Leveraging Network Slicing in SDNs for Handling Application Failures

Bommareddy Lokesh, *Member, IAENG,* and Narendran Rajagopalan

***Abstract*—Unlike traditional networks, Software-defined networks (SDNs) provide an overall view and centralized control of all the devices in the network. SDNs enable the network administrator to implement the network policy by programming applications on top of the SDN controller using generic APIs. One or more controller instances can be deployed to administer the data flow by maintaining unified control of the entire network. The controller is expected to respond quickly to queries from forwarding devices. Presuming quick responses from the controller while enforcing a complex security mechanism is unreasonable. In this paper, the authors propose a unique, adaptive, lightweight, yet efficient technique called ISOLATOR to mitigate the effect of insider attacks and malfunctioning of distributed applications in an SDN-enabled cloud. The proposed security application upon detecting any suspicious activity by a virtual machine, isolates it by removing the interface to its respective shared network and reconnects it through a restricted network operating in a highly selective mode. By subjecting the data traffic to deep packet inspection, the restricted network searches for a match with a known worm pattern. The application is programmed for the OpenDayLight controller and the results show a significant improvement in identifying malicious activities with minimal latency and computational cost.**

***Index Terms*—Openstack, OpenDayLight SDN-controller, North-bound-APIs, OpenFlow.**

I. Introduction

IN an SDN environment, the controller has to supervise the data flows between hosts by populating the forwarding tables of intermediate devices with necessary flow rules. NOX was the first SDN controller introduced and is well-known for C++ as the North Bound API (Application Programming Interface). Subsequently, many other controllers supporting a wide variety of APIs came into existence, namely ONOS, Floodlight, OpenDayLight, and RYU. These broad ranges of SDN controllers and their APIs improved the flexibility and programmability of the network. The generic North Bound APIs allow the network administrator to dynamically manage the network traffic using flexible and policy-driven controller applications. Accordingly, the controller installs or modifies necessary flow rules in the device's forwarding tables using South Bound APIs. Openflow is the standard and predominantly used South Bound Interface (SBI) [1].

Typically, the flow rules remain in the device's forwarding tables for a limited duration and are suspended upon timeout. Usually, two types of timeouts exist in the SDN paradigm, namely idle and hard timeouts. A flow entry hits idle timeout and is removed from the forwarding table if no reference to that particular flow entry arises for a fixed time period. Flow entries are also evicted after a predetermined duration irrespective of its activity. In that case, it is termed a hard timeout. Devices can restore forwarding table entries that have been suspended due to timeout by contacting the controller.

The performance of the controller apparently deteriorates if it is overwhelmed with a large number of flow queries. Li et al. introduced CPMan for reducing the load on the controller [2]. They propose to enhance the switches by equipping them with more information, resulting in fewer queries to the controller. In the case of large networks, a single instance of the controller with practically limited capacity may not be able to handle all the requests alone. In a clustered approach, the data store needs to be instantly updated to ensure a conflict-free platform for all controller instances to maintain a logically centralized view of the network. ONOS, by architecture, can logically centralize the control easily and improve the scalability of the network. However, it is recommended to operate with the minimum number of controller instances possible to achieve better synchronization across the network.

As OpenFlow has evolved, the number of fields in the packet header has gradually increased, leading to a more complex implementation. To address this issue, a high-level language called Programming Protocol-independent Packet Processors (P4) was proposed as a straw man proposal. It is still in the early stages of development [3]. It is an attempt to customize the way switches process their packets. P4 throws light on how OpenFlow should evolve in the future. For better scalability and manageability, cloud providers like Google and IBM have already adopted SDN in their clouds, and many other cloud service providers are likely to adopt SDN technology at their data centers soon [4], [5]. Mininet is the predominantly used emulator for testing the controller applications. Networks with arbitrary configurations can be set up rapidly using Mininet. Moreover, Mininet enables the SDN controller to run as an external application on a separate machine. Despite the benefit of simplified and rapid prototyping, the results obtained in the Mininet are not trustworthy, since the hosts and network components run as kernel processes in the host machines [6].

Evaluating the performance of controller applications in a realistic cloud platform, such as OpenStack, would result in more realistic outcomes when compared to the results obtained in Mininet. This is because in OpenStack, testing can be performed on actual systems [7], [8]. DevStack is a simplified version of Openstack that can be used to build a cloud platform within a stand-alone machine. Since SDNs are very new, networking devices that support SDN functionality are very costly. Traditional devices are relatively cheap but are proprietary in nature. To test the proposed technique, we

Manuscript received February 3, 2023; revised December 20, 2023.

Bommareddy Lokesh is an Assistant Professor in the School of Computer Science and Engineering, VIT-AP University, Amaravati, Andhra Pradesh, 522237, India. e-mail: lokesh.bommareddy@vitap.ac.in

Narendran Rajagopalan is an Associate Professor in the Department of Computer Science and Engineering, National Institute of Technology Puducherry, Karaikal, 609609, India. e-mail: narendran@nitpy.ac.in

opted to use OpenvSwitch (OVS) to configure the desired network. Apart from implementing the proposed security mechanism, this paper also presents some of the vulnerabilities in SDNs. Fuzzers like SCAPY (a Python script used to learn the nature of protocols) are used to fabricate the network traffic for identifying possible vulnerabilities. SCAPY can be used to clone, cast packets and decode the replies by matching them with similar responses to study the controller functionality.

The unique contributions of this work include:

1. A dynamically adaptable statistical-based approach to detect abnormal traffic in an SDN-enabled Data center network (DCN) is proposed and implemented.
2. A simple VM isolation technique to mitigate the effect of malfunctioning distributed applications in an SDN-enabled network is presented.

The remaining sections of this paper are organized as follows. section II reports the related work. Section III discusses the nature of data center applications and their traffic patterns. section IV presents a brief overview of Openstack and details about the experimental setup. section V discusses the process of configuring Openstack in conjunction with OpenDayLight controller and the proposed security algorithms. section VI presents the tests conducted and the results obtained on implementing the proposed model. The concluding remarks are presented in section VII.

II. Related work

In this section, our primary focus is to review existing works proposed to make SDNs resilient.

Porras et al. introduced FortNOX, a framework to dynamically detect and resolve conflicting flow entries injected by Openflow applications in NOX controller [9]. ControllerSEPA, a security plug-in was introduced by Y. Tseng [10]. This plug-in supports application-based Authentication, Authorization, and Accounting services (AAA). A detailed study on benefits from a security perspective through various features of SDN is presented by Shin et al. [11].

Corybantic, a conflict-free modular model for synchronizing controller applications that compete for resources to fulfill their independent objectives is proposed and implemented by Mogul et al. [12]. Seo et al. proposed a solution for security vulnerabilities that result as a part of communication between the identification process and cloud services [13]. Shin et al. addressed possible attacks like control plane saturation and responsive challenge. Connection migration and actuating triggers are the two techniques proposed and implemented for mitigating those attacks respectively [14].

Network virtualization allows multiple isolated logical networks to share the same physical infrastructure while maintaining isolation and hardware forwarding speeds. This is achieved through a novel switch-level virtualization approach that utilizes commodity switching chipsets and does not require programmable hardware[15]. Caron and Cornabas proposed a model to avoid data leakage and modification using VM placement heuristics considering different levels of isolation [16]. The proposed method minimizes performance interference. Yuchi and Shetty proposed security-aware VM placement or VM migration in the cloud to avoid the risk of security exploits on vulnerable virtual machines [17]. The authors recommended the migration of the vulnerable machine image into a separate physical server to reduce the risk of attacks. However, VM migration needs a lot of network's bandwidth and the downtime is even daunting [18], [19]. Woo et al. introduced a framework called RECHECKER to discover bugs and vulnerabilities in RESTful services provided by SDN controllers [20].

Bari et al. proposed "CQNCR" a technique to determine the ideal execution order of workload migration [19]. This model has shown a significant improvement in the availability of workloads and the time taken for migration within a data center. Seungwon Shin and Guofei Gu proposed and implemented CloudWatcher, which acts as an application for network monitoring to provide adequate security [21]. A multi-player dynamic game based on rewarding and penalizing for the utilization of network bandwidth was suggested by Chowdhary et al. [22]. Their model is tested on the OpenDayLight controller and their model do not allow scaling up the network. According to the authors, their algorithmic complexity is linear and is dependent on the system's user count.

Chinese wall policy-based security-awareness VM placement scheme (SVMPS) was proposed and implemented by Yu et al. [23]. Their scheme provides isolation between conflicting users. Due to the overall view of the network, Distributed Denial of Service and Denial of Service attacks can be easily detected and alleviated in an SDN-enabled cloud [24], [25]. An efficient packet-level traffic monitoring using vTAP in Openstack environment is presented by Jeong et al. [26]. The performance analysis of live VM migration in Openstack is presented by He et al. [27].

i Isaac Thulo and Eloff presented a collective mechanism keeping in view the QoS, cost, and security of cloud [28]. Authors have evaluated the existing VM placement algorithms and identify those that show the potential to be considered for future enhancement from a security perspective. SDNs offer flexibility for network virtualization which enables them to create multiple virtual networks on top of the existing physical network [29], [30], [31], [32], [33].

A model for the dynamic restoration of a virtual network as a result of a breakdown in the network hardware is proposed and implemented by Ko et al. [34]. Application-aware prioritization of flows in collaboration with a Deep Packet Inspection is proposed and implemented by Jeong et al. [35]. Benchmark studies of the traditional state-of-the-art data center network virtualization and its importance are presented by Bari et al. [36]. In [37], the authors proposed and implemented a method for embedding Virtual Data Centers (VDCs) by pooling resources from multiple physical data centers using Locator/Identifier Separation Protocol (LISP) and Openflow. The authors presented an efficient method to connect different DCNs as a local area network.

Counters and rate fields in the meter tables introduced in Openflow version 1.3 provide rate-limiting capabilities in SDNs. Rate limiting capabilities of various Openflow protocol versions and the support offered by specific open-source SDN controllers are presented by Karakus and Durresi [38]. Monitoring of both network and host data for detecting abnormal activity through a machine learning algorithm is proposed and implemented by Kim et al. [39]. The importance of SDNs in determining application-centric requirements at a high level is presented by Lopez et al. [40].

Network access control through fine-grained flows in FlowNAC and application-specific network flows in BYOD is presented by Hong et al., and Matias et al. respectively [41], [42]. A unique model to standardize and automate the process of vulnerability identification in SDNs is proposed and implemented by Lee et al. [43].

Post survey of the available literature from a security perspective, and testing using some of the open-source tools for known vulnerabilities in SDNs [44]. In this paper, the authors proposed a scheme to logically isolate the suspected VM from its shared network using the network virtualization feature of the SDNs. The suspected VMs are blocked following a Deep Packet Inspection. SDNs have the ability to allow such dynamic reconfiguration of the network by virtualizing the underlying network hardware.

Algorithm 1 Trust index algorithm.

Data: VMG: List of VMs in a Group.
Data: *ShNw*: Shared Network for VMs in a Group.
Data: *RsNw*: Restricted Network with a Traffic Analyzer.
Data: RSD(T_i): Recorded Statistical Data in time slot T_i.
for all pairs of RSD[$T_1 \rightarrow T_N$] in the VMG **do**
    $r(i,j)$= **Correl**(RSD[i], RSD[j])
if pattern identified from $r(i,j)$ **then**
    $PRT \leftarrow newRepeatTime$
Calculate $trust\_index$ using (1)
for each RSD[x] **do**
    for $i = N$ to 1 and $j = N$ to 1 **do**
        $r(i-1, j-1) = r(i,j)$
    while $N - PRT > 0$ **do**
        $r(N,N\text{-}PRT)$=**Correl**(RSD[N], RSD[N-PRT])
        $PRT = PRT * k$, where $k = 1 \rightarrow n$
    Update $trust\_index$ using (1)
    if $trust\_index <$ critical value and *ShNw* $\leftarrow true$ **then**
        Detach interface to *ShNw*
        Connect to *RsNw*
    else if $trust\_index >$ critical value and *RsNw* $\leftarrow true$ **then**
        Detach interface to *RsNw*
        Connect to *ShNw*
procedure CORREL(a, b)
    $r_i = \frac{n(\Sigma ab) - (\Sigma a)(\Sigma b)}{\sqrt{[n\Sigma a^2 - (\Sigma a)^2][n\Sigma b^2 - (\Sigma b)^2]}}$
    return r_i

Algorithm 2 Proposed Security algorithm.

Data: VMG: List of VMs in a Group.
Data: *ShNw*: Shared Network for VMs in a Group.
Data: *RsNw*: Restricted Network with a Traffic Analyzer.
for each VM in the VMG **do**
    examine the *trust_index*
    if *trust_index* not acceptable **then**
        if VM frequently blacklisted or a worm pattern is detected **then**
            detain the VM and report to the VMG admin
        else blacklist the VM
            discard link to *ShNw* and reconnect to *RsNw*
    else if VM not directly connected to *ShNw* **then**
        retain privileges and reconnect to *ShNw*
    else proceed to next VM

III. Data center applications & Traffic Patterns

In this section, we shall explore various data center applications and their traffic patterns to realize the key points in designing a security model for an SDN-enabled data center network. A comparative study of traffic patterns at various data centers including Facebook and Google is taken into account. [45], [46].

Volume: Survey reveals that large volumes of data are being moved within a data center. For instance, Facebook's application requires an average of 521 distinct internal fetches in processing a user request to load one of its popular pages. Apart from the data center application traffic, data processing tools (like Hadoop, Spark, etc.) that provide useful information for these web applications also transfer massive amounts of data inside the data center. [47] shows that the volume of data at Google's data center is noted to be doubling every year over a decade.

Locality: An attempt to determine the locality of traffic at Facebook's data center discovered that around 13 percent of data center traffic is rack local. Interestingly, about 58 percent of traffic is not within the rack but within the cluster and is the majority of the traffic that Facebook's servers generate. Moreover, approximately 12 percent of the traffic stays within the data center and 18 percent of the traffic is across data centers. [47] More information from Google's data center about locality shows that almost 91 percent of the traffic is non-local at any block. Blocks are larger when compared to a rack, but smaller than a cluster. Google's stats throws light on how they organize their storage for guaranteeing high availability.

Benson et al. analyzed the data center traffic at enterprises, universities, and commercial cloud networks. Their study shows that almost 70 percent of the data center traffic is rack local which is entirely different from Facebook's and Google's measurements [48]. The reason for conflicting numbers might be due to the nature of workloads, the size of map-reduce tasks, or the differences in organizing compute and storage resources, etc.

Concurrent connections: Facebook's data shows that hundreds and thousands of concurrent connections are quite common for web servers in their data center. Facebook's Hadoop-style workloads tend to have an average of about 25 concurrent connections which is much higher when compared to only six correspondents at a different data center running similar workloads (two rack local servers and four non-local servers).

Flow rate: The average flow rate at a server in Facebook's data center is approximately 2*ms*; on the contrary, at a different data center with at least 1000 servers in a cluster, the inter-flow arrival time is much larger by almost ten times [49].

Flow size: Hadoop flows at Facebook's data center are lesser than a kilobyte and are the majority of flows that their servers exchange at irregular time intervals. A small fraction of flows

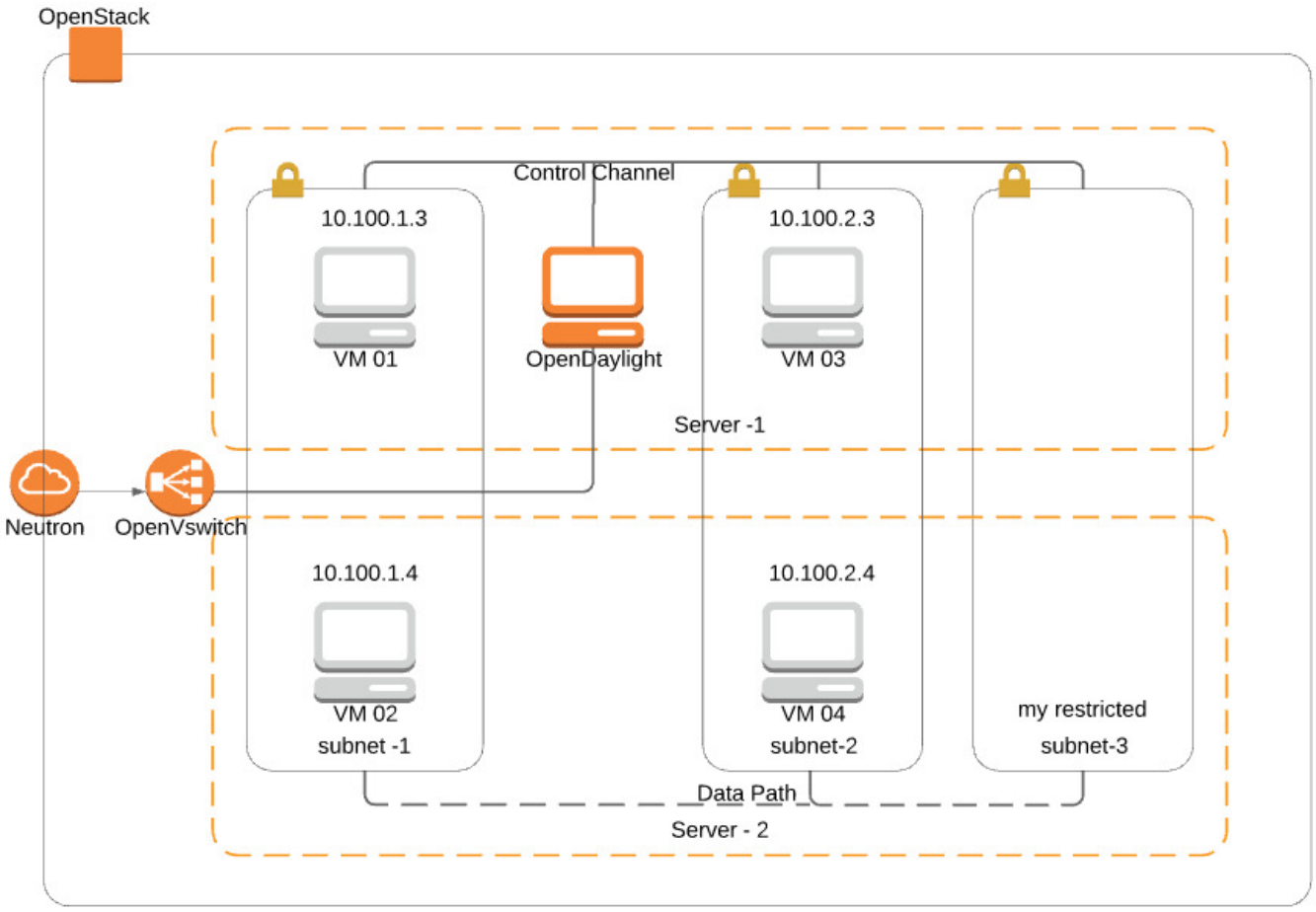

Figure 1: Integration of Openstack and OpenDayLight SDN controller.

exceed a megabyte and persist for more than 100*sec*.

The necessary inference from the above discussion is that the traffic pattern at a data center is dependent on several aspects like the nature and design of web applications, the scale at which they are running, the network's design, etc.

Another significant finding from the analysis of data center traffic is that application-specific traffic tends to persist for longer durations and exhibit a harmonic pattern. Most of them may be using TCP connections and perhaps to avoid the overhead involved in TCP handshakes, they remain connected for a longer duration.

IV. Experimental setup

Openstack is a cloud operating system providing IaaS (Infrastructure-as-a-Service). It can virtualize computing, network, and storage resources throughout the cloud environment [50]. The six core components of Openstack are

i. Horizon: A dashboard to manage the cloud resources. However, in Openstack managing resources via. CLI (Command Line Interface) is also possible.

ii. Neutron: Provides a layer of abstraction on the underlying physical networking devices to achieve desired Network Function Virtualization (NFV).

iii. Glance: An Openstack service that provides services to store, share and handle bootable disk images.

iv. Keystone: An identity validation service that provides Authentication, Authorization, and Accounting (AAA).

v. Cinder: Provides access to block storage resources.

vi. Nova: Provides access to compute resources, including containers.

By default, Openstack does not allow incoming traffic to any of the spawned machine instances. So, a security group rule shall be explicitly added to permit external access to the VMs. To implement the proposed model the experimental setup with the following specifications is considered. The graphical view for the same is shown in Figure 1.

A. *Hardware specifications*

CPU	Cores	RAM	Compute node
i7-2600 @3.4 GHz	4	16GB	server-1
i7-2600 @3.4 GHz	4	8GB	server-2

B. *Software specifications*

Operating System : Ubuntu 18.04.3 LTS
Cloud Platform : Openstack pike
SDN Controller : OpenDayLight Lithium

V. Openstack in conjunction with OpenDayLight controller

SDNs can facilitate the creation of a cutting-edge cloud platform that offers improved security and flexibility. This platform can adapt to dynamically changing workloads and can be controlled from a single point. In this section, the procedure to integrate Openstack with OpenDayLight (ODL)

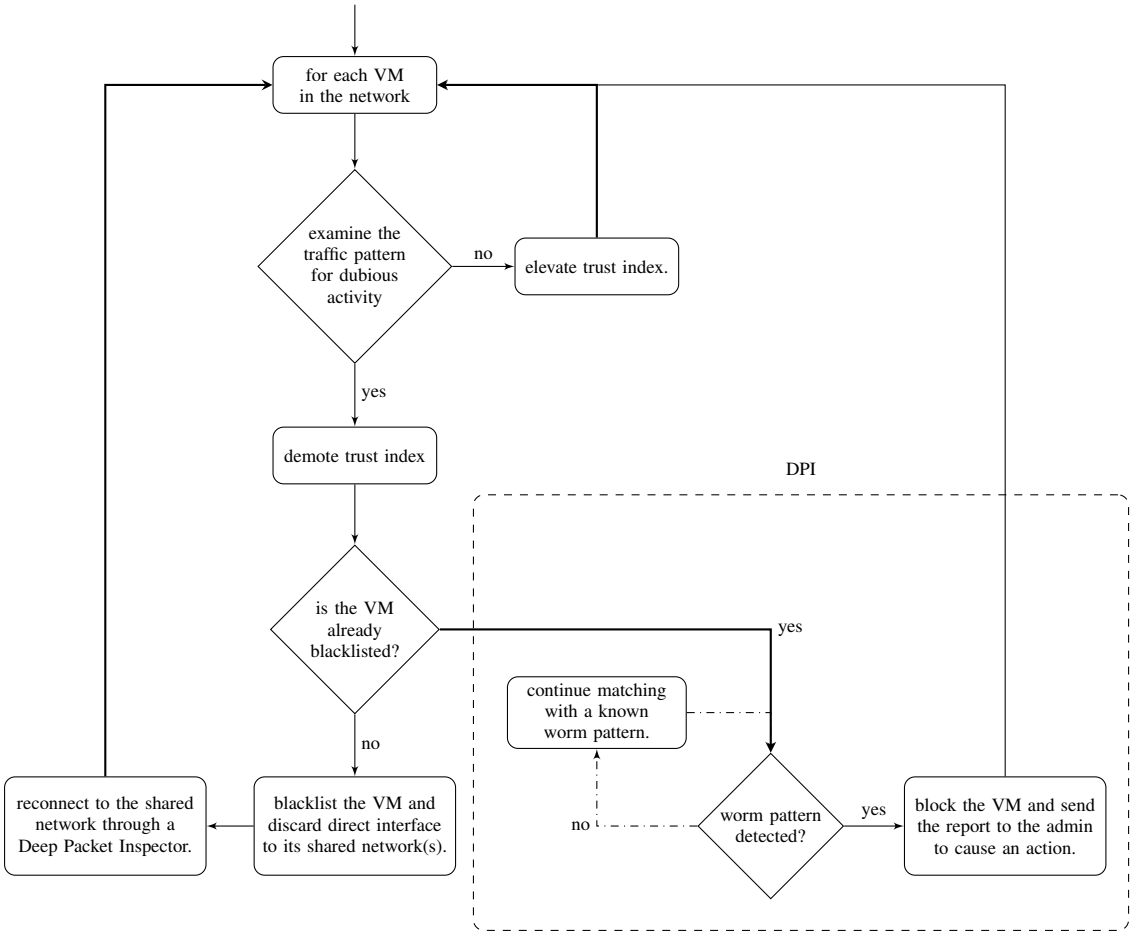

Figure 2: Flow diagram of the proposed security technique.

controller and manage networking services is discussed in detail. Integrating Openstack with OpenDayLight SDN controller facilitates substantial control over monitoring and managing the network resources at layer 2. ODL is a widely used open-source SDN controller developed by the Linux team and runs on the Java platform. OpenStack is a set of open-source projects that can be leveraged to create a cloud environment on a server. Using Openstack, VMs can be spawned rapidly with readily available bootable system images like Cirros, Fedora, Ubuntu, etc.

A. Configuring Openstack and OpenDayLight

Before launching OpenStack or ODL, it is advisable to remove all existing virtual machines (VMs), networks, subnets, and ports. Later the ODL controller is initiated and the following features are installed. To integrate ODL with OpenStack, the *odl-ovsdb* feature is essential, while the *odl-dlux* feature is optional and provides a graphical user interface. After creation, each machine instance is assigned an IP address within the local network. OpenStack only permits a restricted range of IP addresses.

Machine instances can communicate with each other in OpenStack through the use of Linux bridges and OpenvSwitch (OVS). In comparison to Linux bridges, OVS provides improved control at layer 2, such as VLAN segmentation, reuse of IP addresses, and enhanced flexibility for virtual machine communication. Apart from a direct interface to the external network, OpenStack enables machine instances to access the external network via a private network, which is facilitated by a router. Virtual machines that belong to the private network cannot communicate with the external network directly. Therefore, to enable communication with the external network, each machine instance requires a floating IP address. Openstack uses the conventional RSA algorithm for key pair generation.

The Modular Layer 2 (ML2) plug-in of the Neutron module allows the Openstack to coalesce with the ODL controller. ML2 can collectively work with the *OpenvSwitch, HyperV,* and *Linux bridge L2 agents*. By default, ML2 do not load any mechanism drivers. The *odl-neutron-service* feature provides integration support for Openstack via the OpenDayLight ML2 mechanism driver. IP tables are used to realize security groups in the Openstack platform. Security groups help to filter traffic based on the VM group policies. The *reservation module* in the OpenDayLight controller provides dynamic low-level resource reservation capability.

B. Algorithm

The proposed algorithm is based on the fact that harmonic traffic is more likely to be legitimate. We employed four different pattern recognition techniques, in addition to logistic regression, to detect harmonic traffic patterns and then

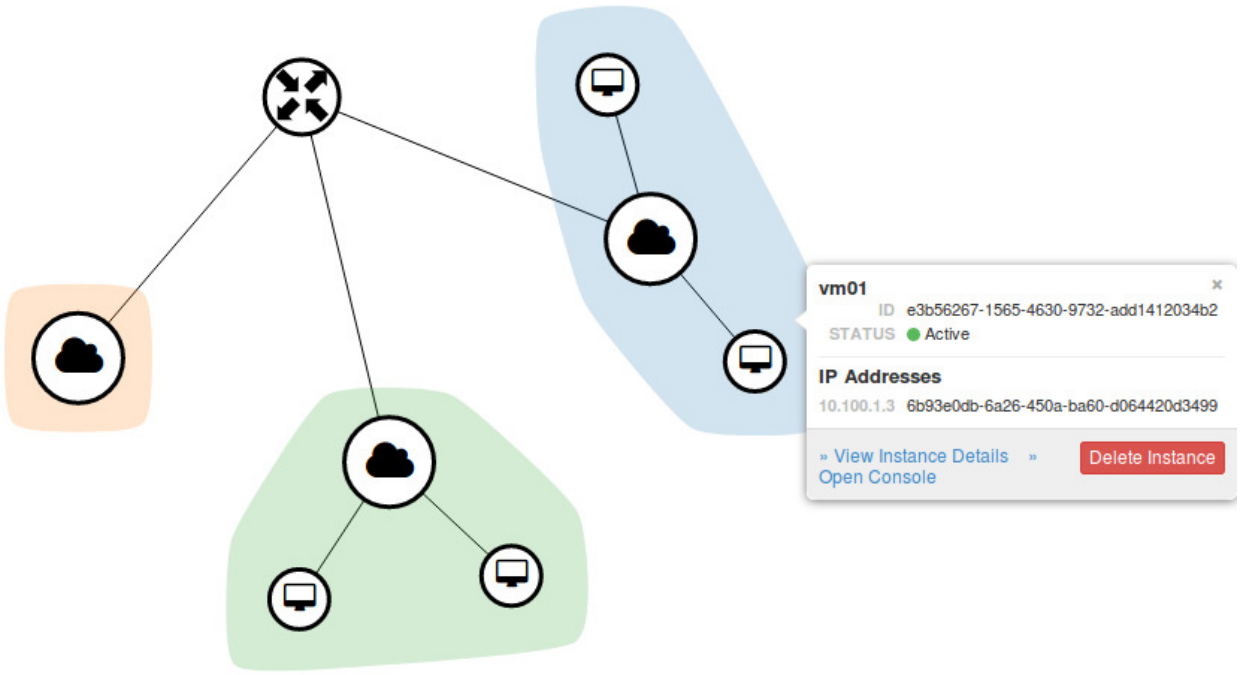

Figure 3: Initial configuration.

Table I: Attacks vs Mitigation Techniques.

Attack Scenario	**Mitigation Technique**
Run a port scanner on a host machine and try to learn the network details or perform DoS attacks on open ports.	Due to the overall view of the entire network in SDNs, port scanners like nmap, superscan can be easily detected and blocked because they flood the traffic.
Create raw packets using SCAPY and flood them to the peer machine instances. IP spoofing attack on peer machine instance performed using SCAPY.	By virtue of *HostLocationService* at the controller and enforcing port classification technique, spoofed packets can be readily identified and dropped.
SDN controller periodically casts LLDP packets in plain text format to learn the network topology using the *LinkDiscoveryService*. Similar control packets are fabricated and broadcasted through different ports for topology poisoning.	Efficient link discovery by guarding against link fabrication attack and improving the performance by eliminating superfluous traffic is proposed by Zhao et al. [51].
In Single, multiplanar or a hybrid private planar network models, one VM demanding more shared resources leads to side-channel attacks.	Algorithms to mitigate similar attacks are proposed by Lopez et al. [40] that allow application-centric sharing of network resources by a group of VMs [40].

Table II: Application Specific Results.

Application Protocol	**Total Flows**	**% of flows classified as benign traffic**	**% of flows classified as dubious traffic**	**Detection Rate**
ICMP	157	100.00	0.00	100%
FTP(active)	97	81.44	18.56	98%
FTP(passive)	126	88.10	11.90	97%
SSH	12,465	99.41	0.59	87%
SMTP	154	98.05	1.95	100%
HTTP	26,254	95.11	4.89	81%
POP	57	100.00	0.00	100%
DNS	5,962	100.00	0.00	100%

compared their outcomes with those of logistic regression.

i. Fourier Transform: The Fourier transform stands as a powerful method for examining harmonic patterns. It dissects time series data into its fundamental sinusoidal constituents, unveiling the frequencies and amplitudes of the harmonic patterns.

ii. Periodogram Analysis: A Periodogram is employed to estimate the frequency components or recurring patterns within time series data. In this approach, we compute the periodogram, which is a tool for frequency-domain analysis, to pinpoint the primary periodic elements within the traffic patterns.

iii. Autocorrelation Analysis: Autocorrelation analysis, also referred to as auto-correlation, is a statistical approach utilized to assess and quantify the level of resemblance or correlation between time series data and a time-shifted version of itself. Autocorrelation assesses the likeness of a signal at various time lags and can unveil inherent patterns, recurring cycles, and trends within the data. Within this method, we calculate the autocorrelation function of time series data to identify recurring patterns or periodicities.

iv. Harmonic Regression: In this method, we employ regres-

sion techniques to represent data as a composite of harmonic functions, such as sine and cosine waves. The amplitudes and frequencies of these functions symbolize the harmonic elements.

The findings indicate that the utilization of logistic regression to identify harmonic traffic patterns in the KDD dataset yielded a higher F1 score.

Based on the VM's historical behavior and resource usage, the controller maintains the *trust_index* for each VM operating under its supervision using Algorithm 1. The credibility of VMs is assessed by examining their *trust_index*. In a VM group, the ACL is used for identifying collaborative VMs. In a supervised approach, along with the ACL, the VM group admin has to provide information about the probability of communication between all pairs of VMs in the group. In an unsupervised approach, the communication probability between all pairs of VMs in the group can be learned by analyzing the network traffic over a certain period.

The controller can retrieve various types of statistics, such as flow, table, or group statistics, from an OpenFlow switch by sending an *OFPT_STATS_REQUEST* message. The switch responds with an *OFPT_STATS_REPLY* message that contains information about the requested statistics. The proposed algorithm tries to identify a harmonic pattern from the correlation coefficients. The recorded statistical data is used to calculate the correlation coefficients 'r' and subsequently the *trust_index* of VMs.

$$\textit{trust\_index} = w_1 * r_1 + w_2 * r_2 + ...w_N * r_N \qquad (1)$$

For,

$$\sum_{i=1}^{N} w_i = 1$$

and 'N' is the number of past correlation coefficients taken into account and $0 < i \leq N$. Where, 'n' is the number of peer machine instances, 'w_i' is the weight attributed for i^{th} correlation coefficient. The Trust index algorithm periodically updates the trust index of each VM by matching the similarity of the recorded statistical information with respect to the detected harmonic pattern.

To have better adaptability to the dynamically changing traffic patterns within the data center the past $'N'$ correlation coefficients calculated for that VM alone are taken into account. A weighted average of the current and historical correlation coefficients contribute to updating the trust index of a VM as shown in equation (1). If a substantial dissimilarity is found in the traffic pattern, then its *trust_index* plunges accordingly. *trust_index* is a numerical value ranging between -1 to +1. A trust index value close to +1 indicates no suspicious activity by the corresponding VM, while a value approaching -1 suggests a discrepancy between the VM's reported probability of outgoing traffic to its collaborative VMs and the actual traffic observed by the controller. A value near 0 implies abnormal changes in the VM's outgoing traffic pattern.

If the *trust_index* of a particular VM in a VM group drops below an acceptable level, it will be blacklisted and restricted from direct access to its shared network unless it regains an acceptable *trust_index* value. All blacklisted VMs will undergo deep packet inspection to identify any potential threats. The *HostLocationProvider* service of the controller helps to maintain a mapping of host IP addresses, MAC addresses, VLAN IDs, and the ports on which they are connected to the network. The *trust_index* is updated by analyzing the network traffic at edge switches and their respective ports using the *HostLocationProvider* service.

A VM is blocked if it is found malicious, and can be retained only after the approval of the VM group administrator. This can be done by altering the probability of an outgoing packet to its respective peer machines. Once a VM is blocked, a report is sent to the VM group administrator for initiating necessary action.

The flow diagram in Figure 2 gives a brief idea of the proposed security mechanism. Traffic analyzing tool *ntopng* is used to capture and analyze flows for detecting malicious activities by VMs. *ntopng* is the next generation version of *ntop* that works based on $libpcap$ to monitor application-specific network traffic. nDPI library is used along with the *ntopng* traffic analyzer tool. Application-specific worm patterns are recorded and registered with the existing nDPI library for more accurate results [52]. Port classification is used to determine hosts connected over a specific port. The proposed model, due to its nature of simplicity may not prevent attacks, but can effectively mitigate them at a minimum computational cost.

VI. Testing and Results

The investigation into the operational flow of the OpenDayLight controller has provided a comprehensive understanding of the potential attack surfaces within the SDN stack. This critical analysis has enabled us to identify and assess vulnerabilities that could be exploited by malicious actors. To delve deeper into these vulnerabilities, a series of rigorous tests were meticulously carried out on the identified attack surfaces. These tests were designed to simulate various attack scenarios and evaluate the system's response under different conditions. In the pursuit of a thorough evaluation, our study focused on an SDN-enabled Openstack cloud platform, a representative environment for real-world SDN deployments. The exhaustive nature of our testing regimen is illustrated in Table I, where an itemized compilation of the conducted tests is presented. The table outlines the specific tests executed and also provides a comprehensive overview of the obtained results. These insights into our study strengthen its reliability and relevance, as the combined analysis of attack surfaces and rigorous testing provides a more holistic perspective on the security posture of SDN systems.

Figure 3, depicts the basic network topology of the VMs spawned in the Openstack cloud platform. VM01 and VM02 are connected to shared network1, and similarly, VM03 and VM04 are connected to shared network2. In addition, both the shared networks are connected through a router. The outgoing traffic from VM01 is captured and analyzed for suspicious activity. In Figure 4, VM01 is moved to the restricted network upon detecting suspicious activity. Results obtained on implementing the proposed model are presented in Table II. The percentage of malicious activities or attacks successfully detected by the proposed method is presented

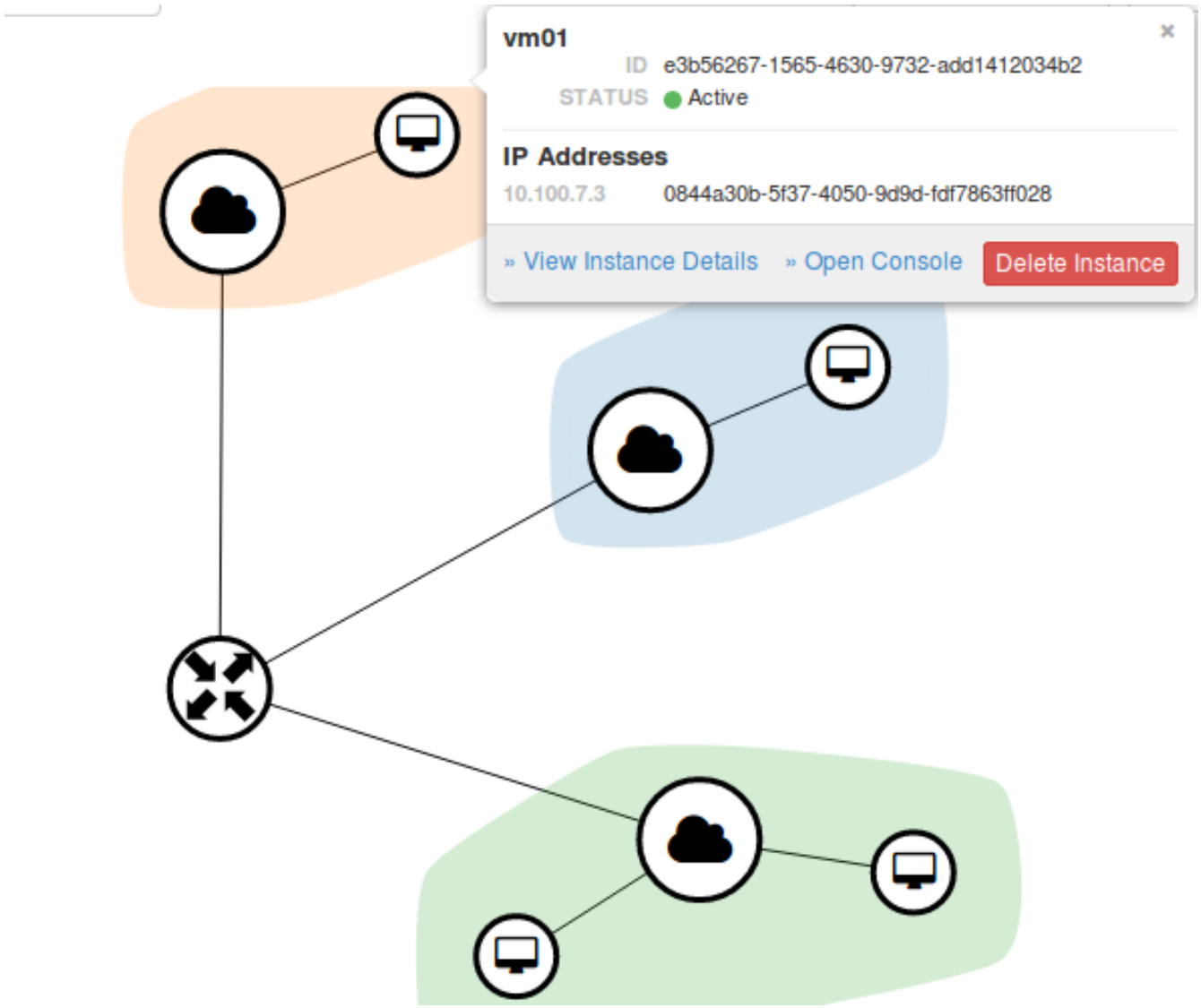

Figure 4: vm01 shifted to a restricted network.

under the Detection Rate column. Default driver *libvirt* in Openstack is used for live migration [27].

SDNs offer flexibility for VM migration by virtualizing the network hardware. As a proof of concept, the proposed technique is validated with real traces that are captured during the experimental evaluation. The resultant *trust_index* is presented in the form of a graph in Figure 6. The average round trip time before and after introducing the security application in Openstack environment are 0.523 ms and 0.581 ms respectively. The average round trip time shows that the additional delay caused due to the proposed security application is negligible. In addition, the ability of the proposed model to withstand and adapt to various attack scenarios, including novel or sophisticated attacks is evaluated and presented in Table III.

A. Further analysis of the proposed technique using the KDD dataset:

We applied the proposed technique to the standard KDD dataset to assess its real-time performance. In Figure 5, the test results and scores of logistic regression, Fourier transform, Periodogram analysis, Autocorrelation analysis, and harmonic regression on the KDD dataset are presented. The results indicate that the proposed technique performed adequately on the KDD dataset, demonstrating its potential effectiveness in real-time scenarios. Furthermore, in Figure 7, a visual representation showcases a segment of the predicted and actual outcomes obtained by employing the logistic regression technique. This visualization not only provides insight into the technique's accuracy but also offers a clear understanding of its ability to differentiate between normal and anomalous traffic patterns. Overall, these findings underscore the viability of the proposed technique as a valuable asset in network security and anomaly detection applications.

VII. Conclusion

From the above results, it is quite evident that insider attacks and orchestration failures can be effectively detected and mitigated with the help of the proposed technique at a minimum computational cost. In particular, the malfunctioning of distributed applications in a cloud platform can be readily reported. Moreover, machines responsible for DoS and DDoS attacks can be easily identified and blocked. Since the device's flow entries themselves are used to filter traffic,

Table III: Proposed Security Model vs Sophisticated Attack Scenarios.

Attack Scenario	**Description**	**Can the Proposed Security Model Detect the Attack?**
Advanced Persistent Threats	Sophisticated and long-term in nature, these attacks involve unauthorized network access by an attacker who evades detection for an extended period while targeting specific information or valuable assets.	No
Zero-Day Exploits	By leveraging vulnerabilities in software or systems that are either unknown to the vendor or have not yet been patched, attackers can clandestinely infiltrate a network without detection.	No
Insider Threats	These attacks or data breaches occur when authorized individuals with privileged access to a network, either intentionally or unintentionally, misuse their privileges, compromising the security of the network.	Yes
Botnets	These are networks of compromised devices that are under the control of a centralized entity. They are frequently utilized to carry out malicious activities, including DDoS attacks, spamming, and the distribution of malware.	Yes
Advanced Malware	This refers to sophisticated and covert malicious software intentionally created to evade conventional security measures. It encompasses polymorphic malware, rootkits, and advanced persistent malware, among others.	Yes
Social Engineering Attacks	This involves manipulating human psychology with the intention of deceiving individuals into revealing sensitive information or engaging in actions that jeopardize network security. Examples include phishing, spear phishing, and impersonation techniques.	Yes
Cryptojacking	This refers to the unauthorized exploitation of a victim's computing resources to mine cryptocurrencies without their knowledge or consent. This activity is typically carried out by injecting malicious code into websites or applications.	Yes
DNS Tunneling	This involves the exploitation of DNS (Domain Name System) protocols to circumvent network security measures and establish unauthorized communication channels or extract data.	Yes
SQL Injection	By exploiting vulnerabilities in web applications, attackers can inject malicious SQL statements into the underlying database, potentially enabling unauthorized access or manipulation of data.	Partially Yes
Cross-Site Scripting	By injecting malicious scripts into web pages accessed by other users, it often results in session hijacking, data theft, or defacement.	Partially Yes
Wi-Fi Eavesdropping	This involves intercepting wireless network traffic with the intention of capturing sensitive information transmitted over unsecured or inadequately secured Wi-Fi networks.	No
Phishing	This refers to the act of sending deceptive emails or messages that have the appearance of legitimacy, deceiving recipients into disclosing sensitive information such as login credentials or financial details.	No

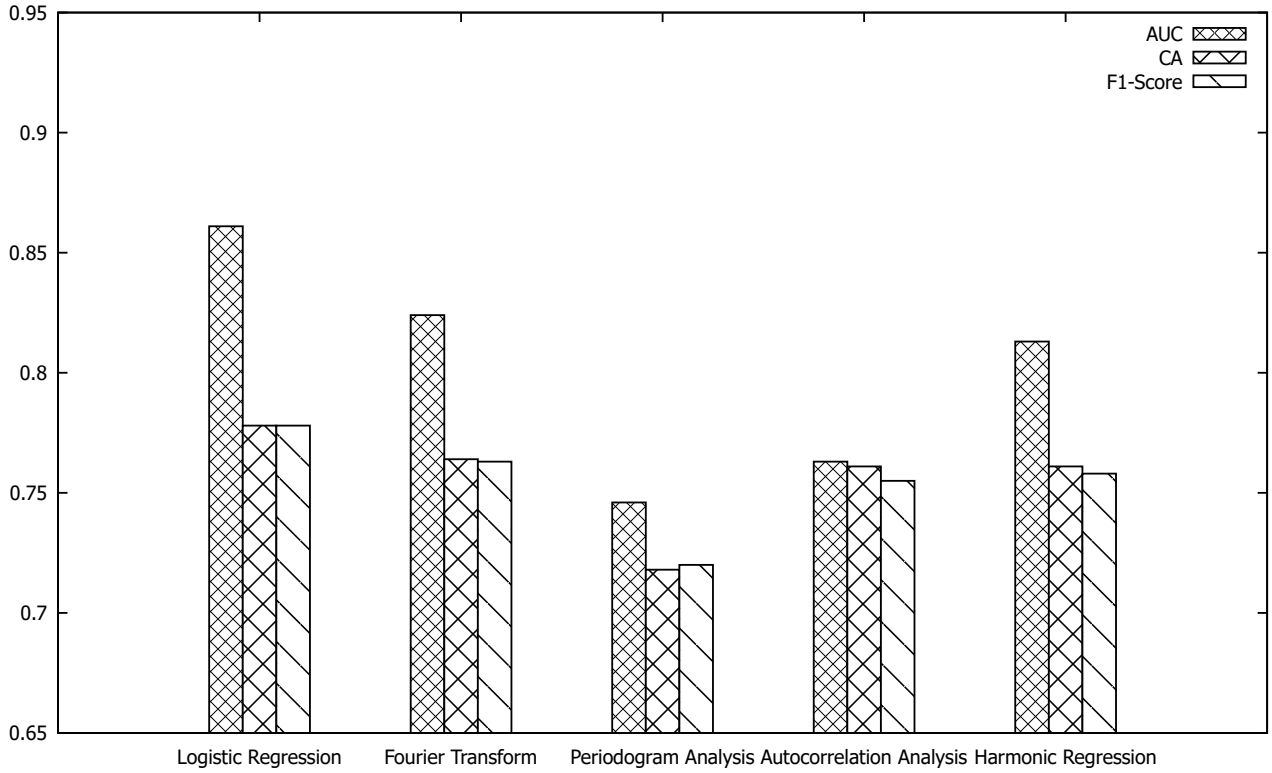

Figure 5: Scores of various harmonic pattern detection techniques on the KDD dataset.

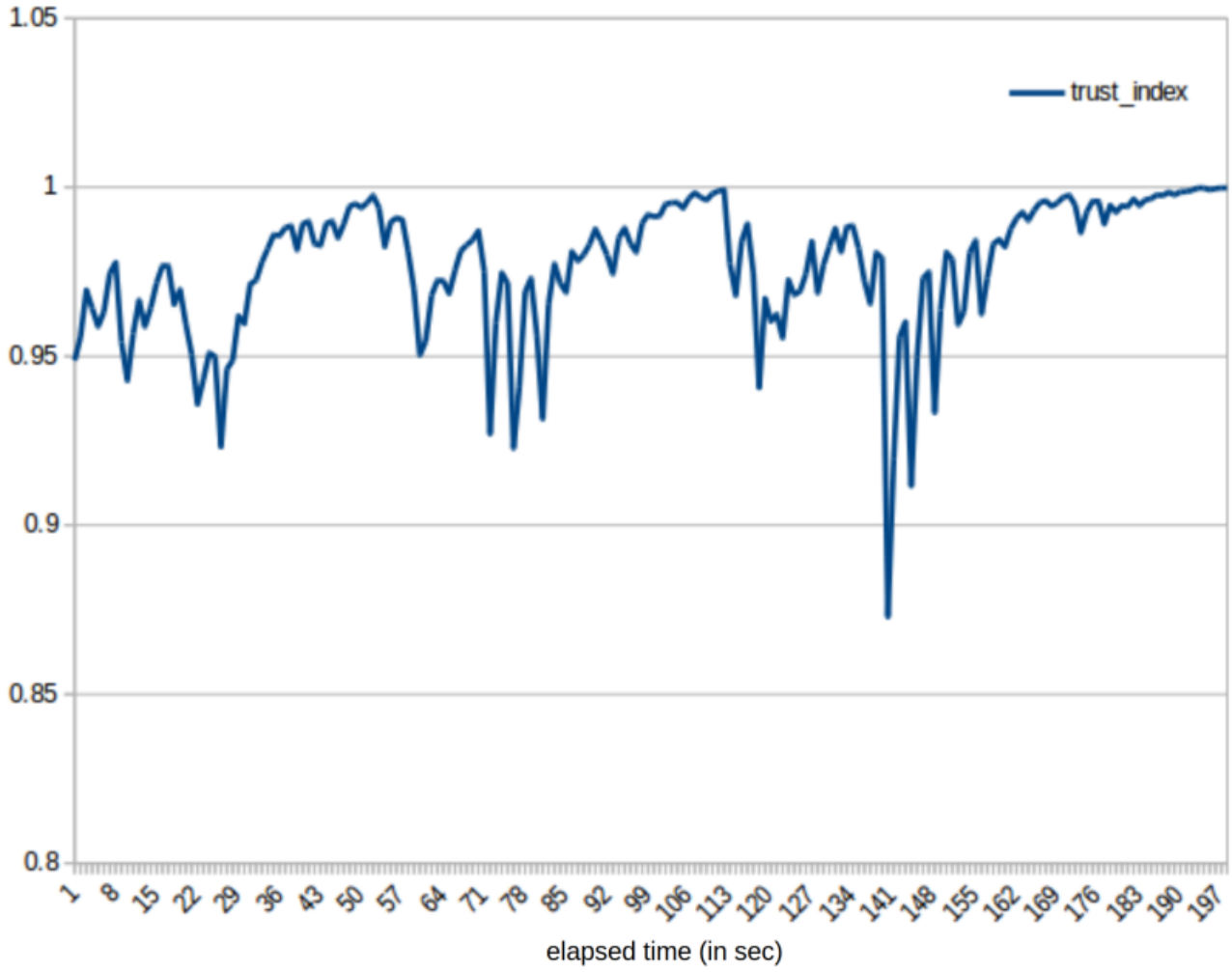

Figure 6: Trust index for the dataset.

	Tree	Logistic Regression	id	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent
1	normal	normal	1	0	tcp	ftp_data	SF	491	0	0	0	0
2	normal	normal	2	0	udp	other	SF	146	0	0	0	0
3	anom...	anomaly	3	0	tcp	private	S0	0	0	0	0	0
4	normal	normal	4	0	tcp	http	SF	232	8153	0	0	0
5	normal	normal	5	0	tcp	http	SF	199	420	0	0	0
6	anom...	anomaly	6	0	tcp	private	REJ	0	0	0	0	0
7	anom...	anomaly	7	0	tcp	private	S0	0	0	0	0	0
8	anom...	anomaly	8	0	tcp	private	S0	0	0	0	0	0
9	anom...	anomaly	9	0	tcp	remote_job	S0	0	0	0	0	0
10	anom...	anomaly	10	0	tcp	private	S0	0	0	0	0	0
11	anom...	anomaly	11	0	tcp	private	REJ	0	0	0	0	0
12	anom...	anomaly	12	0	tcp	private	S0	0	0	0	0	0
13	normal	normal	13	0	tcp	http	SF	287	2251	0	0	0
14	anom...	anomaly	14	0	tcp	ftp_data	SF	334	0	0	0	0
15	anom...	anomaly	15	0	tcp	name	S0	0	0	0	0	0

Figure 7: A sample snippet of Predictions.

restricted. Containers simplify the scaling of applications.

In future endeavors, we plan to expand our experimentation to a container-enabled OpenStack setup, exploring formats like ari, ami, bare, aki, ovf, ova, and docker. This extension will enable us to comprehensively examine security implications within different container environments, enhancing our understanding of potential vulnerabilities and mitigation strategies in the context of software-defined networks.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[2] J. Li, J. Yoo, and J. W. Hong, "Cpman: Adaptive control plane management for software-defined networks," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov 2015, pp. 121–127.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[4] A. Vahdat, D. Clark, and J. Rexford, "A purpose-built global network: Google's move to sdn," *Queue*, vol. 13, no. 8, pp. 100:100–100:125, Oct. 2015. [Online]. Available: http://doi.acm.org/10.1145/2838344.2856460

[5] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an sdn platform for cloud network services," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, February 2013.

[6] P. Patel, V. Tiwari, and M. K. Abhishek, "Sdn and nfv integration in openstack cloud to improve network services and security," in *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, May 2016, pp. 655–660.

[7] S. Wang, "Comparison of sdn openflow network simulator and emulators: Estinet vs. mininet," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, June 2014, pp. 1–6.

[8] J. Son and R. Buyya, "A taxonomy of software-defined networking (sdn)-enabled cloud computing," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 59:1–59:36, May 2018. [Online]. Available: http://doi.acm.org/10.1145/3190617

[9] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 121–126. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342466

[10] F. N.-A. Y. Tseng, Z. Zhang, "Controllersepa: A security-enhancing sdn controller plug-in for openflow applications," in *2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Dec 2016, pp. 268–273.

[11] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing network security through software defined networking (sdn)," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, Aug 2016, pp. 1–9.

[12] J. C. Mogul, A. AuYoung, S. Banerjee, L. Popa, J. Lee, J. Mudigonda, P. Sharma, and Y. Turner, "Corybantic: Towards the modular composition of sdn control programs," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII. New York, NY, USA: ACM, 2013, pp. 1:1–1:7. [Online]. Available: http://doi.acm.org/10.1145/2535771.2535795

[13] J. Seo, J. Nam, and S. Shin, "Towards a security-enhanced cloud platform," in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Dec 2018, pp. 229–230.

[14] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 413–424. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516684

[15] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," 01 2009.

[16] E. Caron and J. R. Cornabas, "Improving users' isolation in iaas: Virtual machine placement with security constraints," in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 64–71.

[17] X. Yuchi and S. Shetty, "Enabling security-aware virtual machine placement in iaas clouds," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, Oct 2015, pp. 1554–1559.

[18] D. Kapil, E. S. Pilli, and R. C. Joshi, "Live virtual machine migration techniques: Survey and research challenges," in *2013 3rd IEEE International Advance Computing Conference (IACC)*, Feb 2013, pp. 963–969.

[19] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "Cqncr: Optimal vm migration planning in cloud data centers," in *2014 IFIP Networking Conference*, June 2014, pp. 1–9.

[20] S. Woo, S. Lee, J. Kim, and S. Shin, "Re-checker: Towards secure restful service in software-defined networking," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2018, pp. 1–5.

[21] Seungwon Shin and Guofei Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *2012 20th IEEE International Conference on Network Protocols (ICNP)*, Oct 2012, pp. 1–6.

[22] A. Chowdhary, S. Pisharody, A. Alshamrani, and D. Huang, "Dynamic game based security framework in sdn-enabled cloud networking environments," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFVSec '17. New York, NY, USA: ACM, 2017, pp. 53–58. [Online]. Available: http://doi.acm.org/10.1145/3040992.3040998

[23] S. Yu, X. Gui, F. Tian, P. Yang, and J. Zhao, "A security-awareness virtual machine placement scheme in the cloud," in *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, Nov 2013, pp. 1078–1083.

[24] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 602–622, Firstquarter 2016.

[25] H. Wang, L. Xu, and G. Gu, "Floodguard: A dos attack prevention extension in software-defined networks," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015, pp. 239–250.

[26] S. Jeong, J. You, and J. W. Hong, "Design and implementation of virtual tap for sdn-based openstack networking," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, April 2019, pp. 233–241.

[27] T. He, A. N. Toosi, and R. Buyya, "Performance evaluation of live virtual machine migration in sdn-enabled cloud data centers," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 55 – 68, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074373151830474X

[28] M. i Isaac Thulo and J. H. P. Eloff, "Towards optimized security-aware (o-sec) vm placement algorithms," in *ICISSP*, 2017.

[29] Y. Han, J. Li, D. Hoang, J. Yoo, and J. W. Hong, "An intent-based network virtualization platform for sdn," in *2016 12th International Conference on Network and Service Management (CNSM)*, Oct 2016, pp. 353–358.

[30] Yoonseon Han, Jonghwan Hyun, and James Won-Ki Hong, "Graph abstraction based virtual network management framework for sdn," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2016, pp. 884–885.

[31] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," 2009.

[32] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: Make your virtual sdns programmable," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620741

[33] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, March 2013.

[34] K. Ko, D. Son, J. Hyun, J. Li, Y. Han, and J. W. Hong, "Dynamic failover for sdn-based virtual networks," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–5.

[35] S. Jeong, D. Lee, J. Hyun, J. Li, and J. W. Hong, "Application-aware traffic engineering in software-defined network," in *2017 19th Asia-*

Pacific Network Operations and Management Symposium (APNOMS), Sep. 2017, pp. 315–318.

[36] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 909–928, Second 2013.

[37] Yoonseon Han, Jian Li, Jae-Yoon Chung, Jae-Hyoung Yoo, and J. W. Hong, "Save: Energy-aware virtual data center embedding and traffic engineering using sdn," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–9.

[38] M. Karakus and A. Durresi, "Quality of service (qos) in software defined networking (sdn): A survey," *Journal of Network and Computer Applications*, vol. 80, pp. 200 – 218, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516303186

[39] T. Kim, Y. Choi, S. Han, J. Y. Chung, J. Hyun, J. Li, and J. W. Hong, "Monitoring and detecting abnormal behavior in mobile cloud infrastructure," in *2012 IEEE Network Operations and Management Symposium*, April 2012, pp. 1303–1310.

[40] V. Lopez, J. M. Gran, J. P. Fernandez-Palacios, D. Siracusa, F. Pederzolli, O. Gerstel, Y. Shikhmanter, J. Martensson, P. Skoldstrom, T. Szyrkowiec, M. Chamania, A. Autenrieth, I. Tomkos, and D. Klonidis, "The role of sdn in application centric ip and optical networks," in *2016 European Conference on Networks and Communications (EuCNC)*, June 2016, pp. 138–142.

[41] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards sdn-defined programmable byod (bring your own device) security," in *NDSS*, 2016.

[42] J. Matías, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "Flownac: Flow-based network access control," *2014 Third European Workshop on Software Defined Networks*, pp. 79–84, 2014.

[43] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. A. Porras, "Delta: A security assessment framework for software-defined networks," in *NDSS*, 2017.

[44] W. Huang, A. Ganjali, B. H. Kim, S. Oh, and D. Lie, "The state of public infrastructure-as-a-service cloud security," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 68:1–68:31, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2767181

[45] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling memcache at facebook," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 385–398. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/nishtala

[46] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of datacenter traffic: Measurements —& analysis," Jan. 2009, pp. 202–208.

[47] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, G. Boving, Seband Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 183–197, Aug. 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787508

[48] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 267–280. [Online]. Available: https://doi.org/10.1145/1879141.1879175

[49] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 123–137, Aug. 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787472

[50] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Article: Openstack: Toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, October 2012, full text available.

[51] X. Zhao, L. Yao, and G. Wu, "Esld: An efficient and secure link discovery scheme for software-defined networking," *International Journal of Communication Systems*, vol. 31, no. 10, p. e3552, 2018, e3552 dac.3552. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.3552

[52] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "ndpi: Open-source high-speed deep packet inspection," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, Aug 2014, pp. 617–622.

Bommareddy Lokesh received a Bachelor of Technology degree in information technology from the PVP Siddhartha Institute of Technology, Vijayawada, in 2011, and a Master of Technology degree in computer science and technology from the Andhra University College of Engineering in 2013. He was awarded a Ph.D. in Computer Science and engineering from the National Institute of Technology Puducherry, Karaikal in the year 2022.

He is currently working as an Assistant Professor in the School of Computer Science and Engineering at VIT-AP University, Amaravati, India. His research interests include software-defined networking, Internet of Things, network function virtualization, and 5G communications.

Narendran Rajagopalan received the Bachelor of Engineering degree in information science and engineering from the Vidyavikas Institute of Engineering and Technology, Mysore, in 2004, and the Master of Technology in networking and Internet engineering from the Sri Jayachamarajendra College of Engineering, Mysore, in 2007, and Ph.D. degree in computer science and engineering from the National Institute of Technology Trichy, Trichy, in 2013.

He is currently working as an Associate Professor in the Department of Computer Science and Engineering, National Institute of Technology Puducherry, Karaikal. His research interests include wireless sensor networks, software-defined networking, data center networks, 5G communications, and blockchain.