applied sciences

MDPI

Article

Robust and Agile System against Fault and Anomaly Traffic in Software Defined Networks

Mihui Kim [1], Younghee Park [2,*] and Rohit Kotalwar [2]

[1] Department of Computer Science & Engineering, Computer System Institute, Hankyong National University/327, Jungang-ro, Anseong-si, Gyeonggi-do 17579, Korea; mhkim@hknu.ac.kr
[2] Computer Engineering Department, San Jose State University, One Washington Square, San Jose, CA 95192, USA; rohit.kotalwar@sjsu.edu
* Correspondence: younghee.park@sjsu.edu; Tel.: +1-408-924-7854

Academic Editor: Christos Bouras
Received: 10 December 2016; Accepted: 2 March 2017; Published: 8 March 2017

Abstract: The main advantage of software defined networking (SDN) is that it allows intelligent control and management of networking though programmability in real time. It enables efficient utilization of network resources through traffic engineering, and offers potential attack defense methods when abnormalities arise. However, previous studies have only identified individual solutions for respective problems, instead of finding a more global solution in real time that is capable of addressing multiple situations in network status. To cover diverse network conditions, this paper presents a comprehensive reactive system for simultaneously monitoring failures, anomalies, and attacks for high availability and reliability. We design three main modules in the SDN controller for a robust and agile defense (RAD) system against network anomalies: a *traffic analyzer*, a *traffic engineer*, and a *rule manager*. RAD provides reactive flow rule generation to control traffic while detecting network failures, anomalies, high traffic volume (elephant flows), and attacks. The traffic analyzer identifies elephant flows, traffic anomalies, and attacks based on attack signatures and network monitoring. The traffic engineer module measures network utilization and delay in order to determine the best path for multi-dimensional routing and load balancing under any circumstances. Finally, the rule manager generates and installs a flow rule for the selected best path to control traffic. We implement the proposed RAD system based on Floodlight, an open source project for the SDN controller. We evaluate our system using simulation with and without the aforementioned RAD modules. Experimental results show that our approach is both practical and feasible, and can successfully augment an existing SDN controller in terms of agility, robustness, and efficiency, even in the face of link failures, attacks, and elephant flows.

Keywords: software defined networks; network measurement; traffic engineering; intrusion detection system; network attacks; reactive routing

1. Introduction

Software defined networking (SDN) has been a major topic of interest in the field of networking for a decade. SDN enables intelligent management and control of network elements such as switches and routers that possess low level functionality. The SDN architecture separates the control and data planes, thereby abstracting the infrastructure from applications. The control plane is logically centralized, whereas the data plane simply follows the decisions of the control plane. The biggest advantage of SDN is that new control functions can be implemented by incorporating software-based logic into the control plane [1]. Thus, SDN can quickly apply new technologies to networks and can easily interact with external control devices.

Recently, greater emphasis has been placed on preserving the availability of networks because of the importance of fault management for huge and complicated networks as well as the increasing numbers of distributed denial of service (DDoS) attacks. In particular, data center networks have a variety of requirements, including high availability of up to 99.999%, simple and quick error detection, reliability, and fast restoration [2]. Thus, enterprises demand a highly available SDN infrastructure to meet growing business needs. This type of infrastructure can be strategized by using an SDN controller. The controller provides a global and centralized view of the network. It transmits information between the network switches or routers and the applications above them. Having an abstract SDN controller enables us to control network elements programmatically and dynamically reconfigure these network elements based on network conditions. However, this shift of intelligence to the controller requires efficient operation therein. Leveraging the agility and programmability of SDN, the controller must provide comprehensive monitoring and management even when abnormalities are present and act as a nimble countermeasure to assure the availability.

Many research groups have concentrated on utilizing the high controllability of SDN or preventing faults or anomalies, based on different goals: *traffic engineering* for efficiency, *measurement and monitoring* for accuracy and low load, and *security and dependability* for specific attacks. Traffic engineering includes load balancing, flow rules and traffic optimization, and application-aware networking. Load balancing has been researched for network efficiency [3–6]. The goal for flow rules and traffic optimization is SDN efficiency [7–15]. Application-aware networking guarantees quality of service (QoS) of multimedia traffic [16,17]. Next, as a basic process for traffic engineering or security, research on measurement and monitoring has tried to improve the accuracy of the process while simultaneously reducing the load of the controller [18–22]. Moreover, because of the vulnerability of the SDN controller itself (i.e., one point of failure) some studies consider new possible attack scenarios (e.g., crash and data compromise attacks [23], host location hijacking attacks [24], or control plane saturation attacks [25]) and design a proper defense for each. However, these studies do not provide a comprehensive system for monitoring and preventing internal faults, anomalies and outside attacks. Thus, the synthetic system research providing an agile and efficient defense is necessary.

Our goal in this study is to develop readily available and stable SDNs that take adept actions against both fault and anomaly traffic, quickly detect anomalies by means of packet inspection, generate automatic flow rules, and directly apply them on switches. Our proposed architecture helps to develop a comprehensive solution for fault and abnormal detection and prevention, minimize latency, and increase network availability and efficiency under any circumstances. Moreover, our architecture adopts proper SDN rule management methods (i.e., proactive and reactive) based on flow types (i.e., mice and elephant flows). To achieve these goals, we propose a robust and agile defense (RAD) system against both fault and anomaly traffic by utilizing SDN technologies. Our RAD system consists of three main modules: a *traffic analyzer*, *a traffic engineer*, and a *rule manager*. The traffic analyzer monitors the traffic using sFlow-RT [26], a real-time network measurement tool, and Snort IDS [27], a signature-based intrusion detection system, to detect elephant flows and attacks, respectively. If an anomaly is detected, the traffic engineer module generates new routing paths based on the network status (i.e., network utilization or delay). The newly determined routing paths enable the rule manager to generate flow rules for the data plane on network devices. These steps are automatically and circularly performed according to the outbreak of anomaly events.

The contributions of this study are as follows:

- We design the RAD system having fast detection and defense against faults and attacks as well as generating reactive flow rules for elephant flows by considering network status.
- We evaluate the RAD system with respect to robustness, agility, and efficiency through experiments with various scenarios by employing each module.

The remainder of this paper is organized as follows. Section 2 reviews previous studies on SDN in relation to fault tolerance and attack defense. Section 3 explains our proposed system architecture

(RAD) with the three modules. Section 4 discusses the experimental results of the RAD system. Section 5 provides concluding remarks and suggestions for future research.

2. Related Work

Several studies have proposed protection tools or attempted to increase the availability and efficiency of SDN networks. These include *traffic engineering*, *measurement and monitoring*, and *security and dependability* [28].

Traffic Engineering: The programmable characteristic of SDN enables the various traffic engineering techniques (i.e., *load balancing*, *SDN rules/traffic optimization*, and *application-aware networking*) to enhance the efficiency of the network. The first application envisioned for SDN traffic engineering is load balancing such as that described in [3–6]. To achieve scalability, the authors in [3] devised concise wildcard rules that adjust to changes in load-balancing policies. The authors in [4] designed load-balanced forwarding using in-packet Bloom filters for scalable forwarding services. Plug-n-Serve [5] attempts to minimize response time by controlling the load on the network.

In addition, the performance of SDN networks depends on the efficiency of rules generated by the SDN controller and the proper placement of rules on the memory of switches. This is because the many rules increase the number of control signals and the switches have memory limitations. Authors in [7] attempted to maximize the amount of traffic delivered by generated rule policies. The survey in [8] considered rules placement research to focus on memory management and reducing the signaling traffic for scalability purposes. To manage memory in switches efficiently, some research [9,10] removed the inactive or less-important rules from flow tables in the data plane. Other approaches [11,12] compressed the required rules that preserve the original semantics using wildcards. Still other mechanisms [13,14] split and distributed the rules to commodity switches. Finally, to decrease the signaling traffic while in the reactive mode of the original SDN/OpenFlow specification [29], research in [15] considered the proactive mode based on predicting and estimating traffic demands or user location.

Meanwhile, application-aware networking guarantees the QoS of multimedia and data stream using multiple packet schedulers [16] or queue management [17]. Most of these traffic engineering schemes promise network performance, scalability, or fault tolerance in certain circumstance, but do not consider immediate counteractions to various anomalies.

Measurement and Monitoring: As a part of network management, measurement and monitoring are major processes. SDN technologies provide new functionality to existing networks for measurement and monitoring (e.g., enhanced visibility of broadband performance [30]), reacting (i.e., traffic shaping) to continually changing network conditions in a home network [31], or dynamic management of a smart grid [32].

In addition, other studies have focused on enhancing measuring and monitoring efficiency itself. The authors in [18] reduced the burden of the control plane using a stochastic and deterministic sampling technique, whereas authors in [19] attempted to achieve high measurement accuracy without incurring overhead on the control plane. Other monitoring frameworks (i.e., OpenSample [20], OpenSketch [21], and PayLess [22]) were designed to deliver real-time, low-latency, and flexible-monitoring capability to SDN networks without impairing the performance and load of the control plane. However, the results from measuring and monitoring should be closely connected to the skillful actions of the SDN controller.

Security and Dependability: The programmability of SDN can improve the security of other networks [33,34]. However, the logically centralized control plane in SDN networks is one point of failure and may become a desirable prey to an adversary. Therefore, some approaches have been examined to protect the SDN controller. Early approaches tried to apply simple techniques (i.e., first classifying applications and then preventing low-priority applications from overwriting rules generated by security applications [35]). As additional research, a framework for developing security-related applications was proposed in [36]. To secure an SDN network, most of these

approaches generate new attack scenarios and focus on defending against each attack. Rosemary [23] defends against system crashes and data compromise attacks. TopoGuard [24] identifies misused APIs and defends against new attacks such as host location hijacking attacks. Finally, Avant-Guard [25] defends against control plane saturation attacks. However, these types of defenses are restricted to new types of attacks.

Policy based approaches (i.e., access control) have been researched to secure the north bound APIs of an SDN controller. SE-Floodlight [37] implements security features that utilize trust model and policy mediation logic. The operation checkpoint permission system [38] ensures that controller operations are performed only by trusted applications. The authors in [39] implemented read, notification, write, and system access permissions for OpenFlow applications in order to isolate the controller and apps in thread containers. Although these policy-based approaches introduce the idea of providing access control for applications, they do not secure the core modules of the SDN controller or do not effectively resist attack traffic on an SDN data plane.

Overall, the mechanisms for traffic engineering, measurement and monitoring, and security and dependability fail to interconnect in order to prevent internal faults and anomalies, defend against outside attacks, and maintain the availability and efficiency of a network even when anomalies are present. Indeed, none of the current approaches provides a reasonable solution to resolve these problems. The primary objective of this study is to provide a comprehensive reactive monitoring and defense system. Table 1 briefly compares existing works to our proposed system.

Table 1. Comparison of existing works.

Type	Work	Goal
Traffic Engineering	Load balancing [3–6]	Network efficiency
	Rules/traffic optimization [7–15]	Efficiency of SDN controller or networks
	Application-aware networking [16,17]	QoS guarantee
Measurement and Monitoring	New functionality provided to existing networks [30–32]	Efficient management with new features
	Efficient measurement and monitoring [18–22]	High measurement accuracy and overhead reduction
Security and Dependability	Reinforcement security of other networks [33,34]	Improvement of security with programmability of SDN
	SDN-specific security [23–25,35,36]	Securing against SDN network-specific vulnerabilities
	Policy-based approaches [37–39]	Access control at SDN controller
Comprehensive Monitoring and Reactive System	Our proposed system, RAD	Agile and robust response against abnormalities (i.e., faults, elephant flows, and attacks) with efficient network performance

3. RAD System

This section presents our RAD system by first providing an overview and then a detailed description of the system design. Our RAD system achieves high availability and flexibility for SDN. In addition, it provides fault-tolerance and reliability for SDN by detecting network abnormalities and attacks. The novelty of the RAD system provides automatic chained processes in order to continuously monitor and control entire networks for dynamic and fault-tolerant network operations in SDN. It aims for defense against attacks using traffic measurement, and automatic rules generation for high availability and reliability. Moreover, our RAD system applies either proactive or reactive flow management based on the flow type.

3.1. Overview of RAD System Architecture

This subsection provides an overview of our RAD system architecture, which is divided into three main modules with one storage, as shown in Figure 1. The three main modules consist of

a *traffic analyzer*, *a traffic engineer*, and *a rule manager*. There is a *Data Storage for Topology/Anomaly Signature*. Furthermore, RAD utilizes the features of sFlow-RT and Snort IDS for flow classification and anomaly detection, respectively. The RAD system jointly operates with the SDN controller through a northbound interface obtaining traffic measurements and providing the new flow rules.

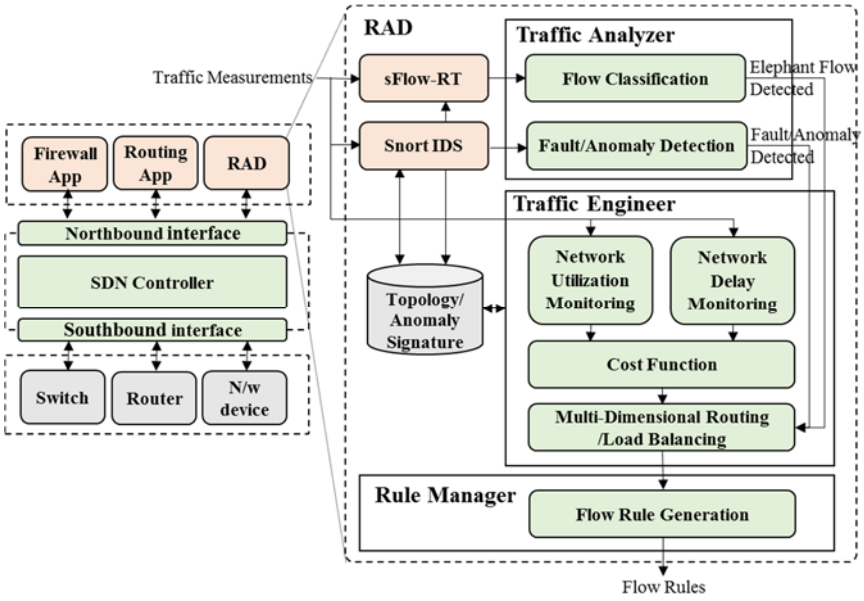

Figure 1. RAD system architecture.

In the *traffic analyzer* module, sFlow-RT monitors traffic and collects and analyzes network statistics from network devices in order to classify both elephant and mice flows. In detecting elephant flows, the flow classification submodule triggers the traffic engineer to reactively generate an optimal routing path for these flows. As a proactive manner, the flow classification submodule periodically provides statistical information regarding flows for the traffic engineer to prepare paths for mice flows through the traffic engineer. Snort IDS also examines traffic to detect attacks based on rules generated, whereas the fault detection submodule monitors the status of devices in order to detect faults. If an attack or fault is detected, the traffic engineer is triggered to generate other optimal routes. The *traffic engineer* calculates optimal routing paths with the aid of a cost function that utilizes current values of network utilization and delay. The load balancing submodule distributes normal traffic on multiple paths to increase performance and reliability. The *rule manager* generates flow rules for network devices based on routing decisions from the traffic engineer, and pushes the rules to network devices through the SDN controller. *Data storage* stores the information for network topology and signatures for the purpose of attack detection.

3.2. Main Modules of RAD System

In this subsection, we provide detailed descriptions of each module of the RAD system: *traffic analyzer*, *traffic engineer*, and *rule manager*.

3.2.1. Traffic Analyzer

We utilize an sFlow (short for "sampled flow") real-time tool, known as sFlow-RT, to detect elephant flows dynamically. The sFlow-RT is a well-known industry standard and incorporates the asynchronous analytics technology of InMon [40]. It thereby enables the real-time management of SDN to reduce network outages and the new classes of performance-aware applications (e.g., load balancing) to improve network performance. Furthermore, sFlow-RT provides network visibility, which is essential to understanding the interactions among scale-out services running on a cloud

infrastructure [26]. The sFlow-RT analytics engine receives a continuous stream from sFlow agents embedded in network devices (e.g., network switches) and monitors traffic information (i.e., the occupied link bandwidth) to detect elephant flows.

An *elephant flow* is a long-living flow that occupies high bandwidth (e.g., 10% of the link bandwidth); the ratio to judge an elephant flow can be controlled. Based on the reports of data center networks, the majority of flows are short-lived latency-sensitive mice flows. They consume small portions of traffic. The other kind of flow is the bandwidth-sensitive elephant flow, which accounts for 80% of the traffic but constitutes only 10% of all flows [41]. Typical examples for elephant flows are large data transfer transactions such as VM migration, video streaming, or peer-to-peer file sharing, whereas mice flows occur in small web requests and VoIP calls. A flow can be categorized as an elephant flow if it occupies more than 10% of the NIC bandwidth or if it occupies a specific bandwidth (e.g., 10 Mbps). In the event of an elephant flow, sFlow-RT notifies the traffic engineer to reactively reroute the elephant flow. The traffic engineer then makes appropriate routing decisions by considering the available link capacity or delay. This ensures that the elephant flow is routed on a better path from that point and thus reduces congestion of regular short-lived mice flows. The rules for mice flows can be proactively installed and reactive flow rules are then used only to route elephant flows and to manage attacks, failures, and changes in topology. This hybrid approach of utilizing both proactive and reactive rule management in an OpenFlow-based network offers better scalability than a completely reactive flow rule management.

sFlow-RT converts the received streams from sFlow agents into actionable metrics, which are accessible through the RestFlow API [42]. The RestFlow API facilitates configuring customized measurements, retrieving metrics, setting thresholds, and receiving notifications. Applications can be external, and thus written in any language that supports HTTP and REST calls, or internal, using the embedded JavaScript or ECMAScript of sFlow-RT. By combining network, host, and application monitoring within an integrated analytics pipeline, sFlow-RT provides visibility into applications as well as server and network resources needed to sustain performance.

To detect attacks in networks, we use Snort Tool. Snort is a free and open source network intrusion prevention system and network intrusion detection system created by Martin Roesch in 1998 [27]. Snort can perform real-time traffic analysis and packet logging on IP networks. Snort performs protocol analysis as well as content searching and matching in order to detect probes or attacks, including but not limited to operating system fingerprinting attempts, common gateway interfaces, buffer overflows, server message block probes, and stealth port scans. Snort can be configured in three main modes: sniffer, packet logger, and network intrusion detection. In sniffer mode, it reads network packets and displays them on the console. In packet logger mode, it logs packets to the disk. In intrusion detection mode, it monitors network traffic using a rule set defined by the user. Snort then performs a specific action based on what has been identified. If, while in intrusion detection mode, Snort detects attacks, it notifies the fault/anomaly detection submodule in the traffic analyzer to properly defend against it (e.g., by dropping the attack traffic).

The following are examples of *Snort rules* used to detect certain attacks (i.e., ICMP flooding and TCP SYN flooding attacks), *Snort input*, and *Snort output* in the log file.

(1) **Snort Rule**:

// ICMP flooding attack with any IP/port to any IP/port exceeding 10 ICMP messages in 1 s

alert icmp any any -> any any (msg:"ICMP FLOODING ATTACK DETECTED"; GID:1; sid:10000011; rev:001; itype:8; threshold: type threshold, track by dst, count 10, seconds 1; classtype:icmp-event;)

Snort Input:

/ / Mounting ICMP flooding attack by hping3 attack tool (SrcIP 10.0.0.1, DstIP 10.0.0.22)

hping3 -1 –flood -a 10.0.0.1 10.0.0.22

Snort Output:
10/07-13:54:37.429751 [**] [1:10000013:1] ICMP FLOODING ATTACK DETECTED [**] [Classification]: Generic ICMP event] [Priority: 3]

{ICMP} 10.0.0.1 -> 10.0.0.22:66530

(2) **Snort Rule**:
// TCP SYN flooding attack with any IP/port to any IP/port exceeding 100 SYN messages in 1 s
alert tcp any any -> any any (msg:"DDOS ATTACK Detected (TCP SYN Flooding)"; flags:S; threshold: type threshold, track by_dst, count 100, seconds 1; sid: 10000003; rev:1;)

Snort Input:
// Mounting TCP SYN flooding attack by hping3 attack tool (SrcIP 10.0.0.1, DstIP 10.0.0.22, interval 100 µs)
Source 10.0.0.1 >> hping3 -i u100 -S 10.0.0.22

Snort Output:
10/07-14:15:37.839869 [**] [1:50000008:0] DDOS ATTACK Detected (TCP SYN Flooding) [**] [Priority: 0] {TCP} 10.0.0.1 -> 10.0.0.22:3334

To defect a fault in switches (e.g., link or switch down), the fault/anomaly detection submodule monitors the status of switches and links between them. This submodule is also supported by a module called "link discovery" in SDN, which monitors all link information, whether it is in regard to the addition or deletion of a link.

3.2.2. Traffic Engineer

The traffic engineer module provides fast and proper routes even for elephant flows, faults, and attacks based on the trigger of the traffic analyzer. The module has two monitoring submodules: *network utilization* and *network delay*. Through the input of traffic measurement, these submodules monitor network utilization and delay for each link. Although a shortest path can be the best choice for routing in the case of networks having the same capacity link and light traffic, considering only a hop-count cannot guarantee a high network utilization rate or short delay for multimedia traffic requiring various QoS. Thus, we design a multi-dimensional routing module considering various metrics. Now, we next employ network utilization and delay. However, the other metrics can be easily added on.

The cost function submodule calculates the cost for each link as given by Equation (1) [41]. This is necessary to choose the best path that considers multi-dimensional metrics based on the values from two monitoring submodules and control parameters. The link incurring high cost is excluded from the route generation. The cost function controls the preference weight of metrics and normalizes the different ranges of metric values.

$$C_i = \sum_k c_k \times f_k(x_i^k) \tag{1}$$

where C_i is the cost value of a link i, x_i^k is a kth metric value of a link i, c_k is the preference weight of metric k ($\sum_k c_k = 1$), and $f_k(\cdot)$ is a normalized function for the kth metric. In this study, we use two metrics (thus, k is 2): utilization U_i and delay D_i, fine-grained by Equations (3) and (4). For $f_k(\cdot)$, we use an S-shaped sigmoid form, which is the most commonly used normalized function and is often used to describe QoS perception [43,44], as Equation (2).

$$f(x) = \frac{(x/x_m)^\theta}{1 + (x/x_m)^\theta} \tag{2}$$

where $x_m > 0$ and $\theta \geq 2$ are tunable parameters that differentiate the utilities. We also assume that the utilities are normalized to their highest limits (i.e., the asymptotic value of $f(x)$ for the largest x is considered to be equal to 1).

To control the utilization and delay metrics in a delicate manner, RAD uses Equations (3) and (4) instead of Equation (2) [45]. These equations can establish the minimum cost in the cases of lightly loaded and speedy links in a sophisticated manner:

$$f(r_i) = \begin{cases} \frac{\left\lceil \left(\frac{r_i}{R_i} - u_{min}\right)^{\alpha} \cdot (C-1) \right\rceil + 1}{C}, & r_i/R_i > u_{min} \\ 1/C, \ otherwise \end{cases} \quad (3)$$

where r_i is a used bandwidth of link i, R_i is the capacity of link i, C is the cost level, α is a tunable parameter that controls how expensive heavily loaded links look relative to lightly loaded links, u_{min} is the minimum-cost utilization level. Note that any link utilization below u_{min} is considered to have the minimum cost. This cost model delicately controls the utilization metric. Larger values of C tend to decrease the blocking rate, particularly when the network possesses accurate link-state information (i.e., frequent link status updates [45]). However, the fine-grained cost metrics do not always enhance the performance. Thus, we use $C = 4$ and $\alpha = 2$, referring to [43].

$$f(d_i) = \begin{cases} \frac{\left\lceil \left(\frac{d_i}{D_i} - d_{min}\right)^{\alpha} \cdot (C-1) \right\rceil + 1}{C}, & \frac{d_i}{D_i} > d_{min} \\ 1/C, otherwise \end{cases} \quad (4)$$

where d_i is the delay of link i, D_i is the maximum delay of link i that has turned on, C is the cost level, α is a tunable parameter that controls how slow links look relative to fast links, and d_{min} is the minimum-cost delay level. Note that any delay ratio d_i/D_i below d_{min} is considered to have the minimum cost. This cost model delicately controls the delay metric.

Finally, the multi-dimensional routing/load balancing submodule generates the shortest paths having the lowest total cost based on Equation (1). The rule manager is then notified about information regarding multiple paths for generating rules and pushing them to switches.

3.2.3. Rule Manager

The rule manager generates the proper rule set for route information received from the traffic engineer module and pushes them to each switch. The following are example general route and action rules after attacks have been detected:

(1) **Switch Rule**:
// This is a general routing rule on switch S5 for host H1(10.0.0.1), which sends traffic to H5(10.0.0.5). The 'actions': *output:1 represents H1 uses port 1 to send traffic to H5. The similar rules are pushed on other switches in the generated paths (Refer to network topology in Figure 2).*
{'ipv4_src': '10.0.0.1', 'cookie': '0', 'actions': output: 1, 'active': 'true', 'eth_src': '3a:e4:de:ed:02:ac', 'ipv4_dst': '10.0.0.5', 'eth_dst': '72:f1:22:9a:95:91', 'name': 'flow1', 'priority': '0', 'switch':'00:00:00:00:00:00:00:05', 'eth_type': '0X0800', 'in_port': '3'}

(2) **Switch Rule**:
// If a host H1(10.0.0.1) connected to switch S5 is identified as an agent to attack a host H22(10.0.0.22), the following rule is pushed into switch S5 for dropping the attack traffic.
{'ipv4_src': '10.0.0.1', 'cookie': '0', 'actions': 'drop', 'active': 'true', 'eth_src': '3a:e4:de:ed:02:ac', 'ipv4_dst': '10.0.0.22', 'eth_dst': '72:f1:22:9a:95:91', 'name': 'flow1', 'priority': '0', 'switch': '00:00:00:00:00:00:00:05', 'eth_type': '0X0800', 'in_port': '3'}

4. Performance Evaluation

This section evaluates the RAD system from the viewpoints of agility, robustness and efficiency through in-detail simulation, in occurring faults, attacks, and elephant flows. We compare the performance in the cases with and without RAD system. Table 2 shows the used tools and technologies for the simulation.

Table 2. Tools and technologies.

Tools and Technologies	Name	Version	Description
Simulator	Mininet [46]	2.2.1	Create a realistic virtual network, running real kernel, switch, and application code on a single machine (VM, cloud, or native).
SDN Controller	FloodLight [47]	1.2	An Open SDN Controller offered by Big Switch Networks that works with the OpenFlow protocol to orchestrate traffic flows in an SDN environment.
IDS	Snort [27]	2.9.8	The most popular network intrusion detection system.
Traffic Generator	Iperf [48]	2.0	A popular network tool that was developed to measure TCP and UDP bandwidth performance. By tuning various parameters and characteristics of the TCP/ UDP protocol, the user can perform several tests that provide an insight on the network's bandwidth availability, delay, jitter, and data loss.
Attack Tool	Hping [49]	3.0	A network tool that can send custom TCP/IP packets and display target replies as a ping program does with ICMP replies. Hping3 handles fragmentation as well as, arbitrary packets body and size, and can be used to transfer files encapsulated under supported protocols. It can thus be used as an attack tool.
Development tools	Python	2.0/3.0	Python scripting language is used to develop the script for load balancing, routing, and preventing an attack.

Utilizing the Mininet simulation tool, we set up our network topology as shown in Figure 2. This topology is general for networks, because the real small network has a standard pattern that consists of backbone (core) part and edge part, and it is quite symmetric. Switches from S1 to S4 consist of a core backbone network with complete connections, whereas switches from S5 to S12 act as edge switches that provide connections to the hosts (H1 to H48). FloodLight, SDN controller used in evaluation part, provides only network utilization, and thus we have no choice to use metrics except the network utilization in this evaluation. All results are computed by averaging the values of 10 simulation runs to obtain statistical results.

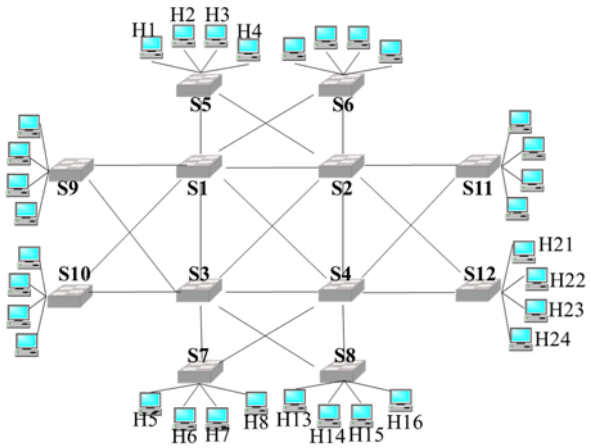

Figure 2. Mininet topology for simulation.

4.1. Effect on Faults

4.1.1. Experimental Setup

In this subsection, we simulate our proposed architecture RAD defending against internal abnormal status (i.e., link down) and evaluate the architecture from three perspectives as we set up the abnormality.

- Robustness: Packet Loss Rate (%)
- Agility: Transfer Delay (UDP), Connection Establishment Time (TCP)
- Efficiency: Throughput

Basically, the general SDN controller (i.e., Floodlight) without RAD sets up the shortest path for routing. We also vary the abnormal level based on the number of downed links.

4.1.2. Results Analysis

Figure 3 shows the packet loss rate (%) of 5-Mbps UDP packets from H1 to H5 at specific times. In this simulation, links are down at the 4th second in cases of "One Link Down (S3-S2)" and "Two Links Down (S3-S2 and S5-S1)". In the case of "No Link Down without (w/o) RAD", only one path (S5-S1-S3-S7) is used. During time 0.0–1.0 s, the route set up via the SDN controller is required and thus minimal packet loss (5.7%) occurs. When links on the used path are downed in the SDN system without RAD, 100% of the packets are lost because only one path is used and fault management for rerouting does not exist.

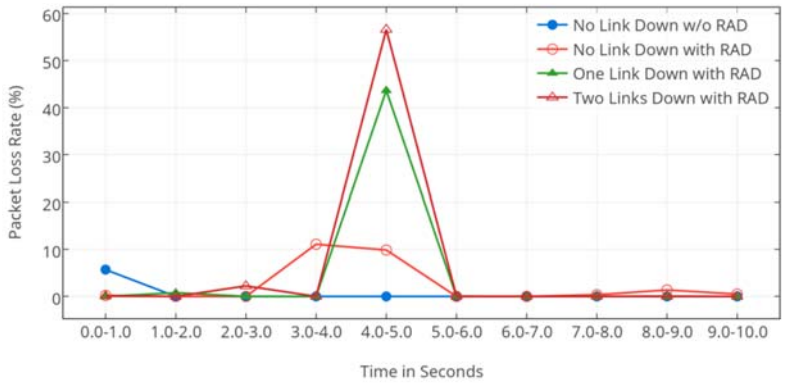

Figure 3. Packet loss rate (%) vs. time.

In the case of "No Link Down with RAD", the RAD traffic engineer distributes traffic on multiple paths that are set up by the cost function, multi-dimensional routing, and load balancing. Thus, even though this is a case of no link down, the trial time in which to find better paths and distribute traffic on paths occurs with only minimal packet loss (average of 10.5%) for 3.0–5.0 s. However, the loss decreases to nearly 0% after this period. In the case of "One Link Down", several paths are used (e.g., S5-S1-S3-S7, S5-S1-S4-S7, S5-S2-S3-S7, and S5-S2-S4-S7). If a link S3-S2 is downed at the 4th second, the fault/anomaly detection submodule of the traffic analyzer detects the fault. Then, the path (S5-S2-S3-S7) is no longer used and RAD redistributes the traffic on the remaining paths. The response time for one link down occur 43.6% for 4.0–5.0 s, but the loss stops after this period. The case of "Two Links Down" shows similar results to those of "One Link Down" and only a single path (S5-S2-S4-S7) is available. However, the increase in the number of downed links produces a greater loss rate (56.6%).

Figure 4 shows the average packet loss for 10 s. Although in the case of "No Link Down" with RAD, more packet loss occurs than without RAD, RAD quickly overcomes link down events such that packet loss occurs at a rate of only 4.9% and 5.8% in cases of one and two links down, respectively. However, in the case without RAD, 100% of packets are lost after links of path are downed.

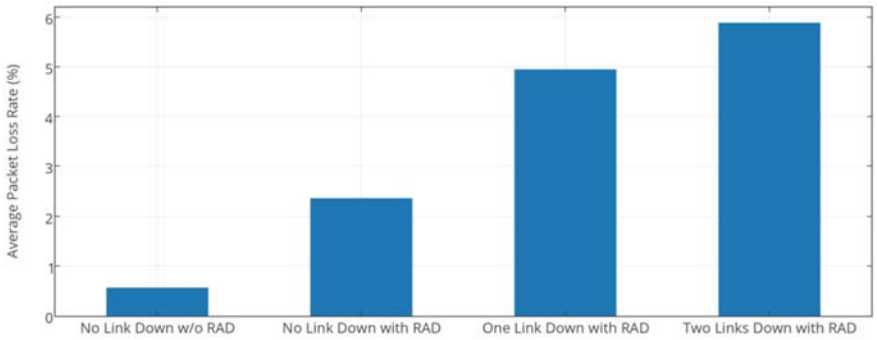

Figure 4. Average packet loss rate (%) vs. the number of links down.

Figure 5 shows the transfer delay of 5-Mbps UDP packets from H1 to H5 in milliseconds (ms). Link down events are the same as those in Figure 1. Because of the links downed at the 4th second, the delays of "One Link Down" and "Two Links Down" increase during the 4.0–5.0 s period, but the delay decreases again after 5.0 s. The delay in the "Two Links Down" case is much greater than the "One Link Down" for the 4.0–5.0 s because the former has only a single path after two links are down, whereas the latter has three available paths. Moreover, the "Two Links Down" case after the link down event has greater delay than in the other cases. This is because only one link is available and blocked traffic at the 4th second is transmitted all together after the event.

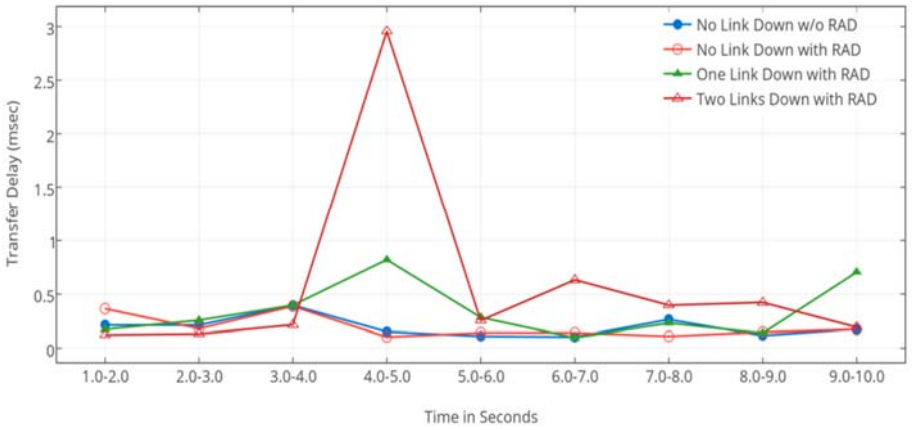

Figure 5. Transfer Delay (ms) vs. time.

Figure 6 shows the average transfer delay in ms during a period of 10 s. The greater the number of links down, the greater is the transfer delay. However, our RAD requires a reasonable increase in time compared to the original SDN system without RAD, in which all packets cannot be transferred after a link down event.

We measure the connection establishment time of TCP packets. We set up a TCP connection trial from H1 to H5 for every 1 s and calculate the average connection established time during a 10-s period. Figure 7 shows that the SDN system without RAD is influenced by the TCP connection establishment time much more during the link down event and for the number of downed links than is the system with RAD under the same conditions. Moreover, the "Two Links Down" case without RAD requires more time than does the "One Link Down" case without RAD. This is because of the decrease in the number of available links resulting from multiple links down. In the system without RAD, the controller traffic between switches and the SDN controller uses multiple paths, whereas the TCP connection packets between H1 and H5 use only a single path.

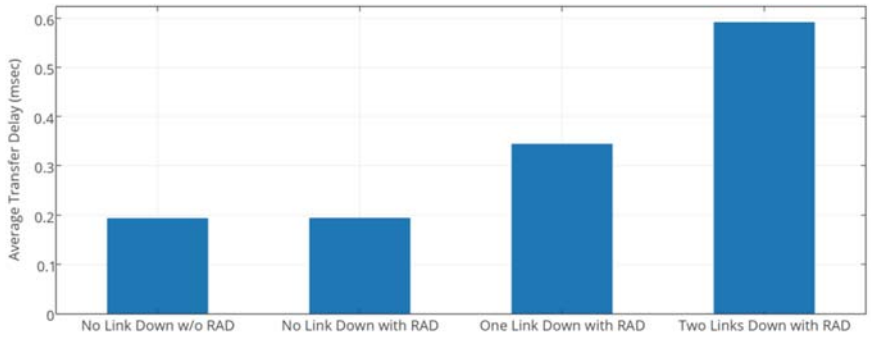

Figure 6. Average transfer delay (ms) vs. the number of links down.

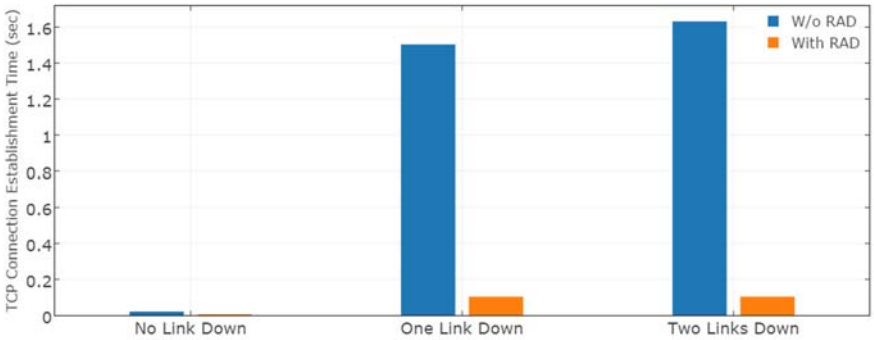

Figure 7. Average connection time (s) vs. the number of links down.

Finally, we evaluate our system in terms of efficiency and throughput. As shown in Figure 8, our RAD system overcomes link down events, where the result with one link down even outperforms somewhat that without a link down. We determine that a link down event triggers to generate the new paths and performs load balancing on them. However, the system without RAD suffers a serious decline in performance in link down events.

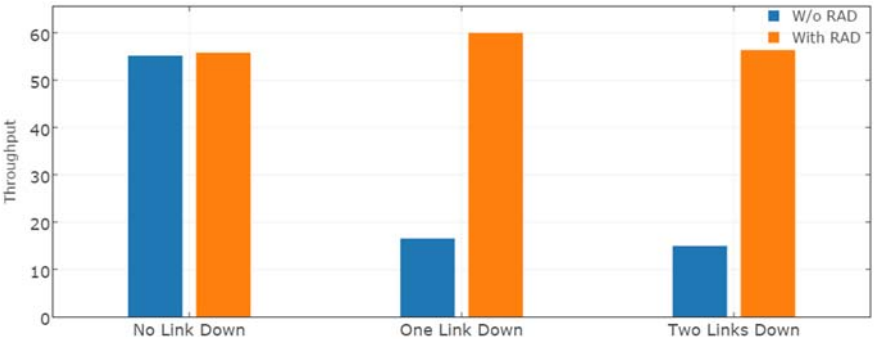

Figure 8. Throughput vs. the number of links down.

4.2. Effect on Attacks

4.2.1. Experimental Setup

In this subsection, we describe an experiment in which we mount a TCP SYN flooding attack on a network. This is a form of denial-of-service attack in which an attacker sends a succession of SYN requests to a target system in an attempt to consume sufficient server resources to cause the system to become unresponsive to legitimate traffic [50]. We then compare the defensive ability against the attack in cases both with and without RAD. We measure the UDP transfer delay and TCP connection time to illustrate defensive agility.

4.2.2. Results Analysis

First, we measure the UDP transfer delay with TCP SYN flooding attack traffic in 1000-microsecond (μs) intervals. UDP traffic sends information from H1 to H5 at a 5-Mbps rate. As shown in Figure 9, the system with RAD has less delay than does the system without RAD over time. Average delay values of the system with RAD at different degrees of attack strength (i.e., 1000-μs < 500-μs < 100-μs intervals) are also less than those of the system without RAD, as shown in Figure 10. Even though the system with RAD is more affected by severe attacks during 100-μs intervals, it suffers from fewer attacks than does the system without RAD. The reason the effect is greater with severe attacks is that many ACK packets return from a target server even as SYN packets are dropped in the system with RAD.

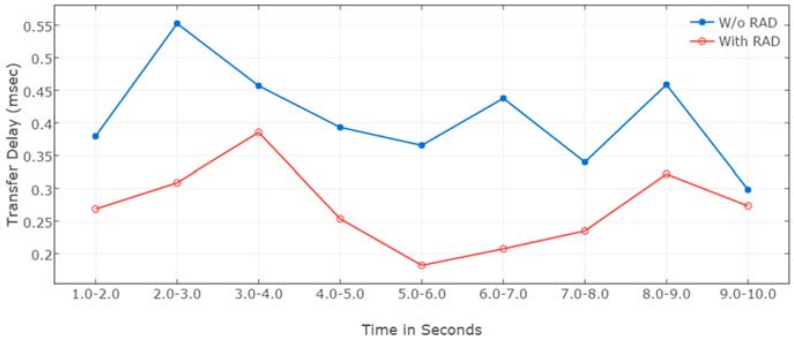

Figure 9. UDP transfer delay in simulation time (With attack in 1000 μs intervals).

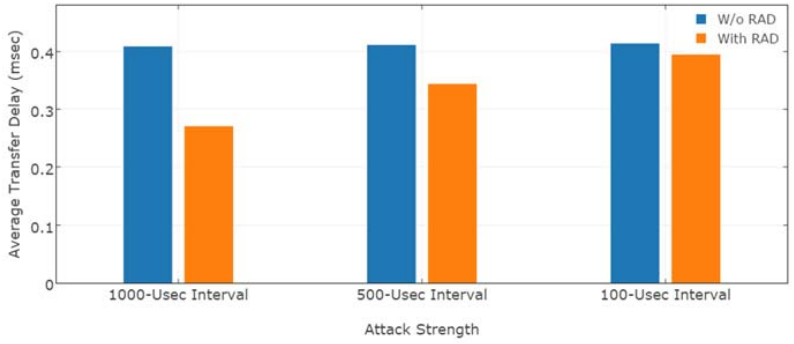

Figure 10. Average transfer delay vs. attack strength.

Next, we measure the TCP connection establishment time from H1 to H5 as we mount TCP SYN flooding attacks from H2 to H5 using three levels of strength (e.g., 2000-μs < 1000-μs < 500-μs intervals). The system with RAD has very small connection times even during attacks. However, the system without RAD suffers an attack and the effect is greater with severe attacks, as shown in Figure 11.

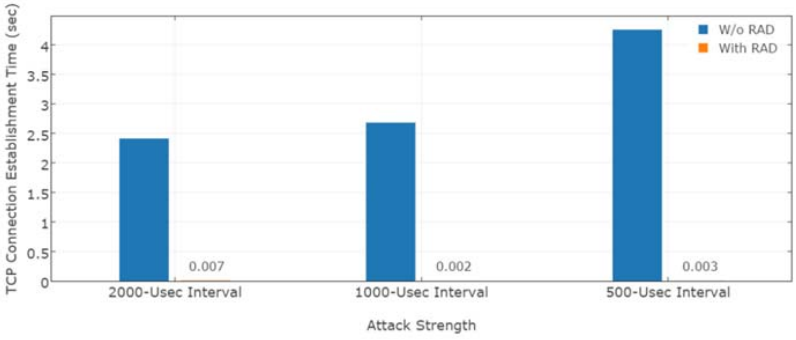

Figure 11. Average TCP connection establishment time vs. attack strength.

4.3. Effect on Elephant Flow

4.3.1. Experimental Setup

In this subsection, we compare the performance of our hybrid proactive–reactive flow rule management with a case involving complete reactivity in the network shown in Figure 2. The performance is measured based on the response time of mice flows (i.e., ping response time). In this simulation, we randomly set up mice flows from hosts on S7 to hosts on S8 (on paths S7-S3-S8 and S7-S4-S8) and an elephant flow from H5 to H13 on the same paths from the 5th to the 10th second.

4.3.2. Results Analysis

Figure 12 shows the average ping completion time of ping requests (mice flows) in msec. With RAD, our hybrid proactive–reactive flow rule management performs better, even with the elephant flow. However, the network with the normal SDN controller is severely affected by the elephant flow (e.g., 83.84 times at the 5th second). With a simple simulation, we conclude that mice flows can be easily influenced by elephant flows and perform better when the elephant flows are reactively routed by the controller.

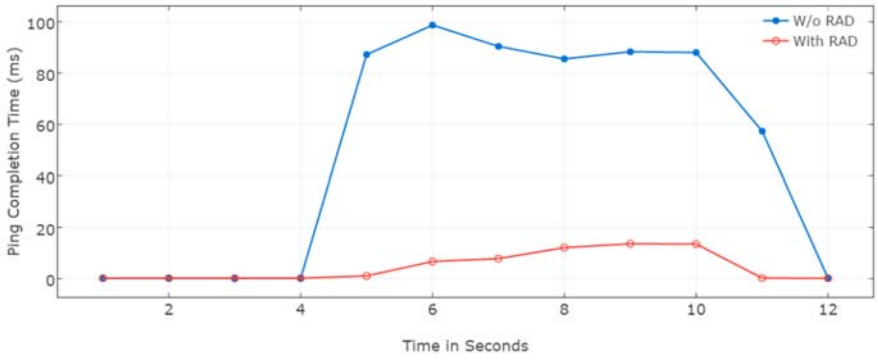

Figure 12. Ping completion time (Effect of mice flows).

5. Conclusions

In this study, we proposed a comprehensive monitoring and reactive defense system called RAD for use in defending various anomalies (e.g., internal faults, elephant flows, or outside attacks) on SDN networks. Such a comprehensive system has not been considered by previous studies on traffic engineering, monitoring, or attack defense systems. RAD offers hybrid proactive–reactive flow rule management with sFlow-RT to identify flow types, where mice flows follow a proactive flow mode and elephant flows are reactively routed according to the current network status. RAD uses Snort IDS to detect attacks. The traffic analyzer module in RAD first identifies elephant flows or detects attacks, the sends this information to the traffic engineer module. Cost function calculates each link cost based on the network utilization and delay of each link on the network. The multi-dimensional routing and load balancing module prepares new paths for the flows. The rule manager then generates the rules for new paths and pushes them to switches. We instantiated our RAD using Floodlight (an SDN controller) and Mininet (a simulation tool) to investigate the properties of the proposed system. We then conducted a series of simulation experiments for two cases with and without RAD. Our simulation results revealed that our approach is both practical and feasible, and can successfully augment an existing SDN controller in terms of agility, robustness, and efficiency. This is true even with respect to link failures, attacks, and elephant flows.

To prove the feasibility of our goal in this paper, we have designed and developed a concrete RAD system with popular exterior tools (i.e., Snort IDS or sFlow-RT). Then, the current RAD is limited by features of the used tools. General IDSs (e.g., Snort IDS and Suricata [51] with a signature-based

methodology and rule/policy driven security) operate based on traffic measurements and IDS rules, and perform misuse and anomaly detection, which is similar with Snort IDS. Moreover, sFlow-RT based on traffic measurements can be modified by other tools (e.g., NetFlow [52]) to classify the specific flows. Our current system does not solve all problems in SDN, but RAD can expand with other tools or methods [53–55] to support new features (e.g., detection against unknown anomalies). Provisioning interfaces to other tools for IDS and flow classification in RAD is a point for future study. Moreover, as further study, it will be valuable to evaluate our system with more complicated topologies, complex traffic generation and various attack scenarios, on SDN networks mixed with non-SDN switches. We will also enhance the cost function and evaluate the performances of our RAD system based on additional factors.

Acknowledgments: This work was partially supported by NSF award CNS-1637371.

Author Contributions: This work is completed by Mihui Kim, Younghee Park and Rohit Kotalwar. Mihui Kim focused on writing the paper with the traffic engineering algorithm. Mihui Kim organized evaluations and analyzed the simulation results. Younghee Park developed and designed the proposed system in this work while helping to review this manuscript. Younghee Park guided this whole project as a corresponding author. Rohit helped us to implement and experiment our proposed system. We also thanked all of students who contributed to this work through their master projects during the academic year of 2015 and 2016 under Younghee Park's directions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ahmad, I.; Namal, S.; Ylianttila, M.; Gurtov, A. Security in Software Defined Networks: A Survey. *Commun. Surv. Tutor.* **2015**, *17*, 2317–2346. [CrossRef]
2. Delivering High Availability in Carrier Grade NFV Infrastructures. In *VMware Technical White Paper*; VMware, Inc.: Palo Alto, CA, USA, 2016.
3. Wang, R.; Butnariu, D.; Rexford, J. OpenFlow-based server load balancing gone wild. In Proceedings of the USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), Boston, MA, USA, 29 March 2011; p. 12.
4. Carlos, A.B.M.; Rothenberg, C.E.; Maurício, F.M. In-packet Bloom filter based data center networking with distributed OpenFlow controllers. In Proceedings of the 2010 IEEE GLOBECOM Workshops (GC Wkshps), Miami, FL, USA, 6–10 December 2010; pp. 584–588.
5. Handigol, N.; Seetharaman, S.; Flajslik, M.; McKeown, N.; Johari, R. Plug-n-Serve: Load-balancing web traffic using OpenFlow. In Proceedings of the ACM Sigcomm Demo, Barcelona, Spain, 17–21 August 2009.
6. Wang, Y.; Zhang, Y.; Chen, J. SDNPS: A Load-Balanced Topic-Based Publish/Subscribe System in Software-Defined Networking. *Appl. Sci.* **2016**, *6*, 91. [CrossRef]
7. Nguyen, X.-N.; Saucez, D.; Barakat, C.; Turletti, T. Optimizing rules placement in OpenFlow networks: trading routing for better efficiency. In Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN), Chicago, IL, USA, 22 August 2014; pp. 127–132.
8. Nguyen, X.-N.; Saucez, D.; Barakat, C.; Turletti, T. Rules Placement Problem in OpenFlow Networks: A Survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1273–1286. [CrossRef]
9. Zarek, A.; Ganjali, Y.; Lie, D. OpenFlow Timeouts Demystified. Master's Thesis, Department of Computer Science, University of Toronto, Toronto, ON, Canada, 2014.
10. Kim, E.-D.; Lee, S.-I.; Choi, Y.; Shin, M.-K.; Kim, H.-J. A flow entry management scheme for reducing controller overhead. In Proceedings of the 2014 16th International Conference on Advanced Communication Technology (ICACT), Pyeong Chang, Korea, 16–19 February 2014; pp. 754–757.
11. Curtis, A.R.; Mogul, J.C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Banerjee, S. DevoFlow: Scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 254–265. [CrossRef]
12. Chiba, Y.; Shinohara, Y.; Shimonishi, H. Source flow: Handling millions of flows on flow-based nodes. In Proceedings of the ACM SIGCOMM Conference, New Delhi, India, 30 August–3 September 2010; pp. 465–466.

13. Nguyen, X.N.; Saucez, D.; Barakat, C.; Turletti, T. OFFICER: A general optimization framework for OpenFlow rule allocation and endpoint policy enforcement. In Proceedings of the IEEE INFOCOM, Hong Kong, China, 26 April–1 May 2015; pp. 478–486.
14. Li, H.; Li, P.; Guo, S. Morule. Optimized rule placement for mobile users in SDN-enabled access networks. In Proceedings of the IEEE GLOBECOM, Austin, TX, USA, 8–12 December 2014; pp. 4953–4958.
15. Benson, T.; Anand, A.; Akella, A.; Zhang, M. MicroTE: Fine grained traffic engineering for data centers. In Proceedings of the 7th Conference Emerging Network Experiments and Technologies (CoNEXT), Tokyo, Japan, 6–9 December 2011; pp. 1–12.
16. Ishimori, A.; Farias, F.; Cerqueira, E.; Abelem, A. Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking. In Proceedings of the 2nd European Workshop on Software Defined Networks, The Hague, The Netherlands, 10–11 October 2013; pp. 81–86.
17. Zeng, H.; Zhang, S.; Ye, F.; Jeyakumar, V.; Ju, M.; Liu, J.; McKeown, N.; Vahdat, A. Libra: Divide and conquer to verify forwarding tables in huge networks. In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Seattle, WA, USA, 2–4 April 2014; pp. 87–99.
18. Mehdi, S.A.; Khalid, J.; Khayam, S.A. Revisiting traffic anomaly detection using software defined networking. In Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection (RAID), Menlo Park, CA, USA, 20–21 September 2011; pp. 161–180.
19. Yu, Y.; Qian, C.; Li, X. Distributed and collaborative traffic monitoring in software defined networks. In Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN), Chicago, IL, USA, 22 August 2014; pp. 85–90.
20. Suh, J.; Kwon, T.; Dixon, C.; Felter, W.; Carter, J. OpenSample: A low-latency, sampling-based measurement platform for commodity SDN. In Proceedings of the IEEE 34th International Conference on Distributed Computing Systems (ICDCS), Madrid, Spain, 30 June–3 July 2014; pp. 228–237.
21. Yu, M.; Jose, L.; Miao, R. Software defined traffic measurement with OpenSketch. In Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI), Lombard, IL, USA, 2–5 April 2013; pp. 29–42.
22. Chowdhury, S.R.; Bari, M.F.; Ahmed, R.; Boutaba, R. PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks. In Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–9.
23. Shin, S.; Song, Y.; Lee, T.; Lee, S.; Chung, J.; Porras, P.; Yegneswaran, V.; Noh, J.; Kang, B.B. Rosemary: A robust, secure, and high-performance network operating system. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), Scottsdale, AZ, USA, 3–7 November 2014; pp. 78–89.
24. Hong, S.; Xu, L.; Wang, H.; Gu, G. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 7 February 2015; pp. 78–89.
25. Shin, S.; Yegneswaran, V.; Porras, P.; Gu, G. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, Berlin, Germany, 4–8 November 2013; pp. 413–424.
26. sFlow-RT. Available online: http://www.inmon.com/products/sFlow-RT.php (accessed on 1 December 2016).
27. Snort. Available online: https://en.wikipedia.org/wiki/Snort (accessed on 1 December 2016).
28. Kreutz, D.; Ramos, F.M.V.; Veríssimo, P.E.C.; Rothenberg, E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
29. OpenFlow Switch Consortium. Available online: http://archive.openflow.org/ (accessed on 1 December 2016).
30. Sundaresan, S.; Donato, W.; Feamster, N.; Teixeira, R.; Crawford, S.; Pescapè, A. Broadband internet performance: A view from the gateway. *SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 134–145. [CrossRef]
31. Kim, H.; Feamster, N. Improving network management with software defined networking. *IEEE Commun. Mag.* **2013**, *51*, 114–119. [CrossRef]
32. Kim, J.; Filali, F.; Ko, Y.-B. Trends and Potentials of the Smart Grid Infrastructure: From ICT Sub-System to SDN-Enabled Smart Grid Architecture. *Appl. Sci.* **2015**, *5*, 706–727. [CrossRef]

33. Casado, M.; Garfinkel, T.; Akella, A.; Freedman, M.J.; Boneh, D.; McKeown, N.; Shenker, S. SANE: A protection architecture for enterprise networks. In Proceedings of the 15th conference on USENIX Security Symposium (USENIX-SS), Vancouver, BC, Canada, 31 July–4 August 2006.
34. Hand, R.; Ton, M.; Keller, E. Active Security. In Proceedings of the 12th ACM Workshop on Hot Topics in Networks (HotNets-XII), College Park, MD, USA, 21–22 November 2013.
35. Porras, P.; Shin, S.; Yegneswaran, V.; Fong, M.; Tyson, M.; Gu, G. A security enforcement kernel for OpenFlow networks. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN), Chicago, IL, USA, 22 August 2014; pp. 121–126.
36. Shin, S.; Porras, P.; Yegneswaran, V.; Fong, M.; Gu, G.; Tyson, M. FRESCO: Modular composable security services for software-defined networks. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 24–27 February 2013.
37. Porras, P.; Cheung, S.; Fong, M.; Skinner, K.; Yegneswaran, V. Securing the software-defined network control layer. In Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 8–11 February 2015.
38. Hayward, S.S.; Kane, C.; Sezer, S. Operationcheckpoint: Sdn application control. In Proceedings of the IEEE 22nd International Conference on Network Protocols (ICNP), Research Triangle Park, NC, USA, 21–24 October 2014; pp. 618–623.
39. Wen, X.; Chen, Y.; Hu, C.; Shi, C.; Wang, Y. Towards a secure controller platform for openflow applications. In Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013; pp. 171–172.
40. InMon. Available online: http://www.inmon.com/ (accessed on 1 December 2016).
41. Cheng, W.; Ren, F.; Jiang, W.; Qian, K.; Zhang, T.; Shu, R. Isolating Mice and Elephant in Data Centers. *arXiv*, 2016; arXiv:1605.07732.
42. RestFlow. Available online: https://github.com/restflow-org/restflow/wiki (accessed on 1 December 2016).
43. Lohier, S.; Rachedi, A.; Doudane, Y.G. A cost function for QoS-aware routing in Multi-tier Wireless Multimedia Sensor Networks. *Lect. Notes Comput. Sci.* **2009**, *5842*, 81–93.
44. Badia, L.; Lindström, M.; Zander, J.; Zorzi, M. An economic model for the radio resource management in multimedia wireless systems. *Comput. Commun.* **2004**, *27*, 1056–1064. [CrossRef]
45. Shaikh, A.; Rexford, J.; Shin, K.G. Evaluating the impact of stale link state on quality-of-service routing. *IEEE/ACM Trans. Netw.* **2001**, *9*, 162–176. [CrossRef]
46. Mininet. Available online: http://mininet.org/ (accessed on 1 December 2016).
47. Floodlight. Available online: http://www.projectfloodlight.org/floodlight/ (accessed on 1 December 2016).
48. Iperf. Available online: https://iperf.fr/ (accessed on 1 December 2016).
49. Hping. Available online: http://www.hping.org/ (accessed on 1 December 2016).
50. TCP SYN Flooding and IP Spoofing Attacks. Advisory, Software Engineering Institute, Carnegie-Mellon University, 1996. Available online: https://www.cert.org/historical/advisories/CA-1996--21.cfm (accessed on 1 December 2016).
51. Suricata. Available online: https://suricata-ids.org/ (accessed on 18 Febrary 2017).
52. NetFlow. Available online: https://en.wikipedia.org/wiki/NetFlow (accessed on 18 Febrary 2017).
53. Groš, S. Anomaly Detection in Snort. Available online: http://sgros.blogspot.com/2015/02/anomaly-detection-in-snort.html (accessed on 18 Febrary 2017).
54. SNORT.AD. Available online: http://www.anomalydetection.info/?home,1 (accessed on 18 Febrary 2017).
55. Ding, Y.X.; Xiao, M.; Liu, A.-W. Research and implementation on snort-based hybrid intrusion detection system. In Proceedings of the International Conference on Machine Learning and Cybernetics, Baoding, China, 12–15 July 2009; pp. 1414–1418.

© 2017 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).