follow a power law distribution [32]. Power law distribution appears widely in physics, biology, economics and finance, computer science, demography, and social sciences.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

In the discrete case, if n is a set of integer variables, we say that n follows a power law distribution if the probability $\Pr[n = n_i]$ of measuring the value n_i obeys

$$\Pr[n = n_i] = C \cdot n_i^{\alpha}, \quad (1)$$

where the constant α is called the exponent of the power law distribution. Power laws of the form $y = x^{\alpha}$ enable a compact characterization of topologies through their exponents. If such (x,y) relationships are plotted on a *log-log* scale, the power law exponent α defines the slope of the resulting linear plot [33]. The constant C is mostly uninteresting; once α is fixed, it is determined by the requirement that the probability $\Pr[n = n_i]$ sum to 1.

In previous works, it has been observed that many properties of network topologies can be described using power laws [33], [34], [35], [36]. Moreover, Newman [36] found that in network topology, the power law behavior was seen in the tail of the distribution. In this case, if the power law behavior is seen only in the tail of the distribution, for values $n \geq n_{min}$, then the equivalent expression of power law distribution is

$$\Pr[n = n_i] = C \cdot n_i^{\alpha} \quad (n \geq n_{min}) \ . \quad (2)$$

4 Flow Table Overloading Attack Mitigation in SDN

In this section, we first discuss how to launch a flow table overloading attack, demonstrating that SDNs are vulnerable to such attacks. Then, we present our mechanism to integrate idle flow table resources of the entire network to mitigate the attack.

4.1 How to Launch a Flow Table Overloading Attack

To demonstrate that SDNs are vulnerable to flow table overloading attacks, and to make our work practical and meaningful, we have conducted experiments to launch such attacks to overpower the switches in SDN.

The basic idea of a flow table overloading attack is to consume the flow table space of the targeted switch, leading to the overloading of flow table and the destruction of network services. As mentioned previously, such attacks can be launched by continuously sending new flows to the targeted switch.

Most of the switches available on the market have relatively small TCAMs, with up to 8000 entries [4]. Therefore, in our experiments, we set the maximum number of flow rules for the switch to 8000. Moreover, we set the attack rate to different values to simulate the attacks with different levels of attack strength.

As presented in Figure 2, we recorded the required time to overpower the switch under different attack rates. We can see that, when the attack rate is higher than 300 requests per second (which can be easily accomplished by contemporary attackers), the required time to overload a switch reduces with increasing attack rate. A switch will be overwhelmed in less than 10 seconds when the attack rate increases to more than 800 requests per second, which indicates that SDNs are vulnerable to such attacks.

4.2 Mitigation Strategy for Flow Table Overloading Attacks

In this subsection, we first investigate the flow table usage in SDN. Then, we propose our mitigation method, and discuss the implementation and feasibility of the proposed strategy.

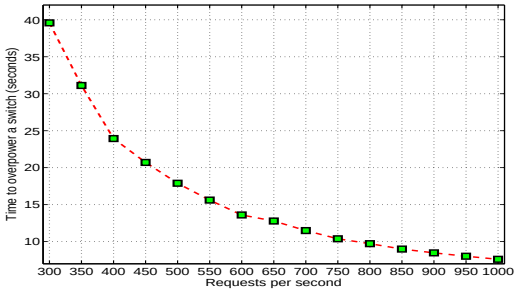


Fig. 2. Required time to overpower a switch for flow table overloading attacks with different attack rates. The chart shows that the attack is feasible and that SDNs are vulnerable to such attacks.

4.2.1 Used and Unused Flow Table Space

As shown in Figure 3, in a typical SDN, a remote, centralized control plane is connected to all switches. In practice, the control plane could be implemented by single or multiple controllers. In this paper, we treat the control plane as a single, logically centralized, remote controller. Each switch serves a certain number of hosts. The flow tables of the switches are controlled by the controller via secure channel [3].

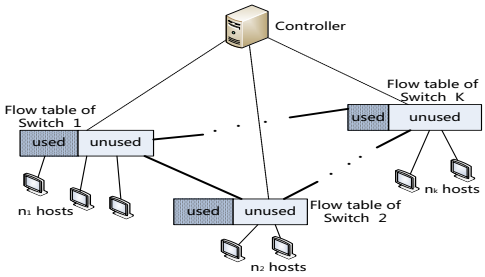


Fig. 3. The usage of flow tables in SDN under nonattack scenario.

In Figure 3, we represent the flow table space that has already been used by legitimate processing as the dashed part, and the unused space as the blank part. The unused space (also called idle space) is the part that can be leveraged to defend against attacks.

4.2.2 Peer Support Strategy

When a flow table overloading attack occurs, the attacker sends lots of spoofed "new" flows to the targeted switch. When a new flow arrives at the targeted switch, a new rule is installed in the switch's flow table to process the flow. Usually, these rules will be set with a finite timeout. In the OpenFlow paradigm, rules will be removed from flow table due to timeout, which can be set by the *hard_timeout* field. Moreover, the controller, depending on the logic of applications running on it, can send messages to switches to delete any rule at any time [5]. As a result, rules installed for new flows have a finite life cycle. Thus, sooner or later, they will disappear from the flow table.

However, when under attack, attacking flows arrive with a very high rate. Thus, the rate of installing new rules is much higher than the rate of deleting rules, and we unfortunately cannot differentiate attacking flows from legitimate flows in real time. Therefore, we cannot delete the attacking rules in a timely way, either. Soon enough, the limited flow table space will be consumed, being almost full of attacking rules. Thus, the targeted switch will be overpowered and the network service will be destroyed.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

Given that the attacking rules are useless for legitimate processing, we can mitigate flow table overloading attacks by installing the new rules into the idle flow tables distributed among the entire network. That is, when a switch runs out of flow table space, we can leverage the unused flow table spaces in other switches to install new rules. Therefore, we can prevent the flow table of the targeted switch from being exhausted and protect the network service from being destroyed.

To this end, we propose our QoS-aware mitigation strategy, *peer support* strategy, under which, the switches treat each other as peers. When a peer switch runs out of flow table resources, other peer switches support the targeted switch by offering their idle flow table spaces to install the new rules (including both attacking rules and legitimate rules), thus to mitigate the attack.

To be more specific, we monitor each switch' status. When a switch's flow table is full, we guide its traffic to other switches. Thus, the traffic will be distributed to the entire network, rather than being aggregated at the targeted switch. Therefore, the rules will be installed into the flow tables of all switches in the entire network. In this way, the idle flow table resources of the entire network can be leveraged to mitigate the attack.

It should be noted that we do NOT actually MOVE the rules from the full switches to other switches. What we do is to install a few rules in the original switches to divert the traffic to other (peer) switches. Therefore, the rules to process the traffic would be installed to the other (peer) switches. When choosing the peer switch, we always prefer to choose the switch that is (1) not full, (2) nearest to the to-be-overloaded switch, (3) less busy and (4) connects to more other switches. Therefore, the overheads on forwarding the malicious traffic can be limited to the minimal, and we can integrate more resources into the battle. We treat all the switches as peers, and they help each other to defend against flow table overloading attacks. As a result, we can leverage the flow table resources in the entire network to defend such attacks.

Also note that we do not actually wait until the flow table is full to begin guiding traffic to its peer switches, since reaching that point is pretty problematic, as the controller needs time to operate on the flow table and this may cause a disruption of service [5]. We have to reserve a space for a few entries for the controller to install rules to guide the traffic. Also, we can reserve some more spaces in each switch for other purposes, and we can set the reserved space to different sizes with minimum modification. Nonetheless, when the unused flow table space reduces to the predefined value, for simplicity, we also say that the flow table is full.

4.2.3 Workflow of the Peer Support Strategy

We present the workflow of the peer support strategy in Figure 4. The peer support process lies on the controller as an application. It contains two core modules: the status collection module and the traffic guiding module. These two modules communicate with each other to implement the peer support strategy.

In the status collection module, a switch status monitor process communicates with the switches to get the status information of the switches. The implementation of this is rather easy with the affluent functionalities of OpenFlow. We describe the implementation details in Section 4.2.4.

The traffic guiding module is the main part of the peer support strategy. When a packet arrives at the switch, if it matches an existing rule, it will be processed according the matched rule. If not, the switch will recognize the new flow and report to the controller. Then, we check whether the switch is full or not. If not full, the controller installs "normal" rules to process the flow. If the switch is full, then we have to guide its traffic to other peer switches. We find the most suitable peer switch(es), and install "traffic guiding" rules to guiding the traffic to the peer switch(es).

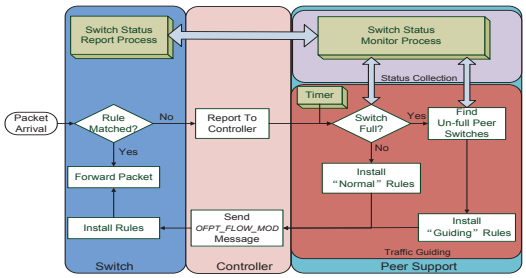


Fig. 4. Workflow of the peer support strategy.

Further, in proactive model, the controller is not invoked on each flow arrival. Therefore, we use a Timer to periodically trigger the processing of checking whether or not the switch is full. We also describe the implementation details in Section 4.2.4.

4.2.4 Implementation of the Peer Support Strategy

To implement peer support strategy, we must take the computation and communication overhead into consideration and make sure that no new security vulnerability is introduced. We list the challenges below.

- First, the scheme should be effective. It should be able to distribute the traffic to the peer switches effectively.
- Second, the scheme should incur small overhead. The computation overhead introduced to the controller, the extra rules installed to the switches, and the communication between the control and data plane should be limited to a small amount to be practical. Also, for the sake of QoS, the overheads on the packet delay should be minimized.
- Last, when distributing the traffic to the entire network, it should not introduce new networking security issues, such as network loops.

To address these challenges, we adequately investigated and exploited the affluent functionalities provided by OpenFlow. For the effectiveness and small overhead challenges, we can leverage the excellent features supported by later version of OpenFlow to reduce the computation and communication overhead.

The later version of OpenFlow [5] supports vacancy events, which enable the controller to get an early warning on the flow table space based on a capacity threshold chosen by the controller. This allows the controller to react in advance and avoid getting the table full. As a result, the controller does not need to query the switch for status information periodically and frequently, which would reduce the extra communication between the control plane and the data plane introduced by the mitigation scheme. Despite this, for the sake of backwards compatibility, we chose the OpenFlow version 1.0, which does not support the vacancy events, to implement the proposed mechanism. We let the controller to send *flow-stats-request*s to the switches to get the numbers of rules that have been installed, thus to check whether or not the switches are full. The experiments results in Setction 6 show that the overheads of our implementation are acceptable.

When the controller is informed that the flow table is full, it installs a few rules to guide the traffic to other peer switches. To

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

reduce the number of rules that need to be installed, we use the rules with wildcards to guide the traffic. A rule with wildcards can process an aggregation of multiple flows. Also, (if necessary) the *table-miss* rule, which wildcards all Match Fields, can be used to guide the other mismatched flows. These rules for guiding traffic to peer switches usually have the lowest priority, and rules for legitimate flows would have higher priority. As such, the processing of legitimate flows will not be affected, which reduces the possibility of QoS violation.

When the traffic is guided to the peer switches successfully, we leverage the sophisticated routing algorithms to find alternative paths (which is out of this paper's scope) that do not contain the to-be-overloaded switches. Therefore, the rules for forwarding the traffic are installed in the switches that exist in the alternative paths, such that the to-be-overloaded switches will not be overwhelmed and the network service will not be destroyed.

For the last challenge, our main concern was to make sure there is no loop introduced to the network. A poorly designed scheme may introduce loops. For example, switch A is full and its traffic is guided to switch B. After a while, B is full and B's traffic needs to be guided to other peers. Without properly handling, there is a chance that B's traffic is guided to A. Therefore, a loop of A-B-A is introduced to the network. To avoid this, our controller maintains a list that contains the switches whose tables are full. When choosing the supportive peers, we perform a loop check, thus to avoid introducing loops from the very beginning.

Moreover, we can leverage previous work to better solve the problem of introducing no security vulnerability. For example, VeriFlow [37] can find the faulty rules issued by SDN applications and optionally prevent them from reaching the network and causing anomalous network behavior.

4.2.5 Feasibility of the Peer Support Strategy

From the resource competition perspective presented in [20], [21], [22], with the peer support strategy, we can remarkably increase the available resource to defend attacks. Therefore, the cost of a successful attack would be much higher, because attackers would have to send many more requests to consume the flow table resources in the entire network, rather than that of a single switch. This means they have to increase the rate of attacking flow requests or extend the duration of the attack, either of which is very costly. They may not be able to make it, since their attacking resource is also limited [19], [23], [24]. Besides, the attackers usually would like to hide their attacking behaviors, and may not immoderately increase the attack strength, since the higher the attack strength is, the easier it is to detect the attack [38].

Meanwhile, under the same attacking strength, we can protect the network for a longer time. If we can protect the SDN longer than an attack's duration time, then we can beat a flow table overloading attack. To this end, our findings suggest to arrange flow table resources, in the terms of flow table size and SDN size, to mitigate attacks with different levels of attacking investment, which is beneficial on system design of SDNs.

In addition, under the nonattack scenario, the peer support strategy introduces no delay to traffic since the traffic is processed normally. Further, when under attack, without the peer support mechanism, all the traffic will be dropped, since there is no place to install new rules to process the traffic and the old legitimate rules will be replaced by attacking rules. However, with the peer support strategy in place, legitimate traffic will not be dropped, at least not all of them. We rely on sophisticated routing algorithms to find alternative paths for them. Even better, we can reserve some flow table spaces on each switch to install new rules for the new legitimate flows, such that the new flows can be forwarded along the optimal paths. As a result, other than dropping the traffic, the peer support strategy only will introduce delays to the traffic.

Finally, other protection progresses can coexist with and benefit from the peer support strategy. For example, peer support makes the network more resistant to attacks, avoiding severe damages at the beginning of the attacks. As such, the network can hold out longer when an attack is ongoing. Therefore, the attack is more likely to be detected before the network services are destroyed, as attack detection is a time-consuming job. Once the attack is detected, further intelligent operations can be executed to better protect the network. More importantly, peer support strategy aims to protect the network devices (switches). Combining peer support strategy and other network protection progresses that attempt to protect the hosts, servers and links, we can provide a more comprehensive protection to the whole network system.

5 System Modeling and Analysis

In this section, we establish an executable and practical mathematical model to represent the system, and conduct a thorough analysis of the system. We list all the notations we use in Table 1.

Like the first case study of realistic attack in SDN [13], we use holding time as the metric to measure the SDN's capacity of defending against an attack, which is the time required for attackers to overpower the switch(es).

We use queueing theory [40], [41] to approximate the flow table spaces that are consumed in each switch under nonattack scenario. Based on the observation that many properties of network topologies can be described using power laws [33], [34], [35], [36], we apply a power law distribution [32] to approximate the number of hosts that each switch serves. Combining these, we approximate the flow table space that is available to defend an attack, with and without the peer support strategy, respectively. Based on this, we calculate the holding time of SDN, with and without the peer support strategy, respectively.

In order to make our modeling, analysis, and the following experiments feasible and practical, we make a few reasonable assumptions as follows.

1) We suppose the arrival rate of new flow requests to the switches follows the Poisson distribution under the nonattack scenario. It is widely considered that the arrival of requests follows the Poisson distribution [19], [42], [43]. Although we only consider the new flow requests, which are a fraction of all requests, it is still reasonable to make the assumption that the arrival of new flow requests conforms to the Poisson distribution.
2) We suppose a rule's duration time follows the General distribution. As we discussed earlier, according to the corresponding flow's feature and the processing logic of the applications, each rule may have a different active time. Therefore, it is nature to assume that the active time of each rule is mutually independent, and follows the General distribution.
3) Last, we suppose that all switches have the same size flow table, as they are in the same network. (In fact, this assumption can be relaxed. We make this assumption only because it would be tidier to describe the system.)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

TABLE 1
Notations we use when modeling and analyzing the system.

K	number of switches in the SDN	L	size of the flow table
L_d^i	used flow table space of switch i under nonattack scenario	L_u^i	unused flow table space of switch i under nonattack scenario
λ	arrival rate of the queue	n_i	number of hosts that switch i serves
λ_0	new flow requests of host	A	offered traffic [39] of the queue
μ	service rate of the queue	π_j	probability of the system when there are j customers in the system
$E[Q]$	average number of customers in the queue system	m	average number of switches that a flow traverses
C	constant parameter of the power law distribution	α	exponent parameter of the power law distribution
S_f	available flow table space without peer support strategy	H_f	holding time of SDN without peer support strategy
S_n	available flow table space with peer support strategy	H_n	holding time of SDN with peer support strategy
R	attack rate	D	attack duration
a	ratio of λ_0 and μ	$f(K,L,a)$	metric of whether we can beat the attack

5.1 System Modeling

In this subsection, based on the assumptions, we establish a mathematical model to represent the system.

5.1.1 Approximation of the Used and Unused Flow Table Space in a Nonattack Scenario

As shown in Figure 3, we suppose there are K switches in the SDN. For any given switch i ($1 \leq i \leq K$), we suppose it serves n_i hosts. The flow table of switch i is composed of two parts, the part already used for legitimate processing and the unused part. We denote the used flow table space as L_d^i and the unused space as L_u^i. Denoting L as the length of flow table, we have

$$L = L_d^i + L_u^i. \tag{3}$$

In the OpenFlow paradigm, when a new flow arrives at a switch, if there is room left in the switch's flow table, a new rule will be installed to process the new flow. Otherwise, no rule will be installed and an error message will be sent to the controller. On the other hand, the rules will be removed due to timeout or when the controller explicitly sends a message to delete it.

We consider this process of the rules' installation and deletion as a queueing process. The rules, which are the "customers" of the queue system, arrive at the queue when they are installed, and depart from the queue when deleted. Therefore, the arrival rate of the queueing process is the rate of rule installation rate, which equals to the new flow rate, and the service time is the rule's active time. Further, there are L servers in the queueing system, since the switch can holds L rules simultaneously. Besides, the customer is immediately served when there are any servers being idle, and the customer leaves the system if all the servers are busy.

Based on our analysis, it is reasonable to use M/G/L/L model to represent the system, which means the queueing process has a Poisson arrival (rules for processing the new flows), a General distributed service time (duration time of rules), multiple servers and a finite buffer. However, the analysis will be complex for this General model, and we may not have computationally attractable methods to calculate the numerical of these models [41]. To date, the M/M/L/L model (exponential service rate) can offer a closed form result [19], [40]. We will also follow this mainstream method for our analysis on the studied system.

As aforementioned, switch i serves n_i hosts. Generally speaking, under the nonattack scenario, the behaviors of these hosts are independent. From an average point of view, we can also suppose that their behavior is identical. Therefore, denoting each host's new flow requests rate as λ_0, the arrival rate of the queue, λ, is

$$\lambda = n_i \lambda_0. \tag{4}$$

We denote the service rate of the queue as μ. Then, we denote notion A (sometimes A is called the offered traffic [39]) as

$$A = \lambda / \mu. \tag{5}$$

Based on queueing theory [39], [40], we get the probability of the system stays state π_j (namely, there are j customers in the system) is

$$\begin{cases} \pi_0 = \dfrac{1}{\sum_{j=0}^{L} \frac{A^j}{j!}} \\ \pi_j = \dfrac{A^j}{j!}\pi_0. \end{cases} \tag{6}$$

where $0 \leq j \leq L$.

From queueing theory, we know that the blocking probability equals to π_L. As a result, the average number of customers in the system, $E[Q]$, can be calculated by

$$E[Q] = (1 - \pi_L)A. \tag{7}$$

In addition, a flow usually traverses more than one switch [44], [45]. Each switch in the flow's path needs a rule to process the flow. If a flow traverses m ($m \geq 1$) switches, there will be m rules. As a result, from an average point of view, the used space of the flow table of switch i, L_d^i, can be calculated by

$$L_d^i = m \cdot E[Q]. \tag{8}$$

Therefore, according to equations from equation (3) to equation (8), we can approximate the usage of flow table spaces in switch i under the nonattack scenario, the used flow table space L_d^i and the unused flow table space L_u^i, as follows:

$$\begin{cases} L_d^i = m \cdot \dfrac{n_i \lambda_0}{\mu} \cdot \left(1 - \dfrac{\frac{A^L}{L!}}{\sum_{j=0}^{L} \frac{A^j}{j!}}\right) \\ L_u^i = L - m \cdot \dfrac{n_i \lambda_0}{\mu} \cdot \left(1 - \dfrac{\frac{A^L}{L!}}{\sum_{j=0}^{L} \frac{A^j}{j!}}\right). \end{cases} \tag{9}$$

5.1.2 Approximation of the Number of Hosts in SDN

As mentioned previously, switch i serves n_i hosts, and n_i follows power law distribution. Further, like Newman [36], we suppose that this power law holds starting from the minimum value n_{min}. This is reasonable and practical, because in the real world there are usually more than one host connecting to one switch.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

Moreover, Michalis et al. [34] and Newman [36] have observed that in the network topology case, the exponent parameter is negative and has a value of roughly -2.1. For convenience, we add a minus sign to α so that it becomes positive ($\alpha = +2.1$). In addition, it should be mentioned that we use the discrete case of power law distribution since n_i is an integer. As a result, in our particular case, n_i flows power law distribution of the form

$$\Pr[n = n_i] = C \cdot {n_i}^{-\alpha} \quad (n \geq n_{min}), \tag{10}$$

where $\alpha > 2.0$ and $n_{min} \geq 1$.

As aforementioned, constant C in equation (10) is determined by the requirement that the probability $\Pr[n = n_i]$ sum to 1. Nonetheless, when designing the network in practice, administrators have to invest enough resources at least to meet the demands of legitimate processing. This indicates that in the nonattack scenario, the demanded flow table space must not exceed the switch's flow table space. We denote n_{max} as the maximum number of hosts that a single switch can serve with its capacity. Combining these requirements and equation (9), we use equation (11) to normalize the power law's constant parameter C.

$$\begin{cases} For\ all\ n_i,\ \ n_i \cdot \dfrac{m\lambda_0}{\mu} \cdot (1 - \dfrac{\frac{A^L}{L!}}{\sum_{j=0}^{L} \frac{A^j}{j!}}) \leq L; \\ \sum\limits_{n=n_{min}}^{n_{max}} C \cdot n^{-\alpha} = 1\ . \end{cases} \tag{11}$$

5.2 System Analysis

In this subsection, we first calculate the holding time of the SDN, with and without the peer support strategy, respectively. The holding time is the metric that we use to measure the SDN's capacity of defending against attacks, Then, we give quantified condition of how to beat a flow table overloading attack.

5.2.1 System without the Peer Support Strategy

When peer support strategy is not applied, the available flow table space for defending against a flow table overloading attack is the idle flow table space of the single targeted switch. Denoting S_f (S stands for space, and "f" means turning off peer support strategy) as the available flow table space without peer support strategy, which equals to the unused flow table space, we get

$$S_f = L - m \cdot \frac{n_i\lambda_0}{\mu} \cdot (1 - \frac{\frac{A^L}{L!}}{\sum_{j=0}^{L} \frac{A^j}{j!}}). \tag{12}$$

When under attack, the new flow request is usually very high, and the flow table space is consumed very fast. The switch will be overridden when the idle flow table space is used up. For simplicity, we approximate the holding time of the SDN as the ratio of idle table space and the attack rate. Therefore, we denote R as the attack rate and H_f (H stands for holding time, and "f" means turning off peer support strategy) as the holding time of SDN without peer support strategy. We can calculate H_f by

$$H_f = [L - m \cdot \frac{n_i\lambda_0}{\mu} \cdot (1 - \frac{\frac{A^L}{L!}}{\sum_{j=0}^{L} \frac{A^j}{j!}})]/R. \tag{13}$$

5.2.2 System with the Peer Support Strategy

When applying the peer support strategy, the idle flow table spaces of all the switches can be utilized to defend against a flow table overloading attack. In this case, the available flow table space is the summary of idle spaces in all the switches' flow tables.

We denote S_n (S stands for space, and "n" means turning on peer support strategy) as the flow table space that is available to defend against a flow table overloading attack when turning on peer support strategy. Therefore, S_n is the summary of all L_u^i ($1 \leq i \leq K$). According to equation (9), we have

$$S_n = K \cdot L - \sum_{i=1}^{K} (m \cdot \frac{n_i\lambda_0}{\mu} \cdot (1 - \frac{\frac{A^L}{L!}}{\sum_{j=0}^{L} \frac{A^j}{j!}})). \tag{14}$$

Likewise, we calculate H_n (H stands for holding time, and "n" means turning on peer support strategy), the holding time of SDN when turning on peer support strategy, by

$$H_n = [K \cdot L - \sum_{i=1}^{K} (m \cdot \frac{n_i\lambda_0}{\mu} \cdot (1 - \frac{\frac{A^L}{L!}}{\sum_{j=0}^{L} \frac{A^j}{j!}}))]/R. \tag{15}$$

Comparing equations (13), and (15), we can see that, H_n is roughly K times of H_f, which indicates that the peer support strategy can improve an SDN's capacity of defending against a flow table overloading attack by a factor of K, which is the number of switches in the SDN.

5.2.3 Quantifying the Condition of Beating Flow Table Overloading Attacks

In general, if a system can hold out longer than the duration of an attack, we say that the attack is beaten. Thus, to prevent the SDN from flow table overloading attacks, we have to make sure that the holding time of the SDN exceeds the attack duration. We denote D as attack duration. Thus, we conclude that, with the peer support strategy, we can beat a flow table overloading attack in SDN if the following expression holds.

$$H_n \geq D. \tag{16}$$

For simplicity, given the attacking strength (represented by attack duration D and attack rate R), let

$$f(K, L, a) = K \cdot L - \sum_{i=1}^{K} (a \cdot m \cdot n_i \cdot (1 - \frac{\frac{(a \cdot n_i)^L}{L!}}{\sum_{j=0}^{L} \frac{(a \cdot n_i)^j}{j!}})) - D \cdot R. \tag{17}$$

where a is the ratio of λ_0 and μ, which can indicate the host's workloads. However, note that a is NOT the same notion of *busy rate* in the M/M/1 model, thus, it is NOT necessary to make sure that a is smaller than 1 to keep the system steady.

Combining equations (15), (16), and (17), the quantified condition of beating a flow table overloading attack in SDN is given by

$$f(K, L, a) \geq 0. \tag{18}$$

6 Performance Evaluation

In this section, we evaluate the performance of our proposed approach. First, we study the effectiveness of the flow table overloading attacks. Then, we evaluate the effectiveness of our proposed peer support strategy with testbed-based experiments and simulations. To investigate the overheads of the proposed strategy, we also evaluate the performance penalty on the peer switch.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

6.1 Key Parameters of Our Experiments

First of all, we summarize some key parameters of our experiments based on empirical data and the previous solid works at top journals and conferences.

As aforementioned, the flow table size, L, is set to 8000 [4]. The rate of new flows of typical network has been studied, concluding that an 8000 hosts network generated a peak rate of 1200 new flows per second [46], [47]. Therefore, we use this conclusion to approximate the host's new flow requests rate λ_0, which is 0.15, roughly.

Benson et al. [48] have studied the characteristics of data center traffic, and data center is the most widely deployed environment of SDN until today. It has been observed that most flows are short flows, lasting less than several tens of seconds. Thus, we set the rules' timeout value (which equals to the active time of the rules) to 30 seconds. Therefore, the service rate μ is $1/30$, and we obtain that a is 4.5 under nonattack scenario.

Further, we summarize the key statistics of DDoS attacks in a global scenario from previous highly referred literature [19], [24]. Moreover, the average number of hops per flow in a data center network is 5 to 6 [44], [45]. Therefore, we set the average number of switches that a flow traverses (m) to 5 if there are more than 20 switches in the SDN. Otherwise, we set m to a relatively small number, 3.

We list these key parameters in Table 2. In addition, we generate a set of n_i that follows power law distribution to represent the number of hosts that each switch serves, using the method proposed by Clauset et al. [49], [50].

TABLE 2
Key parameters of our experiments.

Feature	Value
Flow table size	8000 entries
Ratio of new requests rate and service rate	4.5
Average attack duration	5 minutes
Average attack rate	500 requests per second

As shown in Table 2, under an average situation, the parameter a, which can indicate the host's workloads, is 4.5. To be more practical, we attempt to study the effectiveness of our proposed strategy when the SDNs are under different workloads. Therefore, from now on, we set a to three different values: 3.0, 4.5, and 6.0, to represent the scenarios of SDNs with different workloads.

6.2 Effectiveness of the Flow Table Overloading Attacks in SDNs

To study the effectiveness of the flow table overloading attacks in SDNs, we simulate the attacking process in SDNs with an average attack rate. As discussed previously, we use holding time as the metric of an SDN's capacity of defending against a flow table overloading attack. Further, based on our analysis, the size of an SDN has an impact on the holding time. Therefore, we measure the SDN's holding time with the number of switches (K) from 1 to 50. We present the results in Figure 5.

From Figure 5, we can see that, regardless of the number of switches in the network, the holding time changes very little under the same workload (a). This is because, when not applying the peer support strategy, the SDN can only use the resources of the single targeted switch to counter the attack. Generally speaking, the amount of resources in the single targeted switch are not much

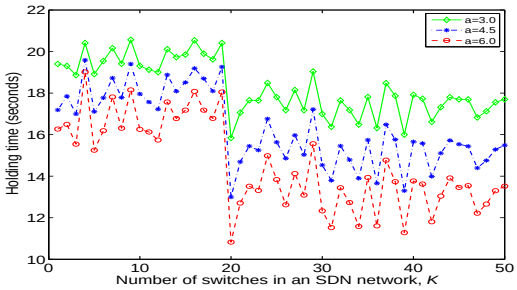


Fig. 5. Effectiveness of a flow table overloading attack (with an attack rate of 500 requests per second).

different no matter how many switches the SDN has. Thus, the holding time would not be much different, either.

Also, recall that different values of a means different workloads of the network. A smaller a indicates lighter workloads and less used flow table space. As a result, a network with smaller a has larger available flow table space for defending against a flow table overloading attack. Therefore, an SDN with smaller a would have a longer holding time, which can be observed from Figure 5.

In addition, there is a obvious decline in each line when the number of switch increases to 20. This is because we set the average number of switches that a flow traverses (m) to 5 when there are more than 20 switches in the SDN. Otherwise, we set m to 3. A larger m means that a flow traverses more switches, and each of these switches needs to install a rule to process the flow. Therefore, a larger m indicates that more flow table spaces in the whole SDN are used, and that less idle spaces are left. As a result, when m increases, the holding time of the SDN decreases.

More importantly, the holding time of an SDN (regardless of the workloads) when not applying the peer support strategy is far shorter than the average attack duration time (5 minutes). This means that without a mitigation strategy, attackers can easily overpower a switch in an SDN, which is also reflected in Figure 2. This is because a single switch has very limited flow table resources. If we can optimize the resource management strategy and integrate more flow table resources to defend against the attack, the SDN will be much more resistant to such attacks.

6.3 Effectiveness of the Peer Support Strategy

Having verified the effectiveness of the flow table overloading attacks, we now validate the correctness of our strategy with testbed experiments (Section 6.3.1). Then, we set the parameters based on empirical data and use Matlab to evaluate the effectiveness of our approach with simulations (Setcion 6.3.2).

6.3.1 *Correctness of the Peer Support Strategy*

To validate the correctness of our proposed strategy, using Mininet (version 2.1.0P1) [51], we created an SDN testbed, which is of mesh topology and contains ten switches. We used POX (version 0.2.0) [52] as the SDN controller. For the sake of backwards compatibility, we implemented our approach as an application on the controller with OpenFlow 1.0. We run the experiments in a Mininet virtual machine running Ubuntu Linux 14.04. The VM instance was allocated a single CPU core and 1 GB of RAM.

In each experiment, we attacked a different switch. We measured the holding time with and without the peer support strategy respectively. To measure the holding time, we recorded the time

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

when the attack starts (denoted as T_0), and periodically send *flow_stats_request* to the data plane to get the number of rules that have been installed in each switch. We record the time when the target switch is full (denoted as T_1) and the time when all the switches are full (denoted as T_2). Therefore, the holding time without peer support strategy is $T_1 - T_0$, and the holding time with peer support strategy is $T_2 - T_0$.

Note that, the holding time we measure includes the delay between the controller and the switches. However, since the delay is at the milliseconds level [53] and the holding time is at the seconds level, we ignore the delay's impact on the holding time.

To simplify the diagram, we normalized the holding time without the peer support strategy. The results are presented in Figure 6. Comparing these two holding times, we can see how our proposed approach can help with mitigating the attack.

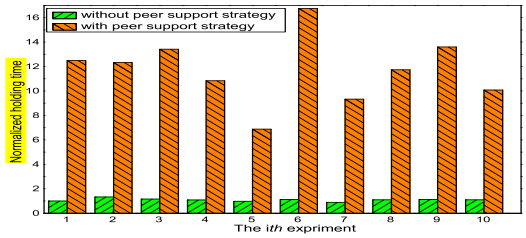


Fig. 6. SDN's enhanced capacity of defending against a flow table overloading attack when applying the peer support strategy. For better understanding, we normalized the holding time.

It should be noted that, in the real world, an attack may stop before it overpowers the switch(es). However, to evaluate the maximum resources that can be integrated when applying our strategy, in our experiments, we kept sending attacking traffic until the all the switches were overpowered.

As we can see from Figure 6, with the peer support strategy, the holding time of the SDN is roughly increased by 10 times (averagely), which is the number of switches in the SDN. The result is consistent with our analysis. This indicates that our proposed strategy can remarkably enhance the SDN's capacity of defending against a flow table attack.

6.3.2 Holding time of SDN with the Peer Support Strategy

With parameters being properly approximated and correctness being verified, we started to evaluate the situation of using our proposed peer support strategy to mitigate flow table overloading attacks. In order not to introduce other influence factors, we used the same configuration and metric that were used when we studied the effectiveness of attacks.

At first, we set the attack rate to current average attack rate (500 requests per second) and present the results in Figure 7. We can see that the holding time increases linearly with increasing K when applying the peer support strategy. This indicates that with our proposed strategy, we can integrate much more idle flow table space in the entire network to counter the attack, rather than that of only the targeted switch's idle space. Therefore, the SDN's capacity of defending against a flow table overloading attack is improved remarkably.

Note that the workloads (a) and flows' average number of hops (m) also have small impacts on the holding time. The reason is very much the same with the situation when not applying peer support strategy.

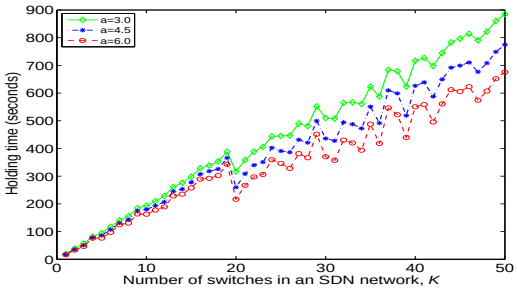


Fig. 7. Holding time of SDN when applying the peer support strategy with attack rate of 500 requests per second.

Then, we let a be constant (since the workloads only have very limited impacts on the results) and set attack rate to different values to better investigate the performance of our proposed mitigation scheme. We set a to 4.5 (Recall that, $a = 4.5$ indicates an average workload) and present the result in Figure 8.

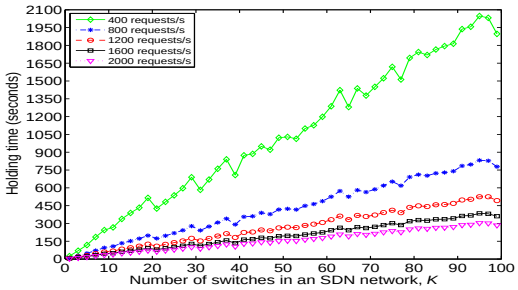


Fig. 8. Holding time of an SDN when applying the peer support strategy with $a = 4.5$; rps (requests per second) is the attack rate.

From Figure 8, we can see that, with the peer support strategy, the SDN can hold for more than 5 minutes when under attack if it has enough switches, regardless of the attack rate. This means that, with proper resource investment, the SDN is capable of beating the flow table overloading attack with average attack strength.

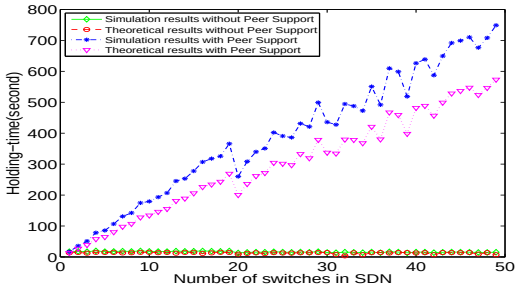


Fig. 9. Holding time comparison of theoretical results and simulation results with $a = 4.5$, $R = 500$ and $L = 8000$.

To illustrate how consistent the theoretical model is to the simulation scenarios, we compare the theoretical results and simulation results in Figure 9. We apply the same parameters used in the simulations to equation (13) and equation (15) to calculate the theoretical results.

From Figure 9, we can see that the theoretical model is quite consistent to the simulation scenarios, only that the simulation holding time is a little bit larger than the theoretical holding time and that the gap increases with the increasing number of switch.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

This is because, in real world networks, there are lots of delays, such as transmission delay, propagation delay, computation delay, communication delay, etc. All of these delays are excluded in the theoretical results. Moreover, in the real world SDN systems, the legitimate rules may be deleted during the attack. Therefore, an attack needs a little extra time to consume these flow table spaces.

6.4 Beat the Flow Table Overloading Attack

The simulation results show that with the proper configuration, we can beat a flow table overloading attack in an SDN. Now, based on equation (18), we quantify the condition of the SDN's configuration in order to beat the attack, thus making a suggestion in the terms of flow table size (L) and SDN size (K).

We focus on the attacks with current average attacking strength. Therefore, we set the attack duration D in $f(K,L,a)$ to 300 seconds and the attack rate R to 500 requests per second. Nonetheless, if the average attacking strength changes, we can reset D and R to re-obtain the results.

To match our previous experiments, we conducted three experiments for $a = 3.0$, 4.5, and 6.0, respectively. As shown in Figure 10, for a given a, we observe the variation of $f(K,L,a)$.

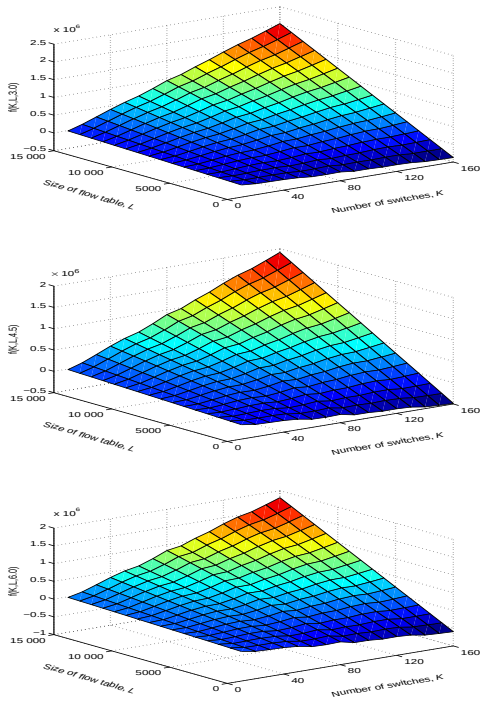


Fig. 10. The SDN's configuration requirements (in the terms of flow table size L and SDN size K) of beating a flow table overloading attack with average attacking strength. $f(K,L,a) \geq 0$ indicates that the SDN is capable of beating the attack.

As we discussed earlier, $f(K,L,a)$ is the metric of whether we can beat the attack or not. If $f(K,L,a) < 0$, the network services will be destroyed by the attack. If $f(K,L,a) \geq 0$, we can beat the attack and protect the SDN successfully.

From Figure 10, we can see that, when both the size of the flow table (L) and the size of SDN (K) are small, $f(K,L,a) < 0$, which indicates that the SDN is not capable of beating a flow table overloading attack with average attacking strength. We can also see that, if the flow table size remains small, to meet the requirements of $f(K,L,a) \geq 0$, the SDN size has to increase tremendously. Likewise, the flow table size has to increase extremely to beat the attack if the SDN size remains small. The ideal situation is that both the flow table size and the SDN size increase at the same time. This way, with reasonable values of flow table size and SDN size, we can beat the attack.

In order to have a straight concept of the configuration requirements on flow table size and SDN size to beat an average attack, we list some of the numerical results from Figure 10 in Table 3.

TABLE 3
Some numerical mitigation requirements on flow table size (L) and SDN size (K) with $D = 300$, $R = 500$ and $a = 4.5$.

Given K	10	20	50	100	150
Minimum L	16286	9294	4232	2834	2297
Given L	2000	2500	5000	7500	10 000
Minimum K	195	120	40	24	17

From Table 3, we can see that, the product of corresponding K and L is always larger than 150 000, which is the product of D and R, which indicates that we have to invest enough resources to beat flow table overloading attacks, namely, an SDN with small size (K) requires switches with larger flow table space (L), and vice versa. Therefore, if SDN size is given, network administrators have to choose the switches that have larger flow table space. Likewise, if the flow table size is given, network administrators have to add more switches to the network.

All in all, we conclude that, to beat flow table overloading attacks, we at least have to make sure $KL > DR$ holds all the time. To be more specific, to beat the attacks with current average attacking strength, we at least have to make sure $KL > 150\ 000$ holds. Further, since the attack strength is increasing all the time and different networks have different characteristics, the network administrators should invest flow table resources dynamically, regarding to the specific attack strength and the unique features of their own networks.

6.5 Performance Penalty on the Peer Switch

In this subsection, we evaluate the performance penalty on the peer switch that supports the target switch to defend the attack.

The main concern on performance penalty is the time that the peer switch needs to process the legitimate packets. There are two major factors that affect the processing time, the table lookup time and the usage of the switch's CPU.

When under attack, there will be much more rules installed into the peer switch's flow table, therefore, the switch needs more time to find the rules for the legal flows. Moreover, the more rules installed in the flow table, the longer it takes to find the corresponding rules. Therefore, when evaluating the performance penalty, we measure the processing time under different scenarios when there are different number of rules in the peer switch.

The peer switch would have a higher CPU usage when under attack since it has to process much more traffic. A higher CPU usage means a longer time that the legal flows have to wait to be processed, and the CPU usage is proportional to the attack rate. Also, according to the peer support strategy, when the peer switch is full, the peer switch only needs to forward the extra traffic to the next peer switch. However, when the peer switch is not full, in reactive model, it has to generate extra *Packet-in* messages to report the new attacking flows and receive *Flow-mod* messages to install rules, and in proactive model, it also has to generate new rules. Obviously, generating messages and installing rules need

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2602861, IEEE Transactions on Services Computing

more CPU resources. As a result, we measure the processing time with different attack rate when the peer switch is full and not full, respectively.

Moreover, the legal flows consist of exiting flows and new flows. The legal exiting flows are processed directly according the existing rules, while the legal new flows need new rules to process them. Therefore, we evaluate the performance penalty on the existing flows and new flows separately.

We measure the processing time of legal flows with the Ping command under different scenarios as discussed above. We also measure the processing time when there is no attack as the comparing base. As aforementioned, there are lots of factors that can affect the results. Each factor's impact on the processing time changes when other factors change. Therefore, we omit the tedious details of the processing time under each scenario in which different factors have different values. Instead, from the average point of view and for simplify of illustration, we calculate the average values of those times and present them in Figure 11.

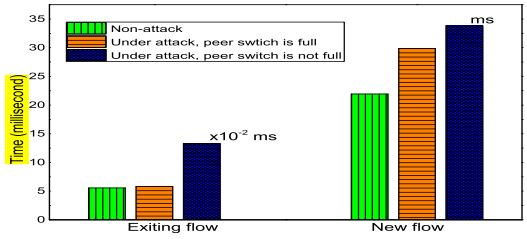


Fig. 11. Average processing times of existing flow and new flow under different scenarios with Ping command.

From Figure 11, we can see that, when under attack the peer switch needs more time to processing the legal packets. Also, when the peer switch is not full, it need more time to process the packets than that when the peer switch is full. However, the max extra time is about 0.08 ms and 12 ms for the existing flow and the new flow, respectively. Moreover, the tolerance of latency in data center can be up to 100 ms [54], [55]. As a result, we believe the performance penalty on the peer switch is acceptable.

7 Limitations and Further Discussion

To the best of our knowledge, this paper is the first work to discuss how to mitigate flow table overloading attacks in SDNs. Due to the space and time limitations, we have only discussed our mechanism in this paper. As a new research field, there are many issues to be further investigated and improved. Based on our understanding, we discuss some of them here.

First of all, to defend against flow table overloading attacks, a straightforward solution is to simply decrease the timeout of the flow entries. However, in our opinion, this is not promising. Decreasing the timeout for flow entries means that the switches have to communicate with controller for multiple times to build up the same flow, which would add distinct extra delay to the flows. What's worse, the superabundant communication between the switch and the controller will consume lots of resources, which may bring new threats the SDN, such as the link saturation attack and the control plane resource consumption attack [14].

Secondly, our proposed solution allows (potentially) malicious traffic enter the network. Since the purpose of our work is to protect the switches from being overpowered, we suppose that the protection scheme of the hosts or servers is ideal. That is, even if the attacking traffic enters the network, the protection system for the hosts or servers is capable of preventing the hosts or servers from being victimized. We are aware of this limitation of our solution. However, our work does can make SDNs more resistant to attacks, such that, the attacks are more likely to be detected before any severe damages are made, thus to be beaten by more complex and intelligent schemes.

Thirdly, the topology of the SDN can affect our solution. For instance, a sophisticated attacker may coordinately attack several switches located in certain area, making peer support strategy less effective. We believe that this problem can be solved by erratically changing the peer switches. That is, based on the switches' status, we can always use the most feasible switches to support the victim switch by changing the action fields of the guiding rules. Alternatively, we can leverage the global view of the controller to calculate a global optimal peer-supporting path. The controller can always know the changes in the underlying network and collect information of the whole network, such as the list of to-be-overloaded switches, the busy rate of the switches, the topology of the network, etc. Based on these information, the controller can periodically and dynamically calculate a global optimal peer-supporting path, and switch to the new path by replacing the old guiding rules with new ones.

Fourthly, though the attack strength is limited in practical, and it is rare that an attacker can launch attacks to consume all the switch table resources among the whole network, a more strict rate limiting scheme is needed. That is, when the number of full switches reaches an upper limit (for example, 80% of the switches are full), the target switch has to stop forwarding traffic to more peer switches. With rate limiting, we can further reduce the possibility of the collapse of the whole network, even when facing with attacks that are of extreme strong attack strength.

Finally, for simplicity, we treat all the switches in the SDN equally. In practice, differently located switches have different status, for example, different workloads, and different used/unused flow table spaces. Researchers may take this into consideration for more practical system modeling and analysis.

8 Conclusion and Future Work

In this paper, we point out that, because of the limitation of flow table resource in OpenFlow-enabled switches, SDNs are vulnerable to the *flow table overloading attack*, which is a transformed DDoS attack that targets SDN devices. We propose a QoS-aware *peer support* strategy, which can integrate idle flow table resources in the whole SDN system as many as possible to mitigate such attacks. At the same time, the proposed strategy can minimize the QoS violation. We establish a practical model to approximate the capacity of SDN in defending against such attacks. We thoroughly analyze the proposed model. Extensive experiments confirm our claim that we can remarkably improve the SDN's capacity of defending against the flow table overloading attack. Based on quantified experiment results, we also provide theoretical guidance on how to arrange SDN flow table resources against different levels of attack strength.

As a new research area, there are lots of work expected to be completed in the near future. As future work, we will first attempt to improve the M/M/K/K model to a more general and practical model, such as G/G/K/K model. Second, we expect to design more sophisticated algorithm to choose the peer switch(es), to reduce the impact of network topology. Third, we attempt to dynamically

invest extra resources when the existing resources are used up, to further improve the resistance of the network. In addition, we expect to find methods to differentiate the attacking flows from legitimate ones in real time, so that we can drop the attacking traffic before they enter further in the network, thus to solve the problem fundamentally.

ACKNOWLEDGMENTS

We thank Weiming Li and ZhenShan Cao for their valuable help. This paper was supported by National 973 Fundamental Basic Research Program under grant No. 2014CB340600.

REFERENCES

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.
[2] O. M. E. Committee, "Software-defined networking: The new norm for networks," *Open Networking Foundation*, 2012.
[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
[4] D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
[5] ONF, "OpenFlow switch specification version 1.4.0," 2013.
[6] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on sdn network utilization," in *Proceedings of the 33th IEEE International Conference on Computer Communications*, 2014, pp. 1734–1742.
[7] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam, "Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane," in *Proceedings of European Workshop on Software Defined Networking*, 2012, pp. 79–84.
[8] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proceedings of the 32th IEEE International Conference on Computer Communications*, 2013, pp. 545–549.
[9] 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 ACM International Conference on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2011, pp. 254–265.
[10] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013, pp. 55–60.
[11] R. Klöti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *Proceedings of the 8th IEEE ICNP Workshop on Secure Network Protocols*, 2013, pp. 1–6.
[12] K. Benton, L. J. Camp, and C. Small, "Openflow vulnerability assessment," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013, pp. 151–152.
[13] S. Shin and G. Gu, "Attacking software-defined networks: a first feasibility study," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013, pp. 165–166.
[14] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 20th ACM Conference on Computer and Communications Security*, 2013, pp. 413–424.
[15] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.
[16] J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the source," in *Proceedings of the 10th IEEE International Conference on Network Protocols*, 2002, pp. 312–321.
[17] C. Jin, H. Wang, and K. G. Shin, "Hop-count filtering: an effective defense against spoofed DDoS traffic," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 2003, pp. 30–41.
[18] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against DDoS attacks," in *Proceedings of the 9th ISOC Network and Distributed System Security Symposium*, 2002.
[19] S. Yu, Y. Tian, S. Guo, and D. Wu, "Can we beat DDoS attacks in clouds?" *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2245–2254, 2014.
[20] S. Yu, S. Guo, and I. Stojmenovic, "Can we beat legitimate cyber behavior mimicking attacks from botnets?" in *Proceedings of the 31st IEEE International Conference on Computer Communications*, 2012, pp. 2851–2855.
[21] Y. Chen, K. Hwang, and W.-S. Ku, "Collaborative detection of DDoS attacks over multiple network domains," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 12, pp. 1649–1662, 2007.
[22] J. François, I. Aib, and R. Boutaba, "Firecol: a collaborative protection network for the detection of flooding DDoS attacks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1828–1841, 2012.
[23] M. A. Fabian, R. J. Zarfoss, and M. A. Terzis, "My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging," in *Proceedings of the 1st USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.
[24] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," *ACM Transactions on Computer Systems*, vol. 24, no. 2, pp. 115–139, 2006.
[25] D. K. Yau, J. Lui, F. Liang, and Y. Yam, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 29–42, 2005.
[26] S. Yu, W. Zhou, W. Jia, S. Guo, Y. Xiang, and F. Tang, "Discriminating DDoS attacks from flash crowds using flow correlation coefficient," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 1073–1080, 2012.
[27] R. Braga, E. Mota, and A. Passito, "Lightweight DDos flooding attack detection using nox/openflow," in *Proceedings of the 35th IEEE Conference on Local Computer Networks*, 2010, pp. 408–415.
[28] J. Suh, H. Choi, W. Yoon, T. You, T. Kwon, and Y. Choi, "Implementation of content-oriented networking architecture (cona): A focus on DDoS countermeasure," in *Proceedigns of European NetFPGA Developers Workshop*, 2010.
[29] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and software-defined networking," in *Proceedings of the 9th IEEE Workshop on Secure Network Protocols*, 2014, pp. 624–629.
[30] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," in *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies*, 2013, pp. 13–24.
[31] SDxCentral, https://www.sdxcentral.com/resources/sdn/what-the-definition-of-software-defined-networking-sdn/.
[32] M. E. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.
[33] A. Medina, I. Matta, and J. Byers, "On the origin of power laws in internet topologies," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 2, pp. 18–28, 2000.
[34] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 251–262, 1999.
[35] A. Li, X. Li, Y. Pan, and W. Zhang, "Strategies for network security," *Science China Information Sciences*, vol. 58, no. 1, pp. 1–14, 2015.
[36] M. Newman, *Networks: an introduction*. Oxford University Press, 2010.
[37] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, "Veriflow: verifying network-wide invariants in real time," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.
[38] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1530–1539.
[39] M. Zukerman, *Introduction to queueing theory and stochastic teletraffic models*, 2007.
[40] L. Kleinrock, *Queueing systems, volume I: theory*. Hoboken, NJ, USA:Wiley, 1975.
[41] J. Kingman, "The first erlang century - and the next," *Queueing Systems*, vol. 63, no. 1-4, pp. 3–12, 2009.
[42] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 266–277, 2011.
[43] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, 2015, pp. 24:1–24:7.
[44] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *Proceedings*

of the 33th IEEE International Conference on Computer Communications, 2014, pp. 2598–2606.

[45] Y. Shimotsuma, Y. Tarutani, Y. Ohsita, and M. Murata, "Evaluation of data center network structures considering routing methods," in *Proceedings of the 9th International Conference on Networking and Services*, 2013, pp. 146–152.

[46] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. M. Parulkar, "Can the production network be the testbed?" in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementations*, vol. 10, 2010, pp. 1–6.

[47] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "A first look at modern enterprise traffic," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, 2005, pp. 15–28.

[48] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference*, 2010, pp. 267–280.

[49] A. Clauset, C. R. Shalizi, and M. E. Newman, "Power-law distributions in empirical data," *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[50] Y. Virkar, A. Clauset *et al.*, "Power-law distributions in binned empirical data," *The Annals of Applied Statistics*, vol. 8, no. 1, pp. 89–119, 2014.

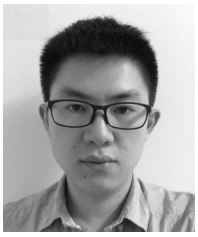
[51] "Mininet," http://mininet.org/.

[52] "Pox," http://www.noxrepo.org/pox/about-pox/.

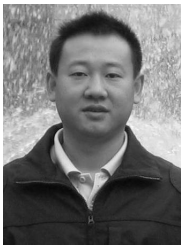
[53] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Latency in software defined networks: Measurements and mitigation techniques," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2015, pp. 435–436.

[54] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 50–61, 2011.

[55] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications,*, 2010, pp. 63–74.



Bin Yuan received the BS degree in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2013, where he is currently working toward the PhD degree in computer science and technology. His research interests include security in SDN and cloud computing.



Deqing Zou is a Professor of Computer Science at HUST. He received his PH.D at HUST in 2004. His main research interests include system security, trusted computing, virtualization and cloud security. He has been the leader of one "863" project of China and three NSFC (National Natural Science Foundation of China) projects, and core member of several important national projects, such as National 973 Basic Research Program of China. He has applied almost 20 patents, published two books (one is entitled "Xen virtualization Technologies" and the other is entitled "Trusted Computing Technologies and Principles") and more than 50 High-quality papers, including papers published by IEEE Transactions on Dependable and Secure Computing, IEEE Symposium on Reliable Distributed Systems and so on. He has always served as a reviewer for several prestigious Journals, such as IEEE TPDS, IEEE TOC, IEEE TDSC, IEEE TCC, and so on. He is on the editorial boards of four international journals, and has served as PC chair/PC member of more than 40 international conferences.

Shui Yu received the B.Eng. and M.Eng. degrees from University of Electronic Science and Technology of China, Chengdu, China, in 1993 and 1999, respectively, and the Ph.D. degree from Deakin University, Victoria, Australia, in 2004. He is currently a Senior Lecturer with the School of Information Technology, Deakin University, Victoria, Australia. He has published nearly 100 peer review papers, including top journals and top conferences, such as IEEE TPDS, IEEE TIFS, IEEE TFS, IEEE TMC, and IEEE INFOCOM. His research interests include networking theory, network security, and mathematical modeling.

Dr. Yu actively servers his research communities in various roles, which include the editorial boards of IEEE TPDS, and three other International journals, IEEE INFOCOM TPC members, symposium co-chairs of IEEE ICC 2014, IEEE ICNC 2013 and 2104, and many different roles of international conference organizing committees. He is a senior member of IEEE, and a member of AAAS.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at HUST. Jin received his PhD in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientists of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security.

Jin is a Fellow of CCF, senior member of the IEEE and a member of the ACM. He has co-authored 22 books and published over 700 research papers. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Weizhong Qiang is an Associate Professor of Computer Science at HUST. He received his Ph.D at HUST in 2005. His research interest is mainly about software system security. He has published more than 30 research papers.

Jinan Shen received his MS degree from computer school of Wuhan University, China, in 2009. Currently he is a PhD candidate at HUST. His research interests are in the area of security in cloud computing, focusing on privacy preserving in the cloud.