SDN-based Dynamic and Adaptive Policy Management System to Mitigate DDoS Attacks

Rishikesh Sahay
Institut Mines-Télécom, Télécom SudParis
rishikesh.sahay@telecom-sudparis.eu

Gregory Blanc
Institut Mines-Télécom, Télécom SudParis
gregory.blanc@telecom-sudparis.eu

Zonghua Zhang
Institut Mines-Télécom, IMT Lille Douai
zonghua.zhang@imt-lille-douai.fr

Khalifa Toumi
Institut Mines-Télécom, Télécom SudParis
khalifa.toumi@telecom-sudparis.eu

Hervé Debar
Institut Mines-Télécom, Télécom SudParis
herve.debar@telecom-sudparis.eu

ABSTRACT

This paper presents a dynamic policy enforcement mechanism that allows ISPs to specify security policies to mitigate the impact of network attacks by taking into account the specific requirements of their customers. The proposed policy-based management framework leverages the recent Software-Defined Networking (SDN) technology to provide a centralized platform that allows network administrators to define global network and security policies, which are then enforced directly to the OpenFlow switches. One of the major objectives of such a framework is to achieve fine-grained and automated attack mitigation in the ISP network, ultimately reducing the impact of attack and collateral damage to the customer networks. To evaluate the feasibility and effectiveness of framework, we develop a prototype that serves for one ISP and three customers. The experimental results demonstrate that our framework can successfully reduce the collateral damage on a customer network caused by the attack traffic targeting another customer network. More interestingly, the framework can provide rapid response and mitigate the attack in a very short time.

Keywords

Security policy, Policy management, SDN

1. INTRODUCTION

In today's Internet, traffic engineering is mainly performed by the Internet Service Providers (ISP), while the customers are usually passive. As we know, one of the major objectives of traffic engineering is to mitigate traffic congestion, which can be caused, among others, by attacks. The lack of collaboration between an ISP and its customers may eventually lead to dissatisfaction among its customers, as legitimate traffic may get dropped. For example, when defending against Distributed Denial of Service (DDoS) attacks that attempt to deplete an ISP's bandwidth, simply prioritizing legitimate traffic or redirecting suspicious traffic for one customer may impact other customers of the same ISP, considering the fact that the same path can be shared between different customers in an ISP network.

As a matter of fact, without collaborating with their ISPs, customers do not have much control over the incoming traffic, apart from blocking the attack traffic at their border router. Therefore, it is in the interest of both the victim network and its ISP to collaborate for traffic engineering to mitigate the effect of congestion. In this case, the customer can express finer requirements that can be addressed as differentiated services by the ISP. Despite a large number of solutions [8, 10] proposed for traffic engineering, they have not been considered for widespread deployment, chiefly due to the complexity involved in the network management task, such as configuring switches and routers for policy enforcement. According to a report from Juniper [12], the network downtime due to human error accounts for 80% of the total network downtime. The manual configuration therefore hinder the dynamic deployment of network services, further downgrading the Quality of Service (QoS) level for the customers of an ISP.

Another fact is that service providers statically provision security devices with the dedicated network devices, and define the ordering constraints that must be applied to the packets [11]. These security and network devices are generally distributed in the network through separate VLANs, while network policy is usually applied per VLAN. This essentially leads to static service chaining with the deployment of static policies for steering network traffic to the security and network devices [11]. This topological dependency with the deployment of *middleboxes* makes ISPs reluctant to deploy new security functions in their networks for providing security services to their customers. Furthermore, all the traffic,

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

whether it needs to be processed through the security devices or not, eventually traverse these devices, which causes processing overhead on these devices. Therefore, a dynamic and automated policy management system is required to overcome these issues.

Software-Defined Networking (SDN) recently emerges as a novel networking paradigm that can simplify network administration and management by centralizing the decision-making process [13]. It also provides the administrator a global view of the network and enables programmability in the data plane. By leveraging features of SDN, we develop an automated policy management system in which the ISP can express high-level policies that can be enforced dynamically as the network environment changes. Specifically, our policy management system provides collaborative and user centric automated response for mitigating the attack traffic and providing the QoS service to the customers of the ISP. Customers can express their requirements to the ISP, and based-on the requirements ISP can deploy the policies to fulfill its customer' requirements. Our policy management system reduces the collateral damage caused by the attack traffic on its multiple customers.

In this paper, we implement our proposed framework for a specific use case, where the ISP and their customers collaborate with each other to mitigate the effect of congestion caused by DDoS attack. We also experimentally demonstrate that the framework can help an ISP to provide good QoS to legitimate traffic, while reducing the impact on other customers' traffic.

The remainder of this paper is organized as follows: Section 2 provides some related works Section 3 describes our policy framework, its workflow and functional components. Section 4 reports our experiments and results. Section 5 concludes the paper.

2. RELATED WORK

To the best of our knowledge, there is a number of works dealing with policy-based network management leveraging the SDN paradigm [1, 2, 7, 9]. They usually exploit key features such as data plane programmability and network visibility to ease the network management process. In this section, we will discuss existing policy-based frameworks, the policy languages that inspired them, and earlier proposals in traffic steering.

Traffic steering is an exemplary instance of how to take advantage of SDN switches to enforce routing policies in an efficient manner for middlebox-specific networks. One such effort, SIMPLE [14], alleviates manual operation from administrators by allowing them to specify a logical routing policy and translates it into forwarding rules, in compliance with physical resource constraints. Additionnaly, FlowTags [4] provide a technique to enforce network-wide policies in spite of packet modifications imposed by middleboxes.

At a higher level, policy languages have been proposed to specifically program software-defined networks. Languages, such as Frenetic [6], free programmers from reasoning with low-level details of the switches and allow them to describe high-level packet forwarding policies on top of the control plane. Another example, Procera [17], extends policy design into event-driven network control, which is not permitted by configuration languages exposed by controllers. These languages usually offer the ability to specify routing policies, in terms that are close to network operations, while we are interested in specifying higher-level policies to enforce security or quality of service operations.

Policy management frameworks would often rely on the above-mentioned technological building blocks to satisfy user-centric requirements. EnforSDN [2] proposes to simplify network service management by decoupling policy resolution (computing concrete rules) from policy enforcement (pushing low-level rules) by remarking that the former deals with flows, e.g., security policies, while the latter forwards packets at the data plane. It tackles middlebox-induced problems but fails to accommodate other contexts than the network. PolicyCop [1] is such a QoS policy enforcement framework that provides an autonomic management of user-centric policies by monitoring and enforcing the users' service-level agreements (SLAs). It relies on common data plane interfaces exposed by OpenFlow forwarding devices for statistics collection and flow information retrieval, and includes a number of controller applications to support policy monitoring and enforcement. Our work is interested in extending the inputs to the policy engine with security events. Additionally, the computation of policy routes do not take into account the availability of security services. Business-level goals are also taken into account in a policy authoring framework proposed by Machado et al. [9]. Their framework matches these requirements with the capacities provided by the network infrastructure in order to decide on an appropriate policy, through abductive reasoning. It is however unclear how network elements are requested and configured beyond the policy path computation (referred to as *Analysis Phase* in their work). OpenSec [7] is close to our proposal in that this framework allows provides a language that blends security services within the autonomic reaction process. However, it seems to focus the deployment on edge switches, while we are interested in distributing the policies across the controller's network domain.

Our work aims at accommodating multiple customer services sharing a network service provider, who doubles as a security service provider. Going beyond routing and QoS requirements, we aim at reacting to security events, with the collaboration of the customers, and offer network-status-aware security reaction policies. The presence of multiple competing customers raises a supplementary challenge in that the reactive policy targeted at a given customer should little to no impact on other customer services. It is important to consider the whole network then, and not only the edge switches, in order to distribute the rules along the policy path. This path traverses a number of forwarding switches and security services (either static middleboxes, or virtualized network functions) in a fashion similar to OpenSec [7] and SIMPLE [14].

3. POLICY MANAGEMENT AND ENFORCEMENT SYSTEM

Our previous analysis shows that most of the existing policy

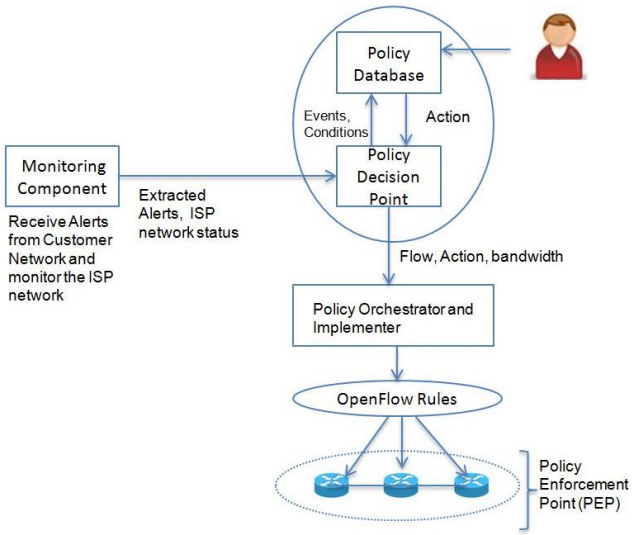

Figure 1: Workflow of the Policy Management System

management frameworks does not take into consideration the real-time collaborations between ISPs and customers. This issue will lead to heavy delay of attack detection and inaccurate response. In particular, the ISP should consider multiple factors like the current network status and policy agreements with its customers. Also, it should be adaptive to the requests of particular customers, so that traffic engineering performed for one customer will not impact the traffic going to other customers' network.

To address the aforementioned issues, we propose a policy management and enforcement framework to achieve dynamic and automated configuration and enforcement of network and security policies in the ISP network. With our solution, the administrator has only to design the high level security or network policies that will be dynamically and automatically deployed.

3.1 Design Overview

The design overview of our framework is shown in Fig. 1, consisting of several functional components: Monitoring Component (MC), Policy DataBase (PDB), Policy Decision Point (PDP), and Policy Orchestrator and Implementer (POI). As most of the operations are carried out within the ISP domain, the customer network is not shown here. To better illustrate the specific functions of the different components, the operational workflow is given as follows:

1. An event is triggered at the ISP controller when a notification is received by the MC from the customer controller. A notification may be a security alert from a customer or a network status detected by the ISP controller. The detection mechanism used by the customer is out of scope of this paper. We assume that a customer uses some detection mechanism to flag whether it is under attack or not [10].

2. The MC module will analyse the notification and extract the important information as the flow information, impact severity of the traffic, security class and attack type information. Then, it will forward the extracted inputs to the PDP.

3. The PDP selects the high-level action from the policy database to be applied on the flow based on the event and its corresponding conditions. It then forwards the high-level action, bandwidth request, and flow information to the POI.

4. Based on the high level action activated by the PDP, the POI (1) will identify the Policy Enforcement Points (PEPs) needed to deploy the policy and (2) will compute one or multiple paths.

5. Based on the resulting paths, the POI will be responsible of the deployment of the specified paths. It will transfer the chosen actions to a set of OpenFlow rules that will be deployed by the different switches.

3.2 Design Components

The functional components of the policy management system are discussed as follows.

3.2.1 Monitoring Component (MC)

The MC will be responsible for receiving alerts and notifications from the different customers. Then, it will analyse them and extract the important information to be used by the PDP as: flow information (source and destination IP addresses, protocol), the security class (suspicious, malicious, legitimate), the type of attack, and the impact severity of suspicious traffic detected on the customer network. Moreover, this component will be able to monitor the status of the switches and paths in the ISP network and provides the network status (congested, normal) to the PDP. Indeed, to monitor the ISP network, we can use a tool like OpenNetMon [16] that permits to maintain the traffic matrix for different paths and switches in the network.

3.2.2 Policy Decision Point (PDP)

The PDP is in charge of global policy decisions in the ISP network. As a matter of fact, it will firstly activate contexts[1] based on the status of the networks and/or the received alerts. Then, based on the activated contexts, the received information and the policy database, it will decide to enforce a set of high-level actions (e.g, redirect, drop, forward). Finally, it will forward the high level actions and flow information to the POI module to be deployed.

3.2.3 Policy Database (PDB)

It is a repository which contains the predefined security policies. It provides the flexibility for a network administrator to define high level security policies without interest to their deployment.

[1]The context allows to fine-tune the policy that should be enforced, so as to minimize the side-effects of policy enforcement.

Listing 1: Syntax used for High-level Security Policy

```
1  Event = {UDP_Flood | TCP_SYN |
       ICMP_Flood | DNS_Amplification |
       QoS_request}
2  Condition = {Security Class |
       Impact_Severity | ISP_Network_Status}
3  Security Class = {Suspicious | Malicious
       | Legitimate}
4  Impact_Severity = {Low | Medium | High}
5  ISP_Network_Status = {Normal | Congested
       }
6  Actions = {Redirect | Block | Forward}
```

Listing 2: A Sample Policy File to redirect suspicious traffic

```
1
2  <Policy PolicyName="Security_policy">
3      <Event event="UDP-Flood">
4      </Event>
5      <Condition>
6          <security class="suspicious"/>
7          <Impact severity="medium"/>
8              <ISP_Network_Status status="
                   normal"/>
9      </Condition>
10     <Actions action="redirect"/>
11     </Actions>
12 </Policy>
```

In this paper, we provided a syntax (see Listing 1) allowing the design of a mitigation security policy. Policies are structured in the Event-Condition-Action (ECA) model which we believe is suitable for dynamic policy management. Specifically, each *event* refers to a specific attack or incident and is associated with a set of rules. The rules are described as a set of *conditions* that match the context in which the attack or incident occurs (e.g., security class, impact severity, ISP network status). `Security Class` contains three classes of traffic:(1) Malicious denotes the flows that are certain to be attacked;(2) Legitimate represents the flows that are benign in nature; (3) Suspicious denotes the flows which can not be classified into the two former cases and may mix both legitimate and malicious packets. `Impact Severity` indicates the impact of the attack traffic on the customer network. It contains three values: (1) low, (2) medium, and (3) high. The last element of ECA model *Action* is essentially a set of actions to be applied to the identified flows given the context.

Listing 2 is a sample policy to address UDP flood attacks. In this example, flows that have been identified as UDP floods will be diverted to other paths in the network (`redirect` action). In the case they may mingle legitimate packets among malicious ones (i.e., the flow is classified as `Suspicious`), they have a medium impact on the network and in normal network conditions.

3.2.4 *Policy Orchestrator and Implementer (POI)*

This module is responsible for path computation and distributing the rules along the switches in the path for an ordered processing of flows. It also considers the middleboxes to traverse, in order to steer the flows in the network. For instance, the suspicious traffic may be redirected to a firewall and then a NAT (Network Address Translator). It gets the flow information (source IP, destination IP), high-level action, and bandwidth requested as inputs from the PDP for path computation. Paths are computed using the policy aware shortest path [3], which enables the traffic to be routed through a path based on the predefined policies. Algorithm 1 takes into account the flow informations, bandwidth requested, and high level action to compute the path. After path computation, it distributes the rules in the switches and insert Network Service Header (NSH) [15] in the packet for processing. NSH helps in steering the flows in from the core network with only labels. Core switches do not need to check the whole packet header for forwarding. It also reduces the number of flow entries in the core switches of the network.

Algorithm 1 *Path_Computation*

```
1: procedure PATH_COMPUTE(flow,bw_req,action,step,Max_rate)
2:     hop ← 0
3:     path ← []
4:     for Flow F and each path p do
5:         p ← compute_Bandwidth(max(all_links)) //Takes
   the maximum bandwidth in the path
6:         d ← Hop_Count(step[i] + step[i + 1])
7:         hop_count ← hop + d // Computes the hop count in
   the path
8:         path.addList(p)
9:         if (C − bw_req) ≥ Max_rate then //New flow should
   not impact other flows traversing the link.
10:            return hop_count, path
11:        else
```

4. EXPERIMENT

The purpose of our experiments is to demonstrate, in a scenario with multiple customers, the effectiveness of our proposed policy engine on mitigating traffic congestion and collateral damage in the presence of DDoS attacks.

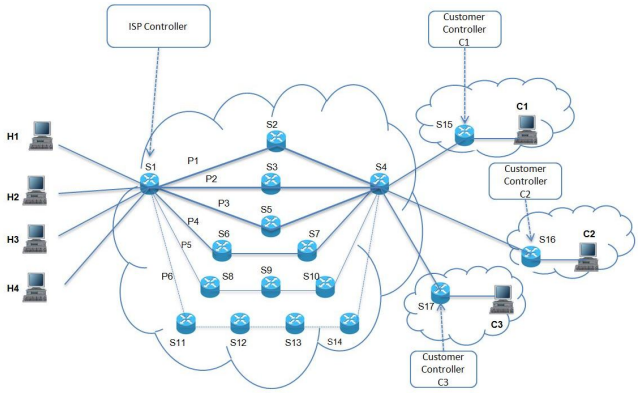

Figure 2: Experimental scenario: one ISP with three cusomters.

Table 1: Traffic paths in terms of bandwidth and link loss probability

Paths	Bandwidth	Link Loss Percentage
P_1	400 Mbps	0
P_2	400 Mbps	0
P_3	400 Mbps	0
P_4	200 Mbps	3
P_5	100 Mbps	5
P_6	50 Mbps	7

4.1 Settings

The policy enforcement framework is implemented in Python and run as an OpenFlow application on Ryu SDN controller. The experiments were carried out in Mininet, which provides prototyping environment for the OpenFlow switches. The experimental scenario is shown in Fig. 2, in which we assume the ISP network contains 14 OF switches and 6 paths, and the bandwidth and link loss probability of different paths are assumed in Table 1. Also, the security alerts received by the ISP are represented in IDMEF format [5].

4.2 Results and Discussions

We use *throughput* and *network jitter*, two well accepted QoS metrics, to evaluate the effectivenss. Some results are reported in the following.

Throughput of legitimate traffic. We measured the throughput of legitimate traffic in the presence of DDoS attacks. As shown in Figure 2, we used H_2 to generate DDoS attack traffic, and observed the impact on the legitimate traffic going to the customers C_1, C_2 and C_3. As we can see in Figure 3, the throughput of all the legitimate traffic dropped sharply to zero as soon as H_2 started to attack. As a result, the SDN controller of customer C_3 sends an alert, which contains the FlowID (source IP, destination IP) and security class (legitimate), to the module *Monitoring Component* in the ISP controller, making *PDP* decide to redirect the flow. Subsequently, *policy orchestrator and implementer* runs Algoritm 1 to compute the best path, i.e., P_3, and inserts the NSH in the packets for redirection. Finally, the corresponding OF rules are loaded to *PEPs* namely OF switches. As shown in Figure 3, the legitimate traffic heading to C_3 therefore was able to quickly return to the normal level.

Similarly, the traffic flow going to C_2 was redirected through path P_2 upon the request of customer C_2 for restoring the throughput to the normal level. Afterwards, the alert of customer C_1 reached at ISP controller, which interestingly redirected the traffic originating from host H_1 (which has the higher rate) to path P_2 as well, pushing the throughput of the traffic (originating from host H_3) to customer C_2 down to zero, as shown in Figure 3. This indicates that, due to the limited availability of high QoS paths in the ISP network, ensuring the QoS for one customer may incur negative impact on other customers.

QoS provisioning for legitimate traffic. Following the previous experiment, we examine how the QoS of legitimate traffic can be provisioned if all the paths with high bandwidth are congested. In this experiment, we assume that customer C_1 requests for getting better QoS of the traffic sent from H_1. As shown in Figure 4, since the legitimate traffic going towards customer C_2 and C_3 were protected from collateral damage, the traffic from H_1 was redirected to the lower bandwidth path P_4, ensuring that the QoS was not heavily impacted despite the congestion of the legitimate path LP_1.

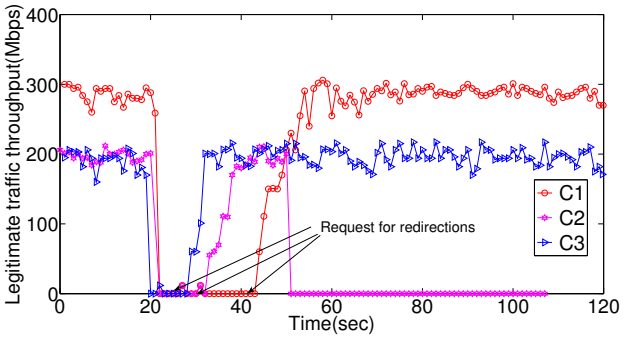

Figure 3: Throughput of legitimate traffic going towards customer network after redirection.

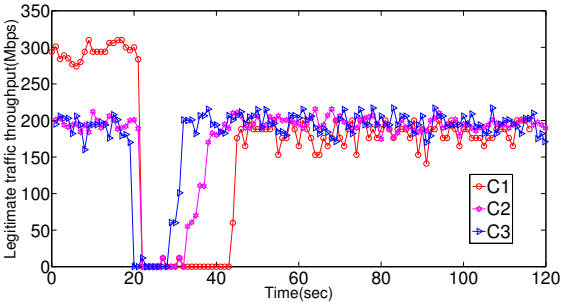

Figure 4: Throughput of legitimate traffic in the case traffic going towards C_1 is redirected through low suspicious path.

Network Jitter of legitimate traffic. Finally, we test how the network jitter of legitimate traffic varies because of congestion in the network. As Fig. 5 shows, the network jitter of legitimate traffic going towards customers C_1, C_2, and C_3 started to increase when the attack traffic from H_2 congested the network. However, all of them immediately decreased when the ISP controller redirected the traffic flows upon receiving the mitigation requests from the customers. Despite the similar changing pattern, the network jitter of the traffic going to C_3 decreased earlier compared to those of C_2 and C_1. This is simply because customer C_3 sent alert earlier than customers C_2 and C_1 did.

5. CONCLUSION

In this paper, we proposed an automated and dynamic policy enforcement mechanism for mitigating DDoS attacks

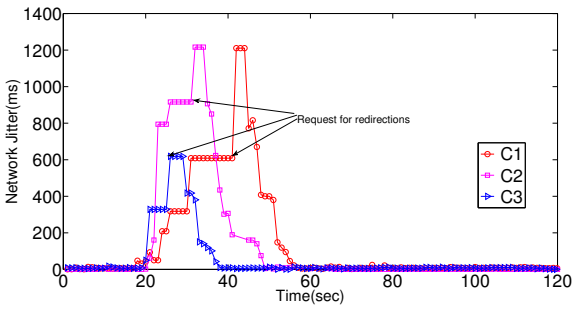

Figure 5: Network jitter of legitimate traffic.

in a scenario which has one ISP serving for multiple customers. One of the major advantages of the mechanism is that it allows high-level security policies of ISP to be dynamically specified based on the security alerts sent from the customers. The policies are then enforced at OpenFlow switches via APIs of SDN controller, which an objective to achieving dynamic security service chaining that is enabled by the service header in the packets. Our future work will be focused on resolving the conflicts of reaction policies due to simultaneous enforcements for different customers. We will also enrich the the policy attributes and study the resulting complexities.

Acknowledgment

This research has been partially supported by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 643964 (SUPERCLOUD).

6. REFERENCES

[1] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. Policycop: An autonomic qos policy enforcement framework for software defined networks. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7, Nov 2013.

[2] Y. Ben-Itzhak, K. Barabash, R. Cohen, A. Levin, and E. Raichstein. Enforsdn: Network policies enforcement with sdn. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 80–88, May 2015.

[3] Z. Cao, M. Kodialam, and T. V. Lakshman. Traffic steering in software defined networks: Planning and online routing. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing*, DCC '14, pages 65–70, New York, NY, USA, 2014. ACM.

[4] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. FlowTags: Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 19–24, New York, NY, USA, 2013. ACM.

[5] B. Feinstein, D. Curry, and H. Debar. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765, Oct. 2015.

[6] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. *SIGPLAN Not.*, 46(9):279–291, Sept. 2011.

[7] A. Lara and B. Ramamurthy. OpenSec: Policy-based security using software-defined networking. *IEEE Transactions on Network and Service Management*, 13(1):30–42, 2016.

[8] S. B. Lee, M. S. Kang, and V. D. Gligor. Codef: Collaborative defense against large-scale link-flooding attacks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 417–428, New York, NY, USA, 2013. ACM.

[9] C. C. Machado, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho. Policy authoring for software-defined networking management. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 216–224, May 2015.

[10] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. dFence: Transparent Network-based Denial of Service Mitigation. In *Proceedings of the 4th USENIX Conference on Networked Systems Design Implementation (NSDI)*, pages 24–24, Berkeley, CA, USA, 2007. USENIX Association.

[11] T. Nadeau and P. Quinn. Problem Statement for Service Function Chaining. RFC 7498, Nov. 2015.

[12] Open Networking Foundation. What's Behind Network Downtime? Technical report, Juniper Networks, 2008.

[13] Open Networking Foundation. SDN Security Considerations in the Data Center. Technical report, ONF, 2013.

[14] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. *SIGCOMM Comput. Commun. Rev.*, 43(4):27–38, Aug. 2013.

[15] P. Quinn and U. Elzur. Network Service Header. Internet-Draft draft-ietf-sfc-nsh-05, Internet Engineering Task Force, May 2016. Work in Progress.

[16] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8, May 2014.

[17] A. Voellmy, H. Kim, and N. Feamster. Procera: A language for high-level reactive network control. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM.