# Adaptive policy-driven attack mitigation in SDN

Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, Khalifa Toumi, Hervé Debar

# Adaptive Policy-driven Attack Mitigation in SDN

Rishikesh Sahay
Gregory Blanc
SAMOVAR, Télécom SudParis,
Institut Mines-Télćom, CNRS,
Université Paris-Saclay
{rishikesh.sahay,gregory.blanc}@telecom-sudparis.eu

Zonghua Zhang
SAMOVAR, IMT Lille Douai,
Institut Mines-Télćom, CNRS,
Université de Lille
zonghua.zhang@imt-lille-douai.fr

Khalifa Toumi
Hervé Debar
SAMOVAR, Télécom SudParis,
Institut Mines-Télćom, CNRS,
Université Paris-Saclay
{khalifa.toumi,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 central network view provided by the Software-Defined Networking (SDN) paradigm. 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

***CCS Concepts*** • **Networks** → **Programmable networks**; **Network management**; • **Security and privacy** → *Denial-of-service attacks*

## 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 collaboration 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 (Lee et al. 2013; Mahimkar et al. 2007) 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 (Open Networking Foundation 2008), human error is the cause for 80% of the total network downtime. Additionally, manual configuration hinders 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 (Nadeau and Quinn 2015). 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. This topological dependency with the deployment of *middleboxes* makes ISPs reluctant to deploy new security functions in their networks



*XDOM0'17* April 23, 2017, Belgrade, Serbia



DOI: http://dx.doi.org/10.1145/3071064.3071068

for providing security services to their customers. Furthermore, all the traffic, whether it needs to be processed through the security devices or not, eventually traverse these devices, which causes them processing overhead. 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 (Open Networking Foundation 2013). It also provides the administrator with a global view of the network and enables programmability in the data plane. By leveraging the 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, from which the ISP decides on the policies to deploy. Our policy management system leverages the global view of the network to assess the impact of a policy deployed for one customer, in order to minimize the impact on the other 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 insights on related work. 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 (Bari et al. 2013; Ben-Itzhak et al. 2015; Machado et al. 2015; Lara and Ramamurthy 2016). 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 (Qazi et al. 2013), 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. Additionally, FlowTags (Fayazbakhsh et al. 2013) 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 (Foster et al. 2011), 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 (Voellmy et al. 2012), 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 (Ben-Itzhak et al. 2015) 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 (Bari et al. 2013) 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 (OF) 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. (Machado et al. 2015). 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 (Lara and Ramamurthy 2016) is close to our proposal in that this framework 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

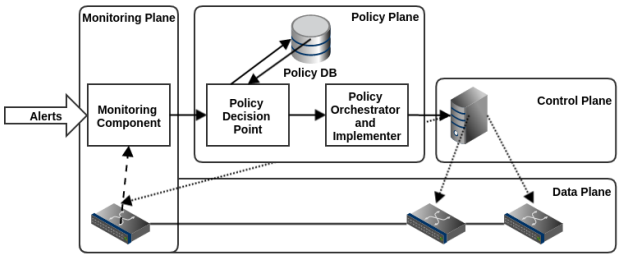


**Figure 1.** Workflow of the Policy Management System

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 cause 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 (Lara and Ramamurthy 2016) and SIMPLE (Qazi et al. 2013).

## 3. Policy Management and Enforcement System

Our previous analysis indicates that most of existing policy management frameworks can not support real-time collaborations between ISPs and customers, thereby leading to heavy latency on attack detection and response. To tackle the issue, the ISP should consider multiple factors, e.g., the current network status and service-level 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. Our purpose is therefore to develop a policy management and enforcement framework which can dynamically and automatically configure and enforce security policies in the ISP network, allowing the administrator to simply specify policies at a high level.

### 3.1 Design Overview

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

1. an event is triggered at the ISP controller when a notification, which can be a security alert sent from a customer controller or a network status update raised by the ISP controller, is received by the MC. We assume that the customer uses some detection mechanism to flag whether it is under attack or not (Mahimkar et al. 2007) [1].
2. The MC module analyzes the notification and extract the information of concern, such as the flow information, its impact severity, its security class and the attack type. The extracted information is then sent to the PDP.
3. The PDP selects the high-level action from the policy database based on the event and its corresponding conditions. The high-level action, bandwidth request, and flow information are then forwarded to the POI.
4. Based on the high-level action, the POI identifies the Policy Enforcement Points (PEPs) and computes the paths.
5. The resulting path(s) is then deployed via POI by translating the chosen actions into a set of OF rules, which can be enforced by the OF switches along the path.

### 3.2 Design Components

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

**Monitoring Component (MC)** handles the security alerts and notifications from different customers. Specifically, it parses them and extract the information of concern for the PDP, including 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 at the customer network. On the other hand, MC monitors the status of the switches and paths in the ISP network and assesses the network status (*congested, normal*) for the PDP. To monitor the ISP network, a tool like OpenNetMon (van Adrichem et al. 2014) can be used as it allows to maintain the traffic matrix for different paths and switches in the network.

**Policy Decision Point (PDP)** is in charge of the global policy decisions. It firstly activates the context[2] based on the status of the networks and/or the received alerts. Then based on the activated context, the extracted information from the alert and the policy database (PDB), PDP decides which actions to take (e.g, *redirect, drop, forward*). The resulting actions, together with the flow information, are finally sent to the POI module to be enforced.

**Policy Database (PDB)** is essentially a repository containing the high-level security policies specified by the network administrator, without detailing the specific deployment strategy.

[1] A detection mechanism is out of scope of this paper.

[2] 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 of high-level security policy

```
Event = {UDP_Flood | TCP_SYN | ICMP_Flood |
    DNS_Amplification | QoS_request}
Condition = {Security Class | Impact_Severity |
    ISP_Network_Status}
    Security Class = {Suspicious | Malicious |
        Legitimate}
    Impact_Severity = {Low | Medium | High}
    ISP_Network_Status = {Normal | Congested}
Action = {Redirect | Block | Forward}
```

**Listing 2.** A Sample Policy for suspicious traffic redirection

```
<Policy PolicyName = "Security_policy">
    <Event event="UDP–Flood">
    </Event>
    <Condition>
        <security class = "suspicious"/>
        <Impact severity= "medium"/>
            <ISP_Network_Status status = "normal
                "/>
    </Condition>
    <Action action="redirect"/>
    </Action>
</Policy>
```

Specifically, security policies are structured using the Event-Condition-Action (ECA) model which we believe is suitable for dynamic policy management. In particular, 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. At last, the *Action*, is essentially a high-level action to be applied to the identified flows.

Formally, we provide a syntax about DDoS mitigation, as shown in Listing 1, in which an *Event* may indicate one of several types of DDoS attacks or QoS requests, a *Condition* set includes the security class – *Malicious* labels the flows certain to be from attacker; *Legitimate* represents the flows that are benign in nature; and *Suspicious* denotes the mixture of malicious and legitimate traffic – as well as the impact severity, of the attack traffic on the customer network (*low, medium or high*) and the status of the ISP network status (i.e., either normal or congested). *Action* may be either of the three following: redirection, blocking, and forwarding.

To illustrate the specification of the policy according to the given syntax, a sample policy addressing UDP flood attack is given in Listing 2, which shows that the flows identified as UDP flood are labeled with a *suspicious* security class, resulting in a *medium* impact on the customer network. If the ISP network is in a *normal* status, the flows will *redirected* elsewhere in the network.

**Policy Orchestrator and Implementer (POI)** is mainly used for path computation and distributing the rules to the switches along the computed path. It also considers the security 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 device. Flow information (source IP, destination IP), policy action, and requested bandwidth are taken as inputs from the PDP for path computation, as detailed in Algorithm 1, which is essentially a policy-aware shortest path algorithm (Cao et al. 2014). As a result of path computation, POI distributes the rules in the switches along the computed path and inserts a *Network Service Header* (NSH) (Quinn and Elzur 2016) in the packets for processing. In particular, NSH helps to steer the flows in the core network with only labels, so the core switches do not need to check the whole packet header for forwarding. It also significantly reduces the number of flow entries in the core switches.

**Algorithm 1** *Path_Computation*

**procedure** PATH_COMPUTE(flow,bw_req,action,step,Max_rate)
  $hop \leftarrow 0$
  $path \leftarrow []$
  **for** Flow F and each path p **do**
    $p \leftarrow compute\_Bandwidth(max(all\_links))$ //-Takes the maximum bandwidth in the path
    $d \leftarrow Hop\_Count(step[i] + step[i+1])$
    $hop\_count \leftarrow hop + d$ // Computes the hop count in the path
    $path.addList(p)$
    **if** $(C - bw\_req) \geq Max\_rate$ **then** //New flow should not impact other flows traversing the link.
      **return** $hop\_count, path$
    **else**
      No path is available

## 4. Experiment

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

### 4.1 Settings

The policy enforcement framework is implemented in Python and run as an OF application on the Ryu SDN controller. The experiments were carried out in Mininet, which provides a simulated OF switching environment. 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 to be as in Table 1. Also, the security alerts received by the ISP are represented in the IDMEF format (Feinstein et al. 2015).

### 4.2 Results and Discussions

We use *throughput* and *network jitter*, two well accepted QoS metrics, to evaluate the effectiveness of our approach.

**Throughput of legitimate traffic**. Throughput was measured 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

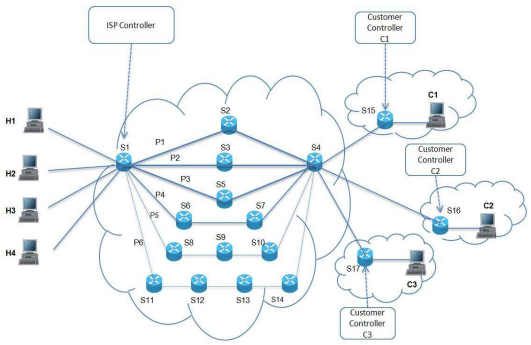


**Figure 2.** Experimental scenario: one ISP with three customers.

**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 |

$C_1$, $C_2$ and $C_3$. As we can see in Figure 3, the throughput of all the legitimate traffic dropped sharply 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 *MC* at the ISP controller, making *PDP* decide on redirecting the flow. Subsequently, *POI* computes the best path, i.e., $P_3$, and inserts the NSH in the packets to redirect them. Finally, the corresponding OF rules are loaded to *PEPs*, namely the OF switches. As shown in Figure 3, the legitimate traffic heading to $C_3$ was thus able to quickly return to its normal level.

Similarly, the traffic flow going to $C_2$ was redirected through path $P_2$ upon the request of customer $C_2$, in order to restore its throughput to the normal level. Afterwards, the alert of customer $C_1$ reached the ISP controller, which interestingly redirected the traffic originating from host $H_1$ (which has the higher throughput) 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 $LP_1$.

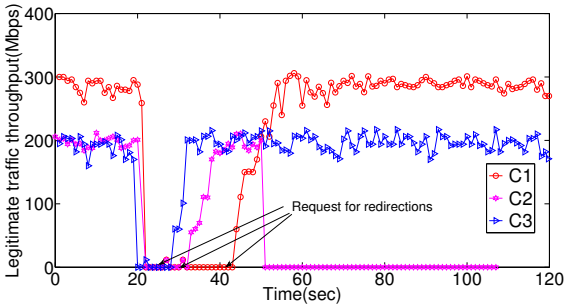


**Figure 3.** Throughput of legitimate traffic going towards customer network after redirection.

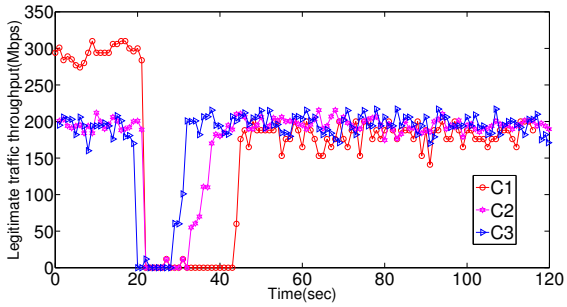


**Figure 4.** Throughput of legitimate traffic in the case traffic going towards $C_1$ is redirected through low suspicious path.

**Network Jitter of legitimate traffic**. Finally, we test how the network jitter of legitimate traffic varies in the presence of congestion. 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

This paper proposed an automated and dynamic policy enforcement mechanism for mitigating DDoS attacks in a scenario where a single ISP is serving multiple customers. One

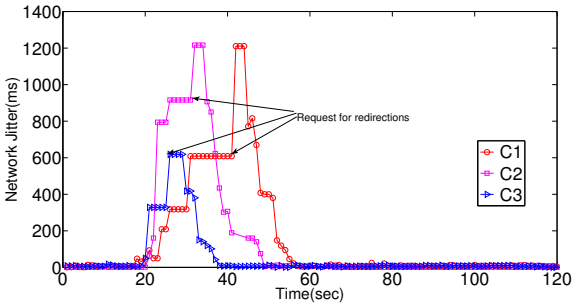


**Figure 5.** Network jitter of legitimate traffic.

of the major advantages of the mechanism is that it allows high-level security policies to be dynamically instantiated based on the security alerts sent from the customers, and adapted upon network changes. The policies are then enforced at OF switches with an objective to achieve dynamic security service chaining, which is enabled by the network service header, positioned in the packets by the SDN controller. 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 policy attributes and survey the associated complexity.

## Acknowledgments

This research has been partially supported by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 643964 (SUPERCLOUD). The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions.

## References

M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. 2013. PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. 1–7. DOI:http://dx.doi.org/10.1109/SDN4FNS.2013.6702548

Y. Ben-Itzhak, K. Barabash, R. Cohen, A. Levin, and E. Raichstein. 2015. EnforSDN: Network policies enforcement with SDN. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 80–88. DOI:http://dx.doi.org/10.1109/INM.2015.7140279

Zizhong Cao, Murali Kodialam, and T. V. Lakshman. 2014. Traffic Steering in Software Defined Networks: Planning and Online Routing. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing (DCC '14)*. ACM, New York, NY, USA, 65–70. DOI:http://dx.doi.org/10.1145/2627566.2627574

Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. 2013. 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*. ACM, New York, NY, USA, 19–24. DOI:http://dx.doi.org/10.1145/2491185.2491203

Benjamin Feinstein, David Curry, and Herve Debar. 2015. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765. (14 Oct. 2015). DOI:http://dx.doi.org/10.17487/rfc4765

Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. 2011. Frenetic: A Network Programming Language. *SIGPLAN Not.* 46, 9 (Sept. 2011), 279–291.

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

Soo Bum Lee, Min Suk Kang, and Virgil D. Gligor. 2013. CoDef: Collaborative Defense Against Large-scale Link-flooding Attacks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*. ACM, New York, NY, USA, 417–428. DOI:http://dx.doi.org/10.1145/2535372.2535398

C. C. Machado, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho. 2015. Policy authoring for software-defined networking management. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 216–224. DOI:http://dx.doi.org/10.1109/INM.2015.7140295

Ajay Mahimkar, Jasraj Dange, Vitaly Shmatikov, Harrick Vin, and Yin Zhang. 2007. dFence: Transparent Network-based Denial of Service Mitigation. In *Proceedings of the 4th USENIX Conference on Networked Systems Design Implementation (NSDI)*. USENIX Association, Berkeley, CA, USA, 24–24.

Thomas Nadeau and Paul Quinn. 2015. Problem Statement for Service Function Chaining. RFC 7498. (10 Nov. 2015). DOI:http://dx.doi.org/10.17487/rfc7498

Open Networking Foundation. 2008. *What's Behind Network Downtime?* Technical Report. Juniper Networks.

Open Networking Foundation. 2013. *SDN Security Considerations in the Data Center*. Technical Report. ONF.

Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. 2013. SIMPLE-fying Middlebox Policy Enforcement Using SDN. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 27–38. DOI:http://dx.doi.org/10.1145/2534169.2486022

Paul Quinn and Uri Elzur. 2016. *Network Service Header*. Internet-Draft draft-ietf-sfc-nsh-05. Internet Engineering Task Force. https://tools.ietf.org/html/draft-ietf-sfc-nsh-05 Work in Progress.

N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers. 2014. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*. 1–8.

Andreas Voellmy, Hyojoon Kim, and Nick Feamster. 2012. Procera: A Language for High-level Reactive Network Control. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*. ACM, New York, NY, USA, 43–48.