Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

OpenStackDP: a scalable network security framework for SDN-based OpenStack cloud infrastructure

Prabhakar Krishnan[1*], Kurunandan Jain[1], Amjad Aldweesh[2], P. Prabu[3] and Rajkumar Buyya[4]

Abstract

Network Intrusion Detection Systems (NIDS) and firewalls are the de facto solutions in the modern cloud to detect cyberattacks and minimize potential hazards for tenant networks. Most of the existing firewalls, perimeter security, and middlebox solutions are built on static rules/signatures or simple rule matching, making them inflexible, susceptible to bugs, and difficult to introduce new services. This paper aims to improve network management in OpenStack Clouds by taking advantage of the combination of software-defined networking (SDN), Network Function Virtualization (NFV), and machine learning/artificial intelligence (ML/AI) and for making networks more predictable, reliable, and secure. Artificial intelligence is being used to monitor the behavior of the virtual machines and applications running in the OpenStack SDN cloud so that when any issues or degradations are noticed, the decision can be quickly made on how to handle that issue, being able to analyze data in motion, starting at the edge. The *OpenStackDP* framework comprises lightweight monitoring, anomaly-detecting intelligent sensors embedded in the data plane, a threat analytics engine based on ML/AI algorithms running inside switch hardware/network co-processor, and defensive actions deployed as virtual network functions (VNFs). This network data plane-based architecture makes high-speed threat detection and rapid response possible and enables a much higher degree of security. We have built the framework with advanced streaming analytics technologies, algorithms, and machine learning to draw knowledge from this data that is in motion before the malicious traffic goes to the tenant compute nodes or long-term data store. Cloud providers and users will benefit from improved Quality-of-Services (QoS) and faster recovery from cyber-attacks and compromised switches. The multi-phase collaborative anomaly detection scheme demonstrates an accuracy of 99.81%, average latencies of 0.27 ms, and response speed within 9 s. The simulations and analysis show that the OpenStackDP network analytics framework substantially secures and outperforms prior SDN-based OpenStack solutions for Cloud architectures.

Keywords SDN, NFV, OpenStack networking, Cloud security, Intrusion detection, Machine learning, Analytics

Introduction

The rapid and broad adoption of Cloud computing services [1] makes this technology infrastructure a target for attacks. Also, in the last few years, dynamic and complex traffic has become a problem for enterprise networks because of the growing number of cloud-based and app-based services traversing zero-trust public networks. The community-driven cloud computing eco-systems such as *OpenStack*, *Eucalyptus*, and *Open Nebula* offer various services, frameworks, and comprehensive

*Correspondence:
Prabhakar Krishnan
kprabhakar@am.amrita.edu
[1] Center for Cybersecurity Systems and Networks, Amrita Vishwa Vidyapeetham, Amritapuri-Campus, Kollam, Kerala, India
[2] College of Computing and Information Technology, Shaqra University, Riyadh 11911, Saudi Arabia
[3] Department of Computer Science, Christ University, Bengaluru, Karnataka, India
[4] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

interfaces. However, most of these solutions lack security and privacy guarantee for the tenants and the hosted applications and users. The recent research report [2] from Verizon concluded that 34% of attacks involved *insider* attacks that exploit cloud networking vulnerabilities. Cloud infrastructure service providers employ security mechanisms such as Intrusion Prevention and Detection Systems (IPS/IDS), perimeter security, threat monitoring systems, and firewall gateways and run anti-malware software/agents on end-point machines or devices or tenant VMs. In the context of Cloud networking, the recent SDN/NFV [3] paradigms are emerging to redefine network architectures and provide new opportunities to defend against cyberattacks. Security and Privacy are essential when dealing with cloud service providers and cyber security. Figure 1 shows how traditional cloud models have evolved into SDN models. SDN separates network intelligence from packet forwarding/routing and offers high-performance programmable networking. The interactions between the control and data planes happen via *OpenFlow* [4] protocol. Traditional network traffic management focuses on device status and tracking the statistic counter values. It can only measure network throughput and capacity, not the full-service carrying scenario. Through end-to-end traffic behavioral modeling and AI functionality, network traffic analysis may construct a tightly coupled association between network operational performance and security, providing comprehensive technical support for optimizing network capacity and early warning of any security threats. The network, user, and application proportions give a risk score that appropriately reflects the current network, user, and application operating conditions. The intrinsic relationship between the data is then merged to produce multi-dimensional correlations that help detect network problems and malfunctions. To deliver stream matching rules in the forwarding path of packets, this paper uses software-based network traffic analytics, monitoring, and security mechanisms, integration of Open Virtual Network (OVN) SDN technologies [5], and an in-line streaming AI Analyzer pipeline to conduct an in-depth analysis of the packets. By performing Intrusion Detection and Prevention (IDS/IPS) system functions as embedded logic (in-line) in the critical path of OpenStack networking - OvS switch/bridge (as compared to VM server/out-of-path model), we shorten the time for dynamically deploying IDS in the network pathways. Using advanced SDN data plane switching (custom OvS bridge) the complete traffic monitoring and analysis are accomplished without any redundant traffic detours across the tenant network.

Modern virtualized data centers have recently started adopting SDN [6] and other virtualized network platforms for complete network orchestration and protection services. The legacy approaches utilized the routers' features and switches to mirror the traffic from one/more ports to a designed port on the same fabric, a technique [7] called *Switched Port Analyzer (SPAN)* filter. Rules and policies can be configured to copy all/specific packets matching the criteria. The system connected to the SPAN port on that switch will receive the packets, scan for any malware or attacks, and enable the response/mitigation action in that network. In a virtualized cloud network, the SPAN function is implemented in the software switch Open vSwitch [8] that mirrors the selected packets. There are several variations between standard computer networks and SDN. The open-source community developed the OpenStack [9] cloud platform, which is also deployed in production data centers. OpenStack consists of dozens of independent parts called the OpenStack services. Internally, each OpenStack service comprises several processes. All services

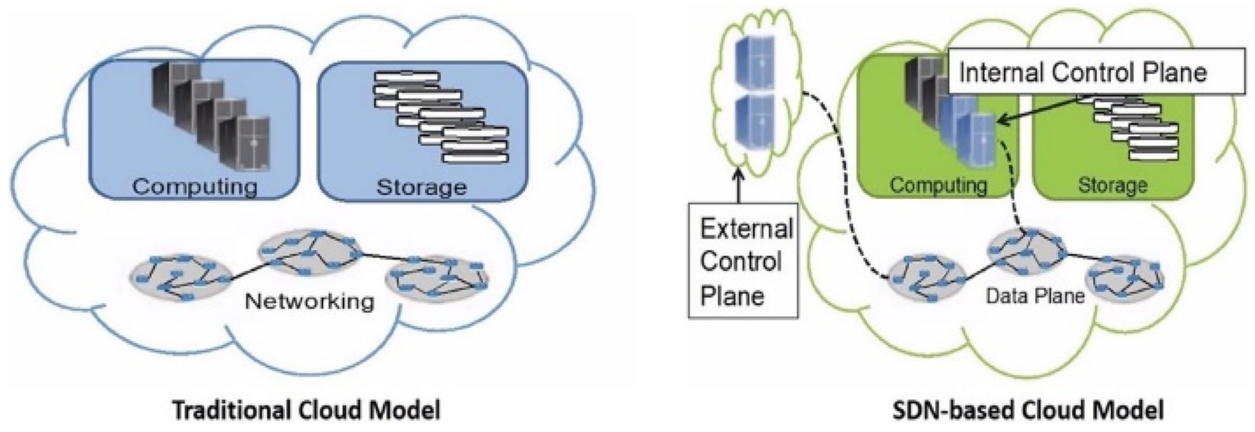


Fig. 1 Traditional vs. SDN View

have at least one API process listening, preprocessing, and passing API requests on to other parts of the service. The main process of the OpenStack Networking service is called neutron-server [10], a Python daemon exposing Neutron API and passing tenant requests to a suite of plug-ins for additional processing. Neutron allows tenants to create advanced virtual network topologies that include services like firewalls, load balancers, virtual private networks, etc. It is one of the most complex components among OpenStack projects since it is built around the core networking concepts. Cinder is the OpenStack Block Storage service [11]. Cinder allows users to create and delete block devices and manage the attachment of block devices to VMs.

The major threats identified in the Open Stack environment include the protection of the data during the transmission of the data through the network API in the Neutron networking and virtualization security during the data replication in the Cinder Block Store. The leading cause for these security risks is the open-source nature and OpenStack widely adopting some security algorithms that include their limitations. Though many issues are being reported across the OpenStack platform comprising some specified components such as Nova, Neutron, Cinder Block store, and Horizon, the security issues mainly tend to occur in the functioning of the Neutron networking component. The OpenStack plugin *Tap-as-a-Service (TaaS)* [12] supports mirroring for the multi-hypervisor cloud environment. The future cloud computing environment will be SDNFV-enabled [5], as indicated in Fig. 2. Cloud infrastructure vendors usually provide reactive solutions to customers' unpredictable traffic by launching pre-made VNF images. While reactive systems can help in some cases, they are not an optimal strategy. So, our research presents a proactive scalable programmable approach for elastically scaling the service chain, monitoring threats, and dynamically deploying defense mechanisms in SDNFV-enabled OpenStack clouds. By separating access restrictions from cloud capabilities at the network protocol level, cloud service providers and tenant managers gain a global view of their security perimeters: the entire data center domain or individual tenants. The unified API and REST interface can also modify security settings, especially policy setups. We advanced from our prior *CloudSDN* [13] proposal and utilized the latest OpenStack *Liberty* [14] framework for benchmarking.

The main contributions of this paper are:

- A comprehensive vulnerability analysis of the OpenStack networking environment.
- Software-based network traffic analytics, monitoring, and security mechanisms in the switching/data plane layer. Native integration of *Open Virtual Network (OVN)* SDN technologies [5] in the OpenStack Cloud platform [15] which consists of monitoring, analytics,

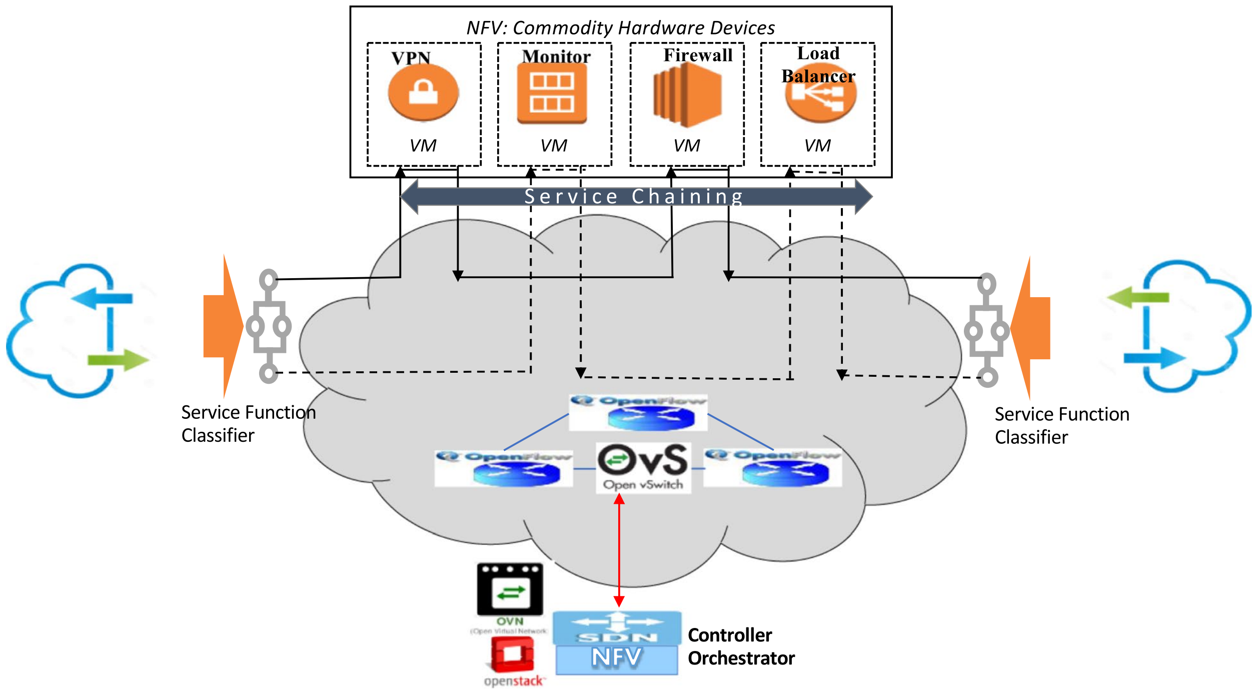


Fig. 2 The emerging SDNFV-enabled cloud

and anomaly detection functions embedded in the switches.

- Threat Score: Applications, the system plane, the data plane, and the control plane all play a role in the comprehensive monitoring and scanning of networking devices.
- Multi-dimensional correlation analysis: A wide range of information, including network device statistic counters, operational data, transmission data, users, tenants, UDP, TCP/IP connections/sessions information, application flows, and metadata of services and policy configuration.
- OpenStack Networking Architecture with redesigned Neutron system consisting of modules: virtualized/software vMon (Monitor), vTAP (Test Access Port)/SPAN (Switched Port Analyzer) for monitoring endpoint hosts or VMs in the tenant networks and authentication functions in the management network.
- A minimally invasive lightweight IDS and security orchestration framework OpenStackDP for OpenStack enables the implementation of various security and access control functions as loadable modules in the data plane.

Through experiments, this paper demonstrates the efficacy of OpenStackDP by putting it through its phases for distinct types of attacks (covering about 90% of all known network attacks). In addition to these situations, OpenStackDP protection capabilities extend far beyond what is covered here. These examples are provided to illustrate how the system performs under a variety of traffic scenarios. We compared the related works and provided a security perspective for SDN concepts for attack detection in the context of the OpenStack cloud ecosystem and well-tested mitigation techniques to tackle them. A comprehensive Security Analysis and Threat Model Mitigation and Validity of our solution are done with the STRIDE method [16]. The results and security analysis based on the popular threat model from the evaluation show that the SDN-based OpenStack system can improve the performance of Cloud infrastructures in terms of threat detection and mitigation compared to traditional Linux networking mechanisms. Our studies were conducted in an actual cloud computing infrastructure utilizing the OpenStack platform and an SDN environment using the OpenDayLight Controller. Still, they could easily be adapted to other controllers. We compared it with DragonFlow [17], a distributed SDN controller for OpenStack Neutron supporting distributed Switching, Routing, DHCP, and more.

The rest of this paper is organized as follows. Background section presents the background discussion of relevant technologies. Related work section gives an overview of the related work. Proposed SDN-enabled architecture and Design and implementation sections describe the proposed solution. Performance evaluation section provides the results of the experiments, result discussion and limitations of the solution, with a sketch of future work. Conclusions section concludes the paper.

Background

With the rapid adoption of SDN architectures, modern data centers have embraced softwarized networking for creating Software Defined Clouds (SD-Cloud). OpenStack is an open-source platform and an eco-system for cloud services and provides an efficient and rich set of APIs to build and manage cloud platforms [15]. OpenStack embraces a modular stateless architecture of services.

OpenStack architecture and SDN integration

Recently the Cloud service providers and enterprise data centers have migrated the services from hardware appliances to softwarized NFV platforms with Virtualized Network Functions (VNFs). The SDN communities have been actively developing open-source controller software platforms such as *Open Network Operating System* (ONOS) and *OpenDayLight* (ODL) that offer promising solutions to virtual resource management and network orchestration problems. A Central DB stores all the configurations and key settings (such as users, credentials, and instances) and is accessible to authorized component services. An OpenStack deployment environment is structured into network domains (ND), as seen in Fig. 3. The public ND includes the external, public, and API networks. The guest ND only has one network. NDs are utilized for servicing activities and internal management communication.

OpenStack requires SDN controllers to connect with Neutron using REST API (Representational State Transfer Interface). Current solutions are Floodlight, OpenDayLight, and Ryu. Each solution supports different OpenFlow requirements and TLS implementations (Transport Layer Security). The OpenDayLight (ODL) platform [18] is the widespread implementation of SDN Controllers compliant with OpenFlow standards, developed and maintained by the community. The *neutron* is one of the core technologies of the OpenStack architecture, which enables the interconnection of all the nodes on the internal network and routes through the gateway to an external internet connection. *Networking-ODL* [19] is an ODL plugin for OpenStack. The ODL controller uses Netvirt, configuring the *Open vSwitch* (OVS) networking settings. The IP/MAC routing, security classes, and other network abstractions are all elements of this

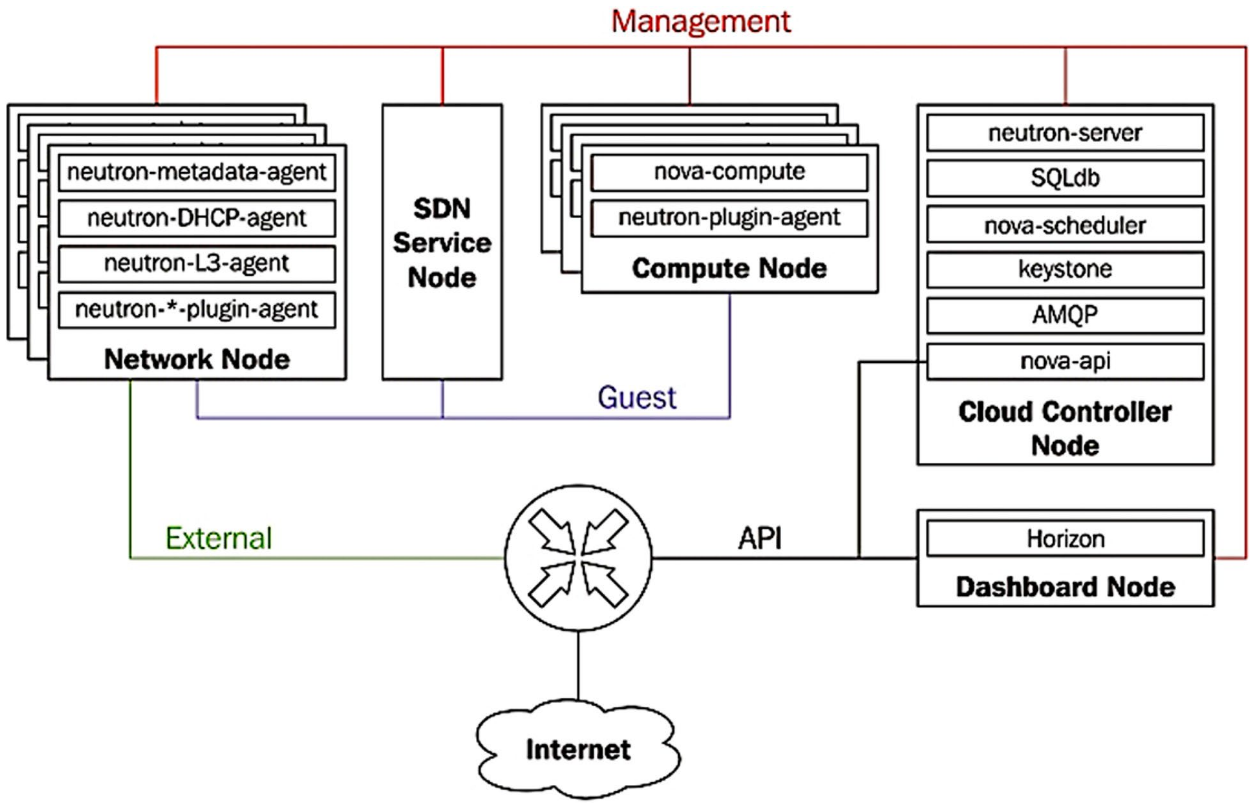


Fig. 3 OpenStack's Network Infrastructure

networking model. The ODL can be deployed in different modes, customizing the components to enable SDN to interoperate with OpenStack architecture. Some specific use-cases are i) Intent-driven data model; ii) OVSDB model that leverages NFV, OvS and includes southbound OpenFlow interface; iii) Virtual Tenant Network: ODL VTN is developed for multi-domain tenant virtualized networks and installed as a plugin to controller application. ODL supports RESTful API functions for Layer2 networking. The main package (driver module) *networking-odl* acts as a redirector/proxying/routing component interfacing between the SDN abstractions and OpenStack security/policy configurations. The *networking-odl* module includes functions to create new networks and establish security groups for tenants. DragonFlow [17] for the OpenStack platform doesn't aim to become a full-featured SDN controller, as it just enables DVR and virtual networking connectivity simply and easily that integrate almost seamlessly on top of OpenStack Neutron without any additional components or installations.

Security threats for an OpenStack cloud

OpenStack is, by nature, a large project with numerous smaller projects. As OpenStack gains momentum, new features are integrated and delivered continuously, bringing new vulnerabilities. Recent Glibc, OpenSSL flaws, and log4j logging library vulnerabilities (most used Java apps) highlight how a single component can expose a whole system. The attack surface is extensive with so many Open-Source new functionalities in OpenStack. Open-source components were found in 99% of codebases in the 2020 Open-Source Security and Risk Analysis Report [20] from Synopsys. Open-Source Software (OSS) benefits from the community approach but creates an attack surface. Malicious developers can introduce back doors into an OSS system, which can be exploited in various ways, including reusing previously used vulnerabilities and simply making mistakes. Attacks against software that relies on a vulnerable piece of OSS are possible because developers reuse open-source software (OSS). Patching open-source bugs become increasingly challenging as the flaws might spread throughout an organization's network. As OpenStack private clouds become increasingly popular in the development and operation (DevOps) among enterprises, so do the risk of incurring attacks. The most common vulnerabilities between 2011 and 2019 reported [21] were Distributed Denial of Service (DDoS) attacks and information-gathering and injection flaws (SQL) vulnerabilities. These flaws affect cloud users and may lead to further harmful attacks. We must first identify potential security concerns and attack routes to offer mitigation methods. With a particular emphasis on software-based networking architecture, it is typical to classify security threats into the following categories by using the standard STRIDE threat model [16]:

- ▪ Spoofing – Pretending to be other than one's real identity.
- ▪ Tampering – Modifying data without authorization.
- ▪ Repudiation – misrepresenting responsibility.
- ▪ Information disclosure – Providing information to someone not authorized to see it. "
- ▪ Denial of service – Absorbing the service resources.
- ▪ Escalation of privileges – Allowing to do critical operations above the allowed level without authorization.
- ▪ Cloud threats – Outsider attacks affecting service providers or tenants.
- ▪ Unauthorized VMs – Malicious VMs on endpoints, un-attested/not-licensed software run by rogue tenants.

OpenStack networking security

Physical switches interconnect virtualized servers in a cloud network, whereas an overlay (virtual) network connects the VMs. OpenStack Networking allows users to manage their networks. Users can be authorized to utilize specific project networking methods and objects by using a policy and configuration manager. This flexible management approach may impact network availability, security, and OpenStack infrastructure posture.

Neutron security

Utilizing cloud infrastructure services, neutron provides networking services and addresses for VMs (tenants). Neutron architecture is based on plugins, so it is essential to understand the required plugins and their usage for third-party solutions and disable any unnecessary plugins. Some potential solutions that can be used to resolve the risks:

- Isolated management network for OpenStack services.
- L2 isolation with VLAN segmentation: VLAN segmentation reduces packet-sniffing capabilities and decreases insider threats. Protocol-level segmentation separates protocols into specific LAN domains/segments. VLAN-enabled L2 bridge learns the association between MAC and vNIC (in compute nodes) and implements a virtual switching interface for the tenant network.
- L2 protection with Generic Routing Encapsulation (GRE): GRE tunnel connects two endpoints (a firewall and another appliance) in a point-to-point, logical link. The packets travel through the GRE tunnel (over a transit network such as the internet) through the cloud service which can enforce policies on the packets. However, GRE tunneling does not support NAT and has no QoS functionalities.
- Using the Security group in Neutron and disabling security groups in Nova (all calls are forwarded to Neutron).
- Securing at the Neutron API endpoints with SSL/TLS.
- Bridging an L2 firewall with iptables and ebtables: *iptables (manipulate the Netfilter Linux modules* at IP level) along with *ebtables* (Filter at Ethernet Frame level) rules to prevent MAC spoofing and ARP spoofing attacks.
- Using rate-limiting network quotas to mitigate DoS attacks.

Security group

The security group capability is very versatile, and it forwards security group calls to the OpenStack Networking API (If not, both services apply competing security policies). A security group's regulations are included within the group definition. Administrators and projects can specify the type of traffic (in/out) that can travel via a virtual interface port using security groups and rules. OpenStack Networking assigns a virtual interface port to a security group. Refer to the Networking Security Group Behavior documentation [22] for further information on port security groups. Security groups encapsulate all components that govern the inbound and outbound traffic to tenants. Using the OpenStack group, the compute instance's firewall rules are enhanced. A single security group is needed to control traffic to multiple compute instances. Various methods can be used to create network security rules and access control policies.

Virtual switch security

Virtual switches enable flexible and "software-defined" interconnection of virtual machines in SDN-based cloud systems. Virtual switches (on servers) provide communication and isolation between virtual machines. A virtual switch is not only more vulnerable to attacks than a traditional switch, but it also has a more significant impact. For OpenStack, Xen, Pica8, and other software systems, Open vSwitch is the default virtual switch. The OVSDB protocol allows the switch controller to handle the OVS database. Figure 4 depicts the security implications of current virtual switch designs. We did a qualitative study of attacker models for virtual switches. We posit that current implementations of virtual switches in Cloud infrastructures are vulnerable. Virtual switches are widely used in data centers and receive unfiltered network packets from virtual machines.

- Virtual switches run software processes such as "controller" on virtualized servers with higher (root) access. These features make data plane attacks risky. In addition to existing security assumptions (OvS runs as root), virtualization (co-location with other critical cloud services), logically centralized control plane (bi-directional channel to the controller), and non-standard routing protocols (e.g., MPLS), the attack can be destructive.
- A software flaw in a virtual switch's packet processing logic can jeopardize both the virtual switch and the operating systems. In the virtualization layer, an attacker can use co-location, centralized control, and complex data processing to launch an assault, scan the network and exfiltrate sensitive data (e.g., cryptographic key, password credentials).
- Attackers can exploit the controller's global routing/flow tables to modify the flow rules, potentially breaking network policies, divert traffic, pivot, or gain lateral entry to other internal systems in the management network, such as the identity service (Keystone) or the VM image files (to install backdoor/hooks).

New opportunities in SDN-managed clouds

Research on virtual switches and their software implementations is carried out extensively, and the popular technologies that came out are Open vSwitch, IP forwarding, iptables, Linux Bridge and OVS-DPDK. The Cloud-based architectures pose some limitations to the network performance due to the suboptimal design of the OpenStack infrastructure. In enterprise cloud infrastructures, the software-agent-based neutron component will become the choking point and not scale to ever-increasing isolation requirements between the tenants of the clouds. The OpenStack Neutron system faces severe limitations in networking and routing functions. The Linux bridge is outdated and is one of the critical architectural flaws in the OpenStack-based Cloud platform. This is where the SDN model can bring in the necessarily centralized orchestration and routing policy decisions. The corresponding routing functions (L2/3) can be distributed across the data plane switches. The southbound OpenFlow protocol can manage the tenant network's routing policies, flow tables, and policies configured through the OpenStack RESTful API. We studied the interplay and integration of the SDN-based Open vSwitch bridge in the place of a Linux bridge. We managed the enforcement of security group and access-control policies through OpenFlow flow rules. This practice improves the efficiency of the interconnect/communication due to programmability and scales well as the switching control configured into the OvS switches of compute nodes [8]. The stateful firewalling function can be accomplished by leveraging the OvS OpenFlow pipeline with group tables on the *br-int* bridge. Integrating diverse components in OpenStack with virtual switches could lead to bottlenecks and reduce scalability and overall performance [23]. The research article [24] defined an improved and scalable software firewall design for OpenStack using the SDN/NFV. Most of the studies that suggested providing end-to-end security and limiting threats used a controller-based approach; consequently, any threats that arise in the network can only be predicted with the controller's participation. This leads to overhead due to additional complexity in flow processing, control plane saturation, and flow table vulnerabilities. Although

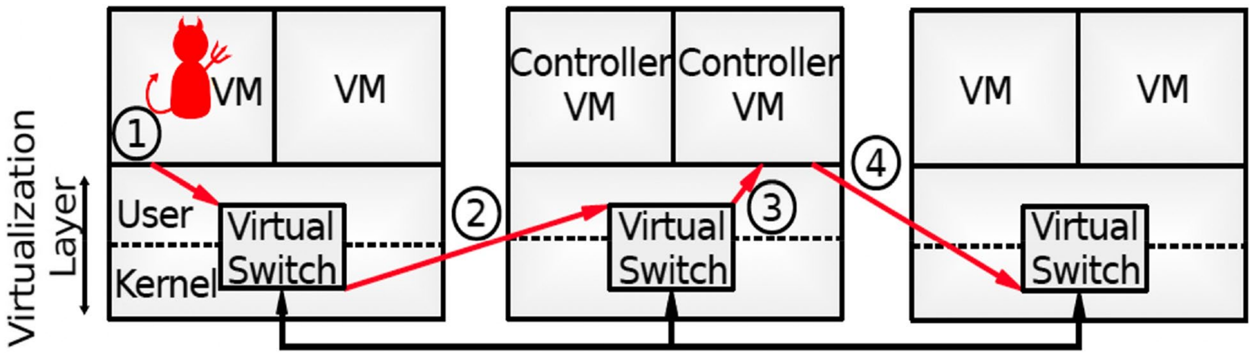


Fig. 4 Overview of Security Implications

using SDN is a feasible way to incorporate a switch-centric structure, it is a great challenge to adapt SDN components to data center networks (DCNs). Securing cloud infrastructures comprises five key aspects- Security Standards, Network, Access Control, Cloud infrastructure, and Data. Moving from a legacy network to an SDN paradigm for Cloud infrastructures has unintended consequences, such as sub-optimal flow-rule management and multi-tenant provisioning in clouds. To this end, we propose natively integrating the SDN and OpenStack platform with a scalable data plane and centralized security orchestration on the controller.

Related work

This section reviews critical studies in the relevant research proposals of software-defined OpenStack architecture for cloud infrastructure. The goal is to draw inspiration from these efforts and identify those gaps that motivated this research work. OpenStack security research [25] states that the bridges between the domains and tenant networks are vulnerable. They identify vulnerabilities in cloud providers' web-based interfaces and conclude in their research that the OpenStack management platform in default configuration is more vulnerable to internal attacks than external threats. Software countermeasures and SDN-based network security [26] may be used to prevent specific attacks against virtual switches in the OpenStack infrastructure. The hypervisor can be detached from VMs and establish a direct link to the I/O pipeline (e.g., network interface hardware adapters). One existing approach is remote attestation for OvS flow tables, as prototyped by Jacquin et al. [27]. With their prototype, the authors perform remote attestation of an OvS flow table in 953 ms. While we did not implement remote attestation, we argue that their work, in principle, demonstrates the feasibility of this technique, also beyond the remote attestation of flow tables. However, we need a way to reduce the overhead, e.g., by using a redundant set of OVS switches, since a one-second network outage is not practically feasible. Isolation of VMs can be done by leveraging an Open-Flow centric IDS architecture for isolating misbehaving nodes (VMs) in the cloud. Hence in the context of SDN, we can leverage a presumed security weakness (centralized operation) into an opportunity to defend infrastructure from misbehaving nodes. Depending on the threat model, multiple IDS/firewalls may be chained together, or virtual appliances may be used for such firewalling.

For network monitoring, most studies proposed using hardware TAP/SPAN port mirroring in switches. But this method requires a set of flow rules to switch OpenFlow tables to sample/capture the flows matching the criteria, thus increasing the usage of critical resources in commodity switches. NetAlytics [28] minimized the burden of real-time surveillance in data centers by positioning the software instantiations of *controller* and *aggregator* in the end-point services, which deliver optimal performance due to load-balancing and bandwidth management. The Packet-I/O transfer across the networking stack from physical NIC is accelerated by exploiting DPDK technology. The authors of vTAP [29] used the DPDK for implementing the high-speed Datapath in the Open vSwitch (OvS bridge) in OpenStack. The policies are installed in the virtual switches as flow rules through a controller application. They demonstrated monitoring applications and use cases for SDN-managed OpenStack infrastructure. The authors of [30] implemented an NFV platform to solve service chaining (SFC) by integrating an SDN controller with OpenStack cloud infrastructure. Since this is not a native solution, it required custom modules to be installed (intrusive and not generic) across many components in the environment. The ONOS team developed SONA [31], which implements Neutron to integrate/bridge between OpenStack infrastructure with OpenDayLight natively. The project includes (i) "*Networking-ONOS*," a modular system with Neutron APIs, and the cluster itself transparently manages the networking. (ii) "*Networking-ODL*" is a package with a driver and plugin for OpenDayLight. They experimented with SDN in their paper [32] Cloud-based IDS solution and contributed new OpenFlow features such as *flow-match/send-to-action* and *flow-match/send-to-to-controller*. OpenStack/SDN integration with Floodlight and RYU controller was conducted by Forester et al. [33], and they utilized Control-plane software to protect the data. At the same time, the administration plane receives an API to enforce the threat detection policies on the firewall. They only present a conceptual design without data and analysis findings.

Extensive research and solution for DDoS attacks, Intrusion detection, and defense for securing OpenStack Clouds are presented in [34, 35]. The *OpenStack-Emu* [36] created a platform combining OpenDayLight SDN Controller and a large-scale network emulator called Common *Open Research Emulator (CORE)*. The OpenStack nodes are interconnected through a TAP interface and the programmable Open vSwitch (OvS) software data plane. This paper [37] presents a simulation environment and tool called *CloudSimSDN* to experiment with SDN-based cloud use cases. In this research [38], OpenStack neutron components (OF-Agent, ML2, and Ryu) in SDN OpenDayLight Controller were tested for the ability to maintain network health and failure recovery features in the face of network disruptions. Another research [39] proposed an SDN-based firewall solution for Cloud security. Most research focused on public domain security, which shows potential customers the benefits of cloud computing. By exploiting the vulnerabilities in the SDN layers (Thimmaraju [21]), the attackers can penetrate the networking components and compromise OpenStack's cloud infrastructure. A Cloud Provider-side vulnerability may also compromise the entire infrastructure, and this exposure prompted a deeper look into infrastructure security flaws, focusing on the provider's procedures. Meridian [23] focuses on cloud data center services utilizing SDN technologies and provides inter-subnet tenant isolation. The OpenStack Security Modules (OSM) [40] project addressed the access-control function and has built a new service called *Patron* and an attachment module called *Access Endpoint Middleware* (AEM). This framework can replace existing OpenStack and other platform permission checks. *Patron*, like the previous efforts, tries to offer cloud access controls. Jin et al. [41] proposed formal ABAC and RBAC standard mechanisms and defined core concepts such as domains, projects, and roles for cross-domain authentication as an *Infrastructure-as-a-Service* (*IaaSad) model* for OpenStack. *IaaSad* provides fine-grained tenant access management. But this work only supports isolated tenants, not cross-tenant access control. Moreover, their model is coupled with ABAC, limiting cloud users' ability to create policies based on unique models. The current complex OpenStack platform must modularize its code by delivering access control, authentication, and firewall as services [42], exactly like other core cloud functionalities. Table 1 lists the prior studies that attempted to solve network security in OpenStack architecture. Unlike the previous studies, our work focuses on the dynamic setting and programmability of tenants' network policies, access control, and security management in cloud environments.

Our research attempted the native integration of SDN architecture within the OpenStack ecosystem. We focused on improving security while sustaining the performance of cloud computation in the network. OpenStackDP consolidated the network analytics, monitoring, and security functionalities to reduce invasiveness into one "huge hook" at the Networking gateway switch/data plane.

Proposed SDN-enabled architecture

Based on our previous works *VARMAN* [47] and *OpenPATH* [48], the data plane (*OpenStackDP*) is designed for the cloud. The overall architecture of the OpenStackDP framework (see Fig. 6) consists of the following major components: (i) Monitoring mechanism vMon operating close to the line rate; (ii) Light-weight Anomaly Detection mechanism vIDS; (iii) Heavy-weight attack classification and Malware analysis system NIDS (Monitor); (iv) Applications for Incident response and policy control (DTARS App). (v) Traffic monitoring and route manipulation modules (deployed as Virtual network functions VNF) within the data plane OvS switches [39].

Threat model

Threat modeling is a technique for optimizing security parameters by defining objectives, metrics, attacks, and countermeasures to prevent or reduce the consequences of system threats. Numerous approaches exist for analyzing the security of a system, including STRIDE [16]

Table 1 Studies on OpenStack network security

Study	Addressed Security Domain				Focused upon			
	Public	Guest	Management	Data	Control Plane	Data Plan	Client	Provider
Meridan et al. [23]	✗	✗	✗	✗	✗	✓	✗	✗
Jiang et al. [32]	✗	✗	✗	✓	✗	✓	✗	✗
Foresta et al. [33]	✗	✗	✓	✗	✓	✗	✗	✗
Xu et al. [34]	✗	✗	✓	✗	✓	✗	✗	✓
Abdulqadder et al. [43]	✓	✗	✗	✗	✓	✗	✓	✗
Jin et al. [41]	✗	✗	✓	✓	✓	✗	✓	✗
Luo et al. [40]	✗	✗	✓	✓	✓	✗	✗	✓
Thimmaraju et al. [21]	✓	✗	✗	✗	✓	✗	✓	✗
Li et al. [39]	✓	✗	✗	✗	✓	✗	✓	✗
Benjamin et al. [44]	✗	✗	✓	✓	✓	✗	✗	✓
Dinh et al. [45]	✗	✗	✗	✓	✓	✗	✗	✗
X. Du et al. [46]	✗	✗	✗	✓	✓	✗	✓	✗
OpenStackDP [This Work]	✓	✗	✓	✗	✓	✓	✓	✓

and PASTA [49]. Modeling techniques are used to generate an abstract representation of the system and, identify the goals and methods of the potential attackers, enumerate a list of possible dangers that could occur. There has been a slew of approaches to threat modeling created. Combined, they provide a complete picture of prospective risks. Some strategies focus on risk or privacy concerns, while others are broad-based. The STRIDE approach categorizes possible risks posed to a system even in the absence of the actual plan for testing. *STRIDE* is an acronym that represents six distinct threat categories: "*Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Privilege Elevation*." STRIDE is currently the most mature threat-modeling method, and it applies a broad set of known threats based on its name, which is a *mnemonic*. The mapping of security properties to the STRIDE threat matrix [50] is illustrated in Table 2.

We assume that the cloud infrastructure management system has implementation flaws and vulnerabilities, which malicious entities can potentially exploit. We trust cloud providers and administrators but think some cloud users and operators may be malicious. We believe that not all tenants trust each other. Although our auditing framework may catch violations of specified security properties due to either misconfiguration or exploits of vulnerabilities, our focus is on detecting specific intrusions attacks. SDN switches, as well as the trade-off between data and control planes, are questioned in this paper. Given the prevalence of virtual switches like OvS in cloud operating systems and that most cloud operating systems like OpenStack are used by IT, telecommunications, education and research, and financial institutions, our threat model is alarming. It is possible to exploit the vulnerabilities found in the OpenStack cloud operating system to harm critical services, such as Managed-compute resources (Hypervisors and Guest VMs), image management, block storage, network management, and identity management (of Hypervisor and Guest VMs). These services are all part of OpenStack, a cloud operating system. Because of the significance and critical location of virtual switches in SDN-based clouds and in general, we present an accurate and appropriate threat model for virtual switches in this study.

Contrary to prior work, we identify the virtual switch as a critical core component that must be protected against direct attacks, e.g., malformed packets. The attacker is looking for a cloud architecture that leverages virtual switches for network virtualization. To restrict the scope of this model, we're going to pretend our attacker has limited access to the public internet. The attacker cannot gain physical access to any of the cloud machines. An attacker can get into a cloud environment in two ways: by renting a single virtual machine or exploiting an existing cloud-based vulnerability, such as a web application vulnerability. We take it for granted that the cloud service provider adheres to industry standards for data security [22]. As a result, at least three separate networks (physical and virtual) for the management, tenants/guests, and outside traffic are to consider. In addition, we take it for granted that all cloud servers are running the same set of applications. We already discussed Fig. 5 in Background section, which illustrates the security risks associated with state-of-the-art virtual switch designs. We did a qualitative study of studies related to attacker models for virtual cloud switches.

Monitoring and anomaly detection components

Using multi-dimensional correlations formed by integrating the data's inherent relationships, network problems and malfunctions can be detected and addressed quickly. With the switch's active tap mechanism (In-Band telemetry), information is sent to the collector module in real-time and at high speed, making it possible to monitor tenant network performance and cloud activities in real-time. Anomaly detection and dynamic prediction of network measurements can be supported by time-series data decomposition and machine learning [51–53]. Analyzing data for anomalies in the past is the primary goal of anomaly analysis, while dynamic forecasting is used to predict future data trends. The historical data gathered by devices is used to develop and train a dynamic prediction AI model for anomaly detection in the traffic analysis using a dynamic baseline technique. Automatic learning

Table 2 Security property mapped into STRIDE model

Security property	Threat category	Interactors	Processes	Data store	Data flows
Authentication	Spoofing	X	X		
Integrity	Tampering		X	X	X
Non-repudiation	Repudiation	X	X		
Confidentiality	Information disclosure		X	X	X
Availability	Denial of Service		X	X	X
Authorization	Elevation of privilege		X		

X denotes threat category for a particular component

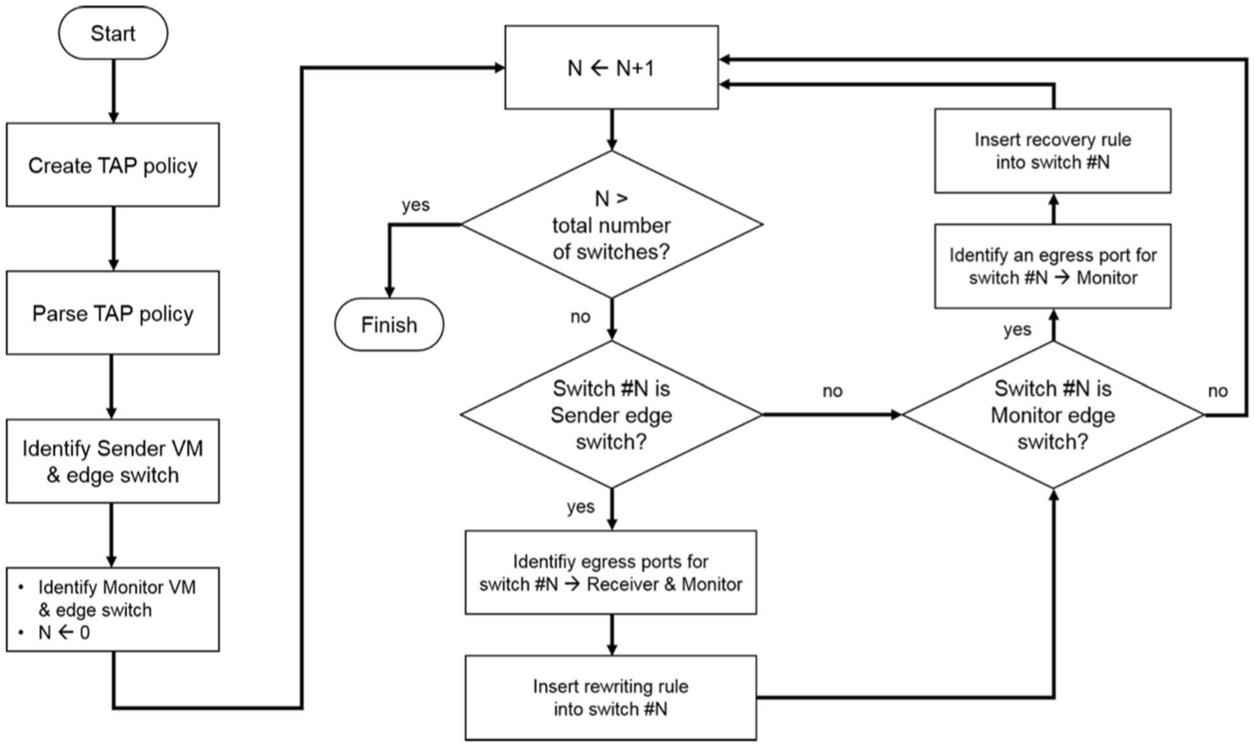

Start
Create TAP policy
Parse TAP policy
Identify Sender VM & edge switch
• Identify Monitor VM & edge switch
• N ← 0
N ← N+1
N > total number of switches?
yes
Finish
no
Switch #N is Sender edge switch?
yes
Identifiy egress ports for switch #N → Receiver & Monitor
Insert rewriting rule into switch #N
no
Switch #N is Monitor edge switch?
yes
Identify an egress port for switch #N → Monitor
Insert recovery rule into switch #N
no

Fig. 5 Flow Chart of the vMon Application

and recurrent self-correction based on past data over a given length of time might enhance the baseline. The dynamic baseline-based anomaly detection algorithm can better reflect the current state of the network's operational conditions than other methods. Figure 5 shows the overall sequence of operations in the vMon application. The central management policies can be interpreted at a higher level, while network packets are handled on a fine-grained level using the controller's global view. These functionalities run as modules in the data plane and are used to track network behavior. Until we perform flow-based detection, the vMon module must collect flow-related statistics from the OvS switches. This data collector collects flow data and exports it directly to the next level. Many data collection techniques can be employed.

Switch monitor

This module monitors the switch counters, manages the switch hardware ports and configurations, static thresholds, alarms, and meters the traffic at the packet level.

Flow monitor

This module classifies short-lived/long-lived malign/benign flows and automatically detects pre-cursors to impending attacks. The flow rules can dynamically update the sampling rate, packet headers, and action handler at the switch flow table.

Match-action

This unit extracts network metadata from the transport layer and application headers of incoming packets and takes actions based on the extended Match field composed of network events besides the original OpenFlow Match field.

We keep the anomaly detection and tracking processes decoupled. Therefore, any flow statistics collection system can be employed if it can access the upstream unit. We used OpenFlow samples exported from the OvS switches coarse-grained tracking (first level of anomaly detection). It can identify abnormalities in the traffic data and act on them. It will trigger the fine-grained attack classification algorithm (second-level IDS) to identify the attack. The architecture is highly flexible and scalable, allowing administrators to pick algorithms depending on the network dynamics and complex needs.

Feature selection

A NIDS's main objective is to choose/extract robust network statistics that can distinguish aberrant behavior from typical network activities. Most existing intrusion detection systems use network flow data (e.g., “Netflow, sflow, ipfix”). These five basic metrics measure the network’s behavior:

- ■ Flow Count: A flow/session consists of packets going from a specific source to a particular destination. (“source IP/port, destination IP/port, protocol”)
- ■ Packet Count: The total packets in a flow.
- ■ Byte Count: Bytes per second in a flow over a given amount of time.
- ■ Packet Size: A packet’s average number of bytes is over a period.
- ■ Flow Behavior: The “Flow Count/Packet Size” Ratio. Anomaly is quantified using this metric. Since most probing or surveillance attacks initiate multiple connections with little packets, the higher the ratio value, the more anomalous the flows will be.

Based on the above five metrics, we define a set of features to describe network traffic behavior. Let F denote the feature space of network flows; a 15- dimensional feature vector f∈F can be represented as {f1, f2, … f15}, where the meaning of each feature is explained in Table 3. A novel experimental hybrid intrusion detection system is presented that combines the highly accurate “signature-based” and “anomaly-based” IDS (that detects unknown attacks based on some specific criteria.

Access control and enforcement services

vACE (Access Control and Enforcement) is a request manager that sits between clients and cloud services in the SDN data plane. The framework provides access control services to enforce policies on other components, such as compute, image, and network. vACE is an attachment module that filters and mediates requests on the target service’s behalf. Based on the policy, the SDN controller chooses whether to authorize this access. SDN Controller’s primary functions are access control, verification, policy storage, and update for the entire cloud. It manages the REST interfaces. It is divided into three sections:

- API: acts as the vACE service’s REST interface.
- Verify: reviews the policy’s access rules and returns an access ruling in response to the vACE inquiry.
- Update: maintains the storage of all access control policies, manages policy updates, and cache timings.

In response to a user request, vACE issues an authorization request to the SDN Controller. All the main OpenStack functions are defined as API calls that may be called via a web server gateway interface (WSGI). It is envisaged that vACE will be able to intercept this interface and filter all incoming requests. All connections to the cloud are mediated and limited as a result. There are protected/privileged cloud functions that are not accessible via REST calls but require access restriction.

Table 3 Feature set collected at DP switches

Features	Description
f_1	Number of TCP Flows per Minute
f_2	Number of UDP Flows per Minute
f_3	Number of ICMP Flows per Minute
f_4	Average Number of TCP Packets per Flow over 1 Minute
f_5	Average Number of UDP Packets per Flow over 1 Minute
f_6	Average Number of ICMP Packets per Flow over 1 Minute
f_7	Average Number of Bytes per TCP Flow over 1 Minute
f_8	Average Number of Bytes per UDP Flow over 1 Minute
f_9	Average Number of Bytes per ICMP Flow over 1 Minute
f_{10}	Average Number of Bytes per TCP Packet over 1 Minute
f_{11}	Average Number of Bytes per UDP Packet over 1 Minute
f_{12}	Average Number of Bytes per ICMP Packet over 1 Minute
f_{13}	Ratio of Number of flows to Bytes per Packet (TCP) over 1 Minute
f_{14}	Ratio of Number of flows to Bytes per Packet (UDP) over 1 Minute
f_{15}	Ratio of Number of flows to Bytes per Packet (ICMP) over 1 Minute

DTARS applications

Policy management

The ODL-based application plays a central role in *OpenStackDP* security. It functions as the Central Policy Manager (CPM) and flow rule auto-configuration for OVS (data plane). A TAP policy management interface is supported (addition, removal, and modification) and is described in terms of OpenFlow rules and version 1.1, which extends it to 44 fields. This application translates the high-level policy into rules installed into OpenFlow switches. For example, TAP policy #1 in Table 4 enables its Monitor to only receive the duplication of the HTTP traffic from 10.1.1.5 to 10.1.1.6 by specifying the IPv4 protocol value as 6 (TCP) and the (TCP) destination port number as 80. Because a TAP policy is applied to the data plane in the form of OpenFlow flow rules, we can dynamically extend the Filtering fields depending on the target network used to offer more granularity on TAP policy specification. The network administrators use a global network view offered by the central SDN controller for creating a policy definition (text/JSON/XML/YANG) manifesto and run-time management. The DTARS application converts these high-level policy definitions into specific OpenFlow flow rules to be installed into related

Table 4 Example of TAP high-level policy specification in DTARS application

TAP Policy	Identifier fields (mandatory)						Filtering fields (Optional)				
	Sender MAC	**Sender IP**	**Receiver MAC**	**Receiver IP**	**Monitor MAC**	**Monitor IP**	**IPv4 Protocol**	**VLAN ID**	**SRC Port no**	**DST Port no**	**Other OF Fields**
Policy#1	74:8f:3c: ba:91:05	10.1.1.5	74:8f:3c: ba:91:06	10.1.1.6	74:8f:3c: ba:91:07	10.1.1.7	6 (TCP)	*	*	80	–
Policy#2	74:8f:3c: ba:91:05	10.1.1.5	74:8f:3c: ba:91:06	10.1.1.6	74:8f:3c: ba:91:08	10.1.1.8	18 (UDP)	108	*	*	MPLS_LABEL = 10
Policy#3	74:8f:3c: ba:91:06	10.1.1.6	74:8f:3c: ba:91:05	10.1.1.5	74:8f:3c: ba:91:09	10.1.1.9	6 (TCP)	200	*	80	IP_ECN = 9

switches. We have two flow rules in the data plane: rewriting and recovery. The rewriting is built into the Sender VM's OVS bridge (sender edge OVS). This kind of flow rule has one flow entry and two matching classes. If there is a match in the target-action edge of the sender with the corresponding OVSDN, access is delegated to that policy. The table method can alter the MAC and IP address (e.g., VLAN tagging). This single flow needs to only adhere to a TAP rule. When the MAC or IP address rewrite is complete, the last rule is applied to restore the contents. The recovery rule is related to the VM table in the OVS bridge.

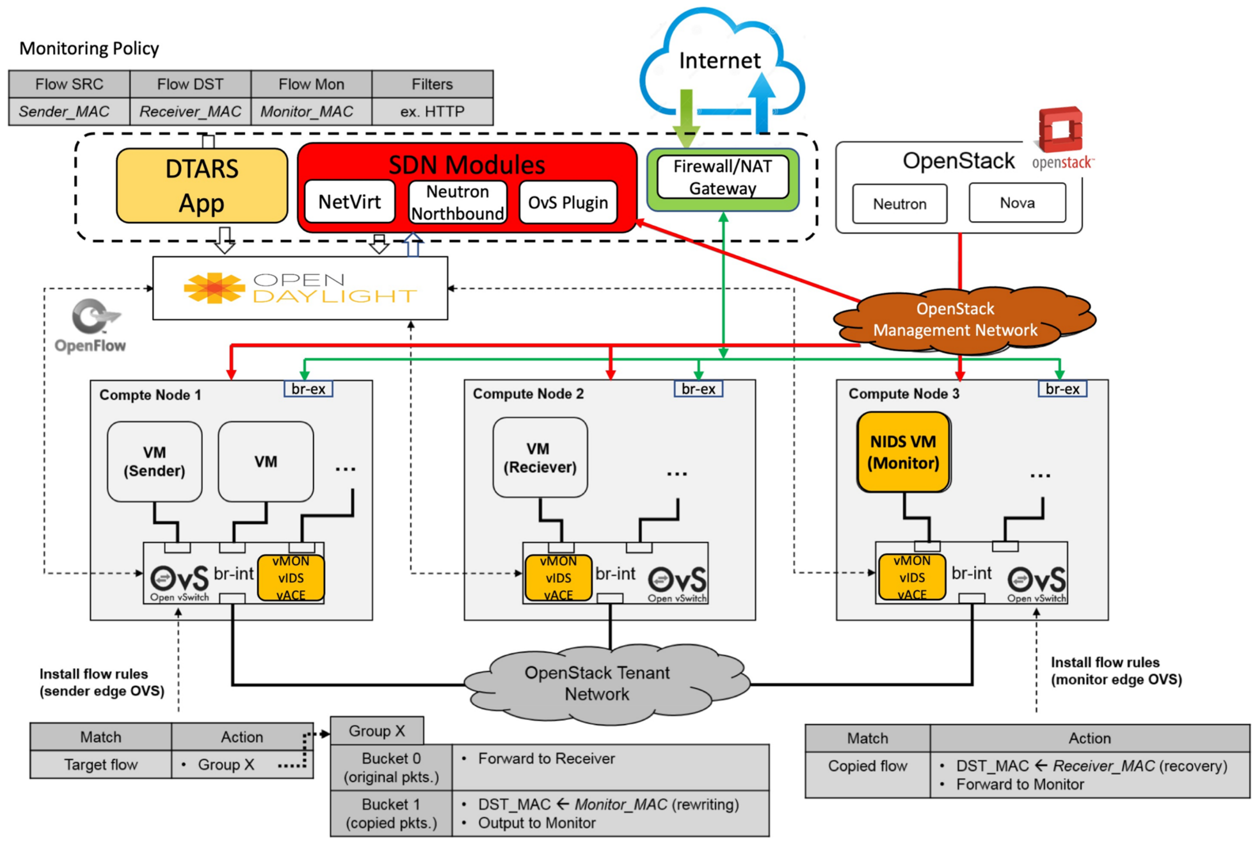
Attack mitigation

Malicious flows can be identified by packet inspection. The second function calculates malicious flows based on the source IP address, while the third creates and returns legit packets. We group more malicious flows into more extensive flow entries to improve scalability. The level of the attack and OvS device restrictions (flow capability requirement) must be met by the operator (e.g., the attack may be of a low rate, or a flooding attack, or the operator might want to block all malicious sources or might leave some unblocked).

Design and implementation

The *OpenStackDP* framework (Fig. 6) comprises the following essential layers.

- *Infrastructure* (OpenStack Nova): This layer consists of virtual/physical machines, switches, and devices.
- *Switches*: OpenFlow/hybrid switches on the Edge. Flows with suspicious packets are flagged for additional investigation.
- *Control Plane*: SDN Controller ("OpenDayLight") program and user applications are customized with extension monitoring, flow classification, and attack detection logic. It invokes the defense action to send commands (encapsulating the -matching- action) to the switch(es) upstream path traveled by the malicious traffic.
- *Security Controller*: This enforces security policies on the tenant network and external access through the OvS switch agents.
- *Cloud Admin*: Cloud computing platform and management applications (e.g., OpenStack) are extended with improved Neutron for collaborating with SDN.
- *SDN Modules:* This includes *networking-sfc, networking-odl, and NetVirt*. The main functions are auto-



Monitoring Policy

Flow SRC	Flow DST	Flow Mon	Filters
Sender_MAC	*Receiver_MAC*	*Monitor_MAC*	ex. HTTP

Match	Action
Target flow	• Group X

Group X	
Bucket 0 (original pkts.)	• Forward to Receiver
Bucket 1 (copied pkts.)	• DST_MAC ← *Monitor_MAC* (rewriting) • Output to Monitor

Match	Action
Copied flow	• DST_MAC ← *Receiver_MAC* (recovery) • Forward to Monitor

Fig. 6 SDN-enabled OpenStack Architecture (OpenStackDP)

scaling and VNF placement. It manages traffic steering in the tenant network and north-south traffic to public internet access.

Once *OpenStackDP* starts running, attack sensors on the OvS switch monitor the packet stream/flows that pass through them. The coarse-grained anomaly detection detects malicious or abnormal flows (e.g., DoS attacks). As the data plane is where packets are forwarded and spend most of the transit time, leveraging the resources on switches for packet inspection and making local routing decisions is the logical/optimal case. Enabling dynamic defense mechanisms and programmable security perimeter in the data plane and tenant network is the critical strategy in our proposal.

Stateful SDN firewall

OpenStackDP framework utilizes the stateful SDN architecture implemented in [54]. We offer an intelligent SDN data plane-based firewall (see Fig. 7) that analyses the contents of a packet to identify malicious traffic. Using OpenStackDP, harmful traffic may be quickly discarded to protect the perimeter of company networks with an intelligent SDN firewall. The payload of a packet can be extracted using Open vSwitch (OvS) before it is matched against flow tables. All payloads will be sent to the SDN controller to make future judgments. The SDN controller has access to a machine learning firewall service to help assess whether a payload is benign or malicious. DPMonitor, Firewall agent, and payload extraction are the three main components.

Payload extraction

The payload of the packets received by the SDN datapath is extracted to perform a more thorough analysis of the data. A datapath module in the OvS stack receives all packets from external networks before

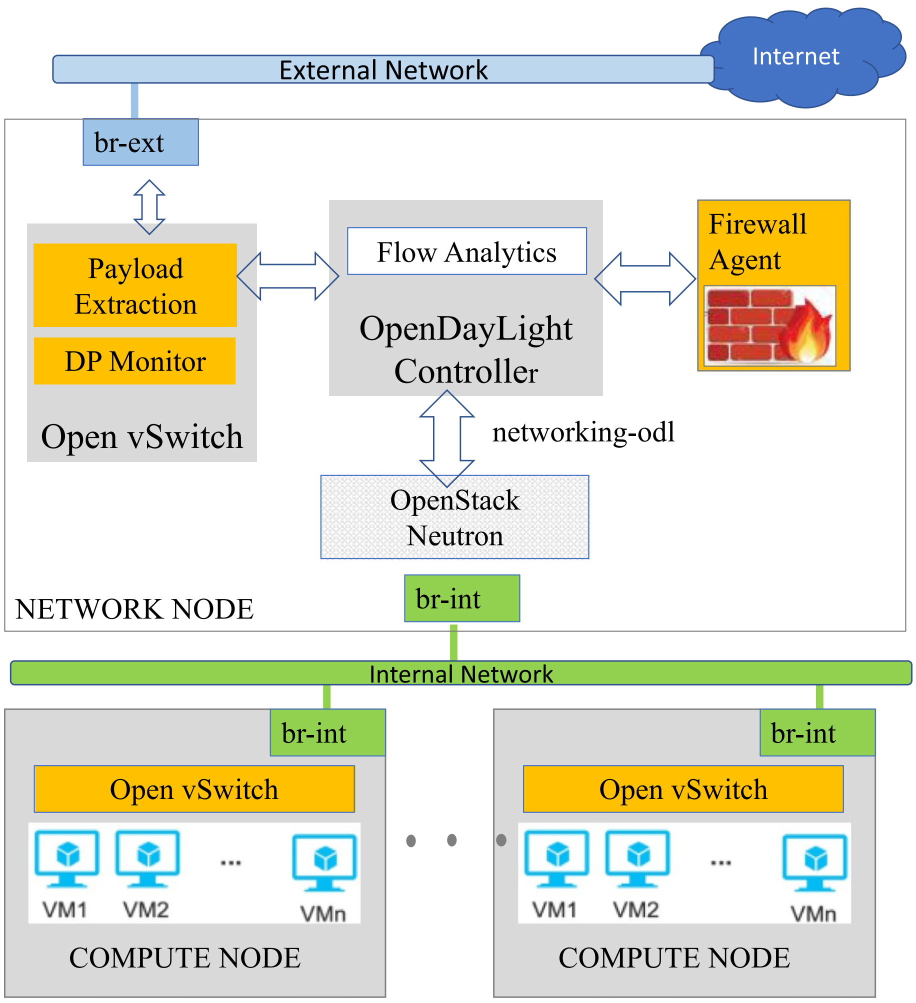


Fig. 7 Intelligent SDN-based Firewall in Network Node

extracting key values (such as MAC-layer and network-layer messages) and matching them with flow tables cached in the kernel.

DPMonitor (vMon)

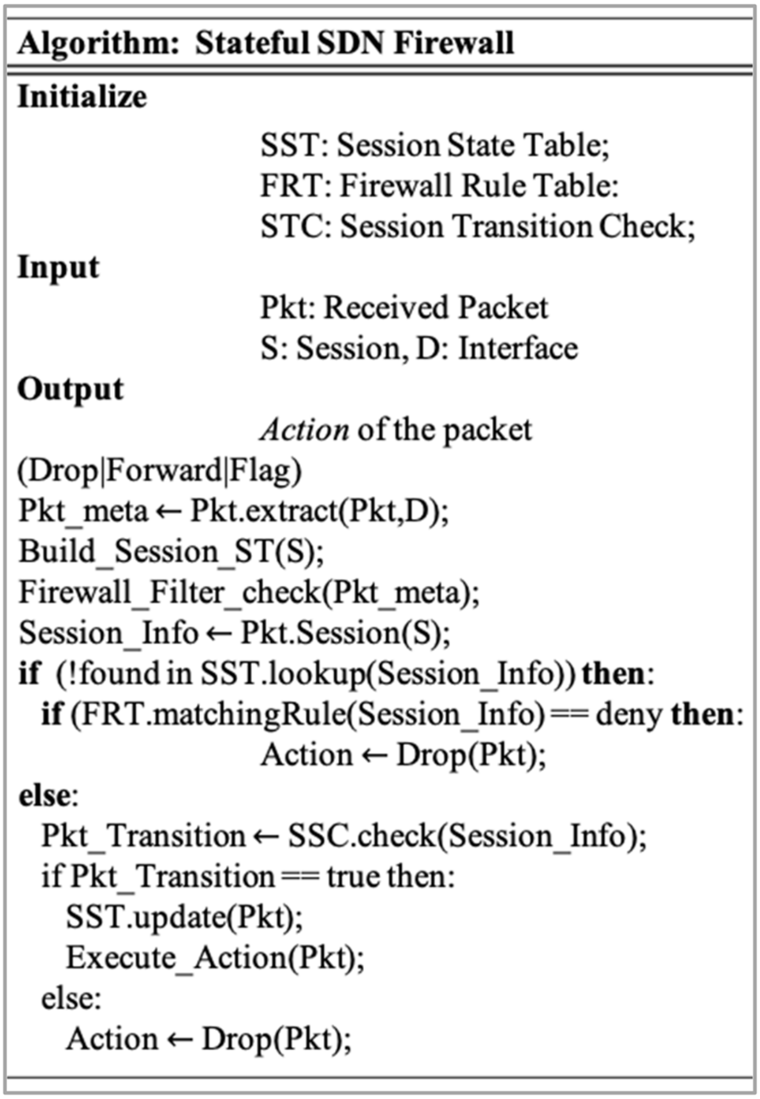
The DPMonitor engine in the datapath monitors payloads from the switch stack. It invokes other application agents such as vMon and vIDS.

Firewall agent application (vIDS)

As described in the algorithm (Fig. 8), the stateful firewall filter application is integrated into the SDN OVS data plane for packet integrity analysis. The packet is dropped or forwarded after the inspection is complete. Next, the IP addresses/ports and session states are tracked and filtered. The application has primarily been instantiated for TCP communications monitoring — states, changes, and behavior.

OpenStack neutron SDN layers

Figure 9 illustrates the connectivity between SDN and OpenStack networking components. SDN services are specified in Python classes and called by the Neutron layer. The main package (driver module) *networking-odl* acts as a redirector/proxying/routing component interfacing between the SDN abstractions and OpenStack security/policy configurations. This module includes functions to create new networks and establish security groups for tenants.



```
Algorithm: Stateful SDN Firewall
Initialize
                    SST: Session State Table;
                    FRT: Firewall Rule Table:
                    STC: Session Transition Check;
Input
                    Pkt: Received Packet
                    S: Session, D: Interface
Output
                    Action of the packet
(Drop|Forward|Flag)
Pkt_meta ← Pkt.extract(Pkt,D);
Build_Session_ST(S);
Firewall_Filter_check(Pkt_meta);
Session_Info ← Pkt.Session(S);
if (!found in SST.lookup(Session_Info)) then:
  if (FRT.matchingRule(Session_Info) == deny then:
                    Action ← Drop(Pkt);
else:
  Pkt_Transition ← SSC.check(Session_Info);
  if Pkt_Transition == true then:
    SST.update(Pkt);
    Execute_Action(Pkt);
  else:
    Action ← Drop(Pkt);
```

Fig. 8 Stateful Firewall in SDN-OpenStack Integration

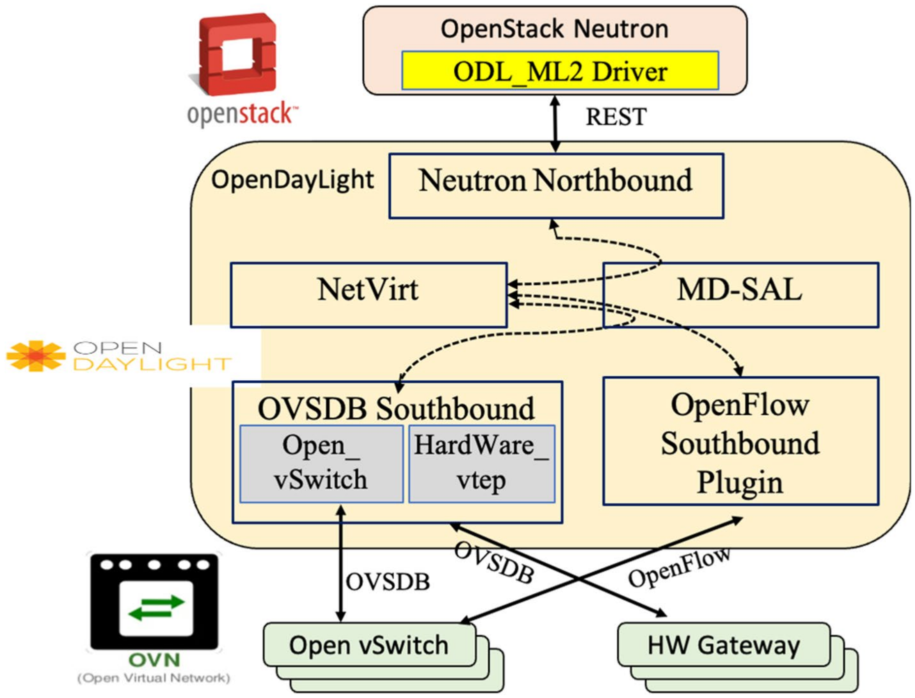


Fig. 9 Interconnection between SDN and OpenStack

NetVirt

Module manages the switching/routing functions.

networking-odl

Module handles IP subnetting across tenant networks, routing VLANs (Layer2) and VXLAN (Layer3 overlay) to interconnect multiple OpenStack domains.

Neutron plugin

This module handles the network functions. It establishes connections for processes in compute nodes through *br-int* (tenant network) and external internet through the *br-ex*. OpenFlow *rules* and the *match-action* set are stored in the ODL controller. The security, access control, and policies are translated into OpenFlow rules.

Neutron agent

This program runs on all the compute nodes and interconnects through the OVS bridge. It is a standalone process with the "*ML2 core backend*" on OVS switches.

Neutron API

A simple "*CRUD* (create-read-update-delete) flow-rule" approach was changed to support ODL calls, and the REST interface was updated for policy administrators.

Anomaly and intrusion detection system

In general, *signature*-based and *anomaly*-based approaches are the most common methods for detecting intrusions in a network system. It's possible to identify assaults based on the patterns or signatures that have already been stored for known incursions. Even though deviations from previously stored profiles of typical activity can detect intrusions, an anomaly-based method can also detect unknown intrusions (or suspicious patterns). To do this, the NIDS scans all incoming packets for unusual data patterns. One of the biggest challenges in statistical anomaly-based attack detection is determining if an outlier is a reality (in our case, an anomaly or malicious activity or attack) or something benign (temporary and natural activity or network spike or jitter). Network traffic is gathered by the gateway firewall and preprocessed to form a dataset. A clustering algorithm is used to build service-based patterns across the dataset. We employ a multistage clustering-based outlier detection technique to distinguish between assaults and network anomalies (spikes/jitter). Static thresholds may not be able to tell the difference between malicious and regular traffic. Figure 10 illustrates that the attack spikes (red) cannot be separated from the harmless ones (blue) using simple thresholds. So, to distinguish between malicious and benign traffic, we use a dynamic model to develop profiles for various traffic circumstances. An unsupervised learning strategy

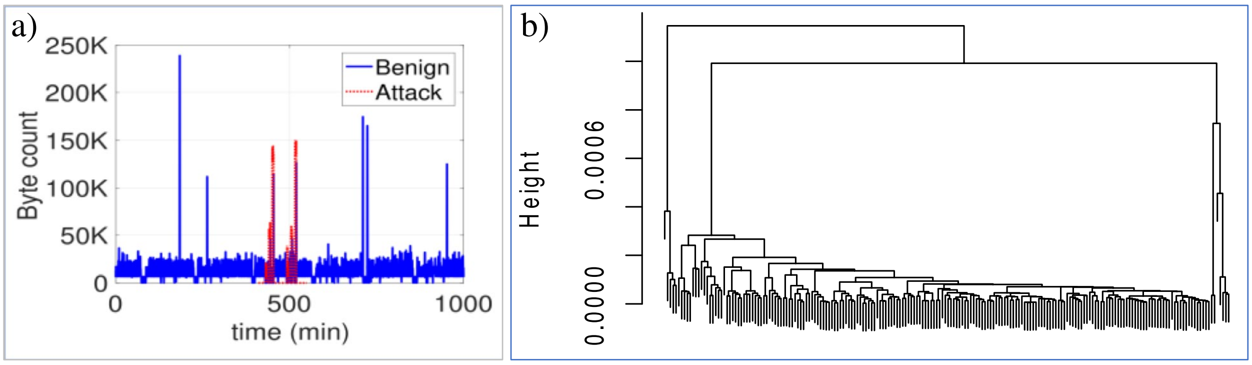


Fig. 10 **a** Traffic Profile **b** Dendrogram Plot

that divides a data set into multiple clusters is called clustering. In hierarchical clustering, each data point is assigned to a distinct set of groups based on Euclidean distances. There are two options for plotting dendrograms when the iterative merging procedure is complete: either the z-axis (which we use) or one of the characteristics (which we use), namely x_1, x_2, x_3, x_4. Figure 10b illustrates the hierarchical link between the clusters using a tree structure (*no. of clusters* on the x-axis, *distances* on the y-axis). An ideal horizontal axis is determined after the hierarchical clustering has been completed.

We offer a hybrid system to combine signature-based IDS with the anomaly detection mechanism and the EM-based clustering scheme [55]. As shown in Fig. 11, the architecture of the system has these major functional blocks: Lightweight IDS (based on packet stats), Heavy-weight IDS (Feature analysis), and Anomaly IDS (Clustering analysis). The first learning phase consists of building a feature set of false positives, which gets updated after a specific time frame, and the threshold of the actual alert is calculated. When categorical attributes are involved, pre-processing the data set is critical to identifying their relevant features. In the next stage of online filtering, the outlier score of each new alert is compared with the threshold value to find when it is a false positive. There must be a proper threshold value to distinguish between various data points in continuous attributes. The closeness, data type, dimensionality, and threshold measure are critical in *distance-based* outlier techniques. It should be highlighted that model reconstruction is unnecessary when using distance-based techniques when modifying the threshold (i.e., changing the outlier factor criterion). A combination of distance, density, and soft computing can provide resilience and

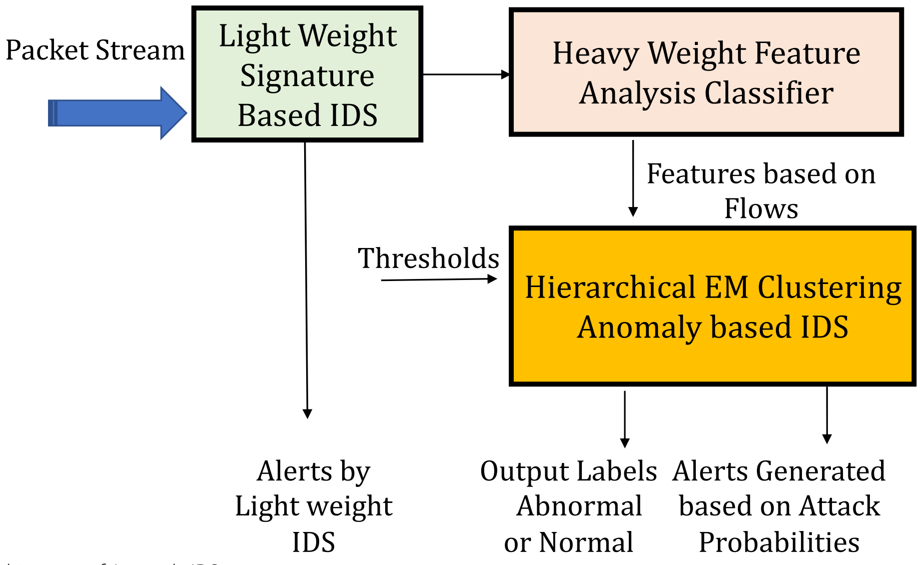
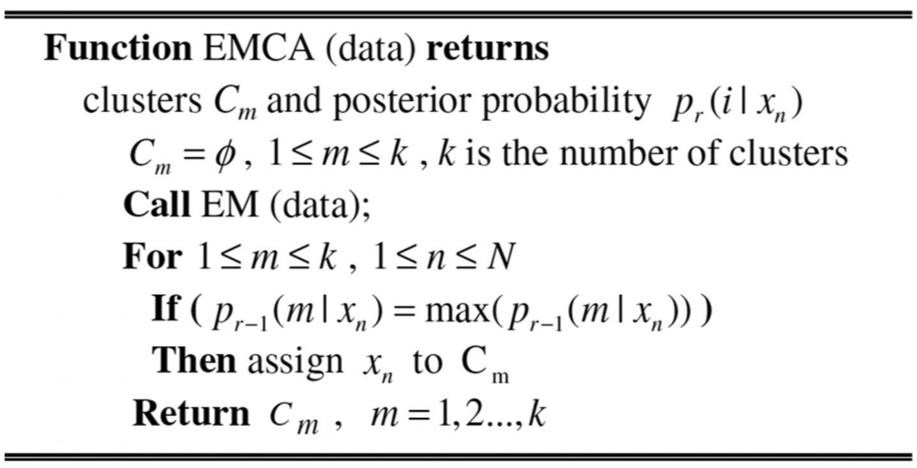

Packet Stream
Light Weight Signature Based IDS
Heavy Weight Feature Analysis Classifier
Features based on Flows
Thresholds
Hierarchical EM Clustering Anomaly based IDS
Alerts by Light weight IDS
Output Labels Abnormal or Normal
Alerts Generated based on Attack Probabilities

Fig. 11 General Architecture of Anomaly IDS

scalability for outlier detection according to numerous circumstances.

To identify outliers, a user-defined cutoff value is produced and compared to each data point. Outliers O_{ij} (including minor outliers) are recognized in terms of the threshold τ and the average of the (95th, 96th, and 99th) percentiles, which are ordered by their scores. As a result, the adaptive or conditional threshold value can improve performance in outlier detection approaches based on distance. Only the data/packets associated with the suspicious flows are sent to the control plane ML application for further study. We can detect abnormal flows. We believe that a more significant number of features will yield a more accurate description of the network's traffic patterns during feature analysis, in which we design and generate 15 different characteristics. Using a set of scoring coefficients, the IDS and EM clustering-based detectors combine these new features. The inference model's fuzzy attacking probability provides the ultimate intrusion decision. We have compared our proposed method with other clustering algorithms like X-Means, Farthest First, filtered clusters, DBSCAN, K-Means, and EM (Expectation-Maximization) clustering to find the suitability of our proposed algorithm. We are using EM to fit the data better so that clusters are compact and far from other clusters since we initially estimate the parameters and iterate to find the ML (Maximum Likelihood) for those parameters. The Gaussian Mixture Model's parameters are frequently assessed using the EM technique (GMM). Data points may have an approximately Gaussian distribution, described by the conditional probability in EM. In most circumstances, the conditional probability of being a GMM component for some data points is closer to 0. These data points are referred to as "noisy data" due to their tendency to stand out. During anomaly detection, the outliers are discarded or labeled as *anomalies*, and the corresponding *attack probabilities* are set to one. The EM model is shown in Fig. 12, where the clustering results are referred to as Cm.

Two key assumptions must be made to use the EM-based clustering technique to discover network anomalies: There are two clusters in the input data: the anomalous cluster, which is smaller than the regular cluster, and the standard cluster. As a result, we can label each abnormal cluster based on its size. We designed a modular anomaly detection scheme and the workflow, as shown in Fig. 13. The OvS core switches in the tenant network collect packet-level and counter statistics and feed them to multi-stage anomaly detection and identification system (shown in orange). The computations are short-lived and linked to *events* that could be launched within the OvS switches without any middleboxes. OpenStackDP data plane uses an outlier detection technique to detect anomalies without supervision. Our proposed framework aims to detect anomalous patterns using the outlier approach. It works first by identifying reference points and by ranking outliers' scores. We identify outliers in the packets collected from the network at the OvS vMon component. An anomaly can occur when observed data deviates from commonly occurring ones. So, on that basis, it is notified that an anomaly has been found beyond these boundary conditions. We flag the flow as suspicious and investigate further by the ML-based Classifier system in the Controller. If the flow falls within these boundaries, it is passed to the final phase. The clusters are developed by running a clustering algorithm, as shown in Fig. 14. We collect data from flow counters regularly and use a sliding window of 1 to 4 minutes to calculate statistical features, resulting in about 15 features per flow



```
Function EMCA (data) returns
  clusters $C_m$ and posterior probability $p_r(i \mid x_n)$
    $C_m = \phi$, $1 \le m \le k$, $k$ is the number of clusters
    Call EM (data);
    For $1 \le m \le k$, $1 \le n \le N$
      If ( $p_{r-1}(m \mid x_n) = \max(p_{r-1}(m \mid x_n))$ )
      Then assign $x_n$ to $\mathrm{C_m}$
     Return $C_m$, $m = 1, 2 \ldots, k$
```

Fig. 12 EM Based Clustering Algorithm

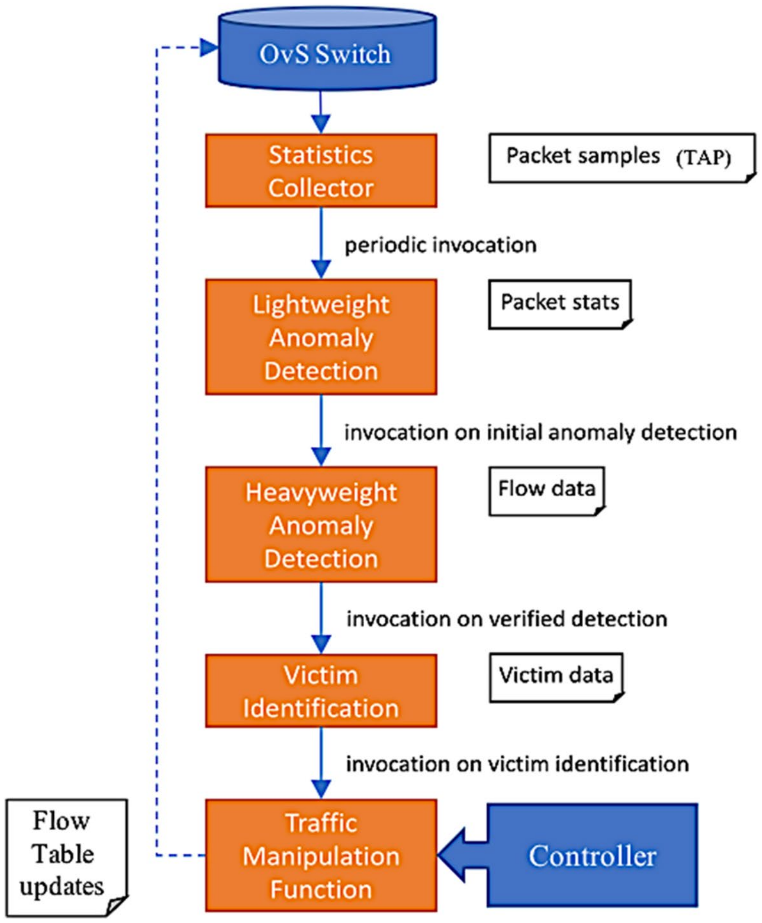
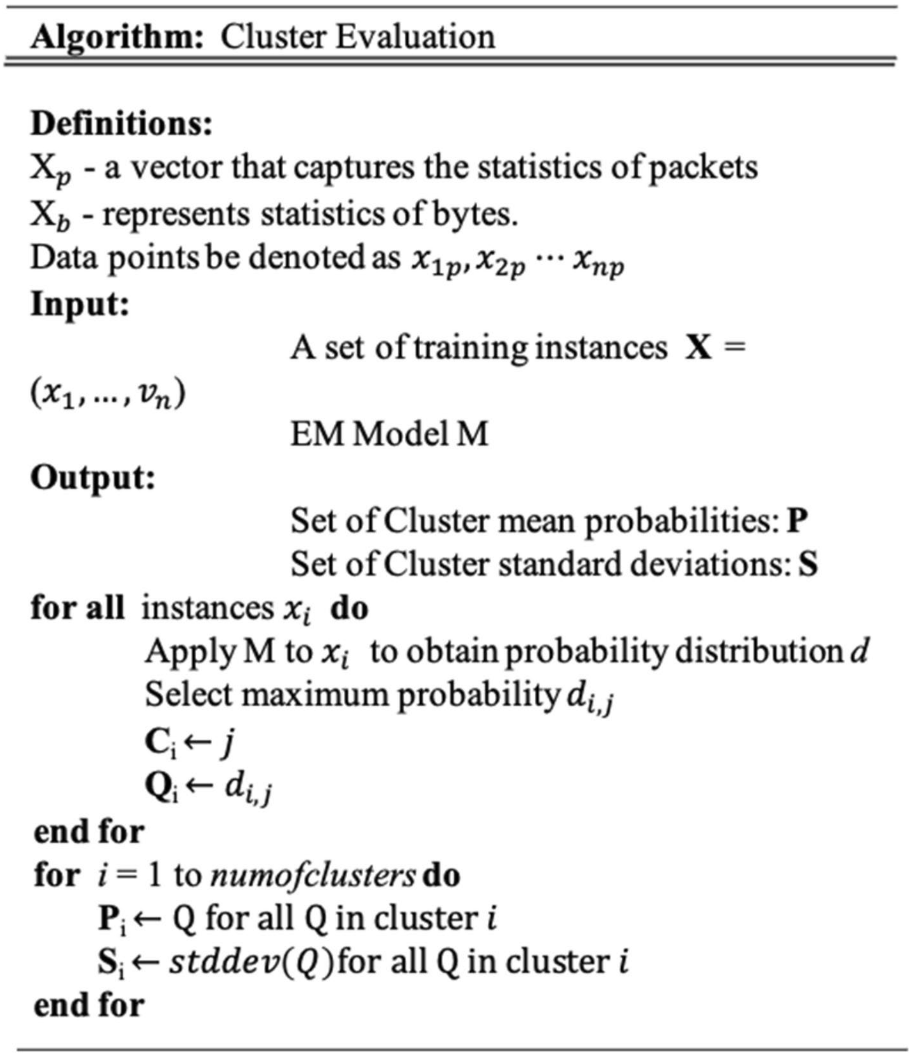

OvS Switch
Statistics Collector
Packet samples (TAP)
periodic invocation
Lightweight Anomaly Detection
Packet stats
invocation on initial anomaly detection
Heavyweight Anomaly Detection
Flow data
invocation on verified detection
Victim Identification
Victim data
invocation on victim identification
Flow Table updates
Traffic Manipulation Function
Controller

Fig. 13 High-level workflow of network anomaly detection

(See Table 4), including bidirectional data. First, we do flow-level analytics to save time on deep packet analysis. Packet-level attributes ("Destination IP addresses, ports or inter-arrival durations") and flow-rule history are only considered if suspicious flows from Stage-1 have been detected. An organization's behavioral profile can be developed by analyzing its operations and performance. As fingerprints, its communication patterns have been consolidated. Packet sequences can be used as valid semantic data for the sessions by generating a single feature vector from the metadata of consecutive packets $p^i \rightarrow p^{i+1} \rightarrow p^{i+2} ... \rightarrow p^n$. The system's observed behavior profile is matched by this set of characteristic vectors, which can be used in ML-based classifiers. We may notice trends in the data and outlier events. When outliers are observed using the algorithm in Fig. 15, the system raises warnings after the outliers are detected and validated (eliminating the noise and false alarms), an anomaly detection algorithm (as shown in Fig. 16) based on thresholds, and the mitigation system is alerted about the attack.

Dynamic service function chaining system

OpenStack-Neutron does not natively support the Service Function Chaining (SFC) functionality. The Virtual Network Functions (VNFs) deployed on the compute nodes must be linked to the network gateway node to enable the chaining/pipeline of network functions/services. The packets/flows must be steered between the link points, which involves complex logic/routing/splitting/merging operations. The *networking-sfc* module developed by the OpenStack community has tackled this problem and uses various drivers for this purpose. Now, drivers are available for the OVS, ONOS, and ODL infrastructures. In OpenStackDP, OVN is being enhanced with service chaining and a driver to communicate between *networking-sfc* and the OVN architecture. The Open vSwitch module is augmented to enable network function chaining. The module interacts with the agent/plug-ins running on the compute nodes. The traffic sequence to enter/exit the chain is determined by using Flow classifiers. A common module is the flow classifier, which can be



```
Algorithm: Cluster Evaluation
Definitions:
$X_p$ - a vector that captures the statistics of packets
$X_b$ - represents statistics of bytes.
Data points be denoted as $x_{1p}, x_{2p} \cdots x_{np}$
Input:
        A set of training instances $\mathbf{X} = (x_1, \ldots, v_n)$
        EM Model M
Output:
        Set of Cluster mean probabilities: P
        Set of Cluster standard deviations: S
for all instances $x_i$ do
    Apply M to $x_i$ to obtain probability distribution $d$
    Select maximum probability $d_{i,j}$
    $\mathbf{C}_i \leftarrow j$
    $\mathbf{Q}_i \leftarrow d_{i,j}$
end for
for $i = 1$ to $numofclusters$ do
    $\mathbf{P}_i \leftarrow$ Q for all Q in cluster $i$
    $\mathbf{S}_i \leftarrow stddev(Q)$ for all Q in cluster $i$
end for
```

Fig. 14 Hierarchical Clustering Method

used for other applications such as Firewalls, Quality-of-Service, Load-balancing, etc. Monitor/DoS Scrubber VMs are deployed on the OpenStack compute nodes. With minimal impact on throughput for big flows, the flow analyzer engine classifies and tags the flows for priority queuing. Embedded NF performs classification and dynamic path adjustment via ChangePath messages in switches based on flow characteristics such as phase change, burst interval, and packet size. Network-function/service sequence/chain service graph is generated by the OpenStack networking-sfc module [56]. Based on the content inspection, the vertex nodes and edges depict the packets' multi-path trajectories. The vertex node's edges determine the next hop for a packet. As a result of the NF Dependency Analysis and the operator's policy template, the ODL controller creates the final service graph (Fig. 17). Service graphs and (potential subgraphs) are translated into flow-table rule definitions by the SDN controller so that the data plane switches can install and manage NFV processing. The NF handler invokes a flow rule <Match: the result of the VNF, Action: Discard/Send to/Default> when the VNF has finished processing a packet. For each node, an ODL controller specifies a Default path (shown in bold) and a Default action (shown in bold).

A custom action or logic can be executed based on the result of the current VNF in other dynamic SFC situations, as demonstrated in Fig. 18, where the ODL Controller can install match-actions rules. Next, the decision is made per packet for the following path or edge to cross. There may be a succession of NFs for

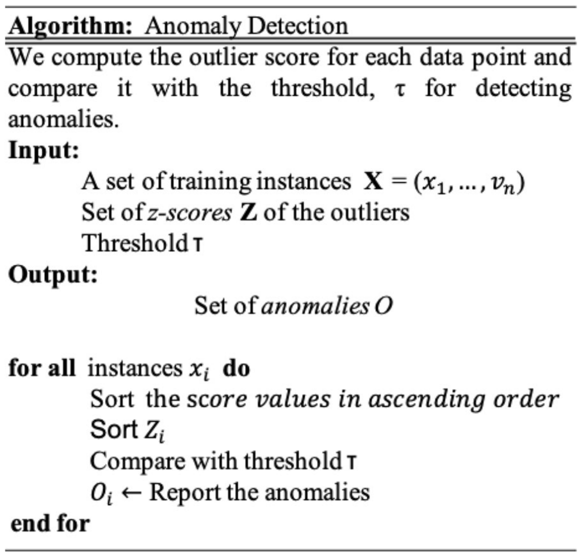
```
Algorithm: Outlier Detection
We do cross-validation and verification by using $Z$-scores
Definitions:
  Lower quartile $Q_1$, Upper quartile $Q_3$
  Inter-quartile range (IQR) ← $Q_3 - Q_1$,
  Lower-outlier← $Q_1 - 1.5(Q_3 - Q_1) = Q_1 - 1.5(IQR)$
  Higher outlier← $Q_3 + 1.5(Q_3 - Q_1) = Q_3 + 1.5(IQR)$
   function Tietjen-Moore test($x_{bi}$)
For the $k$ largest point, $L_k = \frac{\sum_{i=1}^{n=k}(x_{bi} - \bar{x_{bi}}^2)}{\sum_{i=1}^{n}(x_{ib} - \mu_b)^2}$,
where $\bar{x_{bi}}^2$ denotes sample mean removing largest $k$ points.
For the $k$ smallest point is $L_k = \frac{\sum_{i=k+1}^{n}(x_{bi} - \bar{x_{bi}}^2)}{\sum_{i=1}^{n}(x_{ib} - \mu_b)^2}$
return($H_A$)
end function
Input:
      A set of training instances  $\mathbf{X} = (x_1, \ldots, v_n)$
      EM Model M
      Set of Cluster mean probabilities: P
      Set of Cluster standard deviations: S
Output:
      Set of z-scores Z
      Set of outliers O
for all instances $x_i$ do
      Apply M to $x_i$ to obtain probability distribution $d$
      Select maximum probability $d_{i,j}$
      $\mathbf{Z}_i \leftarrow d_{i,j} - \mathbf{P}_j$
      $\mathbf{Z}_i \leftarrow \mathbf{Z}_i \mid \mathbf{S}_J$
end for
for all instances $x_i$ do
      Sort the score values $\mathbf{Z}_i$ in ascending order
      //Do Hypothesis testing to verify outliers
      $\mathbf{O}_i \leftarrow$ Result of Tietjen-Moore test($\mathbf{Z}_i$)
end for
```

Fig. 15 Outlier detection and validation Method

packets to pass through in Enterprise Edge gateway use cases, such as an IDS/Firewall/Sandbox or DoS scrubber/honeypot. When the IDS discovers anomalies or malicious traffic, the packets are diverted to the Sandbox for further analysis (with a *match: action* configured to call a Sandbox VNF). A Sandbox VNF on the IDS directs suspicious packets to the scrubber/honeypot VNF, while the default path is used for all other traffic. That flow of packets would be re-routed or dropped at the Firewall, which is considerably earlier in its chain, using feedback messages.

Performance evaluation

We implemented the testbed (shown in Fig. 19) using *OpenStack Pike* Devstack [15]. For networking, we deployed our custom OVS-DPDK on the compute nodes ("*br-int* bridge ") under Mininet [50]. Some experiments utilize Mininet [50] to simulate large-scale virtual switches and hosts in a cloud computing environment. Multi-tenant is a major feature of the cloud data center environment, and the tenants hope their virtual subnets cloud will be effectively isolated from other virtual subnets.



```
Algorithm: Anomaly Detection
We compute the outlier score for each data point and compare it with the threshold, $\tau$ for detecting anomalies.
Input:
        A set of training instances $\mathbf{X} = (x_1, \ldots, v_n)$
        Set of z-scores $\mathbf{Z}$ of the outliers
        Threshold $\tau$
Output:
                Set of anomalies $O$

for all instances $x_i$ do
        Sort the score values in ascending order
        Sort $Z_i$
        Compare with threshold $\tau$
        $O_i \leftarrow$ Report the anomalies
end for
```

Fig. 16 Anomaly Detection Method in IDS

Experimental setup

- To integrate SDN into OpenStack, we deploy OpenStackDP to manage the networking resources with Neutron. In the setup, software (i.e., Open vSwitch-OVS) and hardware switches are controlled by a single SDN controller.
- OpenStack comprises "compute, storage, and networking" nodes. It keeps evolving to enable more features and higher stability. The operator can choose from abundant components to create the deployment that best suits its requirements. The components run in heterogeneous infrastructure, and new hardware can be easily included.
- We deploy two homogeneous server class machines: the controller and compute node (which in turn houses multiple compute node instances in Mininet Simulator running hosts. We enable the following components: Nova for compute service, Neutron for

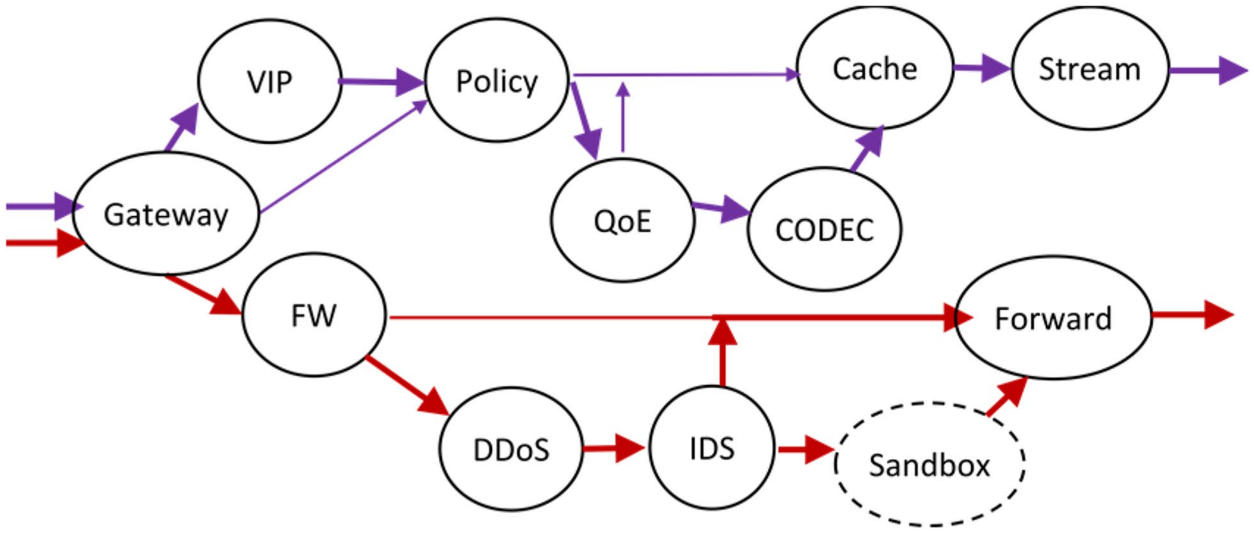


Fig. 17 Default Service Graph Example

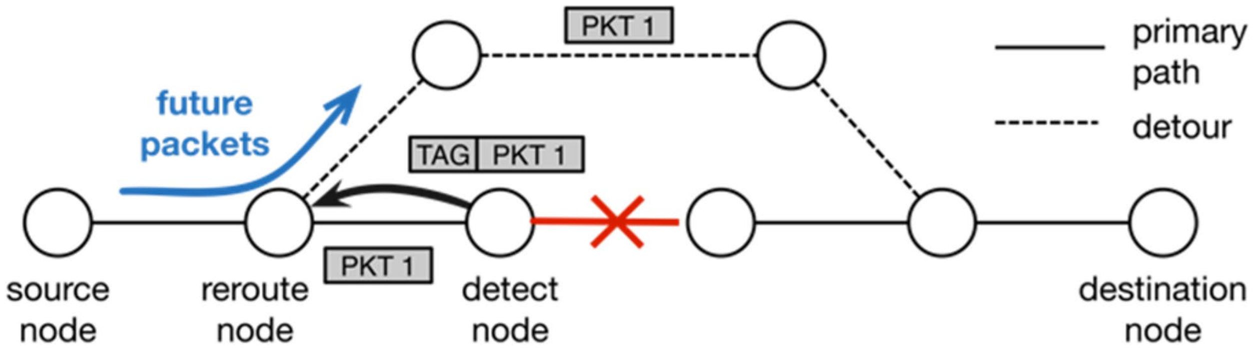


Fig. 18 Dynamically Changing Service graph

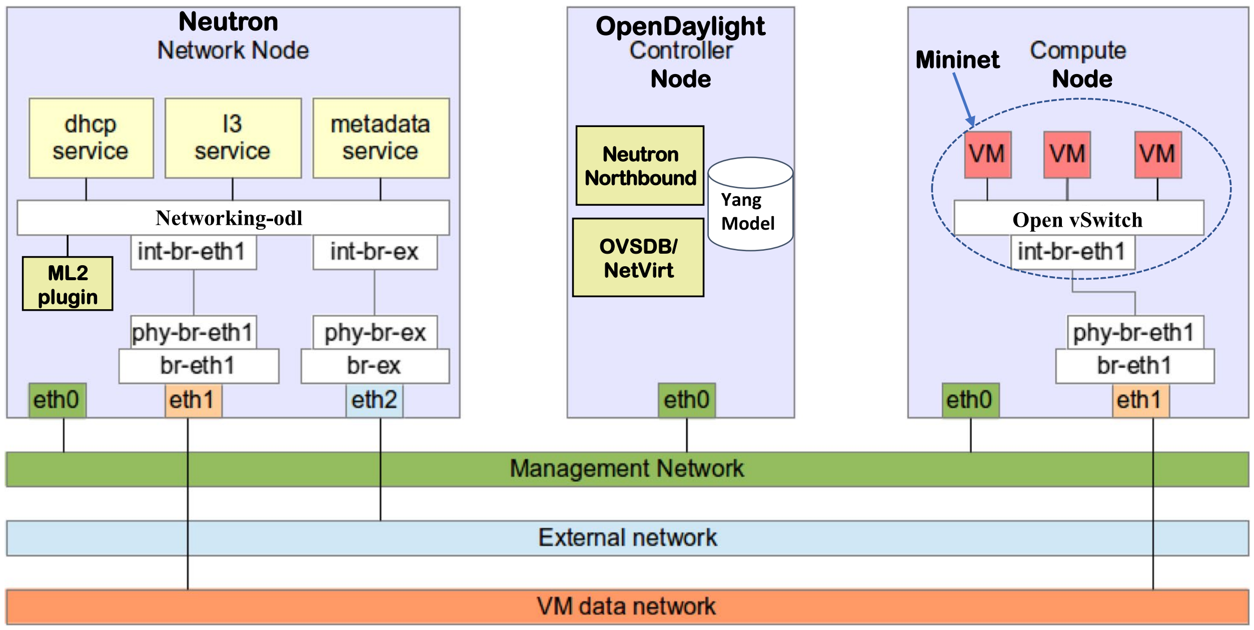


Fig. 19 OpenStackDP Testbed Environment

networking, Keystone for identity service, Glance for image service, Cinder for block storage, and Horizon for the dashboard.

- Each compute node is connected to three networks. The data network which carries the traffic between the VMs is enabled via OVS. The management of software switches, the control of OpenFlow, and the internal communication between OpenStack components are realized by the management network via eth0. We use a custom L3/L3 switch/router running OpenStackDP dataplane software to forward the management traffic and run DTAR, firewall, security, and access control applications.

Basic micro-benchmarks

The main benefit is that with SDN flow rules is that the rules are deployed at various points in constant time. When DDoS-attacked by the attacking hosts, we test the network throughput and latency between two regular hosts with the iPerf application. Our SDN firewall maintained normal host performance, even under high attack conditions. We can obtain up to 20% or 50% more improvement in throughput for homogenous synthetic workloads when the packet size and the number of flows vary. We save about 50% of the cycles for the mixed workload and get around 10-fold better efficiency.

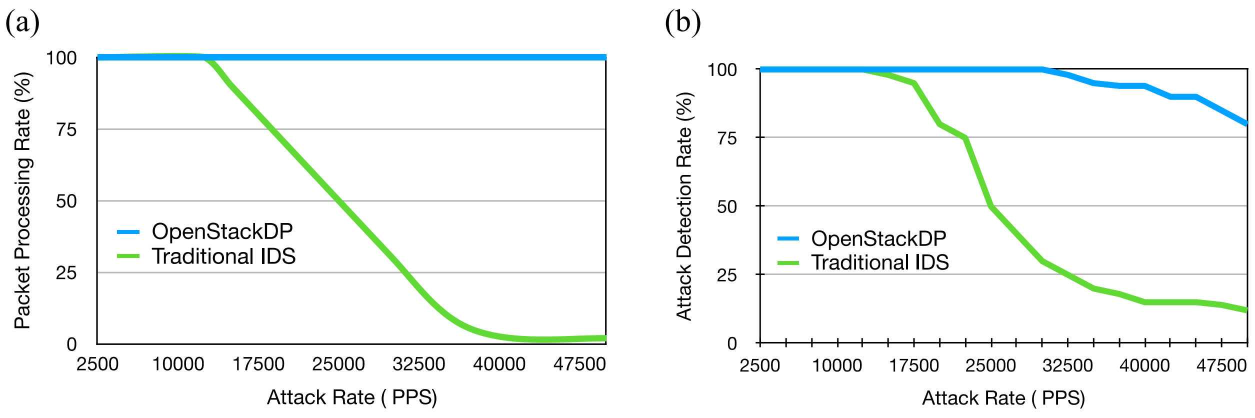
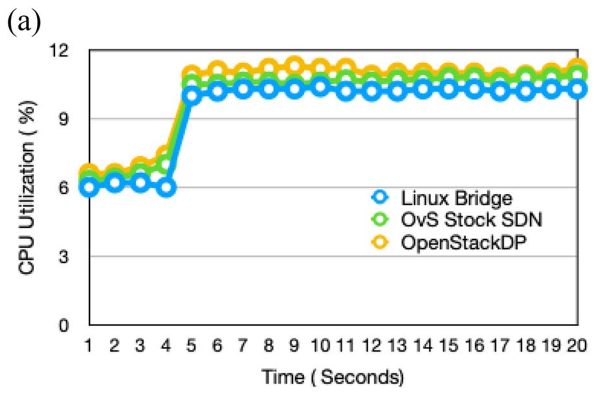


Fig. 20 Performance of Intrusion. **a** Prevention and **b** Detection



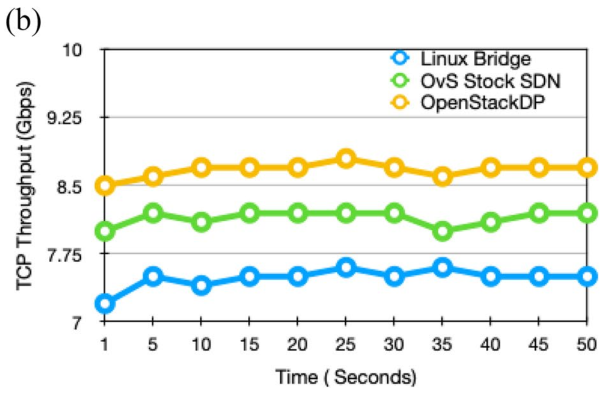


Fig. 21 **a** CPU Usage **b** Network Performance in node

Attack simulation and DDoS attack emulation

Experimenting with various attacks, we simulated the data plane's ability to detect them. Packet sizes, protocols, inter-packet gaps, intensity, and so on were all varied by the traffic generators from the attacker machines, which swamped the network with a variety of traffic. The application traffic emulated client-server protocol behavior (request/response). Experiments are broken down into categories based on the sort of attack and the amount of time spent in the pipeline. Flooding and slow-rate attacks are used in a variety of ways. At predetermined intervals, the number of threads/application processes per computer every five apps steadily increased.

IPS/IDS efficiency

Figure 20a shows the "total-packets processed/sec" variation with attack intensities. OpenStackDP default match-action is configured as "(drop)," as "Snort/Iptables" support only drop filter. The result shows that traditional IPS starts dropping packets from 13 K packets/sec to zero at 36 K packets/sec. For the same attack intensities, OpenStackDP sustained the throughput. Figure 20b When the intensity of attacks increases, the legacy-IDS efficiency gets affected due to the DoS attack packets filling up the "Iptables Queue." OpenStackDP is resistant to heavy-hitting attacks. We conducted efficiency assessment metrics at different demand conditions and network throughput levels. We set up five VMs and have services as plenty as possible. We performed the 30 rounds of evaluation of *netperf*. When the number of tenants per host varies during the attack, the OvS-based firewall achieves higher sustained TCP throughput than the Linux Bridge, proving that OvS is the optimal switching system. As the number of nodes increases, memory use in all three situations is normalized to be equivalent to or less than that of legacy LB.

As illustrated in Fig. 21a, all three techniques (Legacy LB, Native OvS, and OpenStackDP) utilize around the same amount of CPU/Memory. In Fig. 21b, 4 clients send traffic to a single server, resulting in a combined TCP throughput of over 8.6 Gbps.

As shown in Table 5, the amount of PACKET_IN packets on the control channel ranges from 500 and 5000 during the attacks. With NO-IDS, the average round-trip-time (RTT) during a DDoS attack is more significant than 100 seconds; packet drop is 100%. ii) With IDS: a) For attacks of brief duration, the RTT is impacted. b) attacks with a longer duration—the RTT is typical, and there is no packet loss. Table 6 shows that SDN mechanisms require more memory than Legacy LB techniques, owing to the overhead of the SDN/OVS OpenFlow pipeline.

Table 5 Latencies and packet loss

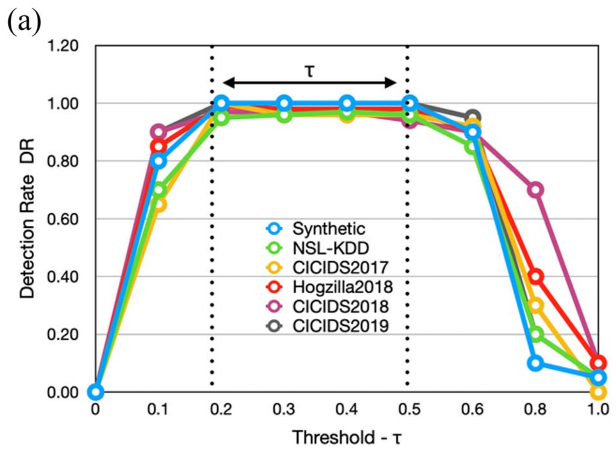
Metric	Flow Table Persistence			
	60 s	120 s	600 s	1500 s
Round Trip	108.67 ms	54.34 ms	5.27 ms	3.90 ms
Packets Lost	4%	2%	0	0

Dynamic threshold vs. detection rate

For IDS, the success of the clustering technique in terms of precision and accuracy largely depends on threshold τ, as shown in Fig. 22a. A synthetic dataset and a few

Table 6 Memory utilization

Nodes	Linux Bridge	Open vSwitch	OpenStackDP
2	9 .3%	22%	23.2%
4	15.8%	26.8%	30.2%
6	27.2%	35%	36.3%
8	42%	49.4%	59.8%



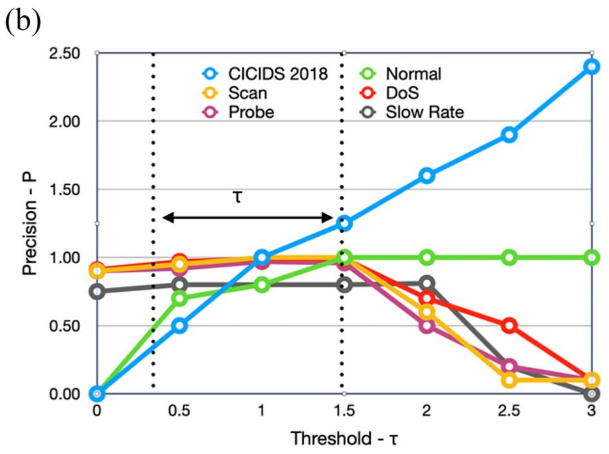


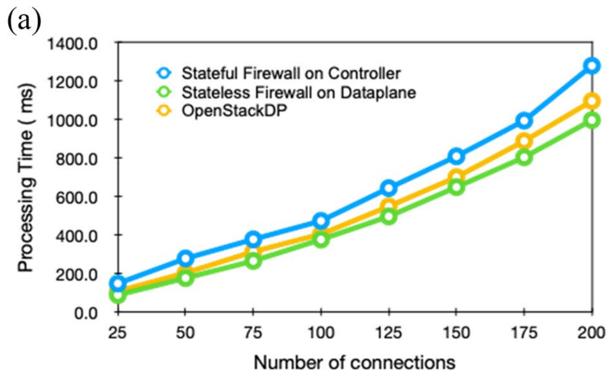
Fig. 22 For different threshold values, **a** Detection rate **b** Precision

real-world datasets were used to study the influence of thresholds (e.g., NSL-KDD [57], CICIDS2017 [58], Hogzilla [59], CICIDS2018 [60], CICIDS2019 [61]). A comprehensive dataset and traffic emulation were conducted to recreate the representative traffic of a real-world network [62] and to be in parity with the latest trends. The threshold τ is dependent on the traffic patterns and the dataset generated. We measure the variations in detection rate and precision for different attack classes (Scan, DoS, DDoS, Probe, R2L, Slow-rate, U2R). Using vertical dashed lines, Fig. 22 depicts the range of possible threshold values for each attack type and for normal data items to achieve good outcomes (b). Threshold values of (0.90 to 2.7) for normal records and (0.39 to 1.08) for attack records were effective in our testing. This estimation helps choose the threshold τ for experiments.

Stateful dataplane performance

We developed and tested a state-based fine-grained firewall. To establish a connection, the switch uses the data packets' state information and the state table's data to make decisions (conformance to firewall policies and updating the state table records). The rule is updated if an internal host begins or ends a connection. This influences TCP connection times. We contrast our strategy with stateless and stateful firewalls regarding the SDN controller. Stateless firewalls allow communication between specific internal and external network addresses. The experiment is set up with a switch and an external Web server. The number of concurrent connections is varied from 50 to 200 per second. We evaluate data for two reasons. The first is the firewall's performance vs. a controller with and without a firewall. In this scenario, we examine the firewall's scalability by tracking the progression of packet processing time zones with concurrent connections. In the second scenario, we want to see how long the Firewall takes to process the packets and connections.

Figure 23a displays the total *connection time* vs. *internal host count*. The findings reveal that our method's connection time is slightly longer than stateless firewalls but much less than a stateful firewall on the controller.



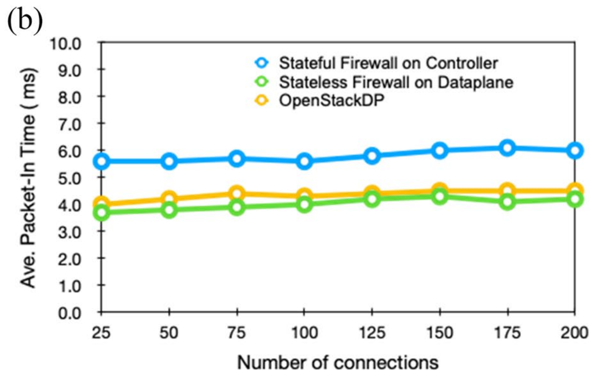


Fig. 23 Dataplane Scalability. **a** Processing time **b** Packet-In delay

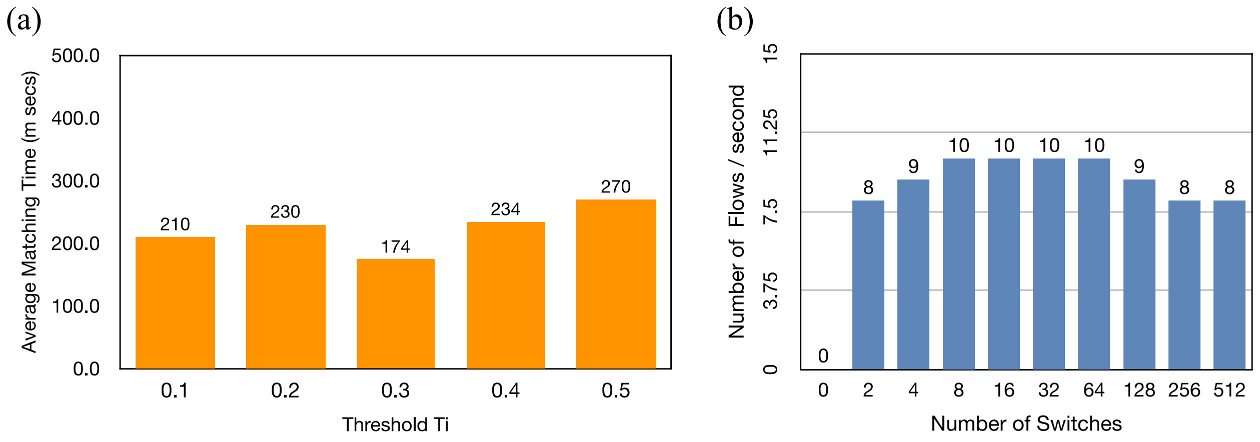


Fig. 24 **a** Firewall Filter Matching Speed **b** Dataplane Scaling

To accept/reject packets from foreign hosts, the SDN must update its flow table rules and connection tables. As the number of internal hosts increases, so does the average *Packet-In* time. Studies in Fig. 23b show a very steady *Packet-In* time for data plane firewalls, in the range of 3.7 to 4.5 ms, and for the control-plane firewall, it takes about 6 ms. Due to the optimal stateful connection table maintained in the data plane switches, our stateful firewall scheme has negligible overhead for *Packet-In* communication.

Four different threshold values and domains are used to accept traffic to hit 80% of the 100 firewall rules. The first and second domains hit their respective rules occasionally, while domains three and four deliver consistent packets to maintain a distinct rule weight at different times. The Average Detection/Matching Time $\left(AMT = \sum_{i=1}^{n} W_i d_i\right)$ is calculated by varying the thresholds. W_i denotes to $Rule_i$ weight and d_i the Flow rule order. We can determine the effectiveness of the number of invocations for each threshold. As seen in Fig. 24a, the criterion of 0.3 has the best AMT and requires the least time in a firewall.

Flow table scalability

When a DDoS attack occurs, switches send "*Packet-In*" messages to the controller through OpenFlow, and the controller sends "*Flow Modification*" messages to switches. To process these "new flows," the controller must use its computing and networking resource, hence a critical SDN metric. The results are shown in Fig. 24b OpenStackDP sustains the "flow installation speed" even while the network is under attack. The classical SDN stack crashed, whereas the OpenStackDP data plane handled up to half-million flow entries simultaneously, and the controller could never be saturated.

Cloud network configuration and neutron node

The main package (driver module) *networking-odl* acts as a redirector/proxying/routing component interfacing between the SDN abstractions and OpenStack security/policy configurations. This module includes functions to create new networks and establish security group for tenants. This experiment measures network configuration,

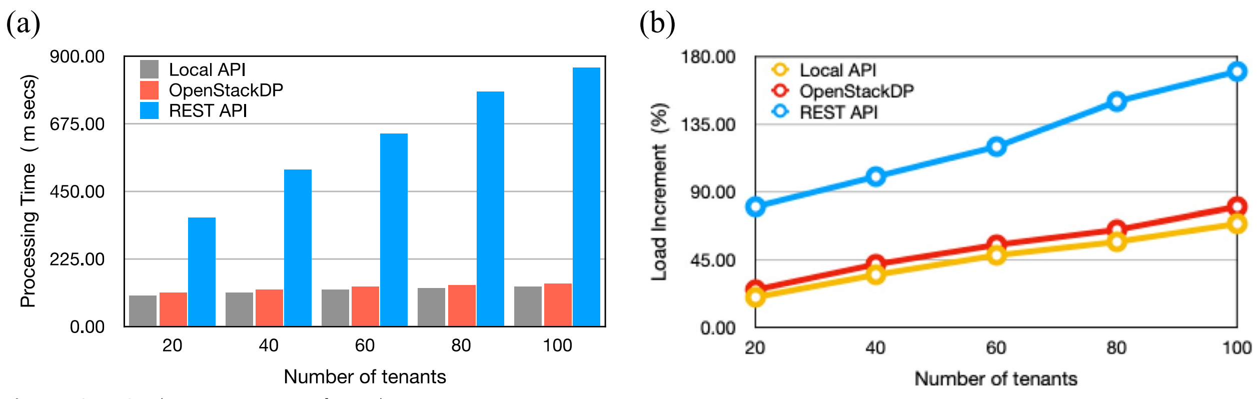


Fig. 25 OpenStackDP API. **a** Latency, **b** Load Increment

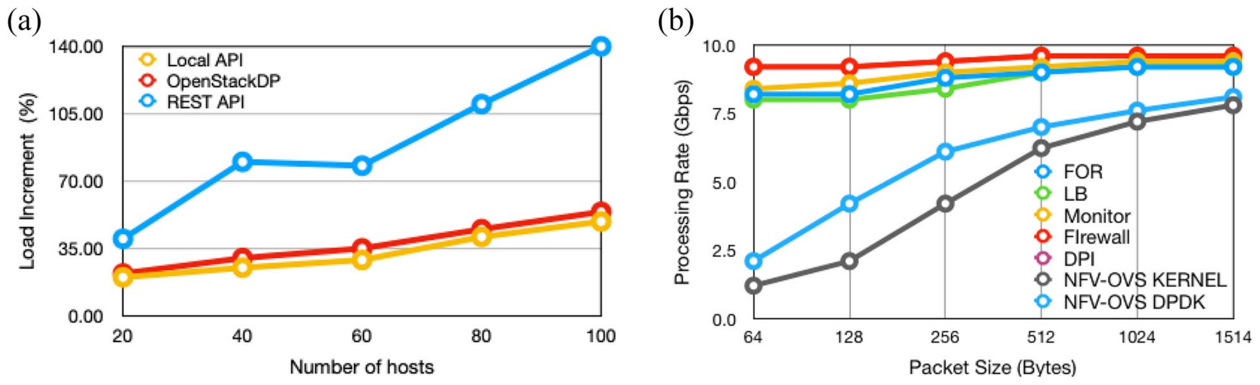


Fig. 26 **a** OpenStackDP API Scaling **b** NFV Throughput

policies processing time, and load increment due to the framework overhead. Calling local APIs to create and configure a virtual subnet takes time as the application run in the SDN control plane. In our framework, it is time for OpenStackDP to process policy requests from cloud tenants and OpenStack API. The processing overhead for REST API calls refers to the SDN controller incurred with handling the tenants' service requests. When configuring virtual subnets, the cloud data center platform has three variables. The three variables are *total tenants, hosts, and virtual machines per subnet*. In an experiment, three variables can change simultaneously. We fix two variables for qualitative research and adjust another for testing. We compare *OpenStackDP* API with *local* API and *REST* API. The number of tenants considerably impacts the total system processing time. As shown in Fig. 25a, the latency of OpenStackDP almost follows the local API but is lesser than REST API. Figure 25b demonstrates that calling the REST APIs increases the demand on the OpenStackDP system by around 40% to 60%. Using OpenStackDP increases system load about the same as contacting local APIs. The findings show that OpenStackDP has greater processing performance than REST APIs. The processing latencies and load increments of OpenStackDP are shorter than the processing time of calling REST API when the number of tenants or virtual subnets for each tenant is increasing. Figure 26a shows that OpenStackDP scales well compared to other API modes.

We compared OpenStackDP with the conventional NFV hosting in compute node VMs. We generate a burst of 64–1514 Bytes TCP packets through one *ingress* port and *egress* through the second port on the gateway. For realistic comparison, we tested with different NFs, from simple forwarding and monitoring to heavy NFs such as Deep-Packet Inspection (DPI).

Throughput

OpenStackDP reached 96% available bandwidth for all packet cases except for composite SFCs (heavy NF being the bottleneck). The graph in Fig. 26b shows a peak rate of 7.6 Gbps for NFV in the OVS-DPDK/Kernel. The classic OVS involved detours and asynchronous process cost, and hence it achieved a lower bandwidth rate.

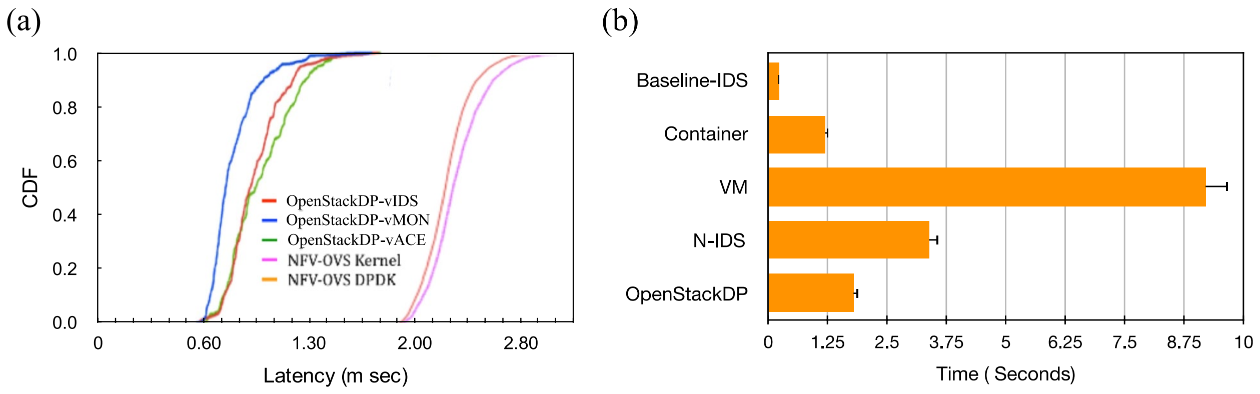


Fig. 27 **a** Latency Curves **b** IDS Launching Speed

Latency

CDF (Cumulative Distribution Function) graph in Fig. 27a shows that more than 98% of packets incur less than 1.30 msecs and average 0.9 msecs delay through the OpenStackDP modules (vMon, vIDS, vACE). The OVS-Kernel switch incurs 2.16 msecs due to detouring and I/O overhead.

Firewall IDS service launch time

This measures the time it takes to start up the IDS application, Container, and VM and then run it. In OpenStackDP, this is the time it takes from sending a user request to starting the IDS service. The averages of each test are displayed in Fig. 27b. The basic scenario has a 0.24 s average launch time due to no virtualization or connectivity costs. The container scenario's average launch time is 1.3 s, 11 times the baseline. Because the N-IDS method requires numerous steps to create an IDS, the launching time is longer, 3.4 s on average. OpenStackDP requires only 1.7 s (approximately 15%) to launch, making it more "elastic." With OpenStackDP, processing and network communication are faster than opening a container.

Security incident response performance

In this experiment, we assess OpenStackDP security monitoring/detection speed and resource utilization performance in an OpenStack cloud with a specific network traffic load. OpenStackDP can easily create, launch, change, and destroy IDS services. We use the "*Mid-Atlantic Collegiate Cyber Defense Competition* (MACCDC)" network trace to mimic the actual security incident and test OpenStackDP applications ("flow-*analyzer and dynamic NF handler*") in terms of response and speed. Each test starts with a traffic sender and recipient VM. *tcpreplay* sends network traffic from the source VM to the receiving VM.

We compare the following scenarios:

- *Baseline-IDS*: Sniffer service running on a computing node.
- *Container-IDS*: Sniffer service running inside a Docker container on a computing node.
- *Virtual Machine*: Sniffer service running as a VM in the tenant network by installing a tap/mirroring on the switch.
- *N-IDS Service*: IDS Sniffer service runs on the network node, mirroring configured to a particular Neutron port.

Dynamic DDoS detection and mitigation

We created a VNF chain for the dynamic firewall pipeline with a rate-limiting filter (threshold) to/from the same domain (potentially a DDoS campaign). Figure 28a shows the traffic network throughput. The attack starts around 10 seconds into the test, and the application detects and responds in 32 seconds, restoring regular service. While the recovery time varies slightly amongst attacks (e.g., TCP, UDP, ICMP), the total recovery time is still reasonable. Figure 28b shows the testbed traffic trace. We started the attack at @1 Gbps and gradually built up to affect the network substantially. The DDoS Detector NF records anomalous flows in packets traveling across the chain. When the traffic hits (4.5 Gbps), the alarm is raised to label it suspicious and redirect it for further scrubbing. An SFC Sandbox VNF to handle suspicious packets is bootstrapped in under 3 seconds. The outbound traffic rate returns at time#29 s (Sandbox deleted the attack), even as inbound traffic grows.

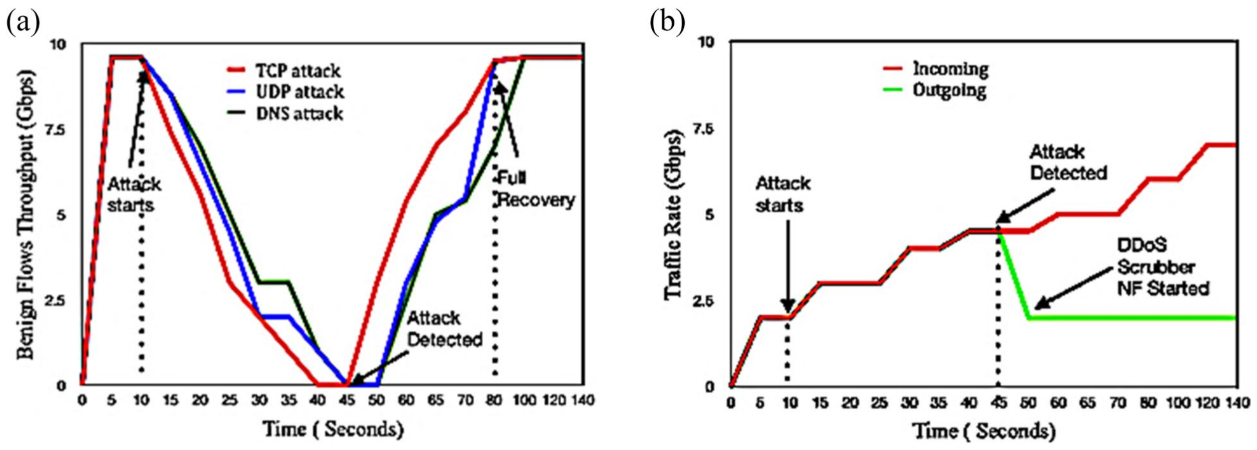


Fig. 28 **a** Dynamic DDoS detection **b** Dynamic NF insertion in SFC

Table 7 Unprotected Ops map

Service	Processing Time (Seconds)		Overhead (%)
	Liberty	OpenStackDP	
Nova	652	680	4.2
Glance	229	249	8
Neutron	230	278	20.8
Cinder	136	140	2.9
Heat	292	300	2.7
Ceilometer	618	628	1.6

Access control performance

According to the research [40], not all OpenStack functions are now under policy enforcement. One hundred five out of 371 operations are uncontrolled. Among the 64 open interface functions, 41 are vital and unprotected by policy, such as modifying arbitrary system or network session metadata. Without authorization, 19 out of 89 functions can be invoked. This puts the system at risk because any cloud user/ adversary can identify unprotected calls and gain entry through those functions without authorization. It allows cloud providers to assess overall security by reviewing management communication calls. Unregistered operations are denied by default. As a result, all accesses are regulated inside the networking node/segment or dataplane of the OpenStack SDN platform.

We benchmarked with the OpenStack *Tempest* [63] and contrasted it with the OpenStack *"Liberty"* [14] platform. Table 7 shows that our security framework *overhead time* is reasonable, given the average cost of 9.7%—the vACE-DTARS app communication delays cause this overhead. At the Glance, the worst case was 8%. This outcome is predicted given the relative speed of each glance call compared to permission verification. The vACE module does not influence the ceilometer because it requires less access control. Compared to the latency through a major public network, the overhead per function call is about 127 ms. We tested the OpenStackDP capacity to filter packets at each protocol layer with the Tempest benchmark, and the results are shown in Table 8. The OpenStackDP service examines the packet transfer using a set of *lightweight* filters. In this test, we observe the advanced filtering capabilities (L2-L7 rules). The service will measure the packet transmission speed between the two networks with the OpenStackDP NIDS Gateway. The filtering criteria will be random without disrupting the TCP connections between the endpoints. The test launched 1000×1024 Mbyte packets and transferred them over TCP using the Linux programs dd and *netcat*. The throughput and the end-to-end time is taken to enforce the rules in the firewall embedded in the data plane are sustained close to the *wire-rate* 9.8 Gbps.

Table 8 Tempest benchmark

Number of Filter Rules				Result	
L2	L3	L4	L7	Time (sec)	Bandwidth (Gb/s)
0	0	0	0	18.09	9.8
100	0	0	0	20.08	9.7
0	100	0	0	21.7	9.6
0	0	100	0	22.2	9.6
0	0	0	100	80.9	9.1
100	100	100	100	110.4	8.8

Comparison with related security solutions

In this experiment, we compare OpenStackDP with five related solutions for OpenStack Cloud and analyze the ability to defend against attacks. All the key metrics, such as network bandwidth, connectivity loss, flow table persistence, packet losses, and latencies, are examined and discussed. We have selected some research works focusing on using SDN technology in the cloud data center with an OpenStack environment [34, 39, 43, 45, 46]. Most of the prior works were based on SDN Controller-based solutions. Our solution is one of the few research projects that exploited the high-speed dataplane and redesigned the network node of the OpenStack architecture. We implemented these solutions (as application modules in the OpenDayLight SDN Controller), from the compared works, in our testbed with the same environmental set-up according to the above observations to retain their original novelties and functionalities. We have implemented the solutions optimally closest to the description in the original paper and with appropriate assumptions and normalizing for the common testbed (hardware/simulator platform).

The strategy of our comparison experiments is:- a) Each experiment evaluates certain performance criteria or metrics. b) Not all these related works can be tested as the scope of their solution is limited, not applicable, or doesn't qualify. So, all these 5 works or only a subset of works are selected to run those tests. The data is normalized and plotted for comparison. e.g., in Threat model validation section, to test the Threat model with the STRIDE approach, only two works, SecSDN-Cloud [43] and FWaaS [39], are compared with the OpenStackDP. These works were chosen because of the completeness of their solution under all major aspects. We have organized and presented the performance in Comparison with related security solutions, Security analysis, and Threat model validation sections with the above assumptions.

- **PDSDN** [46]: *Policy-Driven-SDN-controller* employs a batch processing network scheme. A control plane application is implemented to process the policies according to the user's permissions and operations priority. We implemented a policy assigning module in the SDN Open Daylight Controller, a policy parsing/conflict resolution method, and a policy execution module. The authors of PDSDN mainly tested the performance by creating some virtual subnets. We also compared our scheme with PDSDN regarding system processing time and load increment.
- **SecSDN-Cloud** [43]: *Secure-SDN-Cloud* solves flow table reliability problems, controller saturation, and side-channel attacks. The security mechanisms are implemented in the SDN Controller. The authors of [43] implemented the SecSDN-cloud in the OMNeT++ simulator and evaluated its performance in terms of packet loss, end-to-end delay, throughput, latency, and bandwidth. We adapted their SecSDN cuckoo search algorithm that involves multiple controllers. To guard against this attack type, SecSDN-cloud employs a third-party cloud-monitoring server to execute the EGA-CS algorithm that assigns controllers to switches. We implemented the scheme in the Mininet simulator and compared it with OpenStackDP in the simulation environment. We mainly tested the ability to resist three attack types: flow table overloading, control plane saturation, and Byzantine attacks.
- **ECSD** [45]: *Enhanced-Compromised Switch Detection* is an SDN scheme that applies a multivariate time-series technique for detecting anomalies in traffic passing through switches. This framework aims to detect a compromised switch in SDN architecture synchronized with an OpenStack controller, and it is implemented as an extension in the Control Plane. We adapted their Defense Scheme (ECSD), coded an application module in our SDN environment, and integrated it into the OpenStack network environment to detect compromised switch attacks. To simulate the compromised switch attacks, we manipulated the Open Daylight SDN controller to perform the attacks mentioned in that paper.
- **FWaaS** [39]: *Firewall-as-a-Service* is a Stateful Firewall design. The matching table of the data plane is modified to add state detection logic and a provisioning application in the control plane manages the deployed firewall services. As the authors implement a prototype using P4, we adapted the algorithm of stateful firewall filtering to our environment, implemented in Python. In the experiment, we utilized the Mininet to deploy the virtual network on the OpenStackDP platform. In this case study, we compared the stateful firewall implementations to control the internal network to access the external network.
- **NIDSaaS** [34]: *"Network-Intrusion-Detection-System-as-a-Service"* is designed to run on the designated host(s) directly. The system may generate and remove IDS services dynamically and update rule sets on demand in an OpenStack cloud. The NIDSaaS prototype consists of a user client, a service plugin, a service plugin agent, and a Snort-driven NIDS provider. We implemented their NIDS service in our testbed on the OpenStack network node and sniff on a designated Neutron port to which the target network traffic is mirrored.

Holding time on flow table (Fig. 29a)

We believe holding time to be a crucial statistic for evaluating the resistance of the OpenStackDP to "flow table overloading attacks." The switching design that displays long durations of entries in flow tables even when attack rates are increased suggests a more secure SDN. Consumption of southbound channel bandwidth relates to

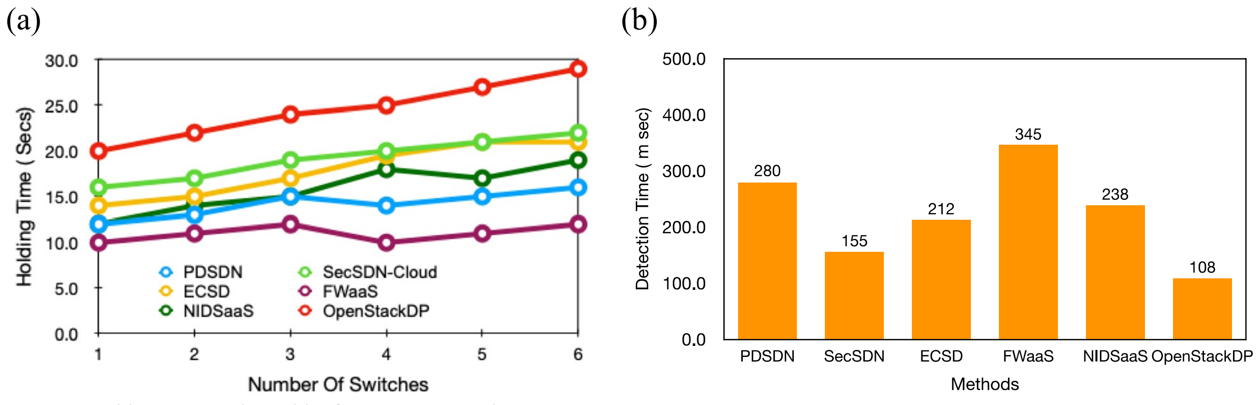
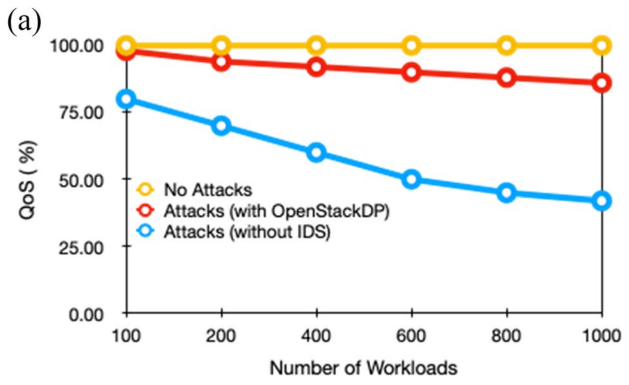


Fig. 29 **a** Holding Time in Flow tables, **b** Detection Speed



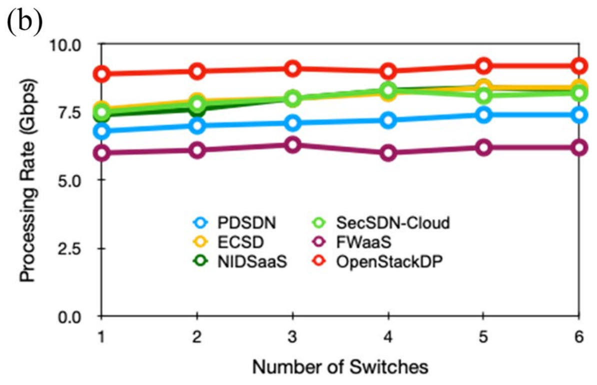


Fig. 30 **a** QoS with workloads **b** Throughput

the attack, and the SDN controller should spend as few resources on network ports as possible.

Detection speed (Fig. 29b)

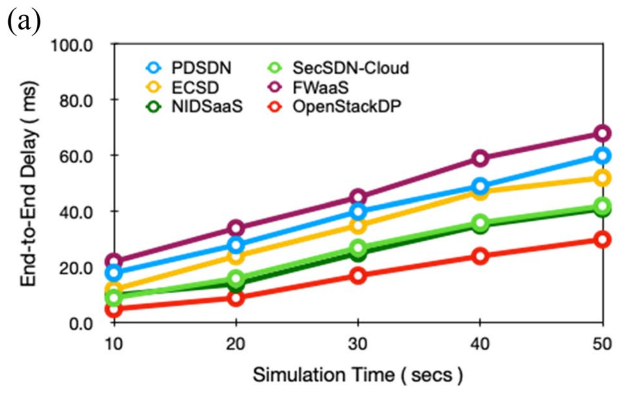
We ran a series of compromised switch assaults to test the *detection speed* of a new attack. The *ECSD* and *SecSDN-Cloud* systems trigger an alarm approximately 0.15 seconds slower than OpenStackDP. The reason is that OpenStackDP uses data plane modules to accelerate its performance. As a result of their intricate design and numerous policy decision-making steps, FWaaS and NIDSaaS systems raise detection alarms slowly. In summary, the detection time is highly dependent on a scheme's lightweight or heavyweight nature, and OpenStackDP outperforms the other alternatives.

Effectiveness of the QoS

We examine how Security affects QoS. The effectiveness of QoS is described in terms of essential characteristics: Fig. 30a workload management, Fig. 30b throughput, Fig. 31a end-to-end delay, and Fig. 31b packet loss. The *response time* represents the time required to finish the workload and produce the required result. We tested three scenarios to assess the response time. (1) No Attack: The system completes the workloads in the required time with normal traffic. (2) Attacks without IDS occur, compromising workload response time and QoS. (3) Attack With IDS: Attacks affect workload response time, but IDS enhances QoS by recognizing and neutralizing them. We counted the packets dropped by the switches, which gave the ratio of benign/malign packet loss due to the attack. OpenStackDP detects the attack pattern and enforces mitigation policies quickly, and other solutions discard fewer malicious packets because they are less efficient at detecting anomalies.

Intrusion detection

To illustrate our outlier detection model, we picked the packet/byte attribute of the flows and replayed the *trace files* with attacks (mixed rate). The points observed outside the boundary conditions are marked as anomalies, and further classification will determine–spikes of benign traffic or attack packets. The comparison of key performance metrics across the solutions is presented in Fig. 32. We



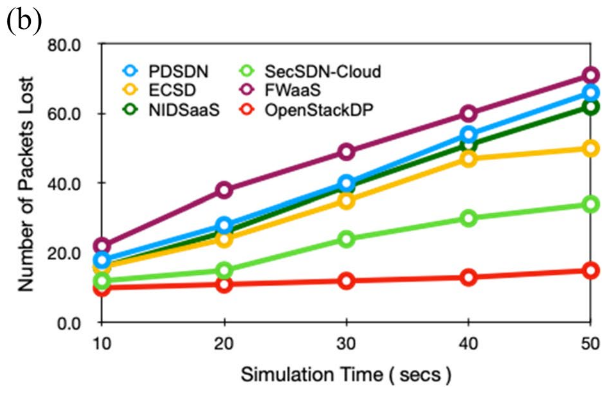


Fig. 31 **a** End-to-end delay **b** Packet loss

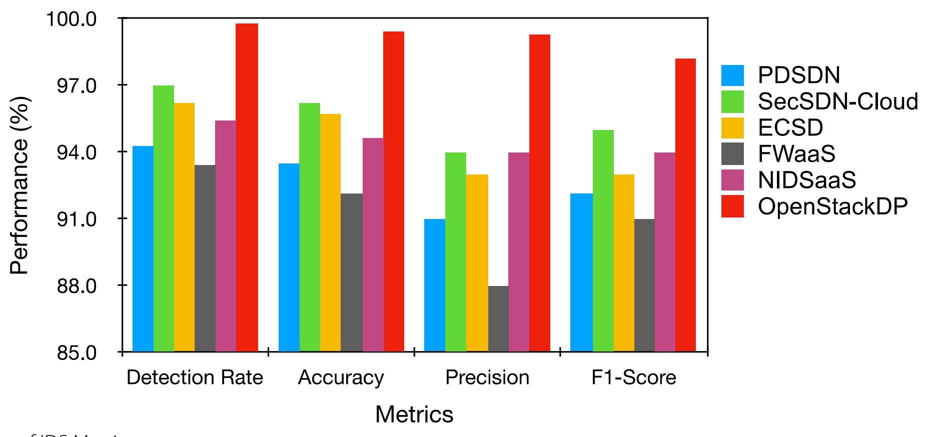


Fig. 32 Comparison of IDS Metrics

Table 9 Classification accuracy

Traffic Type	Dataset Count	Correct	Incorrect	Accuracy (%)
Benign	16,72,234	16,24,074	48,160	97.12
HTTP Flood	18,28,545	17,77,894	50,651	97.23
Slow Read	4,56,567	4,46,933	9634	97.89
Port Scan	8,23,456	8,20,739	2717	99.67
DoS Flood	24,45,678	24,07,770	37,908	98.45

consistently surpass the competition in terms of *detection rate* and *accuracy*, with 99.81% and 99.40%, respectively, being the highest. The *F1-score* is helpful for thoroughly evaluating OpenStackDP performance because it considers both False Positives and False Negatives. Table 9 illustrates the OpenStackDP's detection accuracy.

Security analysis

The OpenStackDP system's monitoring mechanisms detect malware, network-centric attacks, and malicious traffic. As a result of the cross-plane design, response times vary depending on the type of assault being intercepted. Through experiments described in this section, this paper demonstrates the efficacy of OpenStackDP by classifying it through phases in distinct types of attacks (covering about 90% of all known network attacks). In addition to these situations, OpenStackDP's protection capabilities extend far beyond what is covered in Table 10. These examples show how the system performs under various traffic scenarios.

List of attacks and countermeasures

Table 11 lists the most common attacks in the Cloud infrastructure and the corresponding mitigation solution implemented in the OpenStackDP solution.

We track the amount of time fraudulent flows spend in the OpenStackDP pipeline before being mitigated ("Scan, DoS, DDoS, Slow-rate, application-level"). The attacks are recognized and remediated at a certain point in the OpenStackDP detection pipeline (as illustrated in Fig. 33).

Figure 34 shows how the detection speed varies depending on the layer (switch/controller) and IDS

Table 10 Defense capabilities against the various classes of attacks

Attack Type	Attack Identification	Field of Interest	Detection Method	Mitigation Method
Scanning	Increase in Attacker, Host A, ratio to target addresses	IP Address Port	Level 1 (data-plane) Level 2 (NFV)	Block/Drop
DoS	Volume of traffic flows from/to a single IP exceeds a threshold	IP Address TTL	Level 1 (data-plane) Level 2 (NFV)	Block/Drop
DDoS	volume of traffic from multiple IPs targeting exceeds a threshold	IP Address TTL	Level 1 (data-plane) Level 2 (NFV) Level3-(control-plane)	Block/Drop
Slow Rate	opens a great number of half-open connections and initiates request with no replies	IP Address Port	Level 1 (data-plane) Level 2 (NFV) Level3-(control-plane)	Block/Drop
App layer	correlation/asymmetric volume between Request/Response	Port Protocol	Level3-(control-plane) Level 4- (application)	Block/Drop/Remediate

Table 11 List of attacks and countermeasures

Components	Attacks	Countermeasures
Tenants and User Applications	Brute Force	Intrusion Prevention System
	Privilege escalation	Virtualization of Services
	Insider Attacks	Authentication and Security Group
	Policy Violation	Global View and Access Control
Gateway and Internal Network	Injection Attack	Policy Validation and Enforcement
	MITM	Defense Mechanism
	DNS Poisoning	DNS Proxy in the Switch
	Reply Attacks	Flow analysis and Dynamic Rules
	Wormhole	Port Monitoring
	Flooding	Rate limiting and Proxy Firewall
Cloud Servers and Controller Devices	SQL Injection	Input Validation
	Application Persistent Attacks	Packet History Analysis and Stateful Firewall
	Weak Authentication	2-level Authentication
	DDoS	SDN Global View, Flow Analysis & Dynamic Rule Updating
	Backdoors and Exploits	Anomaly Detection system
	Malicious Application	Anti-Virus software modules

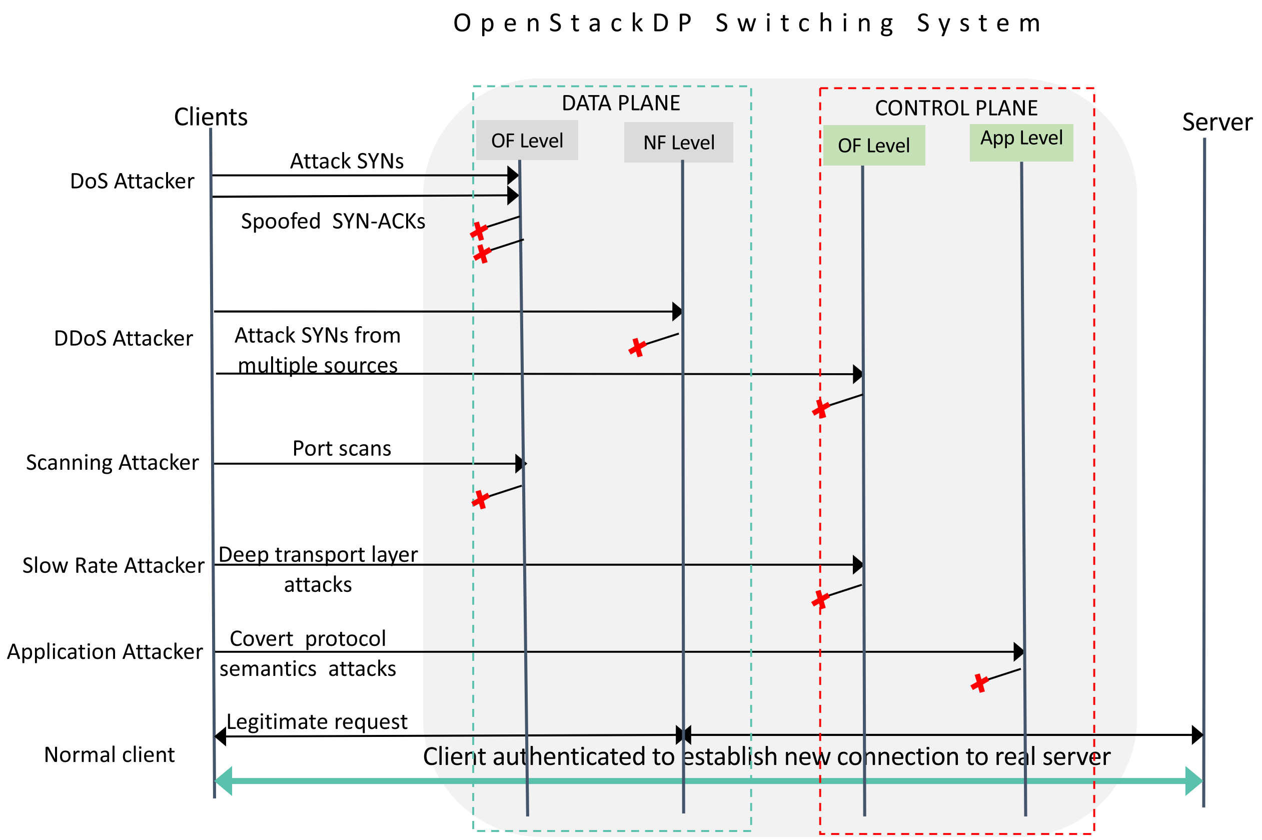


Fig. 33 Depth traveled by attack traffic in the pipeline

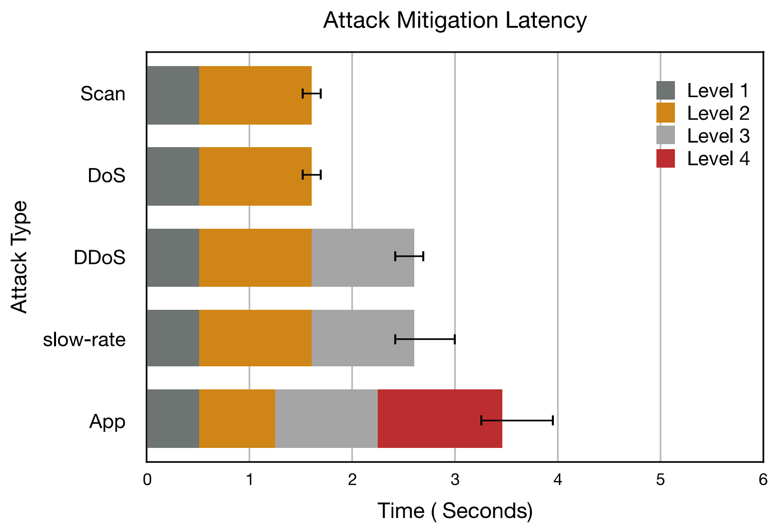


Fig. 34 Detection Times for different classes of attacks

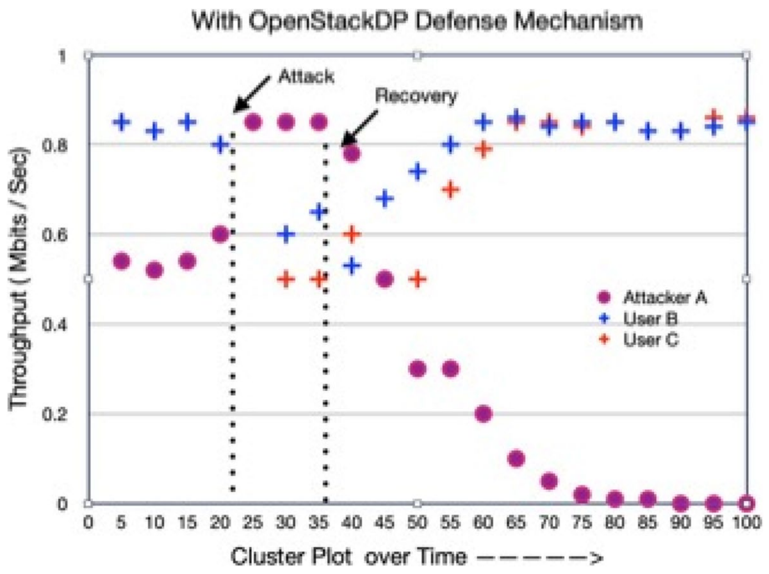


Fig. 35 Throughput Performance during Insider Attacks

Table 12 Components of DFD

Item	Symbol
Process	Circle
Data Flow	Arrow
Data Sore	Two Parallel Horizontal line
Interactors	Rectangle
Trust boundary	Dotted line

classifier overhead (Signature/Anomaly). To detect and mitigate or remediate an application attack, we needed to conduct protocol correlation/analytics, which adds overhead at all levels of the OpenStackDP analytics pipeline.

Figure 35 plots the data points (Bandwidth/Throughput parameter) measured over time, during normal traffic, and when an insider attack began from one tenant (User-A Attacker) to other tenants (User B, User C) in the cloud network and ended. With the OpenStackDP, the throughput for the legitimate users is restored after a brief drop for 8–12 secs.

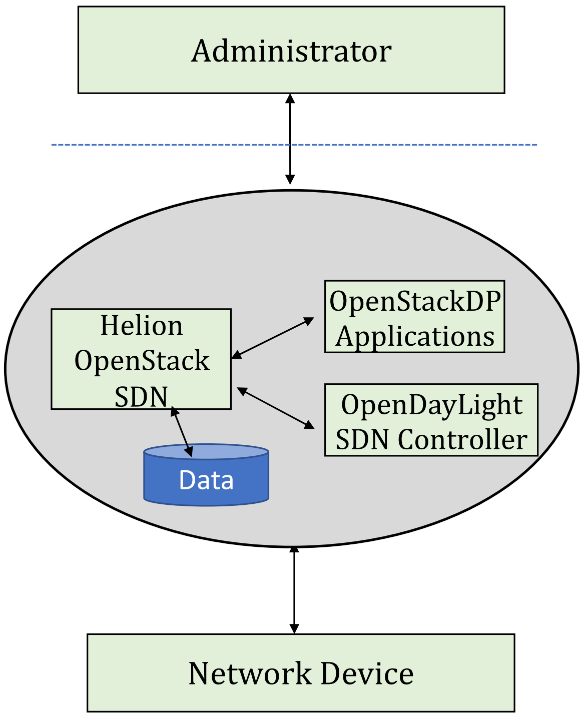


Fig. 36 DFD for OpenStackDP Framework

Threat model validation

Each component should be assessed against a specific threat type to validate and analyze a DFD ("Data Flow Diagram") to detect and mitigate threats. Table 12 lists the five different types of DFD components. Only a small percentage of STRIDE's threat categories affect each DFD's components [16]. For example, users are only susceptible to spoofing and repudiation risks when interacting with each other. The STRIDE approach has analyzed many SDN protocols, architectures, and applications [64, 65]. Two SDN cloud networking platforms, **SecSDN-Cloud** [43] and **FWaaS** [39] are compared with the OpenStackDP. These works were chosen because of the completeness of their solution under all major aspects, closest to our model, and are recent. Using the definitions of the data flows, processes, and DFDs, **SecSDN-Cloud** and **FWaaS**, and OpenStackDP are analyzed using the STRIDE [50] framework. The DFD of the OpenStackDP is shown in Fig. 36. In this case, the OpenStackDP SDN apps and OpenDayLight controller are considered one process. Both the network devices transmitting and receiving data and the administrator using SDN applications are participants in the DFD. The security analysis should consider the communication flow between the administrator, Helion, the SDN controller, and the network device. Hardware and SDN controllers

Table 13 Security analysis for the components of OpenStackDP

Type	Component	Threat					
		S	T	R	I	D	E
Process	OpenStackDP (DP)	✓	✓	✓	✓	✓	*
Data flows	(O) ↔(DP)		✓		✓	✓	
Data flows	(DP) ↔ (N)		✓		✓	✓	
Interactors	Administrator (O)	*		✓			
Interactors	Network Devices (N)	✓		✓	✓		
Data Store	OpenStack db		✓		✓	✓	

*Denotes threat can be mitigated by suggested methods, ✓ denotes threat can be mitigated as architecture provides countermeasures to mitigate

Table 14 Security analysis of SecSDN-Cloud [43]

Type	Component	Threat					
		S	T	R	I	D	E
Process	SecSDN	*	*	*	*	✓	*
Data flows	Network Control Apps ↔SecSDN		*		*	*	
Data flows	SecSDN ↔ (N)		✓		✓	✓	
Interactors	Network ↔ Controller Applications	*		-			
Interactors	Network Devices (N)	✓		✓			
Data Store	Global Network View		*		✓	*	
Data Store	Network runtime		*		✓	*	

*Denotes threat can be mitigated by suggested methods, ✓ denotes threat can be mitigated as architecture provides countermeasures to mitigate, - denotes out of scope

Table 15 Summary of the security analysis for the FWaaS [39]

Type	Component	Threat					
		S	T	R	I	D	E
Process	FWaaS	*	*	*	*	*	✓
Data flows	User ↔FWaaS		*		✓	*	
Data flows	FWaaS ↔ (N)		*		*	*	
Interactors	User	✓		-			
Interactors	Network Devices (N)	✓		✓	-		
Data Store	Cloud Monitor (CM)		*		*	*	
Data Store	Network runtime		*		*	✓	

*Denotes threat can be mitigated by suggested methods, ✓ denotes threat can be mitigated as architecture provides countermeasures to mitigate, - denotes out of scope

are separated by a trust boundary, while a trust boundary separates OpenStackDP and the administrator. The STRIDE model considers the OpenStack components, SDN Controller and Dataplane Switches, OpenStackDP applications, and corresponding threats and mitigation methods.

The following tables show the summary of the security evaluation of the **OpenStackDP** (Table 13), **SecSDN-Cloud** (Table 14), and **FWaaS** (Table 15).

SecSDN-Cloud and FWaaS both allow middlebox interposition in cloud networks (DPI) and vulnerable to security risks. SecSDN-Cloud employs the Libvert API for managing the network components and control. OpenStackDP supports numerous security capabilities that can be utilized with current applications and those built into the protocol. It is up to the customer to configure the network devices that will allow OpenStack's Helion services to be integrated into an existing data center infrastructure. It also defines firewall rules at the deployment's perimeter (to guard against external misuse) and router rules within the OpenStackDP deployment to defend against insider threats or administration errors and misconfigurations.

Result discussion

OpenStack Data plane research has been limited to the control plane [26, 37, 44] and forwarding problems of the data plane switches. Using these insights, we investigated the existing threat models for SDN-based OpenStack platforms and the data plane-to-control plane trade-off. Because virtual switches like OvS are widely used in cloud operating systems. In contrast to kernel-based countermeasures (using group security), our measurements show that user-space countermeasures have no performance overheads. We summarize some key findings in Table 16, which proves the practicality and effectiveness of applying SDN mechanisms in OpenStack networking architecture, especially OpenStackDP, to detect access violations and network-centric attacks in our practical experimental setup. Simulations and analysis show that the OpenStackDP is superior to those of earlier SDN designs. Detection performance, CPU utilization, improved Quality of Service (QoS), and scalability are some of the key metrics of the evaluation. OpenStackDP accurately detects compromised network components and attacks across most factors and will support future applications/use cases in the Cloud ecosystem.

Light-weight flow-based IDS

Compared to the packet-based method, the flow-based IDS [66, 67] deals with a fraction of the total amount of data that needs to be monitored and processed, optimal storage requirements. Also, the IDS is robust against encrypted payload attacks and with fewer privacy issues. Our approach in OpenStackDP is further optimized since we have extended the core switching layer in the data plane to execute first-stage attack detection and DDoS prediction functions, stateful firewall functions (cached flow rules/action set) instructed by the controller on the switches. This has enabled improvements in our system by reducing control channel traffic and CPU processing overhead at the controller. This scheme also contributes to the speed of detection in the data plane as it is critical not to create a larger bump in the wire speed. As discussed in Performance evaluation section and Table 10, the flow records are generated in the high-speed hardware switch. Therefore, no performance overhead from computational resources occurs in IDS. To reduce the negative effect on the flow sampling over the entire packet stream, the sampling process on only a subset of the packets is considered for flow generating, thus reducing the load on the router resources. Pre-filtering and aggregating flows are offloaded to the TAP/probe device on the hardware switch.

Data sets and fine-tuning the ML-based IDS

Our study explored the behavior and application of multiple DDoS datasets for machine learning in the context of intrusion detection. Selecting datasets for this study was

Table 16 Key findings from this research

ASPECTS EVALUATED	DISCUSSION
SDN Performance Section VI A	We investigated the vulnerabilities of the SDN switches on the cloud and the present common types of attacks in a cloud-based SDN environment in a distributed manner. Using multiple hosts under OpenFlow switches, attackers can disrupt the control plane or learn its behaviors without knowing much information about controller applications. These attacks include DoS, topology poisoning, control channel saturation, flow table flooding, and side-channel attacks. Our proposal includes a mechanism to detect and defend SDN compromised cloud network effectively using dynamic thresholds, which brings about high detection rates, low false-alarm rates, and efficiency in resource consumption. Our detection mechanism does not deteriorate network performance. Moreover, switches do not necessarily isolate or turn off for detection. Instead, the administrator can dynamically launch the detect process when the network is running. Therefore, our detection mechanism is very practical. In attack simulation tests, OpenStackDP architecture, for brief duration, the latencies and response times for the applications are impacted. But with attacks with a longer duration—the connections are normal and there are no packet loss or delay. Table 2 and 3 shows that SDN design require more memory than Legacy Linux Bridge (LB) architecture, owing to the overhead of the SDN/OvS OpenFlow pipeline. As the number of nodes increases, memory use in all three situations is normalized to be equivalent to or less than that of Legacy LB. As illustrated in Figure 21 (a), all three techniques (Legacy LB, Native OvS, and OpenStackDP) utilize around the same amount of CPU/Memory. When the number of tenants per host is varied during attack, the OpenStackDP SDN-based firewall achieves higher sustained TCP throughput than the Linux Bridge, which proves that SDN/OvS is the optimal switching system. In Figure 21 (b), four clients send traffic to a single server, resulting in a combined TCP throughput of over 8.6 Gbps. The OpenStackDP cloud environment is robust against different attack types initiated by users, and it also support high-quality communications.
Stateful Firewall Section VI B	We evaluated the Pros/Cons of the stateful SDN architecture of the data plane and compared our approach to Controller-based stateless and stateful firewalls. Stateless firewalls allow communication between certain internal and external network addresses. Because the rule is modified every time an internal host initiates or completes a connection, the update time on the tables on the switch, affects the TCP connection time. We examine the firewall's scalability by tracking the progression of packet processing time zones with concurrent connection we want to see how long the Firewall takes to process the packets and connections. The findings reveal that our method's connection time is slightly longer than stateless firewalls, but much less than stateful firewall on controller. The average packet-in time for the stateful and stateless firewalls is between 3.7 and 4.5 ms, and for the controller firewall it is around 6 ms. Due to the state information preserved by the data plane, our technique has negligible effect on the average Packet-in time. Our tests to demonstrate the flow table scalability in the stateful dataplane switches showed that that traditional IDS takes longer to add new firewall rules and at around 10,000 drop-rules on the classic-IDS firewall simply becomes a bottleneck in the network. With our OpenStackDP Infrastructure, the controller could handle up to 250K flow-rules at a time, and the controller could never be saturated.
OpenStack Networking Section VI C	We measured the network configuration setup overhead, policies processing and load increment due to the stateful dataplane overhead. In conventional OpenStack cloud, Calling local APIs to create and configure virtual subnet takes time as the application run in the SDN control plane. The number of tenants in the Cloud network has a considerable impact on total system processing time. The findings shows that OpenStackDP application and management interface has better processing performance than REST APIs. The processing latencies and load increments of OpenStackDP is shorter than the processing time of calling REST API all the time when the number of tenants or virtual subnets for each tenant is increases. Figure 24 (c) shows that OpenStackDP scales well compared to other API modes. Calling local APIs to create and configure virtual subnet takes time as the application run in the SDN control plane. In OpenStackDP, the overhead involved to process policy requests from cloud tenants and OpenStack API is considerably less compared to the conventional OpenStack stack.
IDS Placement Section VI D	We evaluated resource consumption and impact of the placement models of the IDS. The baseline case and OpenStackDP consume the least amount of CPU and memory. Since the docker container is lightweight, the container model are about the same as the baseline and OpenStackDP cases in terms of memory usage and consume around 4% more CPU than the baseline and N-IDS cases. In the VM model, it is evident that CPU usage sometimes exceeds 100%. This is attributed to that more than one CPU core are being used by the IDS VM. OpenStackDP framework consumes way less resources, approximately only 7-50% of the CPU time and 9-30% of the memory consumed by VM model, no matter whether the sender and receiver VMs are on the same host or not.
Incident Response Section VI D	We assessed OpenStackDP's performance of security monitoring/detection speed and resource utilization in an OpenStack cloud with a specific network traffic load. This test measures the time it takes to start up the IDS application, Container, VM and then run it. In OpenStackDP, this is the time it takes from sending a user request to fully starting the IDS service. In comparison to VM model, OpenStackDP requires only 1.8s (approximately 16%) to launch, making it more "elastic". With OpenStackDP, processing and network communication are faster than opening a container, but with VMs, an IDS VM must be launched first.
Access Control Section VI E	We benchmarked the network access control operations for the Cloud application setup with tempest [61]. Our architecture is contrasted the OpenStack "Liberty" platform. Table 4 shows that OpenStackDP framework overhead is reasonable given the average cost of 9.7%. This overhead is caused by the vACE-DTARS app communication delays. In Glance, the worst case was 8%. This outcome is predicted given the relative speed of each glance call compared to permission verification. The vACE module has no influence on the ceilometer because it requires less access control. When compared to the latency through a major public network like the Internet, the overhead per function call is about 127ms . The throughput and the end-to-end time taken to enforce the rules in the firewall embedded in the dataplane are sustained close to wire rate 9.8 Gbps.
Detection Efficiency Section VI F	All the multivariate hierarchical clustering techniques not only yield high detection rates, accuracies, and F1-scores but also produce low false-alarm rates, in which OpenStackDP outperforms the other schemes. . In terms of detection rate and accuracy, our unique approach constantly accounts for the highest rates, 99.81% and 99.40%, respectively, decisively outperforming the competition. The F1-score is a useful measure for thoroughly evaluating OpenStackDP performance because it considers both False Positives and False Negatives. Table 6 lists out the results, which shows very good classification accuracy and OpenStackDP achieved the highest F1-score, 98.20%.
Quality-of-Service (QoS) Section VI F	The effectiveness of QoS is described in terms of three important characteristics -throughput, end-to-end delay, and packet loss. We also counted the malicious packets dropped by the hacked switches which gives the ratio of benign/malign packet loss in the network due to the attack. OpenStackDP conducts and enforces policies as soon as the attack pattern is discovered. They discarded fewer malicious packets than the three options above because they are less efficient at detecting anomalies. The response time is a QoS parameter that represents the time required to finish the workload and produce the required result. Defending against attackers is the major reason for the improvements in throughput. The OpenStackDP resolves node failure caused by attackers and sustains connectivity. Thus, improvements to these parameters are reflected by a reduction of the end-to-end delay metric. Figure 28 shows that the OpenStackDP minimizes packet losses because it can mitigate harmful attacks. Packet loss results from packets discarded by a device intended to accept the packet due to overloading. Because buffer and queue sizes are limited, packet overloading occurs when the switch is unable to accept packets that exceed the existing level and can cause packet losses for legitimate users.

challenging due to the shortage of DDoS-specific datasets, despite being one of the most devastating security attacks. Moreover, all datasets chosen are recently dated to ensure that all instances and features are relevant and up to date. The classifier was trained to detect a certain attack category using selected features in comparable ML-based NIDS prior works. Only a small portion of the CICIDS 2017 dataset instances was used to evaluate their system. Conversely, in our research, we use all the instances of the CICIDS2017-2018-2019 datasets. A comprehensive dataset and traffic emulation were conducted to recreate the representative traffic of a real-world network [62] and to be in parity with the latest trends.

Feature analysis when compared to other IDS schemes, which selected about 4 to 6 features for classification; we have utilized about 10 to 16 features and created multi-variate classification, and performed anomaly detection based on traffic flow, SDN protocol behaviors, and so on. Although this increased the complexity, our experiments proved that the features monitored to determine anomalies and detect attacks are the right ones and effective in terms of accuracy and predicting the attacks at the dataplane based on coarse-grained security mechanisms. On the contrary, other proposals classified with a limited feature set slipped certain attacks through the IDS and made the downstream services unresponsive to normal users.

Exploit experiment

In our OpenStack stock version, about 9 information flow vulnerabilities were reported after our installation. Six are present in our deployment, and 3 are not. To conduct the comparison, we did not patch the cloud services in our testbed and tried to exploit them in vanilla OpenStack and OpenStackDP, respectively.

Qualitative analysis

To see how our solution can improve OpenStack security, we performed a qualitative analysis of all 78 vulnerabilities identified in OpenStack networking (software versions from 2016 to 2019). We found that almost 30% of OpenStack vulnerabilities are related to information flow problems already studied in this research, and OpenStackDP systematically mitigates those vulnerabilities.

Limitations and future work

Although this study has successfully demonstrated the significance of the feature dimensionality reduction techniques, which led to better results in terms of several performance metrics and classification speeds for an IDS, it has certain limitations and challenges, summarized as follows.

Data sets

The general inadequacy of static attack datasets also introduces severe impediments to machine learning-based IDS deployment. Models trained with labeled data from a specific domain don't usually transfer or generalize to other domains. For example, data streams obtained from cloud-based Linux services cannot be used to predict cyber-attacks against enterprise Windows endpoints due to the intrinsic differences between the operating environments. This limitation impairs IDS model evolution and the adaptation of machine learning defenses against new and emergent attack techniques. Nonetheless, a well-recognized challenge in custom dataset generation is capturing the multitude of variations and features manifested in real-world scenarios.

Fault tolerance

The key aspect of fault tolerance in our system is the ability of the multi-level and cross-plane scheme to detect a large set of well-known attacks. Our models have been trained to detect the 14 up-to-date and well-known attacks. Moreover, deploying distributed intrusion detection systems in the network can enable fault tolerance.

Model resilience

We achieved an FP rate of 0.001, which may reflect a built-in attack resiliency. Moreover, our models were trained in an offline manner. This ensures that an adversary cannot inject misclassified instances during the training phase. On the contrary, such a case could occur with online-trained models. Therefore, it is essential for the machine learning system employed in intrusion detection to be resilient to adversarial attacks.

Hardening the ML-based solution

It is a common trend that cybersecurity and malware analytics systems employ ML/DL-based AI algorithms to analyze correlations and patterns in the traffic data and detect/classify the attacks. As these methods are widely exploited, sophisticated cyber-criminals devise adversarial ML attacks to breach these NIDS. Our study showed that it is possible to derive an unsupervised anomaly detection method built on boosting meta-learning, which has much better detection performance than regular unsupervised algorithms and is robust to zero-day attacks. This opens an interesting scenario and future works on whether and under which circumstances unsupervised meta-learning may achieve detection performance that can compete with supervised solutions. To such extent, we fore- see a validation process which involves more public datasets in the domain of security, as well as widely used supervised algorithms [68] and deep neural network that suit the analysis of tabular data.

Zero-day attacks

Many IDSs show deficiencies in identifying novel, zero-day attacks. These attacks exploit either new vulnerabilities or known vulnerabilities in novel and different ways and cannot be matched against known signatures. Unsupervised anomaly detection algorithms infer patterns from a training set and discover the underlying structure of the data without reference to known outcomes (i.e., labels are unknown at training time). Instead, they assume that ongoing attacks temporarily alter the values of system indicators concerning their expected values. This way, they learn a model decoupled from labels assigned to data points in the training set and therefore fit the detection of zero-day attacks. It is possible to mimic zero-day occurrence by removing specific attacks from the training set and providing them only during evaluation in the test set. Proper tuning and selection have to be derived according to a precise strategy and its application through appropriate tooling and experimental campaigns. In the future, we would like to improve the robustness of our approach by detecting those types of Zero-Day Attacks whose behaviors are independent of existing attacks. We will perform tests to improve accuracy in the case of multi-class classification. The limitation of the proposed work is that the exact category of high/low volume attack variants is not detected due to its implementation approach, which will be explored in the future.

Detection of attacks exploiting the unpatched known vulnerabilities

In the security domain, supervised ML algorithms are commonly adopted to defend against known threats, and they are embedded into IDSs, which aim to detect attackers that exploit known security breaches or vulnerabilities. We demonstrated in our research the security improvement made by OpenStackDP over the off-the-shelf OpenStack through a system exploit experiment and a qualitative analysis. Our ML-Pipeline aims to mine attack patterns from entire data streams rather than merely to classify individual packets as attacks; mixing benign with malicious activities in the environment does not impair our IDS ability to learn attacker patterns, even in the presence of evasive behaviors. This is practical and reveals that OpenStackDP captures the entirety of the attacker's activity, feeding it back to the classifier. We measured our approach's ability to detect previously unseen, unpatched exploits [69]. In this experiment, CVE-2017-5941 is used as an *n*-day vulnerability for which no patch has been applied. The resulting detection accuracy and precision show that the OpenStackDP pipeline helps the classifier learn attack patterns unavailable at initial deployment to learn exploits for which the classifier was not pre-trained. Therefore, we note that by design, OpenStackDP can mitigate unseen/unpatched vulnerabilities to some extent better than the other comparable IDS.

Attacks through cloud service vulnerabilities

One significant problem with this distributed computing environment is that cloud services are complex software components prone to vulnerabilities. Current cloud platforms assume a flawed design where distributed cloud services fully trust each other. Consequently, a security breach in a single cloud service (due to an unpatched component version or misconfiguration) may allow adversaries to propagate attacks to other cloud services, producing security risks for any user's cloud resources. Our solution OpenStackDP provides defenses to protect communications among services, hosts, nodes, and mechanisms but neither defense prevents a compromised cloud service from misbehaving or propagating attacks. Current defenses against such cloud service vulnerabilities are often limited and we intend to study these internal-attacks systematically and more qualitatively.

Conclusions

Automatic network strategy optimization based on artificial intelligence enables engineers to perceive network traffic conditions comprehensively, data response time, service transmission status, and other information through automated learning and data analysis and automatically tune the network based on network traffic and health status changes. In addition, consistency and integrity checks of the tuning strategy are automatically performed to reduce the chance of errors and the risk of network operation.

We proposed an *OpenStackDP* security framework applicable to SDN-managed OpenStack Cloud infrastructures. Softwarization and virtualization of networking functions and services have changed the status quo of enterprise networks. The NIDS service architecture provides cloud tenants with efficient intrusion detection services. OpenStackDP makes it simple to create, manage, and terminate NIDS services.

Experiments conducted under OpenStack demonstrate that OpenStackDP achieved a detection rate of 97% even while maintaining low latency. The results show the potential of software-defined systems to become more widespread and integrated with OpenStack, advancing this Open-Source platform to move fluidly into a complete integrated stack for virtualized data centers. This research combined SDN, NFV paradigms, and ML/AI techniques to provide an intelligent and efficient data plane and networking fabric for securing the OpenStack architecture. As stated previously, our contributions should directly impact how we secure virtual switches, network components, and SDN systems. Our proposed technique OpenStackDP surpassed the previous schemes in terms of detection performance, accuracy, and CPU use. OpenStackDP consumes more resources, but overall, the trade-off is acceptable. However, we evaluate only payloads that have not been encrypted using the Open SSL/TLS traffic in SDN. We plan to apply this high-performance data-plane-based OpenStackDP scheme to 5G Clouds, which demands ultra-low latency, high bandwidth, better authentication, and granular access control. We have advanced the state-of-the-art in cloud security research, designing software-defined OpenStack architectures and improving the agility and security posture of the Cloud Infrastructure.

Authors' contributions

Prabhakar Krishnan conceived and designed this study. He performed the experiments and wrote this paper. Kurunandan Jain contributed to the design and security analysis of the research. Amjad Aldweesh contributed to the threat modeling and validation, designing new experiments. P. Prabu provided critical feedback and helped shape the analysis and the interpretation of the results. Rajkumar Buyya critically reviewed and supervised the overall research, analysis, and manuscript. All authors discussed and contributed to the final manuscript. The authors read and approved the final manuscript.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declarations

Competing interests

The authors declare no competing interests.

Received: 24 June 2022 Accepted: 13 February 2023
Published online: 28 February 2023

References

1. Singh S, Jeong YS, Park JH (2016) A survey on cloud computing security: issues, threats, and solutions. J Netw Comput Appl 75:200–222
2. Data breach investigations report (2019) https://www.verizon.com/business/resources/reports/2019-data-breach-investigations-report.pdf. Accessed 24 Feb 2023
3. Jararweh Y, Al-Ayyoub M, Benkhelifa E, Vouk M, Rindos A (2016) Software defined cloud: Survey, system and evaluation. Future Generation Computer Systems 58:56–74
4. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J (2008) "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Computer Communication Review 38(2):69–74. https://doi.org/10.1145/1355734.1355746
5. Open Virtual Network Project. https://www.ovn.org/en/. Accessed 24 Feb 2023
6. Cheng Q et al (2018) Guarding the perimeter of cloud-based enterprise networks: an intelligent SDN firewall. In: 2018 IEEE international conference on data science and systems, pp 897–902
7. Using the Cisco Span Port for traffic analysis. https://www.cisco.com/c/en/us/td/docs/switches/metro/me3600x_3800x/software/release/15-4_1_S/configuration/guide/3800x3600xscg/swSPAN.pdf. Accessed 24 Feb 2023
8. OpenvSwitch. http://docs.openvswitch.org. Accessed 24 Feb 2023
9. OpenStack: open-source software for creating private and public clouds. https://www.openstack.org. Accessed 24 Feb 2023
10. OpenStack Networking architecture. WWW document. https://docs.openstack.org/security-guide/networking/architecture.html. Accessed 24 Feb 2023
11. Hui K (2013) Laying cinder block (volumes) in OpenStack. Part 1: the basics. WWW document. Available at: https://cloudarchitectmusings.com/2013/11/18/laying-cinder-block-volumes-in-openstack-part-1-the-basics/. Accessed 24 Feb 2023
12. OpenStack Docs: TAP as a Service (TAPaaS). https://docs.openstack.org/dragonflow/latest/specs/tap_as_a_service.html. Accessed 24 Feb 2023
13. Krishnan P, Achuthan K (2019) CloudSDN: enabling SDN framework for security and threat analytics in cloud networks. In: Lecture notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 276. Springer, Cham. https://doi.org/10.1007/978-3-030-20615-4_12
14. Openstack liberty. https://www.openstack.org/software/liberty. Accessed 24 Feb 2023
15. OpenStack. DevStack. https://docs.openstack.org/devstack/latest/. Accessed 24 Feb 2023
16. Urias VE, Van Leeuwen B, Stout WMS, Lin H. "Applying a Threat model to Cloud Computing" OSTI.gov, Sandia National Laboratories Albuquerque, New Mexico, https://www.osti.gov/servlets/purl/1594657. Accessed 24 Feb 2023
17. Dragonflow- Distributed SDN Controller for OpenStack. https://www.openstack.org/software/releases/ocata/components/dragonflow. Accessed 24 Feb 2023
18. OpenDaylight. https://www.opendaylight.org/. Accessed 24 Feb 2023
19. networking-odl. http://events.linuxfoundation.org/sites/events/files/slides/networking-odl.pdf. Accessed 24 Feb 2023
20. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html?intcmp=sig-blog-ossra1. Accessed 24 Feb 2023
21. Thimmaraju K, Shastry B, Fiebig T, Hetzelt F, Seifert J-P, Feldmann A, Schmid S (2018) Taking Control of SDN-based Cloud Systems via the Data Plane. In Proceedings of the Symposium on SDN Research (SOSR '18). Association for Computing Machinery, New York, Article 1, 1–15. https://doi.org/10.1145/3185467.3185468
22. OpenStack Security Guide. http://docs.openstack.org/security-guide. Accessed 24 Feb 2023
23. Banikazemi M, Olshefski DP, Shaikh A, Tracey JM, Wang G (2013) Meridian: an SDN platform for cloud network services. IEEE Communications Magazine 51:120-127
24. Patel P et al (2016) SDN and NFV integration in openstack cloud to improve network services and security. In: International conference on advanced communication control and computing technologies, pp 657–658
25. Tissir N, El Kafhali S, Aboutabit N (2021) How much your cloud management platform is secure? OpenStack Use Case. https://doi.org/10.1007/978-3-030-66840-2_85
26. Lane N, Koslovski G, Pillon M, Miers C, Gonzalez N (2020) Software-defined network security over openstack clouds: a systematic analysis. In: International conference on cloud computing and services science (CLOSER), pp 423–429. https://doi.org/10.5220/0009471304230429
27. Jacquin L, Shaw AL, Dalton C (2015) Towards trusted softwaredefined networks using a hardware-based integrity measurement architecture. In: IEEE conference on network softwarization
28. Liu G, Trotter M, Ren Y, Wood T (2016) NetAlytics: Cloud-Scale Application Performance Monitoring with SDN and NFV. In Proceedings of the 17th International Middleware Conference (Middleware '16). Association for Computing Machinery, New York, Article 8, 1–14. https://doi.org/10.1145/2988336.2988344
29. Jeong S, Yoo J, Hong JW (2019) Design and implementation of virtual TAP for SDN-based OpenStack networking. In: IFIP/IEEE symposium on integrated network and service management, pp 233–241

30. Callegati F, Cerroni W, Contoli C, Cardone R, Nocentini M, Manzalini A (2016) SDN for dynamic NFV deployment. IEEE Commun Mag 54(10):89–95
31. SONA: DC Network Virtualization, The Open Network Operating System (ONOS). https://wiki.onosproject.org/display/ONOS/SONA%3A+DC+Network+Virtualization. Accessed 24 Feb 2023
32. Chi Y, Jiang T, Li X, Gao C (2017) Design and implementation of cloud platform intrusion prevention system based on SDN. In: 2017 IEEE 2nd international conference on big data analysis (ICBDA), pp 1–6
33. Foresta F et al (2018) Improving OpenStack networking: advantages and performance of native SDN integration. In: 2018 IEEE international conference on communications (ICC)
34. Xu C, Zhang R, Xie M, Yang L (2020) Network intrusion detection system as a service in OpenStack Cloud. In: 2020 international conference on computing, networking and communications (ICNC), pp 450–455. https://doi.org/10.1109/ICNC47757.2020.9049480
35. Virupakshar KB et al (2020) Distributed denial of service (DDoS) attacks detection system for OpenStack-based private cloud. Procedia Comput Sci 167:2297–2307
36. Benet CH, et al (2017) "OpenStackEmu — A cloud testbed combining network emulation with OpenStack and SDN." 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), 566-568
37. Son J, Dastjerdi AV, Calheiros RN, Ji X, Yoon Y, Buyya R (2015) Cloud-SimSDN: modeling and simulation of software-defined cloud data centers. In: IEE/ACM international symposium on cluster, cloud and grid computing, pp 475–484. https://doi.org/10.1109/CCGrid.2015.87
38. Malik A, Ahmed J, Qadir J, Ilyas MU (2017) A measurement study of open source SDN layers in OpenStack under network perturbation. Comput Commun 102, C:139–149
39. Li J, Jiang H, Jiang W, Wu J, Du W (2020) SDN-based stateful firewall for cloud. In: 2020 IEEE 6th Intl conference on big data security on cloud (BigDataSecurity), pp 157–161. https://doi.org/10.1109/BigDataSecurity-HPSC-IDS49724.2020.00037
40. Luo Y, Luo W, Puyang T, Shen Q, Ruan A, Wu Z (2016) OpenStack security modules: a least-invasive access control framework for the cloud. In: 2016 IEEE 9th international conference on cloud computing (CLOUD), pp 51–58. https://doi.org/10.1109/CLOUD.2016.0017
41. Jin X, Krishnan R, Sandhu R (2014) "Role and attribute based collaborative administration of intra-tenant cloud IaaS." 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, 261-274
42. Hoang XT, Bui ND (2021) An Implementation of Firewall as a Service for OpenStack Virtualization Systems. ICISN 2021. In: Lecture notes in networks and systems, vol 243. Springer, Singapore. https://doi.org/10.1007/978-981-16-2094-2_12
43. Abdulqadder IH, Zou D, Aziz IT, Yuan B, Li W (2018) SecSDN-cloud: defeating vulnerable attacks through secure software-defined networks. IEEE Access 6:8292–8301. https://doi.org/10.1109/ACCESS.2018.2797214
44. Benjamin B, Coffman J, Esiely-Barrera H, Farr K, Fichter D, Genin D, Glendenning L, Hamilton P, Harshavardhana S, Hom R, Poulos B, Reller N (2017) Data protection in OpenStack. In: 2017 IEEE international conference on cloud computing (CLOUD), pp 560–567
45. Dinh PT, Park M (2020) ECSD: enhanced compromised switch detection in an SDN-based cloud through multivariate time-series analysis. IEEE Access 8:119346–119360. https://doi.org/10.1109/ACCESS.2020.3004258
46. Du X, Lv Z, Wu J, Wu C, Chen S (2016) PDSDN: a policy-driven SDN controller improving scheme for multi-tenant cloud datacenter environments. In: 2016 IEEE International Conference on Services Computing (SCC), pp 387–394. https://doi.org/10.1109/SCC.2016.57
47. Krishnan P, Duttagupta S, Achuthan K (2019) VARMAN: multi-plane security framework for software-defined networks. Comput Commun 148:215–239. https://doi.org/10.1016/j.comcom.2019.09.014
48. Krishnan P, Duttagupta S, Buyya R (2021) OpenPATH: application-aware high-performance software-defined switching framework. J Netw Comput Appl 193:103196, ISSN 1084-8045. https://doi.org/10.1016/j.jnca.2021.103196
49. Real World threat modeling using the PASTA methodology. https://www.owasp.org/images/a/aa/AppSecEU2012_PASTA.pdf. Accessed 2 Jan 2022
50. Mininet. An instant virtual network on your laptop. http://mininet.org/. Accessed 24 Feb 2023
51. Zhang X, You J (2020) A gated dilated causal convolution based encoder-decoder for network traffic forecasting. IEEE Access 8:6087–6097. https://doi.org/10.1109/ACCESS.2019.2963449
52. Pfülb B, Hardegen C, Gepperth A, Rieger S (2019) A study of deep learning for network traffic data forecasting. In: International conference on artificial neural networks, pp 497–512. https://doi.org/10.1007/978-3-030-30490-4_40
53. Mousavi SH, Khansari M, Rahmani R (2020) A fully scalable big data framework for botnet detection based on network traffic analysis. Inf Sci 512:629–640. https://doi.org/10.1016/j.ins.2019.10.018
54. Krishnan P, Achuthan K (2019) Managing network functions in stateful application-aware SDN, security in computing and communications. In: SSCC 2018. Communications in computer and information science, vol 969. Springer, Singapore. https://doi.org/10.1007/978-981-13-5826-5_7
55. Lu W, Tong H (2009) Detecting network anomalies using CUSUM and EM clustering. In: Cai Z, Li Z, Kang Z, Liu Y (eds) Advances in computation and intelligence. ISICA 2009. Lecture notes in computer science, vol 5821. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-04843-2_32
56. networking-sfc. https://opendev.org/openstack/networking-sfc. Accessed 24 Feb 2023
57. Tavallae M, Bagheri E, Lu W, Ghorbani AA. Nsl-Kdd dataset. http://www.unb.ca/research/iscx/dataset/iscx-NSL-KDD-dataset.html. Accessed 24 Feb 2023
58. Sharafaldin I, Lashkari AH, Ghorbani AA (2018) "Toward generating a new intrusion detection dataset and intrusion traffic characterization," International Conference on Information Systems Security and Privacy, pp 108–116
59. P.A.A. Resende, A.C. Drummond, The hogzilla dataset, http://ids-hogzilla.org/dataset. 2018
60. University of New Brunswick. CSE-CIC-IDS2018 on AWS. 2018. Available: https://www.unb.ca/cic/datasets/ids-2018.html
61. University of New Brunswick (2019) DDoS evaluation dataset (CICDDoS2019). unb.ca Available: https://www.unb.ca/cic/datasets/ddos-2019.html
62. Hamza HH, Gharakheili TA, Benson, Sivaraman V (2019) Detecting volumetric attacks on lot devices via sdn-based monitoring of mud activity. In: Proceedings of the 2019 ACM symposium on SDN research, pp 36–48
63. OpenStack Tempest. https://github.com/openstack/tempest. Accessed 24 Feb 2023
64. Brandt M et al (2014) Security analysis of software defined networking protocols OpenFlow. In: OF-Config and OVSDB in IEEE ICCE 2014 (July 2014)
65. Tasch M, Khondoker R, Marx R, Bayarou K (2014) Security Analysis of Security Applications for Software Defined Networks. In Proceedings of the 10th Asian Internet Engineering Conference (AINTEC '14). Association for Computing Machinery, New York, 23–30. https://doi.org/10.1145/2684793.2684797
66. Ring M, Wunderlich S, Grüdl D, Landes D, Hotho A (2017) Flow-based benchmark data sets for intrusion detection. Proceedings of the 16th European Conference on Cyber Warfare and Security, p 361–369
67. Kim M, Kong H, Hong S, Chung S, Hong JW (2004) A flow-based method for abnormal network traffic detection. In: 2004 IEEE/IFIP network operations and management symposium (IEEE Cat. No. 04CH37507), vol 1, pp 599–612
68. Corchado E, Herrero Á (2011) Neural visualization of network traffic data for intrusion detection. Appl Soft Comput 11:2042–2056. https://doi.org/10.1016/j.asoc.2010.07.002
69. CVE security vulnerability database. Security vulnerabilities, exploits, references, and more. https://www.cvedetails.com/. Accessed 24 Feb 2023

Publisher's Note