AMERICAN UNIVERSITY OF BEIRUT

SERVICE PROVISIONING IN VIRTUALIZED AND PROGRAMMABLE NETWORK ENVIRONMENTS

by

MAHA NIZAM SHAMSEDDINE

A dissertation
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
July 2018

AMERICAN UNIVERSITY OF BEIRUT

SERVICE PROVISIONING IN VIRTUALIZED AND PROGRAMMABLE NETWORK ENVIRONMENTS

by

MAHA NIZAM SHAMSEDDINE

Approved by:

[illegible]

Dr. Ayman Kayssi, Professor — Chairman
Electrical and Computer Engineering

Ali Chehab

Dr. Ali Chehab, Associate Professor — Advisor
Electrical and Computer Engineering

[illegible]

Dr. Rouwaida Kanj, Associate Professor — Member of Committee
Electrical and Computer Engineering

[illegible]

Dr. Mohamad Al-Ladan, Dean and Professor — Member of Committee
Sciences and Information Systems, Rafik Harriri University

C/O Sanaa Sharafeddine — Ali Chehab

Dr. Sanaa Sharafeddine, Associate Professor — Member of Committee
Computer Science, Lebanese American University

Date of dissertation defense: July 17, 2018

ACKNOWLEDGMENTS

It’s a challenging life, to achieve great ambitions, hard work and perseverance should be complemented with the encouragement, appreciation, and guidance of great people; my co-advisors, Prof. Ayman Kayssi and Prof. Ali Chehab to you my continual appreciation gratitude and respect.

My family and inspiration, Wassim, Mohammad, and Abed Rahman, your support, patience, and inspiration throughout the tough years of my PhD studies are the ultimate drivers that made this work be accomplished successfully. Thank you.

My Dad and Mom you’ve always believed in my potentials and inspired me to continue the journey of achievements. To you, all the love and respect.

AN ABSTRACT OF THE DISSERTATION OF

Maha Nizam Shamseddine for Doctor of Philosophy
Major: Electrical and Computer Engineering

Title: Service Provisioning in Virtualized and Programmable Network Environments

This work tackles the ossification and complexity problem of communication networks in terms of management, configuration, control, connection and security. These complexities are as a result of the great dependencies on network hardware and vendor-specific operation. In order to support the evolving nature of new networking technologies and the introduction of IoT, Edge Computing, Cloud Computing, Fog Computing, 5G, among others, the network must be transformed to a more flexible, agile, dynamic, and multi-tenant infrastructure. This work addresses the demand for a new networking platform that is capable of adapting to the quick pace imposed by the evolution of the digital world and its communication. Towards the afore elaborated challenge, the work in this thesis addresses the provisioning of new protocols and networking services that would provide a more suitable platform and foundation for the developing communication world. To achieve dynamically reconfigurable and extensible network platforms, network softwarization and virtualization are utilized to provide advanced networking structures and services. The proposed set of networking services that would fulfill the work profound goals include: (1) simplified network management, configuration, and control. (2) Flexible enterprise site connectivity in SDN environments. (3) Malicious data plane detection for enhanced SDN security. (4) Secure authentication and dynamic network configuration for Fog computing in virtualized wireless SDN environments.

CONTENTS

ILLUSTRATIONS

TABLES

Table Page

CHAPTER 1

INTRODUCTION AND THESIS OBJECTIVES

1.1 Introduction

The IP protocol stack has been the main network connecting technology since the inception of the Internet in the mid 90s. IP networks are proving day after day that they are neither efficient nor scalable for supporting the current thriving IT industry [4]. The complexity of provisioning, configuring and dynamically updating networks is one of the main problems hindering the adoption of scalable modern network architectures such as IoT, 5G, Cloud computing, Edge/Fog computing, among others. This necessitates the design and implementation of new generation network configuration and connectivity protocols to support the agility, elasticity, mobility, security, and dynamic management of these new trends in modern networks and the cloud.

There is a growing demand on network agility and flexible network resource allocation and release across and within cloud data centers utilized by multiple tenants. The ossification and the inflexibility of the network architecture and technology adopted by the IT industry impose many limitations and problems for the advancement of the modern networking technologies. This requires a call for action on delivering elastic network connectivity and configuration schemes utilizing the recent advancements in network softwarization and virtualization.

Another major challenge facing current cloud data centers is represented in the flexible management of security mechanisms to ensure the authenticity, confidentiality, privacy, and availability of network data and services. The ongoing advancements and fast development of cloud networking services spanning geographically-dispersed data centers and multi-tenant cloud network architectures together with the new trends in network virtualization and Software-Defined Networking (SDN) raises several security concerns related mainly to network data authentication and accurate data plane operation. In this thesis we propose a set of protocols and services for flexible and dynamic network connectivity, provisioning, and security utilizing the new trends in network virtualization and softwarization.

Computing virtualization has been the main driving force for the dramatic proliferation of the cloud computing model we are familiar with today. The main service execution containers in modern data centers are represented in Virtual Machines (VMs) that can be instantiated, managed, operated, and configured on-demand based on tenants' requirements and workload patterns. The great success achieved in the field of computing virtualization has paved the way for the emergence of networking virtualization platforms that are rapidly laying the ground for specialized Network as a Service (NaaS) [1] cloud offerings. Using the NaaS model, entire Virtual Networks (VNets) can be created and managed instantaneously upon tenants' requests. Many cloud NaaS models exist today and the number is highly increasing with the advancements in network virtualization hypervisor design. Moreover, the emergence of a novel network architecture represented in the SDN [2] concept, where network switching units can be dynamically reprogrammed to control the various aspects of networking operations, had a sizeable positive impact on the NaaS cloud support. This is due to the fact that SDN has shed the light on the significance of coupling the network programmability features with the network virtualization counterparts to truly realize an elastic networking infrastructure.

Accordingly, the world of communication is currently witnessing the evolution of programmable network configuration and management through the advancement of SDN and network virtualization. Today, Virtual Internet Service Providers (ISPs) leverage the SDN centralized network programmability and network virtualization features to provide resilient and easily reconfigurable network connection services. Tenants on the other hand, are seeking communication services that exploit configuration flexibility, performance, scalability, security, and interoperability.

Moreover, the programmability and dynamism features achieved in SDN networks are mainly due to the inherent SDN architecture which is based on the separation of the network control and configuration logic from the network switching logic, by offering dedicated SDN controllers with fine-grained control over network routing and reconfiguration. The network is logically divided into two main planes: the control plane and the switching plane. The control plane is where the SDN controller operates by constructing a global view of the network topology and disseminating a collection of routing action rules over the switching/routing network nodes. The switching plane is mainly responsible of packet forwarding across the set of switches/routers comprising the physical network. This novel architecture lays the ground for a highly appealing concept in networking represented in the ability to easily and remotely configure network topologies in an “elastic” virtualized manner analogous to the techniques used in virtualizing servers and storage in modern data centers. Based on the network virtualization concept, the SDN controller can provide tenants with customized network topology views of the physical network based on their requirements and resource demands thus supporting, for the first time in the history of network computing, the basis of a truly NaaS provider infrastructure.

In spite of the increasing popularity in NaaS offerings, three main issues are still hindering the wide adoption of NaaS services in the cloud, namely, 1) the NaaS model does not provide any assistance or support for the tenants in designing their network topologies or managing the network-wide services, 2) the assorted pricing schemes offered by NaaS providers, as well as the occasional price variations and deals complicate the cost-effective selection of cloud providers, and 3) NaaS services do not support VNet partitioning, whereby the VNet is partitioned into several logical parts that can communicate seamlessly as a single unit. Network partitioning allows for cost-effective and performance-efficient VNet deployment based on the tenants' preferences and constraints.

A second challenge is that traditional ISPs do not support tenants with flexible connectivity solutions that can dynamically adapt to meet tenants' requirements in terms of security, cost, performance, and reliability, which puts extra burdens on the tenants to manually impose their connectivity constraints. With SDN, the whole network connectivity can be reprogrammed utilizing information delivered from the tenants and the SDN providers to deliver seamless mechanisms in the deployment, configuration, and migration of the communication sites on available SDN infrastructures.

On the security front, a major security risk in any packet switching network is mainly represented in the malicious operation of the network forwarding units which is also the case in SDN networks which must be supported with dedicated security services at the data plane to ensure that the switching units responsible of forwarding the packets are not executing or participating in any active attack on the network traffic. One challenge is providing a certifying security service that provides, with high confidence levels, SDN tenants with sufficient guarantees that the network they are running their applications on is free of malicious activities on the data plane. This service should (1) trigger security alarms in real time, (2) be efficient in applying the network monitoring/probing operations using compact data structures, and most importantly (3) be specifically designed for securing SDN networks. Thus it is of high importance to utilize the afore mentioned SDN-specific properties to effectively support the security services in SDN networks. The flexibility and programmability features of the SDN network model provide appealing advantages for the advancement of network autonomous creation and configuration. The introduction of the concept of data plane/control plane separation significantly facilitates network programming and central control over the switching and routing mechanisms of the global network view.

Moreover, SDN can support enhanced security services in the Mobile Edge Computing (MEC) platforms. Despite the great advancements in mobile technology, mobile devices are still relatively limited in computing resources, memory, battery life and power. Together with the bandwidth demands being expected to continue doubling, and the profound development of computing-intensive, and resource and power demanding mobile applications, there is a great need for outsourcing applications on one hand, and bringing the servers to the network edge, on the other hand. Mobile cloud computing has had the strongest impact in outsourcing computation and offloading tasks to more sophisticated and resourceful servers to relief the mobile devices from running such services locally on their terminals. MEC was introduced to address the latency challenge by transferring the cloud data and services to infrastructure providers in the vicinity of the mobile user. Such services can be accessed via cellular or Wi-Fi access points. Fog distributed computing emphasizes on MEC and extends resource and memory servers to support mobile users' expected movement and service demands. Moreover, Fog servers manage autonomous and independent data processing for real-time computing-intensive mobile applications. This results in a three-tier network consisting of three main entities: the mobile user, the Fog server(s), and the cloud service(s).

Security and privacy in Fog/MEC environments are of great concern. A major security problem is the attack from rogue Fog nodes, where a malicious server impersonates a Fog node and advertises itself as a legitimate server to the mobile user. Another important challenge to Fog computing is represented in the vulnerabilities of the underlying physical wireless network. Client access to Fog services is mainly done using Wi-Fi networks and any vulnerability in the wireless security implementation would affect the whole service stack.

1.2 Thesis Objectives

The objective of this work is to tackle the ossification and complexity problem of communication networks in terms of management, configuration, control, connection and security. These complexities are as a result of the great dependencies on network hardware and vendor-specific operation. In order to support the evolving nature of new networking technologies and the introduction of IoT, Edge Computing, Cloud Computing, Fog Computing, 5G, among others, the network must be transformed to a more flexible, agile, dynamic, and multiple-tenant infrastructure. This work addresses the demand for a new networking platform that is capable of adapting to the quick pace imposed by the evolution of the digital world and its communication. Towards the afore elaborated challenge, the work in this thesis addresses the provisioning of new protocols and networking services and technologies that would provide a more suitable platform and foundation for the developing communication world. To achieve a more agile, dynamically reconfigurable and extensible network platforms, network softwarization and virtualization are utilized to provide advanced networking structures and services. The following is the proposed set of networking services that would fulfill the work profound goals:

- Simplified Network Management, Configuration, and Control: To fulfill this objective we design and implement a cloud network configuration and provisioning service in a software-defined networking architecture. This centralized cloud service is proposed for creating virtualized networks in SDN-based cloud architectures that supports dynamic provisioning operations based on QoS, pricing, privacy, reliability, and energy requirements. To fulfill the objective of dynamically provisioning and configuring multiple network topologies according to tenants demands, this service should provide a unified interface through which tenants network specifications and constraints are fed to the service provisioning algorithms thereby creating, configuring and updating their networks dynamically on demand.

- Flexible Enterprise Site Connectivity in SDN Environments: On this front we present a complete SDN-based Internet connection solution for dynamically linking geographically separated enterprise branches. Network virtualization in a softwarized platform is to be exploited in order to provide tenants with a dynamic virtual network connectivity service that would increase the agility and dynamism in network connection.

This service should be designed to connect geographically separated enterprise branches based on the SDN infrastructure with minimum operation cost. Virtual SDNs (vSDN) should be created and dynamically managed for the tenant's sites that are logically decoupled from the network infrastructure, to provide configurable network topologies and isolation on the SDN network virtualization platform.

- Malicious Data Plane Detection for Enhanced SDN Security: For achieving this objective we design and implement a protocol for detecting malicious switching elements in SDN virtualized environments. The introduction of network softwarization and the separation of the hardware from the software network controller brings forwards SDN related security and correct operation breaches. To ensure the correctness of the SDN network data plane performance, network virtualization is to be exploited to provision a cloud security service for detecting malicious switching elements in SDN environments.

- Secure Authentication and Dynamic Network Configuration for Fog Computing in Virtualized Wireless SDN Environments: To achieve this objective we present a secure and scalable authentication and network configuration protocol for Fog and mobile edge computing that leverages the SDN platform and wireless network virtualization. The proposed design addresses the security problem of rogue fog nodes and provide a secure and flexible network creation and configuration mechanisms for cloud service providers at the network edge. This goal emphasizes on promoting a more secure and dynamic communication among the mobile client, the infrastructure provider, and the cloud service provider.

CHAPTER 2

LITERATURE SURVEY

In this chapter we present a literature survey of the main research approaches related to the proposed work. We start, in Section 2.1, by presenting a fundamental survey on network virtualization concepts and architectures. Section 2.2, presents a brief introduction of the new generation networking, SDN networking architecture. In Section 2.3, we introduce network as a service models and the latest advancements in NaaS platforms based on the SDN model. Finally, in Section 2.4 we discuss the security problems that emerged as a result of the SDN centralization of the network control and the research approaches devised to detect and circumvent malicious attacks in SDN environments.

2.1 Network Virtualization: Basic Concepts & Architecture

Network virtualization is the main enabler for the next generation networking in the fields of telecommunication and the Internet [5]. Virtualization facilitates the feasibility of creating multiple heterogeneous networks on the same physical infrastructure. Sharing the physical resources results in "elastic" network topologies composed of virtual nodes connected via virtual links to provide dynamic end-to-end connectivity services. The network virtualization property, imposed by a network virtualization hypervisor layer, provides the spawned virtual networks with full isolation among each other to achieve relatively high levels of privacy and security. In such virtualization environments, VNets are characterised by elevated degrees of flexibility and ease of management, elasticity and dynamism, scalability, isolation, and heterogeneity [6].

Flexibility and ease of management: multiple virtual networks can be created, deployed, and destroyed independent of the underlying physical network. Service providers can provision their choice of network topology and forwarding protocols irrespective of the underlying physical network for the sake of delivering flexible configurations to support tenants' services.

Isolation: VNets can coexist on the same physical infrastructure and their corresponding traffic is logically segregated using the virtual network hypervisor. The latter is responsible for separating the VNets address space to provide higher levels of isolation and to reduce the probability of fault propagation among the configured virtual networks.

Elasticity and Dynamism: Using network virtualization techniques, VNets can be created, scaled up, and scaled down instantaneously using configurable software commands. This expedites the process of resource allocation and release without the need to restructure the underlying hardware infrastructure or network configuration.

Scalability: with network virtualization, VNets can practically scale up to the resources dedicated to the physical network with minimal performance overhead imposed by the network virtualization layer. This is considered a major property to ensure the scalability of the coexisting VNets [7].

Heterogeneity: with network virtualization, Service providers can provision VNets with their arbitrary independent topologies and forwarding protocols irrespective of those of the leased physical network infrastructure.

These technical advantages of Network virtualization has been originated and evolved from the great advancements in server virtualization [8] which in turn was the main enabler for infrastructure and platform services in cloud computing [9]. Server computing virtualization has been introduced by the virtual machine software that abstracts and decouples the software from the underlying machine hardware. Multiple virtual machines can coexist on the same underlying physical machine thus providing full isolation, on-demand provisioning, and flexible management.

Network virtualization has witnessed extensive attention in academia and industry due to the dynamic nature of connectivity achieved which remarkably aids in reducing the OPEX (operational expenses) and the CAPEX (capital expenses) as explained in the Network Function Virtualization (NFV) work presented in [10]. The dynamism and flexibility properties provided by network virtualization push for a better utilization of the network resources and ease of management and mobility of the network components. This high level of flexibility that can be achieved by VNets, and supported lately by appealing network programmability architectures represented in the SDN networking paradigm, is the primary reason behind the proliferation of virtualization techniques in the networking research and academia projects.

Network virtualization had a highly positive impact on the cloud computing industry. Today cloud service providers are separated into two major roles: (1) the Infrastructure Providers (InPs) that mange and offer wide varieties of physical network resources and substrates that meet the requirements and needs of tenants, and (2) the Service Providers (SPs) which provision end to end network services by aggregating resources from different InPs according to tenants' requests related mainly to cost, flexibility, programmability, security, and privacy, among others.

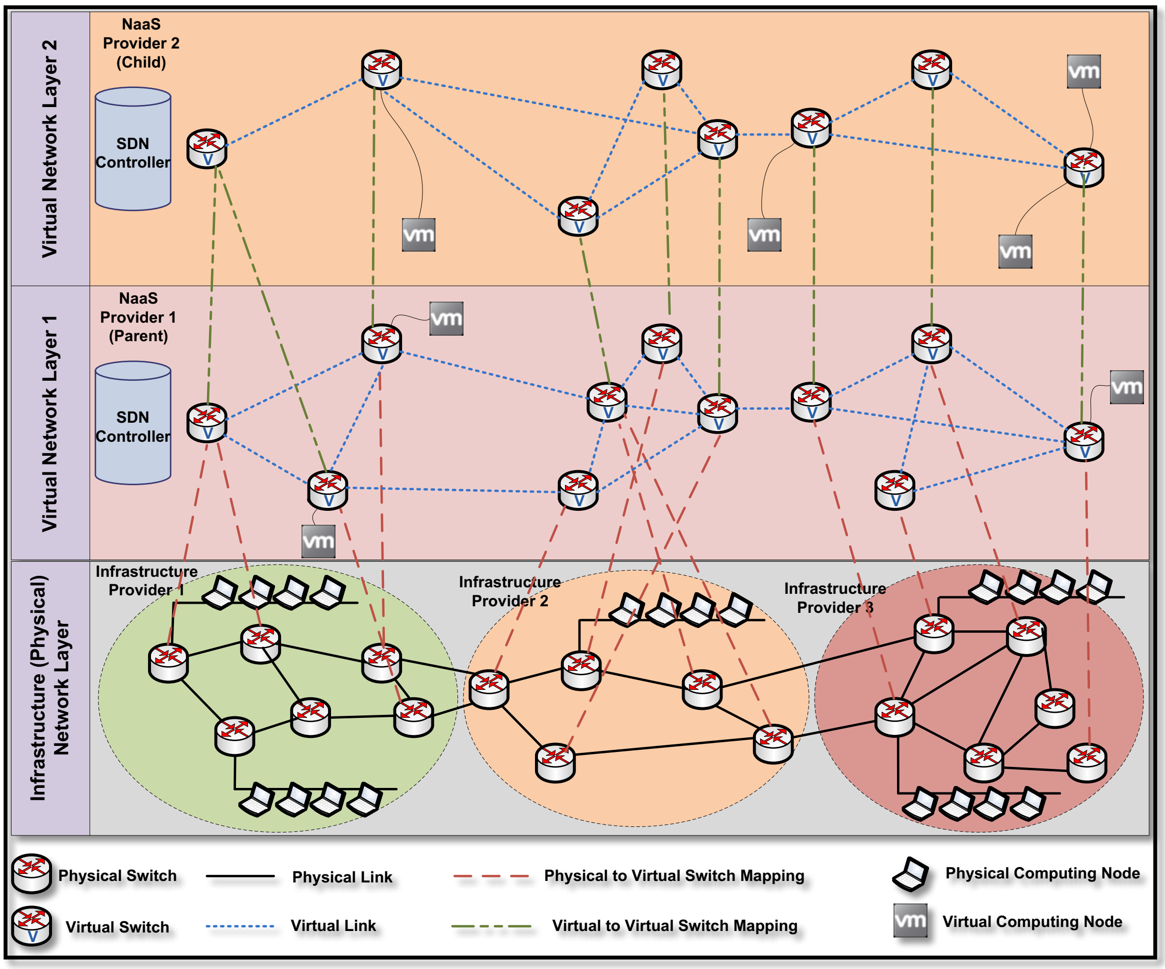


Fig. 2.1 - Network virtualization layers and provider categorization

As a result, the virtual network environment (VNE) [11] greatly facilitates the utilization of a dynamic network infrastructure where multiple SP's provision multiple heterogeneous VN's that can independently coexist on the same network substrates but with full isolation and flexibility. This is illustrated in Figure 2.1.

In [6], the authors classify network virtualization into four main network categories namely: Virtual Local Area Network (VLAN), Virtual Private Network (VPN), active and programmable networks, and overlay networks. VLANs are networks that connect nodes and hosts together using logical addressing and have a single broadcast domain. VLANs coexist on the same physical LAN where each separate broadcast domain is enforced by the layer 2 switches to achieve traffic isolation. Broadcast domains are enforced by mapping a multiport virtual bridge on the underlying physical switch and by dedicating this bridge to a distinct VLAN. VPN is a network created on top of other networks that are located in different geographic locations. Hosts in a VPN from different network sites appear to be on the same virtual network that share the same broadcasting address. The VPN technology was initially created to ensure a secure encrypted network connection between institutional sites and their remote clients over a public internetwork such as the Internet. Many protocols currently support the VPN functionality, the most popular are IP Security (IPSec) [29], Point-to-Point Tunneling Protocol (PPTP) [30], and Layer 2 Tunneling Protocol (L2TP) [31]. It should be noted here that the Secure Socket Layer (SSL) and Transport Layer Security (TLS) [17] protocols can also be used to support the secure network communication of a VPN connectivity solution by employing their authentication, encryption, and integrity mechanisms. Active and programmable networks are a new generation of network technologies, protocols, and platforms that are primarily proposed to support the dynamic and active inclusion of software and hardware services into network elements. These types of networks are supposed to enhance the resiliency, security, management, and customization of the network components to ultimately enable a form of seamless plug-and-play functionality. Active and programmable networks had a great role in the demand for network virtualization to provision dynamic services decoupled from the underlying network resources. Overlay networks are networks consisting of a set of nodes with logical direct connectivity possibly over a path of multiple physical nodes. Overlay networks are mainly used to implement new features on the Internet and for experimental fixes, research, and deployment of new functions and architectures. Popular examples of overlay networks are represented in Peer-to-Peer (P2P) networks, Voice-over-IP (VoIP) networks, and content distribution networks.

2.2 Software-Defined Networking Architecture

SDN is the new generation networking architecture that is based primarily on the concept of separating the forwarding plane from the control plane and centralizing the latter in a set of one or more controller units. This pushes the network evolution greatly by facilitating the programmability of the network functions, controls, management, and configuration. This is achieved by abstracting the operation of the switching elements in the data plane by constructing their forwarding tables and actions centrally at the control plane. Centralizing the control plane and functionally separating it from the data plane is considered a major contribution brought forward by SDN to the networking domain which is the main enabler for further advancements represented mainly in network virtualization and programmability.

According to [12], the SDN architecture is targeted by all the members of the networking industry namely: the Internet service providers, network infrastructure providers, network vendors, enterprises, and end users. This great demand is expected after the long-termed ossification of the internet that will fend as a result of the following innovations of the SDN architecture:

1 *The separation of the control and management plane from the data plane*: In traditional IP networks, control and data plane functions are featured in the network nodes. The separation introduced by the SDN relies on introducing network nodes as forwarding elements that utilize the open flow protocol for data transfer in the network. A central controller or set of controllers represented in the control and management plane are responsible for setting the control rules adequate to transport the network messages in the data plane. The L3 routing protocols and the management functions are the responsibility of the controller in this architecture. This results in a simplified set of forwarding elements in the network that can be easily managed, maintained and replaced.

2 *Centralizing the control plane*: The Internet architecture has been based on distributing the communication elements in order to avoid the vulnerable nature of a centralized communication system. It is only in the recent years that SDN proposed the concept of centralizing control. Accordingly, the network is divided into subnets, controlling and managing these sub-networks is aided by the statistics data that is collected from the switching elements using the Northbound API [13] of the controller that enables the efficient management and utilization of the underlying network resources. The centralization advantage lies in (1) the fast propagation of maintenance activities, (2) the introduction and

enforcement of new policies and updates, (3) the flexible registration, revocation, and reconfiguration of network switching elements and (4) and the better visibility of network topologies which facilitates enhanced internetworking and connectivity establishment.

3 *Northbound APIs for control plane programmability* [13]: The network operating system in the SDN architecture provides a set of user-friendly API's that facilitates programming the underlying network components. The controller provides a software representation of the switching hardware in order to make it vendor independent. An example is the OpenDaylight [14] SDN controller. This is analogous to the computer operating system that witnessed the programmability hype after it was hardware specific. This is the key to prevent the ossification of the Internet and provision for the exploration of network programmability virtues.

4 *Flow based control*: this is enforced by the SDN controller on the network switching units and is governed by the specifications of the OpenFlow [15] protocol. OpenFlow defines a collection of low-level operations for updating and maintaining the switches flow tables which contain the entries that define the packet forwarding mechanisms fed to the switch by the SDN controller. Accordingly, SDN-compliant switches started emerging to support the flow-based, software-controlled forwarding mechanisms imposed by the specifications of the OpenFlow protocol. Typically, to be OpenFlow-compliant [16], a network switch must implement:

- A flow table data structure to store packet forwarding rules
- A secure mechanism to exchange control traffic with the SDN controller, mainly ensured using the TLS protocol [17].

- The necessary software modules for consuming OpenFlow messages and executing their corresponding semantics.

The OpenFlow protocol operation in the data plane relies on three main fields comprising the entries of the switch's flow table:

- The header field: this field identifies network flows for the purpose of packet forwarding. It is worth mentioning here that the header field can consist of a set of subfields to define flows spanning different layers in the network protocol stack, typically layers L1 to L4.
- The flow counter field: the counter field is incremented whenever a packet header matches the respective flow table entry. This leveraged by the SDN controller to manage the network by dynamically maintaining fresh network flow statistics.
- The flow action field: this field specifies the actions that must be executed as a result of a match on the header field of the respective flow table entry. Three principal actions are defined namely: forward to port, forward to controller, and drop packet. When a packet arrives at an ingress port with no matching flow table entry, the packet is forwarded as a "PacketIn" to the SDN controller over the TLS secure channel. Consequently, the controller takes the responsibility of updating the switch flow table with the suitable forwarding rules.

2.3 Network as a Service Models in Software-Defined Networks

The NaaS concept was firstly introduced by Costa et al. in [1]. With this service, tenants are granted an isolated and secure access to the underlying network resources by providing them with virtualized network views on top of the physical network infrastructure. NaaS services are mainly implemented by integrating advanced packet inspection and forwarding policies that isolate the network traffic of the respective tenants' virtual network views and provide them with their own virtual routing and switching network elements. The main benefits provided by the NaaS model are represented in the following supported functionalities:

- Virtualized and dynamic network topologies: with NaaS, tenants can have a virtualized logical view of the physical network that can be "elastically" up scaled and down scaled based on the tenants' application requirements. All the necessary abstractions required to implement this functionality are hidden from the tenant in the network virtualization hypervisor layer. This layer is responsible of maintaining the mapping between the logical network views and the underlying physical infrastructure.
- Customized packet forwarding mechanisms: The network virtualization hypervisor implements customized packet forwarding policies and pushes it to the physical switching units to provide a traffic isolation functionality among the tenants' virtual network views. It is this functionality that makes NaaS services very suitably implemented in SDN environments where the SDN controller aids the network virtualization hypervisor in pushing the forwarding rules to the physical switches to support traffic isolation.
- In-network processing: the packet inspection capability of NaaS services can greatly assist the SDN controller in providing in-network processing functionality such as supporting network aggregation, opportunistic caching, packet filtering firewalls, content-based networking, among others. These functions can be better supported in NaaS due to their dependency on the application specifics. Hence tenants directly controlling the network infrastructure via the virtual network views in NaaS can provide highly efficient in-network

processing functions since they are the entities aware of the application types running on the virtual network views.

Extensive research work focused on providing virtualized network services in SDN-based networks. In [18], Drutskoy et al. presented FlowN, a NaaS architecture that utilizes SDN concepts and database mapping techniques to provide tenants with their own virtual network topology, address space, and controller on top of a single physical data center. FlowN is lightweight due to the utilization of container-based virtualization which allows the system to instantiate isolated SDN controllers for the different tenant networks using one physical controller. In [19], Sherwood et al. presented FlowVisor, a switch virtualization architecture that maps the forwarding services of hardware switches to multiple virtual switches that can be used to construct multiple virtual networks with distinct switching logic. FlowVisor can run on commodity switches supporting the OpenFlow protocol, which makes it an attractive network virtualization solution for existing production data center networks.

In OpenVirteX [3], the authors followed a novel network virtualization approach that made it mimic, to a high degree, computing virtualization mechanisms and operations, such as dynamic configuration, instantiation, destruction, snapshotting, and on-demand migration. This gives network virtualization a leading edge in being capable of supporting flexible NaaS solutions whose configuration, infrastructure management, topology specification, and addressing schemes are completely under the tenants' control. It should be noted here that NCaaS relies on the OpenVirtex solution in developing the prototype test bed implementation. In CoVisor [20], the authors followed an SDN controller virtualization approach that allows a single network controller to be virtualized into multiple controller applications supporting different operational platforms and development languages. CoVisor allows for the control on shared network traffic by enabling the enforcement of multiple policies on separate controller applications such as firewall, load balancer, SSL accelerator, application gateway, router, and traffic analysis policies. The work proposed in this paper compliments the above schemes by providing a unified interface that hides the complexities of VNet creation and management from the tenant while maintaining the same levels of tenant control over the provided VNets.

OpenStack Neutron [21] is undoubtedly the most popular SDN networking project for implementing NaaS services in virtualized cloud environments. OpenStack [22] is a dedicated cloud operating system designed for controlling the compute, storage, and networking resources of big cloud data centers via a standardized Web interface known as the OpenStack dashboard. The main OpenStack projects are:

- SWIFT and CINDER for controlling object and block storage respectively
- NOVA for controlling computer resources
- NEUTRON for providing virtualized networking services
- KEYSTONE for supporting identity management services
- GLANCE for providing image services

OpenStack Neutron (originally codenamed Quantum) provides cloud tenants with direct access on the physical network infrastructure to control the network topology and addressing schemes in multi-tenant cloud environments. Neutron is the main OpenStack component for providing network virtualization and isolation in SDN cloud platforms. Neutron supports tenants with an advanced plugin-based API for programming multiple virtual networks with isolated addressing mechanisms. Such APIs are the main pillar that provides tenants with the necessary tools to have better control over additional network functionality such as security, privacy, QoS, intrusion detection and prevention, packet filtering firewalls, and advanced logging and auditing.

2.4 SDN Security Threats and Models

Extensive research work has tackled the security problems that emerged as a result of the SDN centralization of the network control.

In [37], the various SDN threats and vulnerabilities are discussed with a thorough analysis. The work proposed a secure mechanism that targets each introduced SDN threat vector including: network OS replication, application level replication, software and hardware solutions on the control plane to avoid common mode faults and bugs and to increase the network tolerance to hardware and software accidents and malicious behaviors. Moreover, the authors introduced self-healing mechanisms, isolated security domains, fast and dynamic network recovery, and redundant switch-controller association mechanisms. This work represents a call for action to trigger further research in SDN security solutions.

In [42], the authors present a comprehensive security survey that summarizes the security threats of SDN frameworks and categorizes them based on the layers and SDN interface vulnerabilities. On the other hand, the survey discusses and categorizes the security solutions based on the SDN network programmability infrastructure. The centralization of network programming has introduced both security threats and at the same time new and dynamic security solutions. Most of the solutions involved middle boxes that enforce the network security policy and adjustment in the security monitoring and prevention capabilities.

In [36], the authors tackle the problem of lack of trust in the network OS and applications running on top of them where redundant controllers were introduced to the SDN network and a new layer is created to compare the output of the controllers and ensure consistency among the controllers and the network state and policy. This paper lays the ground for designing a trust scheme for redundant controllers in SDN.

The work in [38] presents a formal verification methodology to ensure the safety, security, and reliability of SDN applications that have access to network monitoring APIs using the OpenFlow semantics. However, it introduces some limitations in verifying the network reliability properties, which was justified due to the complexities and non-standardized network topologies in SDN architectures. FRESCO in [39] introduced an OpenFlow security application framework, which facilitates rapid and dynamic creation and deployment of security functions for attack detection and mitigation on the OpenFlow layer.

In [40], the authors introduce a framework of multiple distributed controllers that coordinate SDN control to achieve high scalability and security measures. This is achieved via a cluster-based mechanism that allows the dynamic addition and removal of controllers to the network without network interruption and down time. Any type of OpenFlow controllers can be used in the proposed framework where the switches and applications are unaware of the underlying reassignment of controllers. JGroups are used to synchronize controllers and ensure correct controller-switch mapping. This work recommends the deployment of multiple redundant controllers in the SDN infrastructure without any consideration to the performance and security implications. In [51], the authors propose a security framework to detect suspicious changes in network topology and the SDN data plane. The work uses the flow graphs abstraction to approximate the network operations and thereby detect any suspicious deviation that may be considered as an attack. The main limitation in this work is that the detection mechanism is non-deterministic and is dependent on the accuracy of the flow graph approximation mechanisms. In [34], the authors present a traffic monitoring system in SDN based on sketches. This model, named "Open Sketch", can support the detection of suspicious traffic surges that may spring as a result of a denial of service (DOS) attack on a particular network part. The main limitation in this work is mainly related to the following set of points:

1. It operates on the physical network layer in the SDN model, which, as stated previously, renders it a traditional network security solution with no focus on the SDN particularities.
2. Software engines running on the SDN switches themselves carry out the sketch calculation and updates. Trusting the switches in calculating the sketches can falsify the resulting traffic monitoring measurements by malicious switches and accordingly can mislead any security decision related to the source of possible attacks.
3. This work mainly focuses on detecting suspicious deviations in network traffic and does not shed light on traffic dropping, augmenting, and modification attacks.

In [46], a non-SDN sketch-based solution for detecting attacks on routers is proposed. The approach is typically analogous to that followed by Open Sketch. Several research works have proposed the application of Machine Learning (ML) techniques to provide intrusion detection services in the SDN network architecture [70]. These approaches mainly focused on

Deep Learning (DL) and classification algorithms to enhance the accuracy of the intrusion detection system and maintain low false posities rates. In [70] the authors present a DL-based system for detecting distributed denial of service attacks in SDN. The system is implemented as a network application on top of the POX controller. DL is mainly employed for traffic classification and for reducing the large set of features extracted for the packet headers and needed for attack detection. The main limitation in [70] is the high processing resources it requires on the SDN controller in the packet collection and features extraction phases. In this DL model, every network packet across the whole network is collected for feature extraction which imposes a sizeable load on the SDN controller. This fact is aggravated in vast SDN networks composed of a large number of forwarding switches in the data plane which results in a serious bottelneck on the SDN controller. The VISKA SDN attack categorization model presented in this paper targets the detection of more attack types in addition to denail of service attacks such as interruption attacks, blocking attacks, and man-in-the-middle attacks. Moreover, VISKA imposes minimal overhead on the SDN controller by isolating the source of attack using a highly efficient probing mechanim before proceeding with attack categorization. As a result, the attack categorization module on the SDN controller needs to collect the ingress/egress network packets of a small set of switches that are detected malicious instead of collecting the entire network traffic.

A similar DL-based approach is presented in [71]. In this work Tang et al. propose a flow-based anomaly detection system based on deep neural networks for intrusion detection in SDN. This model uses a limited number of network features for attack detection for the purpose of ehancing the feature extraction process. The main limitation in [71] is represented in the accuracy of attack detection which reaches 75.75%. This renders it infeasible for competing with existing intrustion detection systems or for application in commercial products. In [72] the authors propose a framework for detecting and classifying anomalies in SDN using information theory and machine learning techniques. The network traffic profiles are collected using Sflow [73]. The process consumes high memory and processing power to analyze traffic information and inspect packets. The resulting framework identifies flows as malicious, benign, or unknown to be further analyzed. In [74] the authors address DDoS attacks based on Support Vector Machoine (SVM) classification algorithms in SDN environments. The design is based on the information collected from the switches flow table states. The flow table information are used to create six-tuple characteristic values based on which the SVM algorithm classifies traffic as normal or attacker abnormal traffic. The disadvantages of this work is that it only addresses the DDoS attack on one hand on the other hand, the training phase has to be executed on real network data and on predetermined periods of time to ensure the correctness of the resulting classifier model. This necessitates more computing resources and processing power on the SDN controller. Similar ML-based intrusion detection approaches are presented in [57, 75].

Mobile applications and services are witnessing a massive growth that is triggering a proliferation in mobile network traffic. Many of the mobile traffic result from applications and services that are location-based [76] and are resource and network demanding. This motivated the provisioning of Fog or MEC computing which according to [77] is a result of the evolution of the Cloud and the dramatic growth of mobile services. In [77], the authors discuss the importance of Fog computing to accommodate the dramatic growth of mobile traffic and the resource demanding nature of mobile applications and services. The ultimate aim of reducing latency and ensuring efficient network operation and service delivery is addressed by utilizing more robust and short communications with MEC servers, thus, replacing the traditional Internet-long thin connections [78].

In [79], Roman et al. provide a comprehensive survey on the security threats facing MEC/Fog implementations today. The main focus in this survey is on the Denial of Service (DOS), Man-in-the-Middle (MITM), and rogue Fog node attacks. The survey also presents some recommendations for mitigation techniques to such threats. Another survey on MEC security is presented in [80]. In this survey, the authors provide a brief description of the MEC security threats inherited from cloud computing (authentication, reputation and trust, network security) as well as those specific to the Fog architecture such as the rogue Fog node attack. In [81, 82], MEC security is addressed in the context of smart grid applications and machine-to-machine (M2M) communication. In [82], the authors demonstrate the feasibility of rogue node attack in Fog environment by compromising the gateways on the path to the mobile client. The major problems hindering the provisioning of a feasible and reliable prevention technique of fake Fog node attacks are: 1) the complex trust scenarios for various MEC server deployment choices among the mobile clients, infrastructure providers, and the Cloud service providers, 2) the dynamic nature of MEC in creating, deleting, and upgrading virtual machine (VM) instances of Fog nodes, which further complicates trust/reputation scenarios, 3) authentication is hard to enforce in Fog computing as presented in [82], where authentication at the three different tiers of the Fog architecture using PKI is non-scalable and inefficient. Authentication on the various gateways on the network and on the data collectors at the client side is deployed to control malicious users, falsified data reporting, and spoofed IP addresses attacks. The work in [81] implements and elaborates on MITM attacks in Fog/MEC networks. Servers at the network edge can be compromised and may be replaced by malicious ones. In this attack, clients connect to deceptive SSID, which is hijacked by a third party that controls the gateways and compromises all the communication by inserting a network program into the TCP/IP stack of the compromised network to hijack and replay communicated data. The authors simulate the attack in a Fog environment and discover how stealthy it could be by measuring the CPU, memory, and power consumption exerted by the attacker and which was found to be minimal and not detectable by an IDS anomaly system. In [83], the author presents an efficient security protocol for mutually authenticating mobile edge clients and fog servers. This work does not take into consideration authenticating the MEC client and the fog server to the main cloud service provider.

A recent vulnerability in the Wi-Fi security protocol, WPA2, was discovered by Vanhoef and Piessens [84]. The attack is known as the key reinstallation attack (KRACK). Briefly, the attack exploits a vulnerability in the WPA2 4-way handshake that is used to establish a Pairwise Temporal Key (PTK). The PTK in addition to a packet number nonce are considered the main components for generating the encryption key used to secure the wireless data stream. The adversary blocks message 3 of the 4-way handshake and tricks the wireless access point to retransmit it, thus causing the reinstallation of the PTK and resetting the packet number nonce. As such, the adversary can collect a set of packets encrypted using the same PTK packet number combinations. Thus, the WPA2 data confidentiality stream cipher can be attacked. In some WPA2 implementations, the key reinstallation attack can be executed up to

19 times in a row before the wireless access point throws an exception. Moreover, a similar attack scenario can be executed on the Group Temporal Key (GTK) and the Integrity Group Temporal Key (IGTK). More details on this attack are presented in [84, 85].

CHAPTER 3

NETWORK CONFIGURATION AND CONNECTIVITY SERVICES

This chapter provides a description of the design schemes and algorithms proposed to target the challenges represented in the virtualization of various networking services in SDN environments. These algorithms and services address the essential drivers for future communication technologies, such as 5G and IoT, by providing more dynamic, agile and flexibly reconfigurable and mobile networks. In Section 3.1 we propose the design and implementation of NCaaS, a centralized cloud service for creating VNets in SDN-based Cloud architecture. This is followed, in Section 3.2, by proposing VNCS, a network service that connects geographically separated enterprise branches based on the Software-Defined Networking (SDN) infrastructure.

3.1. NCaaS: Network Configuration as a Service

In this section, we present NCaaS, a centralized cloud service for creating VNets in SDN-based cloud architectures. The proposed service relieves tenants from the burdens and complexities of VNet creation and management by supporting dynamic provisioning operations based on QoS, pricing, privacy, reliability, and energy constraints set by the tenant. Moreover, it provides a unified interface through which tenants' network specifications and constraints are fed into the service provisioning algorithms. These algorithms, in turn, handle the negotiation with the different SDN NaaS providers and dynamically apply the necessary VNet creation, partitioning, and migration mechanisms to ensure the satisfaction of the tenants' preferences. A proof of concept test bed implementation of the proposed service will be provided on top of the OpenVirteX network virtualization platform. The presented NCaaS service paves the way for a more dynamic network slicing and NaaS services which represent a hot research topic for future 5G and IoT networks.

NCaaS provides tenants with their predefined VNets that meet their constraints while guaranteeing a minimum cost. The tenant VNet is composed of a set of *n* software services that interact together to achieve the desired functionality of the tenant's business logic. These *n* services are to be deployed on a set of VMs distributed among *t* providers. NCaaS algorithms utilize provider-related information as well as tenants VNet specification and constraints to create a minimum cost virtual network that complies with these specifications and constraints. Figure 3.1 demonstrates the set of NCaaS algorithms responsible for creating the VNet based on the providers'/tenants' input and specifications.

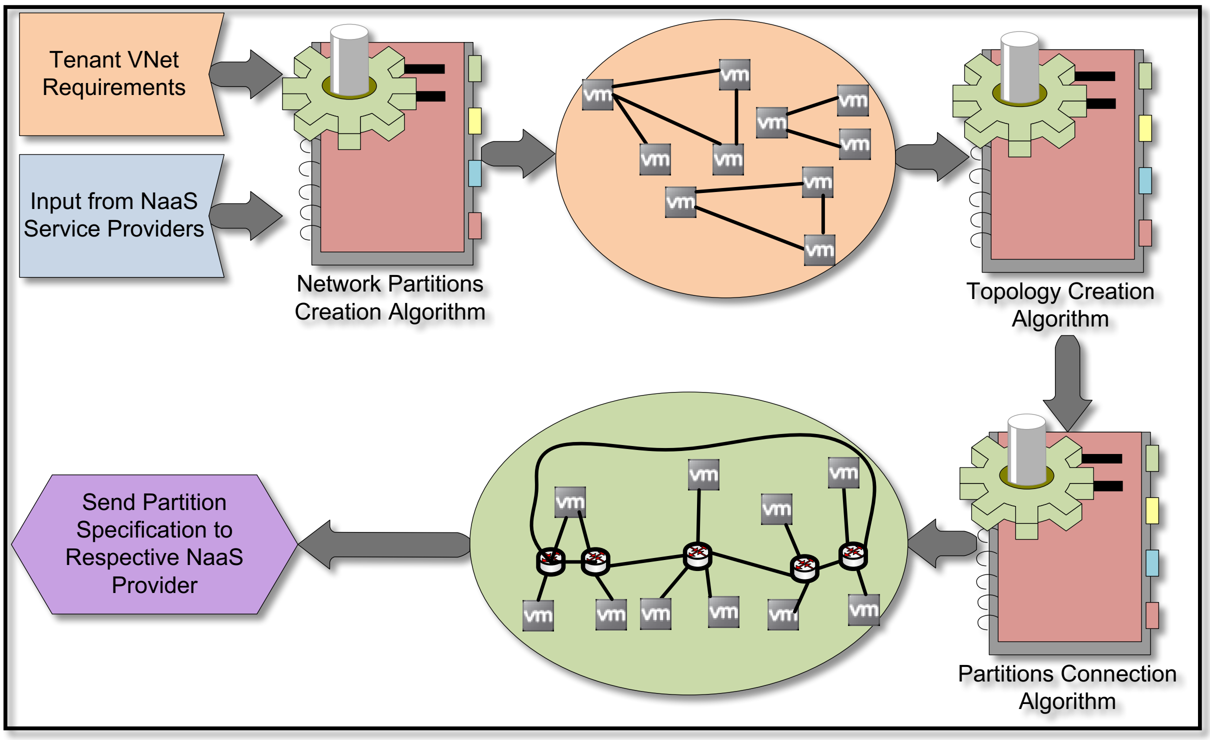

Tenant VNet Requirements
Input from NaaS Service Providers
Network Partitions Creation Algorithm
Topology Creation Algorithm
Partitions Connection Algorithm
Send Partition Specification to Respective NaaS Provider
vm

Fig. 3.1 - NCaaS algorithms for dynamically creating the tenants' virtual networks.

3.1.1NaaS Providers Offers

The NCaaS algorithms operates based on input information retrieved from a set of NaaS providers representing 1) their offered VM profile specifications and their respective cost including the hardware/software specifications of the VM, such as the number of virtual CPUs and their speed, the amount of RAM available, the storage capacity, the operating system and supporting software, and the network bandwidth, 2) the upload/download data communication rates, and 3) the energy resources used by the provider, and the privacy services offered. This information is either input from the providers, or gathered and updated by the NCaaS service. This information is further arranged in three main tables or matrices as follows:

- The Profile-Cost Matrix (CP): this matrix (Figure 3a) shows the cost of the different VM profiles offered by the various cloud providers. Each element in this matrix $CP_{i,k}$ represents the cost of the k^{th} VM profile along with its bandwidth rates offered by the i^{th} cloud service provider.

- The Upload/Download Rate Matrix (UDR): this matrix (Figure 3b) specifies the data upload/download rate offered by each of the τ cloud providers. Element UDR_i consists of the upload/download rate per data unit offered by the i^{th} cloud service provider. Without loss of generality, we assume same rates for data upload and download.

$$\begin{bmatrix} CP_{11} & \cdots & CP_{1\tau} \\ \vdots & \ddots & \vdots \\ CP_{\lambda 1} & \cdots & CP_{\lambda\tau} \end{bmatrix} \begin{bmatrix} UDR_1 \\ \vdots \\ UDR_\lambda \end{bmatrix} \begin{bmatrix} 0 & SD_{1,2} & \cdots & SD_{1,n} \\ \vdots & 0 & & \vdots \\ SD_{n,1} & & \cdots & 0 \end{bmatrix} \begin{bmatrix} SP_1 \\ \vdots \\ SP_n \end{bmatrix} \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix}$$

(a) (b) (c) (d) (e)

Fig. 3.2 - Input information to the NCaaS algorithms: a) The Profile–Cost matrix; b) The Upload/Download Rate matrix; c) The Service Dependency matrix; d) The Service Profile matrix; e) The Performance matrix.

- The Provider Profile Matrix (PP): this matrix provides three main records related to 1) the reputation rank of the provider, 2) the level of privacy supported, and 3) the eco-friendly compliance of the providers with green energy saving policies. Such metrics are mainly provided by trusted third parties as described in [23, 24]. In addition to the reputation, privacy, and energy metrics, the PP matrix specifies the geographical physical location of the corresponding providers' data centers.

3.1.2 Tenants Preferences and Constraints

The second form of input provided to the NCaaS VNet creation algorithms is a set of tenants' specifications and constraints. These are represented in three main matrices:

- The Service Dependency matrix (SD): The SD matrix designates the degree of dependency among the different network services. Dependency in this context reflects the amount of data communicated between the respective services. This matrix is an nxn symmetric matrix for a tenant predefined set of n services (S1, S2, …, Sn) as shown in Figure 3.2c. Each entry SDi,j in the service dependency matrix indicates the amount of network traffic exchanged between services i and j per unit time. SDi,j of zero value indicates no linking between services i and j. Thus, obviously, the values of the matrix main diagonal are set to zeros. Moreover, assuming that SDi,j is the same as SDj,i, renders the SD matrix symmetric.

- The VNet service profile matrix (*SP*): the matrix (Figure 3.2d) represents the profiles and resource requirements of the underlying VMs running the network services. SP_i is the service profile of the i^{th} tenant service. As indicated in Section 3.1.1, the service profile designates the hardware/software specifications of the VM such as the number of virtual CPUs and their speed, the amount of RAM available, the storage capacity, the operating system and supporting software, the network bandwidth, among others. NCaaS allocates network resources on various providers based on this matrix.

- The performance constraint matrix R: this matrix corresponds to the tenant specified services that require high performance and minimum network delay. The R matrix entries are set in the range $1 \leq R \leq 10$ where 10 represents high performance services and a rate of 1 indicates relatively lower service performance demands. The performance constraint matrix (Figure 3.2e) is used in the performance constraints and the topology creation algorithm as will be presented in Section 3.1.4.

In addition to the matrices described above, NCaaS presents a unified interface that prompts tenants to identify a set of network provider constraints as listed below:

- Location of physical provider sites according to tenant's region preferences as well as specifying tenants' black-listed provider set, if any.
- The minimum accepted provider's reputation rank (refer to [24] for more details on reputation ranking schemes in cloud computing).
- Energy requirements where the tenant may select NaaS providers with environmental friendly data centers, renewable energy options and energy consumption limits.
- The desired level of privacy on specific parts of the network (refer to [23] for more details on privacy level specifications).

3.1.3 NCaaS Partitioning Algorithm

NCaaS supports an adaptable VNet partitioning mechanism that provides flexibility in placing the different VNet services on different provider sites depending on the performance and cost requirements of these services. For instance, the online network services that require high performance qualities are placed on top-notch provider sites with relatively high service pricing, while the backup and archiving components can be placed on provider sites with lower service price. Partitioning also allows for the essential VNet services to be deployed on provider sites that are geographically closer to the tenant, which plays a role in enhancing the performance of these services. Moreover, VNet partitioning enhances the reliability and fault tolerance capabilities of the network by allowing the NCaaS service to replicate, clone, and snapshot only small slices of the VNet (such as those with critical online operation) and hence avoiding the expensive operation on the whole VNet. The NCaaS partitioning problem aims at assigning the set of *n* services composing the tenant VNet to a set of partitions on a collection of selected cloud providers. The next subsection presents a mathematical formulation for the NCaaS partitioning optimization problem by specifying the objective cost function and the set of constraints on this objective function. This is followed by devising a greedy approximation algorithm for solving this partitioning problem.

3.1.4 Cost Optimization Formulation and Objective Function

In this section, we formalize the objective function representing the overall cost of deploying the tenant's VNet services on the different cloud providers. The main objective is to minimize this cost while considering the tenants' constraints and providers' specifications. Using the set S of n services ($S_1 \rightarrow S_n$) and the set *PR* of τ cloud providers ($PR_1 \rightarrow PR_\tau$), the cost of mapping a service S_i with VM profile P_i to provider PR_k is defined as:

$$cost(i,k) = (CP_{i,k} + \sum_{j=1\,;j\notin provider\,k}^{n} SD_{i,j} \times (UDR_k + UDR_{prov(j)}) \qquad (1)$$

Equation 1 represents the cost of deploying service *i* on provider *k* in addition to the communication cost of service *i* with all the dependent services *j* as specified in the *SD* matrix. The function *prov(j)* returns a reference to the cloud provider hosting service *j*. Next, we define the binary constraint X_{ik} as:

$$X_{ik} = \begin{Bmatrix} 1 \;\; if\; S_i\; is\; deployed\; on\; PR_k \\ 0 \;\; otherwise \end{Bmatrix} \qquad (2)$$

The objective function of the optimization problem to be minimized is presented as follows:

$f(x) = \sum_{V} cost(S_i, S_j, PR_k, PR_{m=\text{prov(j)}}) \times X_{ik} \times X_{jm}$ (3)

Where $V = \{i, j, k, m | i < j , i \leq n, j \leq n, k \leq \tau, m \leq \tau\}$

Minimizing $f(x)$ will result in placing the services of the VNet on providers with minimum overall deployment and communication cost in addition to the following constraints:

$\sum_{(k,m)}^{1..\tau} X_{ik} \times X_{jm} = 1 \; \forall \; i, j \; in \; S$ (4)

Where $X_{ik}, X_{jm} \in 0,1$ *and k≠m*

Equation 4 indicates that service *i* can be hosted by only one provider at a time. The same applies for service j.

Another essential constraint is the performance constraint directly indicated by the performance constraint matrix *R* and influenced by the distance between the providers PR_k, PR_m hosting the connected services S_i, S_j:

$Distance(PR_k, PR_m) \times SD_{i,j} \leq \delta \times \max(R_i, R_j)$ (5)

where δ is a normalization constant that tunes the range in the *R* matrix to a suitable (distance × data dependency) factor.

This problem is a binary integer optimization problem, which can be reduced to a Multiple Knapsack problem [25] whose computational complexity is exponential in terms of the number of services and providers. The optimal solution of a Multiple Knapsack problem is usually obtained via branch-and-bound [26] techniques. Greedy algorithms are well-suited for approximating such type of problems in terms of computing times and storage resources.

In the next section, we propose the Network Partitions Creation algorithm, which is an approximate greedy algorithm for solving this NP-complete problem.

A sample example consisting of 4 services {s1, s2, s3, s4} and 3 providers {p1, p2, p3} is fed to the NCaaS optimization algorithm using the following matrix configurations:

CP: The profile-cost matrix: $\begin{bmatrix} 2.1 & 2.5 \\ 1.24 & 1.5 \\ 0.9 & 1.2 \\ 0.7 & 0.91 \end{bmatrix}$\$/hr

SP: The Service Profile matrix: $\begin{bmatrix} \mathrm{P1} \\ P2 \\ P3 \\ P4 \end{bmatrix}$

SD: The Service Dependency matrix: $\begin{bmatrix} 0 & 0.5 & 1.2 & 14 \\ 0.5 & 0 & 5.7 & 12 \\ 1.2 & 5.7 & 0 & 19 \\ 14 & 12 & 19 & 0 \end{bmatrix}$ GB/hr

UDR: The Upload/Download Rate matrix: $\begin{bmatrix} 0.07 \\ 0.1 \end{bmatrix}$ \$/GB

The NCaaS optimization algorithm produced the following solution:

Service to Provider mapping :

cost $/hr	p1	p2	p3
s1	4.486	2.355	4.4
s2	2.99	2.18	4.566
s3	2.073	4.43	4.892
s4	4.88	6.3	6.17

Based on the above costs, the service distribution among the providers is found as follows:

s1	p2
s2	p2
s3	p1
s4	p1

The resulting minimum cost based on the above service allocation is 11.488$/hr. To compare the above solution to the optimal solution, we generated all the 64 possible service to provider allocations using a Matlab script, calculated the cost of each allocation, and picked the allocation with the minimum total cost. The optimal solution found for this specific problem is equal to that generated by the optimization algorithm. These results prove that the Multiple Knapsack approximation is fit to the optimization problem. The results were compliant due to the relatively small problem size. Applying the NCaaS optimization algorithm on larger problem sizes should consequently provide a near optimal solution [25].

3.1.5 Network Partitions Creation Algorithm

NCaaS utilizes provider data (Section 3.1.1) along with the tenant VNet specification and constraints (Section 3.1.2) in order to achieve the minimum cost objective function presented in Equation 3.

To serve this goal, the NCaaS's Network Partitions Creation algorithm starts in phase 1) by short listing the set of candidate cloud providers based on the *PP* matrix and tenants requirements. The algorithm proceeds by checking providers' offers in the resource *CP* matrix against the tenant specified *SP* matrix to map each of the *n* VNet service to the provider of minimum cost satisfying the corresponding resource profile. At the end of this phase, all the services of the respective VNet would be arranged in partitions that are mapped to minimum cost providers. In phase 2) and to achieve minimum cost, the highly connected services are rearranged on the set of partitions to satisfy minimum connection cost among providers while ensuring tenants' specified performance criteria. Thus, the tenant specified services' dependency matrix, along with the provider upload/download cost rate, are utilized to achieve the minimum cost objective. The pseudo code of the Network Partitions Creation algorithm is presented below:

Algorithm 1: Network Partitions Creation Algorithm

Phase 1 Minimum cost profile selection:

Step 1

Select the candidate provider list based on the tenants' constraints on the providers' qualifications in terms of reputation, energy, physical location and privacy.

Step 2

Create the Profile-Cost Matrix (CP) with rows corresponding to *p* providers and columns to *n* services where $CP_{i,j}$ is the cost of deploying service *j* on provider *i*.

Step 3

Create the Partitions matrix *PT* of dimension $\tau \times n$. Each row *i* in the *PT* matrix indicate the set of services selected to run at provider *i*. At this stage, this selection is merely based on the minimum cost offered by the various cloud providers for different service profiles. Effectively, each row *i* in the PT matrix represent a network partition to be deployed at the i^{th} cloud provider site.

Phase 2 Achieving least inter partition communication cost:

Step 4

Create dependency hash table *D [1→n]*. Each entry D_k in *D* represents the amount of dependency that service *k* has with the other $(n-1)$ services in the network. To calculate D_k we utilize the *SD* matrix as follows:

```
for i from 1→n
    for j from 1→n
        if j ≠ i
            $D_i$ += $SD_{i,j}$
```

Step 5

Sort *D* in decreasing order to start with the service of maximum connectivity influence on the other services

```
for each service i in D
```

```
    Create a cost vector V [1→t] where each entry $V_k$
    represents the cost of deploying service i in
    partition k added to the cost of interconnection
    between service i and the rest of the services in
    the other partitions.
  for each k in PT
    $V_k = CP_{k,i}$
  for each service j in SD excluding those in partition k
    if $Distance(k, Partition(j)) \times SD_{i,j} \geq \delta \times \max(R_i, R_j)$
        increment k (change partition)
        Break
        $V_k$ +=$SD_{i,j} \times UDR_k$
Step 5
Find the entry of minimum cost in V. The entry's index
represents the optimum partition/provider for
running service i.
```

3.1.6 Topology Generation

After running the partitioning algorithm, the VNet is divided into partitions each of which contains a set of the VNet services to be deployed on a particular cloud provider. The topology creation algorithm arranges the services on the provider site in a fat-tree-based topology (popular in today's data centers), which involves a branching factor *EBR* that is inversely proportional to the number of ports involved in the topology creation. *EBR* is the ratio of the number of edge switch ports that are connected to servers to those connected to core switches. The minimum value for *EBR* is 1, which indicates that there is a dedicated link connecting the edge switch to the core switch for each node in the network. This directly contributes to minimizing the data transfer delay between the services across the edge and core switching layers. To fulfill the performance constraints specified by the tenant and represented in the performance matrix *R* (refer to Section 3.1.2), the maximum performance value R_{max} of the services in the partition is used to calculate *EBR* according to the following equation:

$$EBR = trunc(\ \alpha\ R_{max}) \quad (6)$$

Where α is a normalization factor that bounds the *EBR* value to a predefined range.

3.1.6.1 Topology Creation Algorithm

This section presents the base algorithm for creating each VNet partition topology. The parameters used in the topology creation are listed below along with the topology creation algorithm:

1. ***k***: number of edge switches.
2. ***NE***: number of ports per edge switch.
3. ***NC***: number of ports per core switch.
4. ***NP*** is the total number of servers in a partition.
5. ***ES***: maximum number of edge switch ports to connect to servers deduced from *EBR* by:

   ES=trunc(NE×EBR/(1+EBR)).

6. ***EC***: number of edge switch ports to connect to core switches obtained from ES such that: *EC=NE-ES.*
7. ***B***: maximum number of ports that can connect to each edge switch where *B = ceiling(NC/k)*.

Algorithm 2: Topology Creation Algorithm

Phase 1 Initialization:

```
Compute $EBR = trunc(\alpha\ Rm)$
Set Scounter = 0  //equivalent to the number of services connected
ES = trunc(NE×EBR/(1+EBR))
Set k =1        // (edge switch counter)
```

Phase 2 Creation of edge switches and their corresponding port connections:

```
Let i = Scounter
Create edge switch $ES_k$
While Scounter ≤ ES and Scounter ≤ NP
    if $S_i$ is not connected, connect it to $ES_k$
        increment Scounter, increment i
        for each j ≤ NP and j>i
            find j with maximum $SD_{i,j}$
            if $S_j$ not connected, connect it
                increment Scounter
                If Scounter < NP then increment k   //
                                    //add another edge switch
        Goto Create edge switch
```

Phase 3 Creation of the core level

```
Set c=0 as the core switch counter
EC=NE-ES
B= NC div k
if B ≥ EC then increment c, create core switch $CS_c$ and connect each edge switch to it via EC ports
else
    let r = EC
    while r ≥ B
        increment c, create core switch $CS_c$ and connect each edge switch to it via B ports
        decrement r by B : r = B - r
        if r > 0
            increment c, create core switch and connect to each edge switch via r ports
            Redistribute edge switches over core switches by EC/c connections per edge switch
```

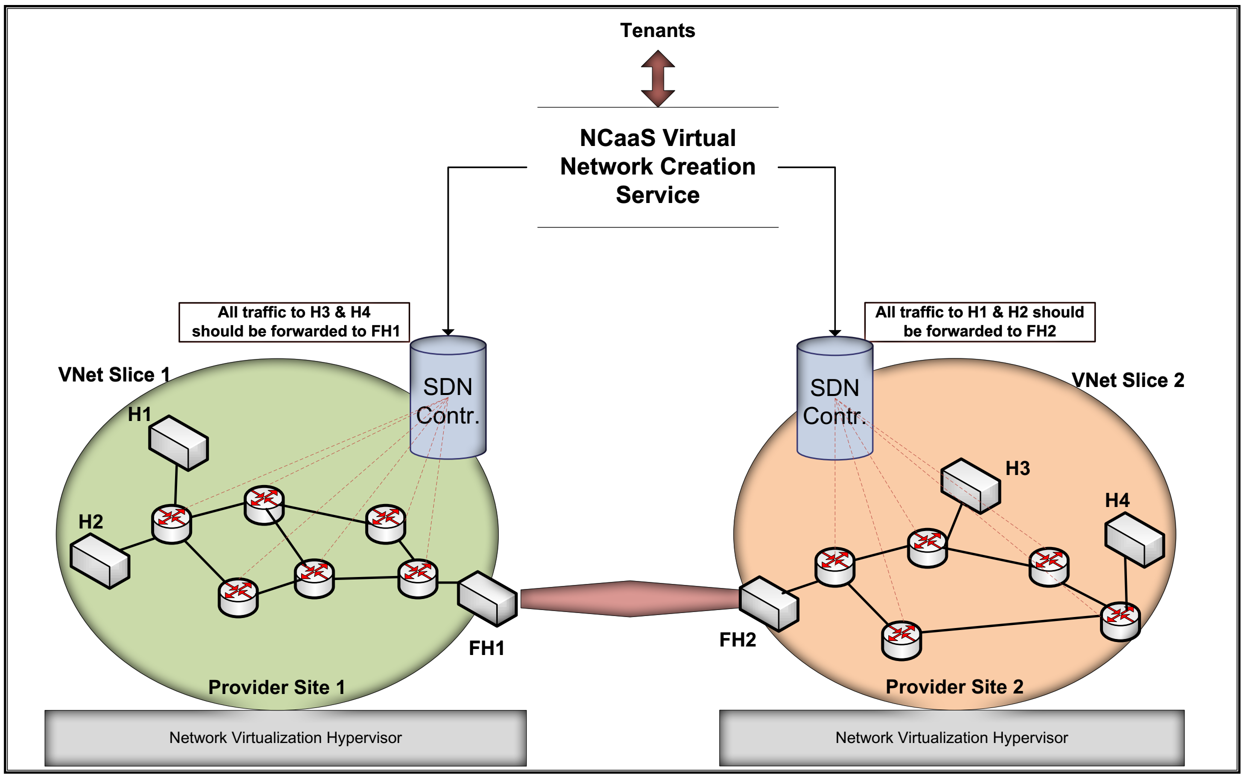

Tenants
NCaaS Virtual Network Creation Service
All traffic to H3 & H4 should be forwarded to FH1
All traffic to H1 & H2 should be forwarded to FH2
VNet Slice 1
VNet Slice 2
SDN Contr.
SDN Contr.
H1
H2
H3
H4
FH1
FH2
Provider Site 1
Provider Site 2
Network Virtualization Hypervisor
Network Virtualization Hypervisor

Fig. 3.3 - Partitions connection algorithm demonstration

3.1.7 Partitions Connection Algorithm

Using the SDN network programming features, VNet partitions can be easily joined to provide a complete VNet view to the tenant. The following example (see Figure 3.3) demonstrates how multiple VNet partitions implemented on possibly different provider sites can interact to provide the tenant with a single compositional VNet. Without loss of generality, the example demonstrates the algorithm for achieving connectivity between two VNet partitions, VNet partition 1 and VNet partition 2 consisting of two hosts each.

For functionally joining the two VNet partitions we use two forwarding hosts FH1 and FH2 at VNet partition 1 and VNet partition 2, respectively. At VNet partition 1, the SDN controller sends the rule “forward all traffic to H3 & H4 to FH1”. Similarly, at VNet partition

2, the SDN controller sends the rule “forward all traffic to H1& H2 to FH2”. The FH1-to-FH2 link can be implemented as an IP tunnel as follows: All frames arriving at FH1 are encapsulated in respective IP packets and sent to FH2 and vice versa. All the packets received by FH2 are de-capsulated and the resulting frames are forwarded to their designated hosts. Analogously, all traffic received by FH1 is de-capsulated and forwarded to its destination. With this procedure, the tenant will view a single VNet composed of the corresponding partitions.

To prevent any performance bottlenecks due to the utilization of a single forwarding host, the Partitions Connection Algorithm can be extended by adopting more than one forwarding host per partition. Communication with other services (residing on other partitions) in the network is assigned equally among the available forwarding hosts in the source partition. In other words, the SDN controller will send the forwarding rules to the network switches to divide the outgoing load among the forwarding hosts based on the destination service receiving the traffic. In the same sense, the encapsulation algorithm on the source forwarding host is configured to forward the traffic equally among the forwarding hosts in the destination partition. This is also based on the recipient service receiving the traffic in the destination partition. That is, the source forwarding host is configured to assign the set of services deployed in the destination partition among the set of forwarding hosts available in that partition. This ensures fair distribution of workload among the forwarding hosts on the source as well as on the sink sides.

3.1.8 NCaaS Implementation

The NCaaS algorithms are fully implemented using Mathworks Matlab 2014 [27]. To demonstrate a full-fledged functionality of the NCaaS service and to test for the fulfillment of the NCaaS objective function of minimum cost as the systems scales, we simulated three main prototype cloud configurations. Configuration 1 consists of 5 providers and 10 services, Configuration 2 consists of 7 providers and 14 services, and Configuration 3 consists of 14 providers and 28 services. The NaaS architecture utilized is the OpenVirteX network virtualization platform. This choice is due to the great flexibility provided by OpenVirteX in terms of address space isolation, topology specification, and dynamic network reconfiguration at runtime.

Table 3.1 Profile offers based on real provider pricing data

Profile Offer ($/h) Range	1	2	3	4	5	6	7	8	9
Min	0.005	0.015	0.025	0.05	0.056	0.13	0.26	0.51	1.25
Max	0.013	0.026	0.053	0.104	0.126	0.252	0.504	1.008	2.53

OpenVirteX is installed on a VirtualBox [28] VM and runs on top of the Mininet network emulator [54]. For each VNet partition, the NCaaS algorithms ultimately produce a .JSON file that formally describes the VNet partition topology. The partitions' files are fed to the OpenVirteX network embedder to automate the process of mapping the respective VNet partition onto the physical data center network. It is worth mentioning here that NCaaS may in theory rely on any available NaaS services. The only requirement here is to feed the NCaaS Topology Creation algorithm with the standard network deployment interface used by the respective NaaS services. The matrices representing the tenants and providers input (refer to Section 3.1.1) are populated with reasonable values for each experiment run (40 runs in total for each of the three configurations). The most important matrices fed into the NCaaS algorithms are populated as follows:

1. The *CP* matrix profile values are randomly generated in the ranges specified in Table 3.1 which are based on real pricing values extracted from the websites of the top 12 cloud providers in the IaaS market.
2. The *UDR* matrix values are randomly generated in the range \$0.03/GB to \$0.11/GB. This range is based on real data transfer rates extracted from the websites of 6 cloud providers.
3. The *SD* matrix entries linking communicating services are populated with assumed values in the range of 100 KB to 24 MB per hour.
4. The *SP* matrix is initialized with reasonable profile settings for the services in the three simulated configurations.
5. The *R* matrix is instantiated with random values in the range of 1 to 10 (this is based on the specification of the *R* matrix in Section 3.1.1).

For the performance constraint implementation, the physical distances among providers are randomly generated in the range 1 to 20. These distances are implemented as the weights on the links connecting the partitions in the network configuration files. To achieve the required normalization level, the normalization constant δ used is 48.

For each of the three prototype configurations, the cost of deploying the entire tenant's VNet on a single cloud provider is computed for the whole set of candidate providers. These costs are compared to the cost of deploying the same VNet using the partitioning algorithms of the NCaaS service. On all the tested cloud configurations, the NCaaS service always produced the minimum cost in comparison with single NaaS provider deployment while satisfying the tenant's constraints and requirements. The results attained for configurations 1 – 3 are respectively presented in Figures 3.4, 3.5, 3.6. On average, the cost saving achieved for the first configuration (5 providers, 10 services) is 26.58%, for the second configuration (7 providers, 14 services) is 30.68% and for the third configuration (14 providers, 28 services) is 37.62%. This demonstrates the scalability of the NCaaS algorithms in increasing the cost savings as the problem size increases. These results are presented in Figure 3.7.

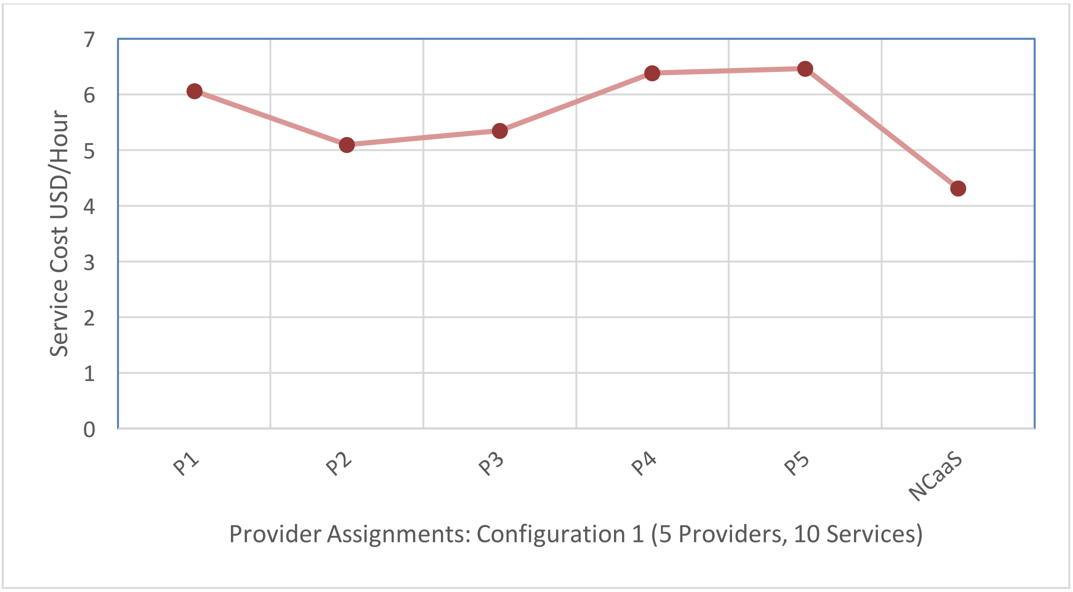


Fig. 3.4 - Cost (\$/h) achieved in the first configuration when deploying the VNet services on the candidate providers and when applying the NCaaS algorithms.

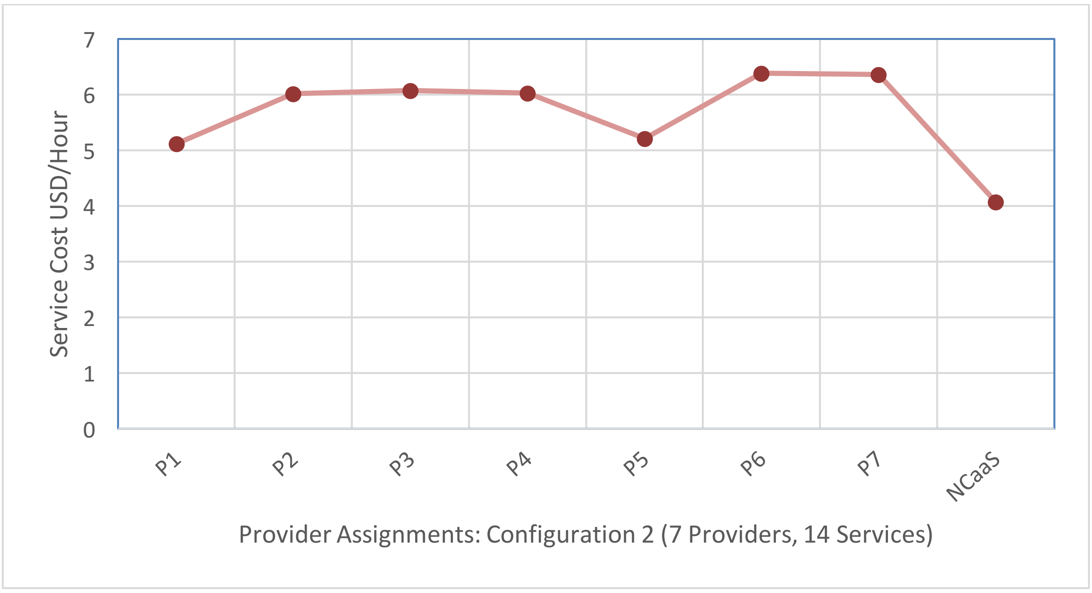


Fig. 3.5 - Cost ($/h) achieved in the second configuration when deploying the VNet services on the candidate providers and when applying the NCaaS algorithms.

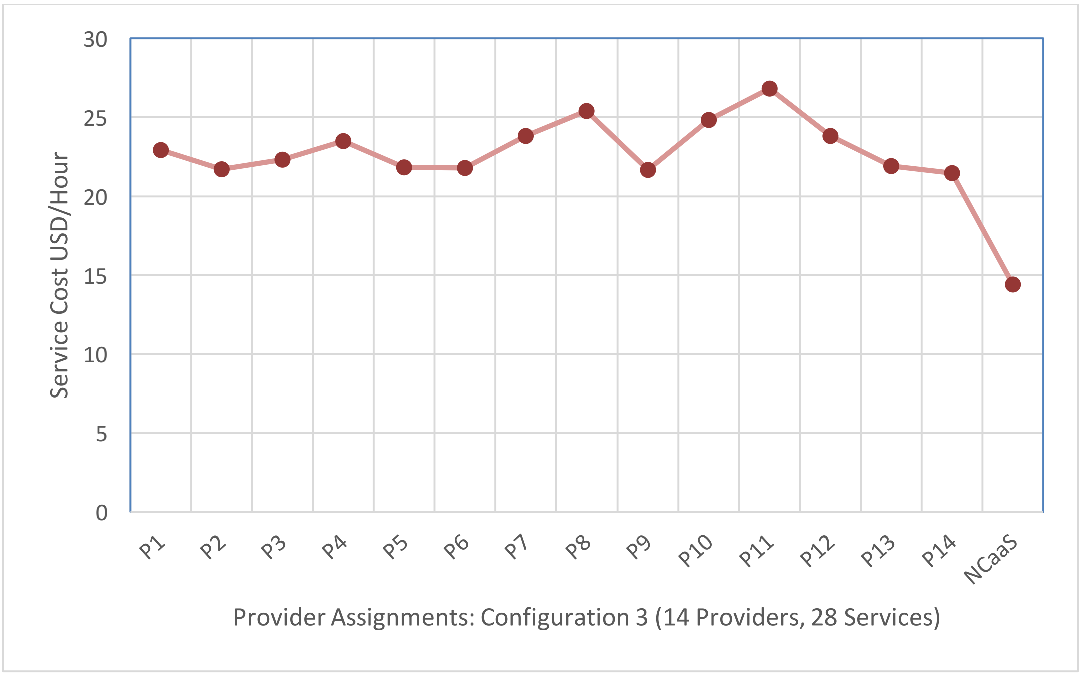


Fig. 3.6 - Cost ($/h) achieved in the third configuration when deploying the VNet services on the candidate providers and when applying the NCaaS algorithms.

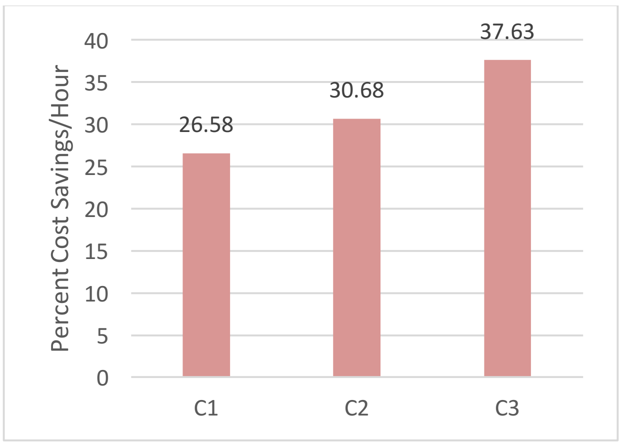


Fig. 3.7 - NCaaS Cost savings per hour achieved when deploying the three VNet configurations.

3.2 Virtual Network Connectivity As A Service Using A Software-Defined Networking Approach

The VNCS service creates and manages networks that efficiently connect tenant remote sites using the vISP SDN-based network providers' infrastructure and presents the tenant with a unified and reliable virtual network which satisfies its constraints and preferences while periodically upgrading and updating the underlying SDN networks according to providers' offers and tenants' service agreement purviews. To achieve this level of network creation flexibility with remarkably personalized features, VNCS algorithms provide a centralized interface to input tenant's network specifications, constraints, preferences, and restrictions on one hand, and SDN providers' offers and their corresponding network features and updates on the other hand. Figure 3.8 illustrates VNCS's network connectivity mechanism.

3.2.1 Tenant Network Design Constraints

The first step in designing the vSDN network, is to set the tenant network information and constraints, which are represented in the following matrices:

a) *Location indices matrix (LI)*: this matrix represents the tenant's branches' geographic locations where each entry LI_i is the location index of branch i. The LI matrix is used in the initial phase of the VNCS network creation algorithm in selecting the candidate SDN providers' sites that can connect each branch.

b) Maximum number of interconnecting hops matrix (MH): in this matrix, the tenant specifies the maximum permitted number of interconnecting SDN provider sites $MH_{i,j}$ between each pair of the tenant's branches i and j. The VNCS service uses this value to restrict the number of connecting hops as a performance measure.

c) Branches connectivity level matrix (CL): this matrix designates the degree of connectivity among the different network branches. This matrix is a $\tau \ x \ \tau$ symmetric matrix for a tenant predefined set of τ branches $(B_1, B_2, \dots, B_\tau)$. Each entry $CL_{i,j}$ indicates the amount of network traffic exchanged between branches i and j per unit

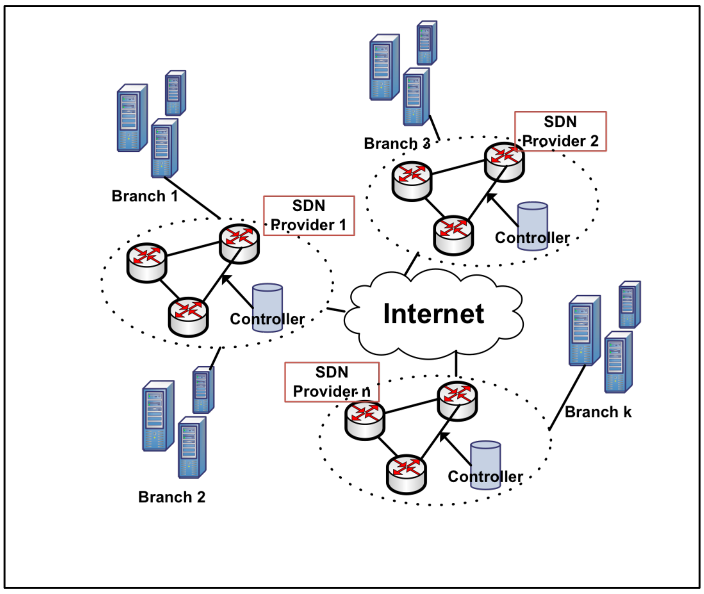


Fig. 3.8 - VNCS network connectivity mechanism.

time. Obviously, the values of the matrix main diagonal are set to zeros.

d) *Performance factor matrix (P):* this performance information is specified by the tenant based on the connectivity and dependency level between branches in the enterprise. $P_{i,j}$ is the communication performance level reflecting the network delay that is requested for the path connecting branches *i* and *j*. This constraint must be satisfied in the VNCS network creation algorithm.

e) *Reliability factor matrix (R)*: in this matrix, the tenant specifies the communication reliability level requested from the VNCS service for each branch. This reliability constraint matrix must be satisfied in the VNCS network creation algorithm, through redundant connecting paths on one hand, and the SDN providers' specified number of redundant communication paths they offer in each of their connecting sites on the other hand.

f) *Security index matrix (SI)*: this matrix indicates the levels of communication security between different branches of the enterprise on one hand and the physical location restriction of the connecting SDN sites and black listed providers on the other hand. The value $SI_{i,}$ is an index that represents the security index required for branch *i*.

g) *Geographic distance matrix (D)*: this matrix represents the distance between the tenants' branches where $D_{i,j}$ is the geographic distance between branches *i* and *j*. This distance is an essential constraint in the VNCS network creation algorithms where SDN providers are selected in order to minimize latency and meet the tenant's performance constraints.

h) *Provider reputation level (RpI):* in this matrix, the tenant indicates for each branch *i*, the SDN network providers' reputation and ranking index RpI_i that is required. *RpI* is significant in the selection of SDN network providers in the VNCS service algorithms.

In addition to these essential tenant specified input matrices that are utilized in the VNCS network creation process, the tenant can personalize the behavior of the VNCS service in terms of the network update process according to seasonal changes, network components migration feasibility and its timing, and the service occasional feedback information and its frequency.

3.2.2 SDN Network Provider Information

The VNCS service optimizes the tenant network creation by ultimately utilizing information that is periodically gathered from SDN network providers. This information involves pricing, available connecting sites, reliability network specification, and geographic location of the SDN network infrastructure that can be delivered to the VNCS service. The following matrices define the information input from SDN providers:

a) *Connectivity cost matrix (C)*: every SDN network provider should provide the VNCS service with the location of the areas that it connects and the corresponding connection sites' costs. For provider *i* and site *j*, the value $C_{i,j}$ corresponds to the cost of the connecting site given by an SDN network provider.

b) *Site spanned connected area* (*SA*): for a provider *i* and site *m*, $SA_{i,m}$ represents the geographic area that site *m* can connect. This matrix is used by the VNCS algorithms in the step of selecting the candidate providers that can connect the tenant sites according to their locations.

c) *Site geographic specification matrix (GS)*: this matrix represents the specific geographic location $GS_{i,j}$ of the sites j network infrastructure that are used by the SDN provider i in the network connecting service. $GS_{i,j}$ matrix is used by the VNCS service to meet the security measures and restrictions that are identified by the tenant in matrix *SI*.

d) *Upload/Download Communication Cost Matrix (UDC)*: this matrix represents the upload/download data rates $UDC_{i,j}$ offered by providers i for site j.

e) *Site Reliability factor (SR)*: for each given provider k, the matrix values SR_i represent the reliability and redundancy measures it provides for each of its available connecting site infrastructure i. The *SR* values of the candidate sites contribute in accomplishing the tenants' reliability requirements and constraints that are set in matrix *R* in Section 3.2.1.

3.2.3 VNCS Branch Connection Mechanism

The VNCS service provides tenants with a completely adaptable and personalized VNets that fully connect their geographically separated remote sites exclusively using SDN infrastructure. The VNCS adaptable network creation mechanism is achieved by the Network Distribute and Bond algorithm (NDB) that enables it to flexibly deploy, replicate and migrate parts of the connecting networks among different SDN providers. In the next subsection, a mathematical formulation of the VNCS distribute and bond network optimization problem is described. This is followed by developing a greedy algorithm that approximates the VNCS optimization solution.

3.2.4 Problem Optimization Formalization

The problem formulation of the VNCS network creation is described in this section. The VNCS objective function is to minimize the overall cost of connecting tenants' sites by configuring the connecting VNets on different vISP providers. Let the set *B* represent the τ branches ($B_1 \rightarrow B_\tau$) of the tenants' enterprise that are input to the VNCS service. The set V*P* represents the φ candidate vISP network providers ($VP_1 \rightarrow VP_\varphi$) each having a set of sites *S*, the cost of connecting branch B_i to VP_k in the VNet is defined by:

$$cost(i,k) = C_{i,k} + \sum_{b=1\,;b\neq i\;at\;prov(b)\neq k}^{\tau} CL_{i,b}\times(UDC_k + UDC_{prov(b)}) \quad (7)$$

Where $C_{i,k}$ represents the VP_k's cost of connecting branch *i* to provider *k* and *prov(b)* is the provider that branch *b* is connected to. The connectivity level $\left(CL_{i,b}\right)$ between branch *i* and the rest of the branches in the VNet is multiplied by the upload/download costs of involved providers.

To formulate the objective function of minimizing the VNet connection cost and fulfilling the tenant's constraints, consider the following binary variable X given by:

$$X_{ik} = \begin{Bmatrix} 1 \;\; if\; B_i\; is\; connected\; on\; P_k \\ 0 \;\; otherwise \end{Bmatrix} \quad (8)$$

The optimization problem objective function *f* is thus given by (9) which represents the cost of vSDN deployment for the tenant's branches' communication, such that all the feasible sites' deployment for all possible combinations of the tenant's branches are considered in a pairwise calculation of the branches' connection costs (7). The problem is to minimize *f(x)*:

$$f(x) = \sum_{\partial,\emptyset} cost\big(B_i, B_j, VP_k, VP_p\big) \times X_{ik} \times X_{jp} \qquad (9)$$

where B_i and B_j represent all possible pairs of branches in the VNet that are deployed on VP_k, and VP_p respectively where p can be equal to *k*. ∂ and $\emptyset$ represent the sets of all possible combinations (10, 11) with the constraint that a branch can be directly connected to one SDN provider at a time (12):

$$\partial = \{i, j, | i < j, i \leq \tau, j \leq \tau\} \qquad (10)$$

$$\emptyset = \{k, p | k \leq \varphi, p \leq \varphi\} \qquad (11)$$

$$\sum_{(k,p)}^{1..\varphi} X_{ik} \times X_{jp} = 1 \; \forall \; i, j \; in \; B \qquad (12)$$

where $X_{ik}, X_{jp} \in 0,1$

Along with minimizing the objective function, the following set of constraints has to be satisfied:

a) *Geographic-security constraint*: this is the security constraint for connecting branch B_i on site S_g of provider *p*, which states that, S_g geographic location should satisfy the security restrictions for branch B_i in the SI matrix: $Location\big(S_g\big) \notin \delta(SI_i)\,, and\; X_{ip} = 1$

where δ is the geographic-restriction factor in the SI entry.

b) *Distance-performance constraint*: the performance requirement $P_{i,j}$ on the connection between each pair of branches i and j has to be met by restricting a threshold on the ratio of the distance between the sites S_m and S_n at providers p and k that are hosting the branches to distance $D_{i,j}$ between the branches:

$$Distance(\, S_n, S_m) \times \sigma \leq P_{i,j} \times D_{i,j} \;\; and \;\; X_{ip} = X_{jk} = 1$$

where σ is a normalization factor that tunes the computed distance value to the range of the matrix *P* values.

c) *Area constraint*: this restraints the set of possible sites that can be assigned to connect the geographically separated tenant branches. This constraint is established by utilizing the tenant's *LI* matrix and the provider's *SA* matrix:

$$LI_i \subset SA_m \; and \;\; X_{ip} = 1$$

$$where \; site \; m \; \in provider \; p$$

d) *Number of hops constraint*: this constraint is indicated in the tenant *MH* matrix and should be satisfied when connecting each branch i to site m on provider *p*:

$$num\_of\_hop(\, S_m \longrightarrow S_n) \leq MH_{i,j} \; and \;\; X_{ip} = X_{jk} = 1$$

$$where \;\; sites \; m \in provider \; p \; and \; site \; n \; \in provider \; k$$

Ultimately, this problem is an integer optimization problem, which is reduced to a multiple knapsack problem [32]. This is an np-complete problem having a computational complexity of exponential order in terms of the number of branches and sdn providers. To solve this problem, taking into consideration the computational power and resources limitations involved in the proposed service, a greedy algorithm, distribute and bond, is devised. This algorithm (illustrated in the following section) approximates an optimal solution to the VNCS problem.

3.2.5 Network Distribute and Bond Algorithm

VNCS distribute and bond algorithm starts connecting the tenant branches by optimally selecting SDN providers' sites to achieve minimum cost and at the same time meet network performance and behavioral constraints. The distribute and bond algorithm starts in phase 1 by finding the set of all feasible sites *(GF)* that each branch can connect to by taking into consideration the branch's geographic position, security, and performance constraints. In phase 2, the branches are ordered in decreasing order of expected communication traffic. The algorithm starts connecting the branches with higher network traffic levels. In phase 3, to connect a branch, the vSDN deployment and communication costs of connecting it to each feasible provider site in *GF* is computed. The algorithm finds the total minimum cost SDN provider site and at the same time satisfies the distance-performance constraint and the maximum allowable number of hops (*MH*) to each branch in the VNet. After mapping each branch to the corresponding SDN provider site, the cost of that SDN provider site is decremented to allow for adjustable incremental costs offered by the respective SDN providers. The algorithm is detailed below:

Algorithm 3: Network Distribute and Bond Algorithm

Phase 1 SDN feasible site selection for the VNet branches:

Step 1

Select the candidate provider list based on the tenants' constraints on the providers' qualifications in terms of reputation, reputation index (*RpI*), and physical location. To identify the respective providers' sites, matrix *ID* is generated where each entry $ID_{k,x}$ represents a unique identifier that designates site *x* at provider *k*.

Step 2

Create *GF,* the geographically feasible sites matrix for each branch *i* that consists of the providers' sites whose connectivity spanning area *SA* complies with the branch location LI_i and SI_i.

```
    for k from 1 → φ
        for x from 1 → sites(k)
            if $LI_i \in SA_{k,x}$
                if $SA_{k,x}$ complies with $SI_i$
                    $GF_{i,k}$ = site $ID_{k,x}$
```

Phase 2 find the branch with maximum communication traffic:

Step 3

Construct the vector *T [1→ τ]* of expected network traffic size (connectivity level) to each branch in the VNet where each entry T_i represents the total amount of dependency that branch *i* has with the other $(\tau - 1)$ branches in the network. To calculate T_i we utilize the *CL* matrix as follows:

```
    for i from 1 → τ
        for j from 1 → τ
                $T_i$ += $CL_{i,j}$
```

Sort *T* in decreasing order to connect the branches starting with the branch of maximum connectivity influence on the other braches in the VNet.

Construct the connection branch matrix *CX* of size $\tau \times \tau$, each element $CX_{m,n}$ represents the list of provider sites that connect branches *m* and *n*. This matrix is initialized to *0*. Construct matrix *SB* that indicates the site-branch mapping, where each entry SB_g corresponds to site *g* id linked to a vector of the *h* branches that connect to it.

Phase 3 Connect the branches T_i to the minimum cost provider:

Step 4

Start with the maximum T_i branch and find in the GF matrix the provider site that results in minimum connection cost for branch *i*:

Create a cost vector *CV* where $CV_{i,k}$ represents the cost of the VNet deployment for branch *i* on site $GF_{i,k}$ added to the cost of the interconnection between branch *i* and the rest of the network branches depending on the connectivity level matrix *CL* excluding branches that are connected to the same provider site and abiding with the performance and distance constraints:

```
    for each $GF_{i,k}$ at provider p site l
        $CV_{i,k} = C_{p,l}$
        for j from 1 → τ
if $Distance(S_l, CX_{j,j}) \times P_{i,j} \geq \sigma \times D_{i,j}$
```

```
            $CV_{i,k}$=MAX, break
            $CV_{i,k}$ += $CL_{i,j} \times UDC_{p,l}$
**Step 5**
Find the minimum of the cost vector $CV_i$ and check if the number of hops involved in connecting branch $i$ with the current connected branches (in the CX matrix) is compliant with the maximum number of allowed hops in the $MH$ matrix. If not, continue choosing the next minimum until the hop constraint is satisfied.
Map branch $i$ to the corresponding provider site, which thus provides minimum cost. Update the $CX_{i,i}$ with the provider site that is assigned to branch $i$. Add the branch-site mapping to the $SB$ matrix and decrement the provider's site cost in the $C$ matrix according to the following formula:
        $C_{p,l} = (\propto +(n-1)\times\beta)\times inital\_site\_cost$
where:
$\propto$ is a percent of the initial cost offered by provider p for connecting a branch to site $l$.
$\beta$ is the incremental charge factor for connecting subsequent branches to the same site $l$
$n$ is the number of branches connecting to site $l$.
**Step 6**
Update the total NVet price by adding the total cost of connecting branch $i$, $CV_{i,k}$.
**Step 7**
If a new provider was selected, connect it to the closest provider site in the resulting VNet by applying the Inter vSDN Connection algorithm described in the next section.
**Step 8**
The next branch $i$ to connect to the VNet is the next item in the $T$ vector which is the next maximum connected branch (step 3). Go to **step 5**
```

3.2.6 *Inter vSDN Connection Algorithm*

The NDB algorithm postulates that the sites hosting the vSDNs will be connected in order to deliver a connected graph topology of the tenant's branches. The tenant must be provided with a complete network view of the connected branches transparent to the underlying SDN provider sites. In order to connect the vSDNs hosting the branches, the SDN network programming feature is utilized by developing a routing application on top of the controllers of each vSDN site. The site connection algorithm uses tunneling to connect the branch vSDNs that are deployed on different SDN provider sites. For instance, two tenant branches B_1 and B_2 whose communication vSDNs are deployed on sites S_1 and S_2 are joined as follows: data communicated from branch B_2 to branch B_1 is routed to the routing application running on the controller of B_2 which encapsulates the frame into an IP packet and sends it to the vSDN controller of S_1. At S_1, another controller application decapsulates the packet and delivers the frame to the designated branch B_1. It should be noted here that we are utilizing the SDN controller at the provider site to route the inter-branch traffic. This design choice leverages the specifics of the SDN architecture to handle the routing mechanisms across the branches, though dedicated routing hosts can be feasibly employed to handle this task.

3.2.7 VNCS Implementation

The VNCS NDB algorithm is implemented using Matlab. To test the efficiency and convergence of the NDB algorithm, we implemented 5 network configurations as shown in Table 3.2.

Table 3.2 Network configurations simulated

Configuration	**# of Branches**	**# of SDN Provider Sites**
1	20	5
2	50	10
3	100	20
4	150	25
5	300	30

The C matrix of the providers' sites' connection costs are set to random values in the range of \$20 to \$90 per month. The UDC matrix values are also set randomly in the range \$0.05/GB to \$0.14/GB (this is based on a survey on actual ISP rates in the US) [33]. The CL matrix of the expected communication levels among the branches is set in the range 100 KB to 24 MB per hour. The values of $\propto$ and β in the $C_{p,l}$ formula in Step 5 of the NDB algorithm are respectively selected in the range of (0.1 to 0.5) and (0.1 to 0.3). The site/branch distances are selected in the range of 0.5 to 20,000 Kms. P and R, the performance and reliability values are set randomly in the range (1 to 5) and σ was set to 5.5 which means that the maximum distance allowed between the sites hosting branches with maximum performance communication constrains (P=5) is 0.1 of the branches' separating distance; for minimum performance values (P=1) the maximum allowed site distance is 5 times the branches' one. The SA and LI matrices were assigned values from an array that was randomly initialized to adequate values. For each cloud configuration in Table 3.2, the NDB algorithm is executed on ten different sets of random matrix values (in the ranges described above). All the NDB simulations output the representation of the VNet connectivity graph of the tenant SDN connected branches. We measured the convergence time of this algorithm on a 1.3 GHz Intel Core i5 MacBook Air laptop supported with 4 GB of RAM. The results are plotted in Figure 3.9. The convergence time increases with the increase in the number of branches and SDN providers which is anticipated since these parameters represent the NDB problem size and hence source of complexity. The total cost of the resulting VNet deployment on the selected SDN providers using the NDB optimization algorithm is compared to that resulting from the unoptimized solution where the different branches are connected to the same ISP (the average of three providers was considered). The percent savings are presented in Figure 3.10.

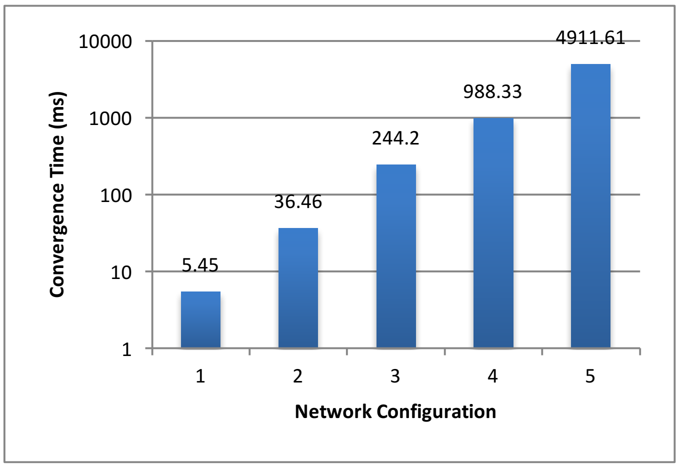


Fig. 3.9 - Convergence time of the DNB algorithm.

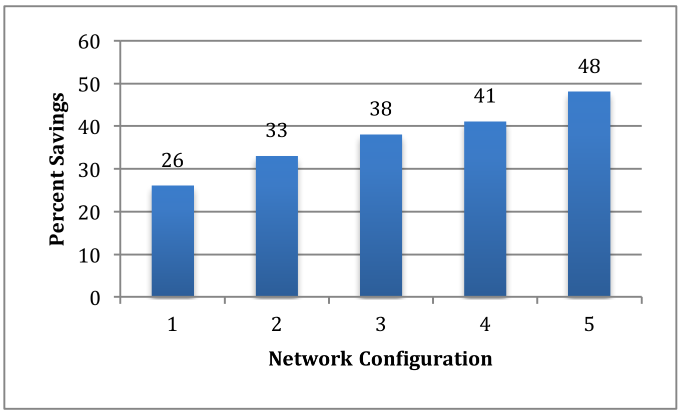


Fig. 3.10 - Percent savings achieved over the unoptimized solution.

The VNCS architecture utilized is the OpenVirteX [3] network virtualization platform. This choice is due to the great flexibility provided by OpenVirteX in terms of address space isolation, topology specification, and dynamic network reconfiguration at runtime. OpenVirteX is installed on a Virtual machine and run on top of the Mininet network emulator [53]. Based on the OVX network mapping mechanisms, the VNets respectively representing the branch and site networks are described using a .JSON representation and fed to the OpenVirteX network embedder to automate the process of network mapping in Mininet.

Future implementation extensions include: (1) enhancing the system model to support the online/non-disruptive migration among SDN providers, (2) dynamically upgrading/degrading the leased SDN provider's resources based on the load or traffic intensity, and (3) augmenting security protocols to support the confidentiality, authentication, and integrity of network traffic exchanged among the enterprise branches and across the SDN providers' sites.

CHAPTER 4

NETWORK SECURITY IN SDN AND MEC ENVIRONMNETS

This chapter presents the design and implementation of security mechanisms that enhance the security of networks that comprises netwrok programmability components. The significance of the presented services and mechanisms extends to NaaS services reliability and secure agility which is a major requirement for future communication and networks.. Section 4.1 presents the design and implementation of a Cloud security service for detecting malicious switching elements in SDN virtualized environments in realtime while essentially building on network slicing and programability. Section 4.2 introduces a secure authentication protocol for edge computing using progrmmable networking platform and wireless network virtualization. The prospposed protocol provides a more secure edge environment for Fog providers and clients by targeting the major problem of rogue Fog nodes. The proposed authentication protocol supports the well-being of future edge/cloud service implementations.

4.1 Network Programming and Probabilistic Sketching for Securing the Data Plane

In this section we present, VISKA, a cloud security service for dynamically detecting malicious switching elements in software defined networking (SDN) infrastructures. The main contributions of VISKA lie in: 1) utilizing network programming and secure probabilistic sketching in SDN environments to dynamically detect and isolate parts of the data plane that experience malicious behaviour, 2) applying a set of focused packet probing and sketching mechanisms on isolated network partitions/views rather than focusing the security mechanisms on the whole physical network, 3) efficiently analyzing the network behavior of the resulting views by recursively partitioning them in a divide-and-conquer fashion to logarithmically reduce the problem size in order to localize abnormal/malicious switching units, and 4) providing an attack categorization module that analyzes live ingress/egress traffic of solely the maliciously-detected switch(es) to identify the specific type of attack, rather than inspecting the whole network traffic as is done in traditional intrusion detection systems. This significantly enhances the performance of attack detection and reduces the load on the controller. A test-bed prototype implementation is realized on the Mininet network emulator. The experimental analysis corroborated the algorithms' convergence property using the linear and FatTree topologies with network sizes of up to 250 switches. Moreover, an implementation of the attack categorization module is realized and achieved an accuracy rate of over 90% for the different attack types supported.

4.1.1 Threat Model

The VISKA security service operates in a typical SDN network composed of a set of physical switching elements (data plane) configured and controlled by one or more controllers (control plane). The control plane is responsible of configuring the data plane with the necessary flow rules that form the basis of the switching units' flow tables. The communication between the control and data planes is governed by the rules of a protocol such as *OpenFlow*.

The adversary model we consider in this work is represented by a set of switching nodes within the SDN physical network. VISKA is capable of detecting, with high confidence levels, active attacks related to malicious or misbehaving switch operation and localizing the source(s) of the attacks. Active attacks mainly include packet modification, packet dropping, and packet injection, which induce a deviation from the normal network behavior. These attacks are analyzed and categorized in order to further secure the data plane and the control plane in the underlying SDN network. Examples of such active attacks consists of one or more switches colluding to:

1. Inject malicious packets for the purpose of instigating DoS attacks on both layers, data and control, of the SDN network.
2. Drop network packets for the purpose of maliciously occluding particular network flows.
3. Augment network flows with padding packets to conceal the malicious effect of packet dropping.
4. Modify the contents of packets to cause traffic rerouting, to execute man-in-the-middle attacks, or to poison particular network flows.
5. Delay the forwarding of network traffic to disrupt the quality of service (QoS) of the SDN network.

VISKA assumes that the SDN controller is trusted and free of malicious security vulnerabilities. In other words, the controller is expected to be operated by legitimate administrative authorities and that it executes valid code that delivers authentic flow rules to the data plane switching units.

The VISKA service can be divided into two complementary modules: 1) packet probing-based security module for detecting malicious data plane elements, 2) real network data-based module for categorizing attacks and creating signatures for novel attacks within the SDN network.

The VISKA security algorithms are designed to operate in a highly malicious SDN environment and can tolerate relatively large number of misbehaving switches. This comes at the expense of the time complexity of the attack localization algorithms as will be demonstrated in Section 4.1.2.4. The accurate localization of the attack source highly facilitates the process of mitigating the attack. This is one of the great security advantages provided by the VISKA service. The mitigation strategy proceeds by firstly ceasing the malicious switch(es) forwarding activities and reporting this action, together with the details of the categorized attack, to the SDN service provider. The latter can administratively execute the necessary technical actions to inspect and possibly rectify the configuration and operational context of the malicious source(s) to resume its/their forwarding activities by leveraging the control plane global network view and the OpenFlow protocol.

4.1.2 System Design

The VISKA service architecture, as depicted in Figure 4.1 utilizes network programming for recursively partitioning the SDN data plane. The controller routing and forwarding achieved through OpenFlow messages on the data plane, allow for the segregation of network partitions, consequently isolating parts of the SDN network referred to as views that would recursively map to the malicious switches, if any. To achieve the goal of localizing maliciously behaving switches, a graph-theoretic partitioning algorithm recursively divides the data plane network into two equal-degree network partitions that have minimal interconnecting edges. Each network partition is probed by a set of data packets dynamically generated by a probing module on top of the SDN controller. The controller probing module consists of two processes, a sender process (P_s) responsible of generating and pushing the probing packets into the data plane, and a receiver process (P_r) responsible of receiving the probing packets from the data plane. The routing of the probing packets from Ps to Pr is transparently updated by the SDN controller in the switches' flow table entries.

To achieve the goal of real time detection of malicious activities in the network data plane, the Tug-of-war sketch data structure is employed on the probing streams. Sketches are probabilistic data structures that compactly represent the frequency of occurrences of items in data streams using a hashing function in sub-linear space. Sketching algorithms are adopted in this work due to their efficient summarization of large data sets which allows VISKA to detect deviations in detect abnormal switch behavior in real time. For each probing time interval *t*, the computations of summarized sketches of the probing packets are generated at P_s and are appended with a timestamp accumulator indicating the packets' transmission time, t_s. The sketch data structure and the timestamps are sent to the active security service in the cloud for inspection. Analogously, the receiver process, P_r, computes and sends the sketch of the received probing packets and the corresponding packet receipt timestamps. The VISKA cloud security service algorithms compile the data structures received from P_s and P_r in order to:

1) Recognize the levels of deviation between the data sketches of the sent and received probing packets, and

2) Compute the average time delay on the probe path based on the sent and received timestamp accumulators.

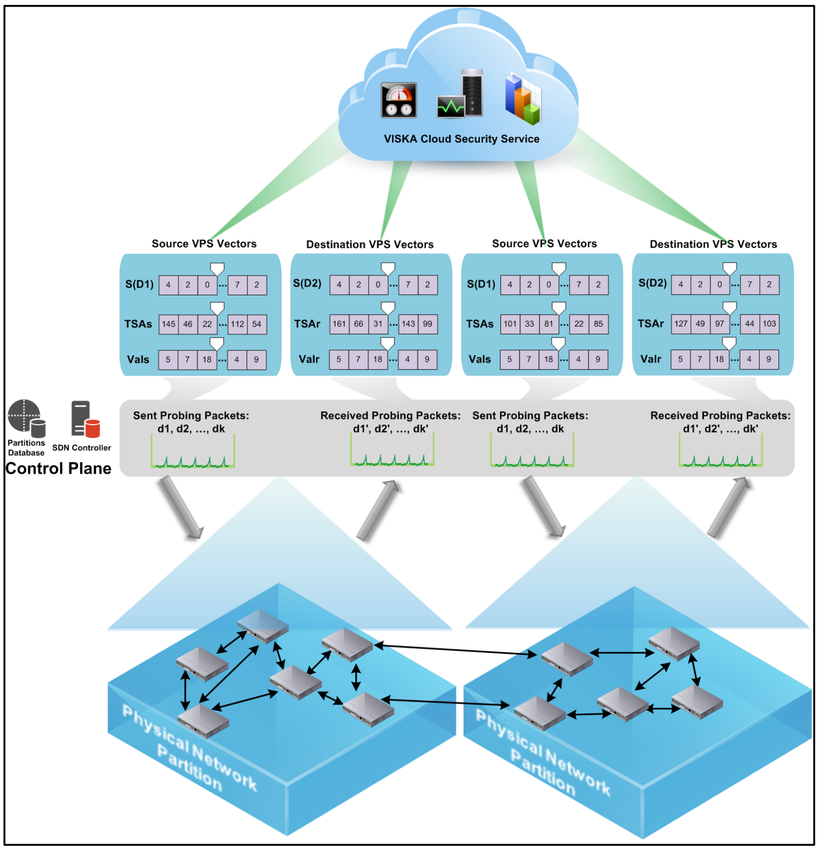

VISKA Cloud Security Service
Source VPS Vectors
Destination VPS Vectors
Source VPS Vectors
Destination VPS Vectors
S(D1)
TSAs
Vals
S(D2)
TSAr
Valr
Partitions Database
SDN Controller
Control Plane
Sent Probing Packets: d1, d2, ..., dk
Received Probing Packets: d1', d2', ..., dk'
Sent Probing Packets: d1, d2, ..., dk
Received Probing Packets: d1', d2', ..., dk'
Physical Network Partition
Physical Network Partition

Fig. 4.1 - VISKA's conceptual SDN design. S(D1) and S(D2): the source and destination sketch data structures, TSAs and TSAr: the sender and receiver time stamp accumulator data structures, Vals and Valr: the sender and receiver validity vectors.

The VISKA algorithms use these computations in order to decide on the probability of malicious switch behavior in the corresponding network partition and furthermore categorize the type of attack in the infected regions of the network by inspecting the real traffic on the egress and ingress ports of exclusively the maliciously-detected switch(es) and not of the whole network traffic.

The VISKA algorithms are thoroughly elaborated in the following subsections. It is worth mentioning here that the VISKA procedures utilize sizeable probing data streams and timestamps to ensure the accurate detection of misbehaving switches along the recursively generated network partitions. Sketching data structures in such setup leads to major reduction in computational complexity, better utilization of storage, and as a consequence, a performance-efficient real time malicious detection.

4.1.2.1 The View Probing and Sketching Algorithm (VPS)

The View Probing and Sketching (VPS) algorithm produces sketch summaries of the probing data at the source and destination controller processes by utilizing the Tug-of-war sketching algorithm [17]. The probing packet stream $(D\colon (d_0 \rightarrow d_v))$ is sent from P_s to P_r by traversing all the switches in a given network view. A probe-route module running on the SDN network controller, pushes the necessary forwarding rules to ensure that the

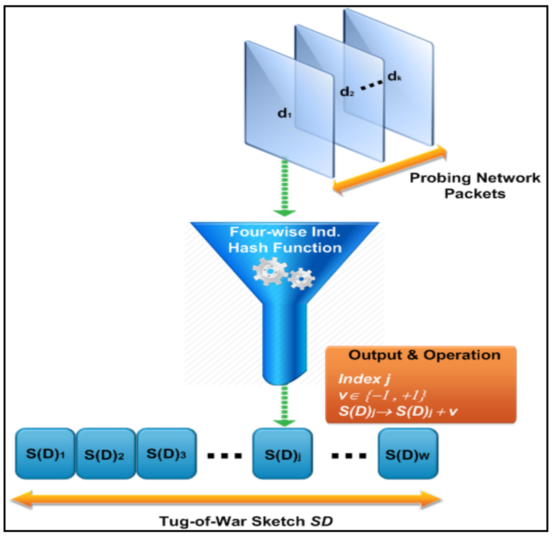


Fig. 4.2 - VISKA’s sketch data structure on probing data

probing packets visit each switch in the corresponding network partition.

At P_s and P_r, the probing stream D is fed to a sketch engine to produce a compact sketch representation $s(D)$ that is sent to the cloud security service for analysis. For each probing packet d_i, a four-wise independent hashing function $\eta[d_i]$ is applied, which uniformly

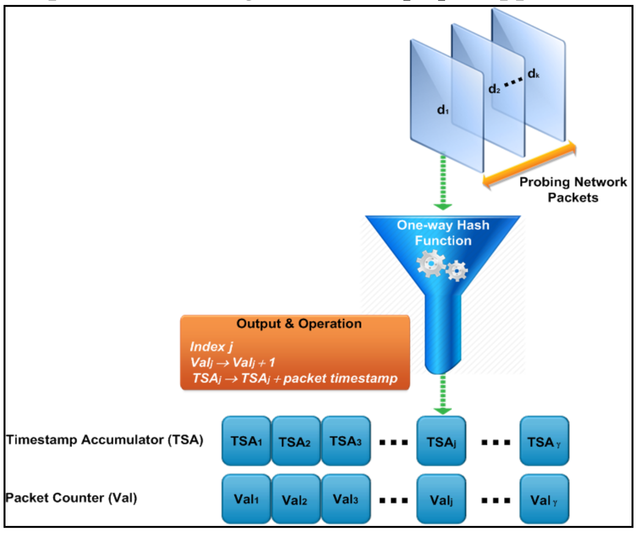


Fig. 4.3 - : VISKA's Timestamp Accumulator data structure

maps to a pair of values: an index j in the sketch vector and a value v in {-1, +1}; v is added to $s(D)$ at index j. The timestamp is appended by the controller probing processes to the sketch data structure corresponding to the sent and arrival times, respectively using acumulator data structures, as will be explained later in this section.The sketch representation of the corresponding probing data stream is evaluated as the summation of the dot product of the hashed values and the data stream as follows:

$$s(D) = \sum_{i=0}^{v} d_i . \eta[d_i]$$

which is a randomized linear projection of the input data stream. The resulting $s(D)$ vector at time t is sent to the cloud security service for analysis. The linearity property of tug-of-war sketch indicates that using the same family of pseudo-random hashing functions, η, on two sets of data streams, D_1 and D_2, then for any constants a and b:

$$aD_1 + \mathrm{b}\mathrm{D}_2 = \mathrm{as}(\mathrm{D}_1) + \mathrm{bs}(\mathrm{D}_2).$$

This linearity property is essential for estimating the difference between the two probe data streams (sent and received) along a network partition.

As a result of the linearity of this sketch based on [47], the second norm difference between the two received sketches reflect any deviation between the sent and received data streams subject to an ϵ error and with minimum probability of $(1 - \delta)$ thus:

$$|\mathrm{s}(\mathrm{D}_1) - \mathrm{s}(\mathrm{D}_2)|^2 = \Delta, \quad \text{where} \Delta = (1 \pm \epsilon) \times |\mathrm{D}_1 - \mathrm{D}_2|^2 \qquad (1)$$

The sketch's second norm difference estimation results in a more accurate representation of the deviation between the corresponding data streams. Such deviation indicates a malicious activity if Δ is greater than a preset threshold, τ. This probing and sketching procedure is repeated every time interval *t* to detect abnormal switch behavior in real time.

The Tug-of-war probabilistic sketching algorithm was adopted in the security model because of its light-weight processing requirements which typically consists of simple hash function calculations. Moreover, the relatively small sized sketch data structure representation relative to the number of probing packets it summarizes induces minimal overhead for network transmission and reception as well as storage. The sketches $s(D_1)$ and $s(D_2)$ represent a compact hash representation of the probing data streams with $O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ computations where ϵ is the error and $(1-\delta)$ is the confidence level. Considering the network and stroage overhead imposed by the sketch representation, each sketch data structure is comprised of W counters of z bits each where $z = 1 + \frac{1}{2}\log\left(4\frac{k}{W}\ln\left(\frac{200W}{\delta}\right)\right)$. This is comprehensively described in Section 4.1.2.4.3.

The timestamp accumulator (TSA) data structures are created and computed at the sender and receiver processes concurrently with the sketch creation for the aim of detecting delay-causing attacks or otherwise nework congestion. Each probe packet is further hashed to a value *c* which is utilized as an index in the TSA_r and TSA_s vectors. These vectors represent the summation of the timestamps of the probing packets that map to the hash value *c* at the controller sender and reciever processes, respectively.

Figures 4.2 and 4.3, respectively describe the sketch and timestamp accumulator data structures on the probing data. Each outgoing probing packet is passed to the sketch engine represented by the funnel symbol in Figure 4.2, to output an index *i* and a value *v* in {-1,1}, which is appended to $s(D_1)$ at position *i*. The hashing function *h* is invoked on each packet to yield the index *c* at which the timestamp of that packet is appended in the TSA_s and TSA_r vectors. When the probing stream is entirely transmitted, the sketch $s(D_1)$ comprises the corresponding sketch values and each TSA_s entry represents the sum of the time stamps of the packets according to the hash value mapping to TSA_s entry indicies. In order to ensure the correctness of the packets timestamps, a validity vector (Val) is updated for each probing packet to count the number of packets according to their hashed value. Each time a timestamp is appended at index c, the validity vector is incremented by one at that same index. Ultimately, the probing packets are sent to the controller process P_r while $s(D_1)$, TSA_s and the validity vector Val_s are transferred to the VISKA cloud service for analysis.

Analogously, at the controller probing process P_r, the data structures $s(D_2)$, TSA_r, Val_r are created and calculated. These data structures are sent to the VISKA service for comparison and malicious activity detection.

The VPS algorithm calculates the second norm difference Δ of the two sketches $s(D_1)$ and $s(D_2)$ received at the VISKA service. If Δ is greater than a preset threshold τ, the network is considered malicious and the attack categorization and summarization module MACM is invoked. On the other hand, the difference in the sent and received timestamps is calculted on the probing stream by:

(1) First, checking that the validity vectors from sender and reciever at each index j are equal; this indicates that the corresponding packets are successfully received and the timestamp counters at that index j are valid.

(2) The timestamp diffence $TSA_r\,[j] - TSA_s[j]$ is calculated and added to the total Δ_{TSA}, which is the total difference in timestamp of the current probing stream. The total time Δ_{TSA} is divided by the sum ValCount of the corresponding valid packet counts in vector Val_s and Val_r. This results in the average time delay of the correctly received probing stream as described in Equation (2).

$$\partial = (TSAr - TSAs)/\text{ValCount} \qquad (2)$$

If the value of ∂ exceeds a threshold Γ_{d}, the network is considered malicious. Otherwise, if the time delay is greater than the congestion threshold Γ_{c}, the corresponding data plane elements are considered congested.

The values of Γ_{d} and Γ_{c} depend on the overall network round trip time (RTT). Since RTT can change from one probing period to the other (even within an individual probing period), the values of Γ_{d} and Γ_{c} dynamically change based on this variation in RTT. To maintain a smooth variation in the values of Γ_{d} and Γ_{c} we followed an algorithm analogous to the retransmission timer calculation algorithm followed in TCP [42]. The details of the Γ_{d} and Γ_{c} calculations are presented in the following smoothing equations using the estimators mRTT and vRTT respectively representing the mean and variance of RTT in the i^{th} probing period.

$$\mathrm{mRTT}_{\mathrm{i+1}} = a \times RTT + (1-a) \times \mathrm{mRTT}_{\mathrm{i}} \quad (3)$$

$$\mathrm{vRTT}_{\mathrm{i+1}} = b \times (|RTT - \mathrm{mRTT}_{\mathrm{i}}|) + (1-b) \times \mathrm{vRTT}_{\mathrm{i}} \quad (4)$$

$$\Gamma_{\mathrm{c}_{\mathrm{i+1}}} = \mathrm{mRTT}_{\mathrm{i+1}} + 4 \times \mathrm{vRTT}_{\mathrm{i+1}} \quad (5)$$

$$\Gamma_{\mathrm{d}_{\mathrm{i+1}}} = 2 \times \Gamma_{\mathrm{c}_{\mathrm{i+1}}} \quad (6)$$

The gains a and b are set to $\frac{1}{4}$ and $\frac{1}{8}$ respectively.

In the first probing period the mean and variance estimators are set as follows:

$$\mathrm{mRTT}_1 = RTT$$

$$\mathrm{vRTT}_1 = RTT/2$$

(3) Finally, if the two values Δ and ∂ are within safe boundaries, the network is interpreted as normally operating.

After sending each sketch data and timestamp data structures, the probing hosts flush them to compute the following interval sketch.

The TLS protocol is implemented on the controller probing processes to ensure the integrity and authenticity of the source and destination sketches when transferred to the cloud service over the network links. When the VISKA service is to be adopted and executed in a real world environment, it is very important to masquerade the patterns of the probing packets introduced in the network to prevent any malicious node ability to recognize VISKA functionality. This is addressed in the same sense wherein the software-based control on the probing processes and their parameters provide the VISKA probing module the control on randomizing certain fields in the probing packets IP header (eg. Identification, TTL, Options and Padding, and ECN fields), the data sections are randomized to conceal any deterministic features that may reveal the probing nature of these packets.

It is worth noting here that the feasibility of using the Timestamp Accumulator data structure relies on the accurate time synchronization among the system clocks of the nodes in the network. To achieve this, we utilize the control-plane centralization property of the SDN architecture by deploying a Network Time Protocol (NTP) [58] server on the SDN controller. NTP is the defacto standard in achieving high-quality time synchronization in modern Internet networking infrastructures. Relying on the local SDN controller in time synchronization instead of utilizing a remote NTP public server aids in a more precise time synchronization by avoiding the asymmetrical latency delays incurred by the NTP time packets exchanged between the probing module and the public time server. This results in a maximum of 0.5 to 1.5 msec time lag between any two network nodes. This is sufficient for correct operation of the VISKA Timestamp Accumulator realization. The VPS algorithm is presented in Algorithm 4.

4.1.2.2 The Network Views Partitioning Algorithm (NVP)

To guarantee malicious-free switch behavior at the network data plane, the correct network functionality should be checked at the switch granularity level. In order to minimize the executions of the VPS algorithm, described in section 4.1.2.1, in checking the physical switches, the SDN network is recursively partitioned into two semi-uniform network partitions. Each resulting partition is probed for possible sources of deviation in the forwarding mechanism. Only partitions marked as malicious will be further subdivided using the NVP algorithm. In this sense, the NVP algorithm generates the network partitions and feeds them to the probing algorithm VPS to check any deviation in the corresponding current network partition. If the VPS algorithm indicates an above-threshold deviation in the sketch calculations and/or the timestamp analysis, the VISKA service recursively executes NVP to partition the respective network topology into two separate network views with minimum interconnecting edges. The two resulting network partition are fed separately to the VPS algorithm at the controller for subsequent behavior check. This recursive partitioning method is applied until the algorithm isolates the source of maliciousness in the data plane, if present.

The efficient real-time detection and localization of malfunction was fully achieved in VISKA by ensuring the segregation of the network partitions owed to the programmable

SDN network architecture. Two main approaches contributed to the feasibility of isolating the problem within separate network views:

1. The Karger's randomized algorithm for generating minimum graph cuts [48] was adopted in the NVP algorithm in order to partition the network graph into two separate sets (mapped to the corresponding network views) with minimum interconnecting edges.
2. A probe-route module is executed on the SDN controller for the probing packets to traverse the switches incorporated within a network view. The controller pushes the necessary rules to the data plane switches to restrain the probing data stream to the corresponding network partition thus, ensuring network views segregation.

This recursive partitioning of the network views, isolates the malfunctioning views which results in a near logarithmic time complexity in the size of the network.

The Karger's graph cut algorithm [48] is based on the contraction of edges and merging the nodes in a connected unidirectional graph $G(n,e)$. This algorithm continues randomly selecting nodes and merging them iteratively until the graph is reduced into two sets represented in two vertices. These two sets remain connected; to minimize the connecting edges between the two sets, the Karger's algorithm repeats this contraction procedure a predefined number of times until a minimum graph cut is produced between the two partitions with a high probability. For a graph of n number of vertices and e number of edges, Karger's contraction method returns a minimum graph cut with a probability of success:

$$\mathrm{P_c} \geq \binom{\mathrm{n}}{2}^{-1}$$

and a probability of not attaining a minimum cut of:

$$P_{nc} \leq \frac{1}{n}$$

The algorithm randomly repeats the contraction procedure $R = \binom{n}{2} \ln(n)$ times in order to arrive at a minimum cut in time complexity of $O(R.e) = O\ (n^2 e \log n)$.

In summary, the network topology graph is input to the NVP partitioning algorithm, which outputs two sets of switches constituting two separate network partitions with minimum interconnecting edges. Each partition is fed to the VPS algorithm to check if any malfunctioning is present. Depending on the output of the VPS algorithm, a network view/partition is either rendered correctly functioning and is thus discarded, or malfunctioning and thus, it is recursively partitioned by the NVP algorithm until the size of the switches in the respective partition is less than or equal to a minimum, m. The NVP algorithm is presented in Algorithm 1.

4.1.2.3 The Malfunction and Attack Categorization and Summarization Module(MACM)

The first stage of the VISKA service operation is the VISKA malicious switch detection where the VISKA VPS algorithm returns the switch(es) that was/were classified as malicious. The second stage of the VISKA service is the MACM module which is responsible of identifying and categorizing the attacks induced by the malicious network elements. The MACM module is invoked in Algorithm 1 in the VPS function when a malicious activity is detected. This stage is essential for securing the SDN network provider services. The SDN provider utilizes the VISKA service to detect malicious operation in its data plane. VISKA aids in guarding against an important set of attacks that initiate in these network infrastructures such as DoS, interruption, blocking, delay, and Man-in-the-Middle attacks.

The second stage of the VISKA service, the MACM, primarily identifies two classes of data plane malfunction:

(1) The distorted traffic malfunction class (CatI), where the second norm difference of the sent and received sketches is beyond a preset threshold τ, which reflects a malicious deviation in the probing stream introduced by the data plane elements of the current network partition.

(2) The time delay malfunction class (CatII), where the packets are received by the probing host correctly (no significant sketch difference is recognized $\Delta < \tau$), however, the average time delay of the transmitted probe packet stream is beyond a normal network congestion value, $\partial \geq \Gamma_d$.

In the case of CatI malfunction detection, the maliciously categorized data plane switching elements are further investigated to summarize the attack in order to deduce and block probable network wide malfunction.

The egress and ingress traffic of the malicious detected switches are collected for certain time periods t_i in order to categorize and summarize the investigated attack. The SDN infrastructure is utilized in investigating the traffic ingress and egress of the malicious

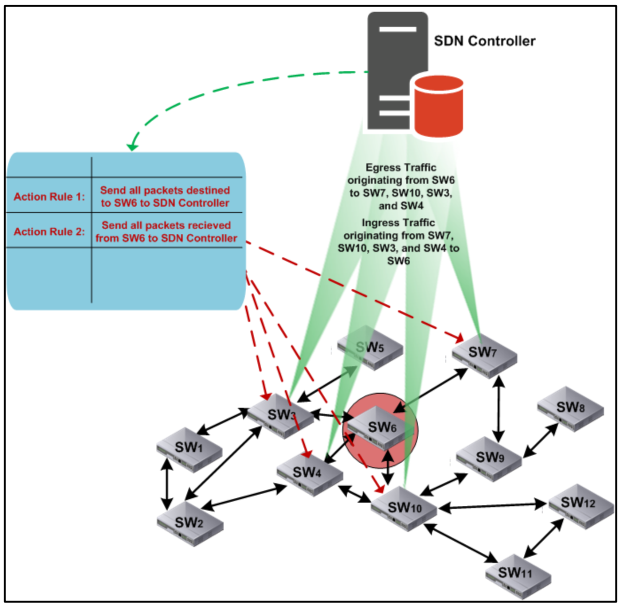

SDN Controller
Action Rule 1:
Send all packets destined to SW6 to SDN Controller
Action Rule 2:
Send all packets recieved from SW6 to SDN Controller
Egress Traffic originating from SW6 to SW7, SW10, SW3, and SW4
Ingress Traffic originating from SW7, SW10, SW3, and SW4 to SW6
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8
SW9
SW10
SW11
SW12

Fig. 4.4 - : Ingress/Egress traffic capture at the controller of the *m* switches neighboring switches mc.

switches by forwarding traffic in and out of the malicious switch to the controller. The set of switches, mc, that are one-hop away from the malicious switch in the network, are primarily identified. A data collecting module running on the controller sends the mc switches the necessary action rules that dictate sending all packets having the malicious switches as their next hop to the controller. The controller monitoring module utilizes data mining features on the periodically collected data to categorize and summarize the attack, or otherwise specify the malfunction as a benign behavior.

In order to achieve anomaly categorization and early stage attack detection, the VISKA cloud service primarily identifies the malicious switches in the VPS and NVP algorithms. Next, the VISKA MACM algorithm exploits the SDN controller centralized network programmability to capture the egress and ingress traffic of the malicious switches on the controller as depicted in Figure 4.4. These packets are first prepared and analyzed by grouping them according to source address, destination address, source port, and destination port. Important packet header fields are stored and grouped at the controller and are prepared for comparison and categorization at the VISKA cloud service to identify and specify potential network attacks. The MACM process consists of the following three phases:

Phase 1: The analysis of the egress and ingress traffic of the m switches is demonstrated in Figure 4.4. The packets are sent by the malicious switches neighbouring switches (mc) to the controller. The controller in turn captures these packets and passes them to the MACM module, which stores the necessary header information of the captured packets. The collected data is prepared and grouped according to specific header fields in information tables.

In Figure 4.5, two hashing functions are applied on specific fields of the packets' headers in order to find certain patterns and characteristics in the captured traffic for time intervals $\mathrm{t_i}$. First, the packets' destination IP addresses are input to the hashing function hashd. Packets with the same hashed destination IP address are aggregated. The resulting E_Dest table includes the packet information categorized by their corresponding destination IP addresses. Analogously, another hash function hashs is applied on the source address of the captured packets to categorize and prepare the source address aggregation table (E_Src).

This procedure of hashing and aggregation is done on both egress and ingress traffic to prepare the data for analysis by the MACM algorithm. The prepared attributes that are stored for analysis in the aggregation tables for each packet include the source IP, destination IP, transport protocol, SYN packet and ACK packet flags. This collected packet information is ready to be analyzed in phase 2.

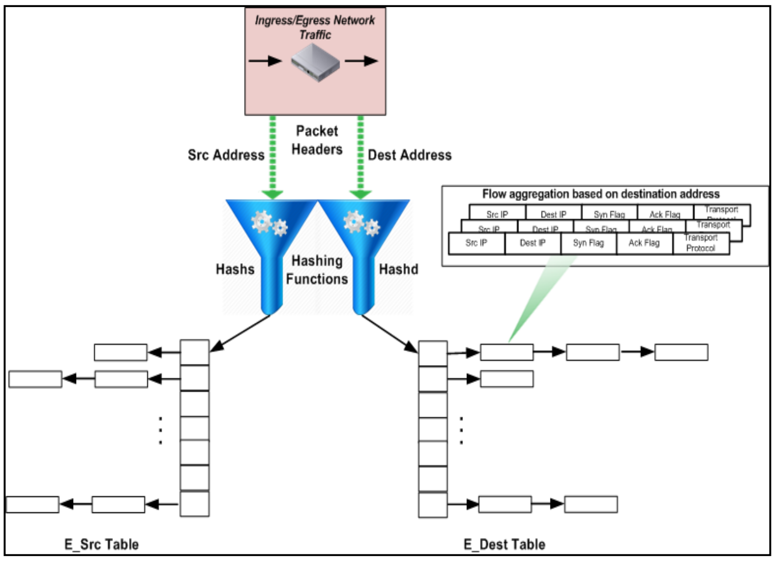


Fig. 4.5 - : Egress flow summarization aggregated based on source and destination IPs (analogous hashing structures are applied on the Ingress traffic).

Phase 2: The traffic information tables collected in phase 1 from the egress and ingress ports of the malicious data elements are analyzed, and certain features are extracted for the purpose of attack categorization. The following is a list of the parameters and features characterizing the collected packets.

1. I_Sum(): the sum of the size of the ingress packet flow in bytes.

2. E_Sum(): the sum of the size of the egress packet flow in bytes.

3. I_count(destIP): the number of ingress packets with the same destination IP, destIP

4. I_count_srcIP(destIP): the count of the different source IPs for the ingress flows with the same destination IP, destIP.

5. I_Sum(destIP), I_Avg(destIP), I_SD(destIP): The sum, average and standard deviation, respectively, of the size of the ingress packet flow with the same destination IP address in bytes.

6. I_count_SYN(destIP), I_count_ACK(destIP): the number of ingress packets of type SYN and ACK respectively with the same destination IP address.

7. I_count(srcIP): number of ingress packets with the same source IP, srcIP.

8. I_count_SYN(srcIP), I_count_ACK(srcIP): the number of ingress packets of type SYN and ACK respectively with the same destination IP address.

Similarly, the following parameters are computed for the Egress traffic:

E_Sum(destIP)	*E_count(srcIP)*	*E_count_srcIP(destIP)*
E_Avg(destIP)	*E_count(destIP)*	*E_count_SYN(destIP)*
E_SD(destIP)		*E_count_ACK(destIP)*

Phase 3: The information tables, the features and the characteristics of the ingress and egress data, which are collected and computed in phases 1 and 2, are subsequently utilized and analyzed to categorize the network attacks induced by the m switches. The following is a list of the MACM identified attacks:

(1) DoS on the controller: The egress traffic is checked for packets where the controller is the destination address including packet-in messages from the malicious switch. If the number of these packets exceeds a certain permitted threshold for normal network operation Γ_c, then the switch is considered "DoSing" the controller.

(2) DoS on network node: This attack could be on a host or switching element in the network. This is identified by checking the count of the egress packets with the same destination address. When this count is larger than that of the ingress packets and at the same time it is beyond a threshold, Γ_{dos}, the corresponding destination is detected to be DoSed, and further analysis is done by the network administrator to identify the rival attack. The VISKA algorithms utilize the SDN centralization and programmability features for early detection of such DoS attacks on the network nodes in real-time and in early stages. As a result, the VISKA service secures the tenants SDN network against presumable DoS attacks.

(3) Data Blocking attack: This intruder attack selectively blocks traffic to specific destinations in the network for the aim of inducing erroneous network operation. Such attacks decrease the tenants' trust in the corresponding SDN network provider. To avoid this, the VISKA MACM algorithms inspect the ingress and egress packets of the malicious switches (detected by the VPS VISKA algorithms) and if the egress traffic is found to be less by a certain network permissible threshold, Γ_{bh}, from the ingress one, the malicious switches are identified to be blocking network traffic. The blocking attack is therefore detected on the destination address or from the specific source address in the investigated network traffic. The IP addresses of the captured packets that were blocked are further analyzed and the involved network elements are inspected by the network administrator.

4) Man-in-the-middle (MITM) attack: VISKA algorithms detect MITM attacks at very early stages in real-time. MITM attack prevention is of high significance to network tenants to ensure network confidentiality and integrity in the cloud. The MACM captures packets at the malicious switches and checks the homogeneity of all source-destination packet flows on the egress and ingress ports of the malicious switch. If the flows are not homogeneous within a certain permissible network limit, the packets are further investigated by checking the size of traffic to a single unique destination address with the same group of source addresses in the corresponding flows. This destination IP does not appear in the consequent flows on the ingress ports and concurrently, the count of the source addresses is the same as the investigated source address classified packet counts. Therefore, the same packets are being forwarded to another destination host, which indicates a MITM attack. Note that the attack detection mechanism is extendable to address additional attack categories based on tenants' demands and for specific network requirements. This is made feasibly by leveraging the controller's programmability features in SDN platforms and the modular VISKA attack detection module design.

The VISKA attack detection and categorization algorithms are summarized in Algorithm-4. A block diagram summarizing the architecture of the VISKA algorithmic modules and their interaction is illustrated in Figure 4.6.

ALGORITHM 4: VISKA MALICIOUS SWITCH(ES) DETECTION/ATTACK CATEGORIZATION ALGORITHM

```
Let G be the graph representing the SDN network
Let n be number of switches in G
Let e be the number of edges in G
Let V be the granular network view/partition to be
checked for maliciousness. Initially V = G
```

```
Let **m** be the maximum malicious partition size (m=1 to
reach single switch granularity)
Let $P_s$, $P_r$ be the controller probing processes allocated
for probing the bootstrapping network view V
VISKA(V, $P_s$, $P_r$)
  If (VPS(V, $P_s$, $P_r$) = malicious)
      if(n<=m)
          return V (containing malicious switch)
      else
        (V1, V2, $P_s$, $P_r$) = NVP (G, n, e)
        VISKA(V1, $P_s$, $P_r$)
        VISKA(V2, $P_s$, $P_r$)
  else return (correct or congested partition behavior)
```

VPS: View Probing and Sketching function

```
VPS(V, $P_s$, $P_r$)
     Randomly Generate probing data D1($d_1 \ldots d_k$) at $P_s$
     and send to $P_r$
    sketch: create sketch data vector s=0 , and TSAs =0,
counts=0 at $P_s$
    for each v packet $d_v$ in D1
            compute (i, value) = $\eta_j[d_v]$
            insert at index i in sketch $s(D1)$ vector:
                     $s(D1)_i \mathrel{+}= value$
            compute (j)= h($d_v$)
                $TSAs_j \mathrel{+}= sent\ timestamp$
                     $(V_s)_j \mathrel{+}= 1$
    send: $s(D1), TSAs, Val_s$ to VISKA service for analysis
Compute: (steps sketch through send) on $P_r$ to generate
and send $vectors\ s(D2), TSAr,\ and\ Val_r$
At VISKA
 count=0
  $\Delta = |s(D1) - s(D2)|^2$
  for i from 1 to k
      if $Val_{s_i} = Val_{r_i}$
        $\Delta_{TSA} \mathrel{+}= TSAr_i - TSAs_i$
        ValCount+= $(V_s)_i$
 $\partial = \Delta_{TSA}$/ValCount
if $\Delta \geq$ threshold $\tau$ or $\partial \geq \Gamma_d$
            return malicious, call MACM ($\Delta, \partial$)
       else if $\partial \geq \Gamma_c$
            return correct, congested
      else
          return correct
```

NVP: Network Views Partitioning Function

```
NVP (G, n, e)
        (G1, G2)= Karger (G, n, e)
        Insert forwarding rules for the probing packets
        on controller
        At the SDN network controller
```

```
    - Isolate network partitions V1, V2 corresponding
      to the Karger output (G1, G2)
      return (V1, V2, $P_s$, $P_r$)
```

MACM: Malfunction and Attack Categorization Module

```
MACM ($\Delta, \partial$)
 if n $\leq$ m
  if $\Delta \geq \tau$
     $CatI = 1$ (attack ∈ to Category I where an active attack is being initiated in the network)
 else // $\partial \geq \Gamma_d$
     $CatII = 1$ (attack ∈ to Category II where a time delay introducing attack is introduced in the network)
if $CatI = 1$

malicious switch(es) ingress and egress traffic is collected and mined:
if E_Sum( )/I_Sum( ) > ΓE$d$
 for each destIP in table E_Dest
   if (E_Sum(destIP)-I_Sum(destIP)> Γdos) AND
      (E_count(destIP) – I_count(destIP)> Γp)
  alarm: DoS on destIP (Flooding)
  if (E_count_ACK(destIP)-E_count_SYN(destIP) < Γcon)
     AND (E_Avg(destIP)< Γsyn) AND ((E_count(destIP)
     – I_count(destIP)> Γp)
   alarm: DoS on destIP (SYN attack)
else if E_Sum()/I_Sum() < ΓId
  for each destIP in table E_Dest
     if (E_count(destIP) – I_count(destIP))< Γbh
       alarm: interruption of traffic to destIP
else // E_Sum()/I_Sum() within boundaries)
 for each destIP in table I_Dest
       if (dest-IP is not in E_Dest) AND (I_count(destIP)- E_count(destIP) < Γbh)
       alarm: blocking on dest IP
 for each srcIP in table I_Src
       if (srcIP is not in E_Src) AND (I_count(srcIP)-
          E_count(srcIP) < Γbh)
       alarm: blocking on srcIP
 for each destIP in E_Dest
    if destIP is not in I_Dest AND E_count(destIP)> Γmitm
       alarm: MITM attack at destIP
```

Algorithm-4. VISKA Algorithms

The values of the thresholds in the MACM module are set empirically and are bounded in the ranges depicted in Table 4.1. The threshold values are dependent on the maximum throughput of the switching elements in the data plane, the average packet size, the time of ingress/egress flow collection, and the minimum and maximum percent of packets generated at the maximum switch throughput in the collection time period that are necessary to initiate a particular attack. This network-specific specification of the threshold ranges facilitates a more accurate threshold selection mechanism in real SDN network environments. The individual threshold range parameters for each attack type are presented as follows:

Table 4.1 Attack Detection Thresholds Ranges

Attack type	**Threshold(s) Range**
Denial of Service Attack	$(1+\alpha_l) \leq \Gamma\mathrm{E}d \leq 3\times(1+\alpha_l)$ $\lfloor \mathrm{P}_{avg}\times X_{max}\times t_i \times \alpha_l \rfloor \leq \Gamma\mathrm{dos}\times 10^5 \leq \lfloor 2\times \mathrm{P}_{avg}\times X_{max}\times t_i\times \alpha_h \rfloor$ $\left\lfloor \frac{\Gamma\mathrm{dos}_{low}}{\mathrm{P}_{avg}} \right\rfloor \leq \Gamma\mathrm{p} \leq \left\lfloor \frac{\Gamma\mathrm{dos}_{high}}{\mathrm{P}_{avg}} \right\rfloor$
Interruption/Blocking Attack	$\lfloor X_{max}\times t_i\times \beta_l \rfloor \leq \Gamma\mathrm{b}h\times 10^5 \leq \lfloor X_{max}\times t_i \times \beta_h \rfloor$
Man-in-the-Middle Attack	$\lfloor X_{max}\times t_i \times \gamma_l \rfloor \leq \Gamma\mathrm{mitm}\times 10^5 \leq \lfloor X_{max}\times t_i \times \gamma_h \rfloor$

$\boldsymbol{X_{max}}$ is the maximum throughput of the switching elements in the deployment network.

$\boldsymbol{t_i}$ is the data collection time period used by the MACM module.

$\boldsymbol{\alpha_l}$**,** $\boldsymbol{\alpha_h}$ are respectively the minimum and maximum percent of packets generated by the malicious switching element at the maximum throughput X_{max} in the time period t_i (multiplied by a factor of 10^{-5}) necessary to initiate a DoS attack.

$\boldsymbol{\beta_l}$, $\boldsymbol{\beta_h}$ analogous to α_l and α_h, β_l and β_h are respectively the minimum and maximum percent of packets generated by the malicious switching element at the maximum throughput X_{max} in the time period t_i (multiplied by a factor of 10^{-5}) necessary to initiate an Interruption/Blocking attack.

$\boldsymbol{\gamma_l}$, $\boldsymbol{\gamma_h}$ analogous to α_l and α_h, γ_l and γ_h are respectively the minimum and maximum percent of packets generated by the malicious switching element at the maximum throughput X_{max} in the time period t_i (multiplied by a factor of 10^{-5}) necessary to initiate an Interruption/Blocking attack.

According to the nature of the detected attack in the network, the controller mitigates the attack by sending the necessary flow rules to isolate and suspend the operation of the maliciously-detected forwarding element in the data plane.

4.1.2.4 Algorithm Complexity and Convergence Analysis

Analysis of the Malicious Switch Detection Function:

In this section, we provide a mathematical complexity analysis of the worst case, average case, and best case running times of the VISKA malicious switch detection algorithms as a function of the network size. Moreover, we present the cost of the MACM malfunction and attack categorization algorithm invoked when a malicious activity is detected.

Worst case analysis: The worst case scenario arises when a malicious behavior is detected by the VPS function in every recursive network partition provided by the NVP partitioning function. The worst case runtime complexity $\mathrm{T(n)}$ is given by:

$$\mathrm{T_w(n)} = 2\mathrm{T_w}\left(\frac{\mathrm{n}}{2}\right) + \mathrm{Cost(NVP)} + \mathrm{Cost(VPS)} \quad (7)$$

where n is the network size, $\mathrm{Cost(NVP)}$ and $\mathrm{Cost(VPS)}$ are the costs of running the NVP and VPS functions, respectively. The multiplicative factor of 2, in $2\mathrm{T_w}\left(\frac{\mathrm{n}}{2}\right)$, indicates that the recursive steps are applied on the two network partitions produced by the NVP function. This case arises when the VPS algorithm detects malicious activity in both network views. From the previous subsections:

$\mathrm{Cost(NVP)} = \mathrm{O}\ (\mathrm{n}^2\mathrm{e} \log \mathrm{n})$ and $\mathrm{Cost(VPS)} = \mathrm{O}\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$.

Substituting in Equation (7) we get:

$$\mathrm{T_w(n)} = 2\mathrm{T_w}\left(\frac{\mathrm{n}}{2}\right) + \mathrm{O}\ (\mathrm{n}^2\mathrm{e} \log \mathrm{n}) + \mathrm{O}\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right) \qquad (8)$$

Note that $\mathrm{O}\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ depends on the size of the sketch data structure allocated on the controller probing module. Since it does not depend on the network size n, it can be replaced by a constant C in Equation (8) to get:

$$\mathrm{T_w(n)} = 2\mathrm{T_w}\left(\frac{\mathrm{n}}{2}\right) + \mathrm{O}\ (\mathrm{n}^2\mathrm{e} \log \mathrm{n})\ + \mathrm{C} \qquad (9)$$

Solving the recursive equation in (5), we get a closed form worst case runtime complexity of $\mathrm{O}\ (\mathrm{n}^2\mathrm{e}\ (\log \mathrm{n})^2)$.

A summary of the worst case complexity equations and their meaning is presented in Table 4.2.

Table 4.2 VISKA worst case complexity analysis summary

Term	**Meaning**	**Value**
$T_w(n)$	The worst case runtime complexity of the VISKA malicious switch detection algorithms. n designates the network size.	$2T_w\left(\frac{n}{2}\right) + \mathrm{Cost(NVP)} + \mathrm{Cost(VPS)}$
$\mathrm{Cost(NVP)}$	The worst case runtime complexity of the Network Views Partitioning function. This is typically the cost of the Karger algorithm.	$O\,(n^2 e \log n)$
$\mathrm{Cost(VPS)}$	The worst case runtime complexity of the View Probing and Sketching function. This cost depends on the size of the sketch data structure and not on the network size. Accordingly it can be replaced by an constant C.	$O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$

Average case analysis: The average case scenario arises when a malicious behavior is detected by the VPS function in one of the two recursive network views provided by the NVP partitioning function. The average case runtime complexity $T_a(n)$ is given by:

$$T_a(n) = T_a\left(\frac{n}{2}\right) + \mathrm{Cost(NVP)} + \mathrm{Cost(VPS)} \qquad (10)$$

Note that $T_a\left(\frac{n}{2}\right)$ is not multiplied by a factor of 2 since the recursion is only applied on one network partition and not both partitions as is the case in the worst case scenario.

Replacing the values of $\mathrm{Cost(NVP)}$ and $\mathrm{Cost(VPS)}$ in Equation (10), we get:

$$T_a(n) = T_a\left(\frac{n}{2}\right) + O\,(n^2 e \log n) + C \qquad (11)$$

Solving the recursive equation in (11) we get a closed form average case runtime complexity of $O\,(n^2 e\,(\log n))$.

A summary of the average case complexity equations and their meaning is presented in Table 4.3.

Table 4.3 VISKA average case complexity analysis summary

Term	Meaning	Value
$T_a(n)$	The average case runtime complexity of the VISKA malicious switch detection algorithms. n designates the network size. Note that $T_a\left(\frac{n}{2}\right)$ is not multiplied by a factor of 2 since the recursion is only applied on one network partition.	$T_w\left(\frac{n}{2}\right) + \mathrm{Cost(NVP)} + \mathrm{Cost(VPS)}$
$\mathrm{Cost(NVP)}$	The worst case runtime complexity of the Network Views Partitioning function.	$O\ (n^2 e \log n)$
$\mathrm{Cost(VPS)}$	The worst case runtime complexity of the View Probing and Sketching function. This cost depends on the size of the sketch data structure and not on the network size. Accordingly it can be replaced by an constant C.	$O\left(\frac{1}{\epsilon^2} \log \frac{1}{\delta}\right)$
$T_w(n)$	After solving the recurrence relation in Equation 5.	$O\ (n^2 e\ (\log n))$

Best case analysis: The best case analysis scenario obviously occurs when the whole network does not contain any malicious activity. In this case, no recursive partitioning is going to be carried out on the network view and thus, the base case will be reached by first applying the VPS function on the network view. As such, the best case runtime is simply a constant C.

The convergence of the VISKA malicious switch detection algorithm is evidently guaranteed by having a deterministic base case step in Algorithm 4 that ends the recursion on a particular network view when either (1) a set of source malicious switches of size m is isolated by the VPS function, or (2) a network view is found to be non-malicious by the VPS function.

Analysis of the Attack Categorization Module:

The MACM module is executed once the malicious switches are detected and identified. The main complexity in this module is related to the transmission of the ingress/egress traffic to the SDN controllers by the mc switches, and the analysis of such traffic by the controller for the purpose of attack categorization. The complexity mainly depends on the number of mc

switches and the flow size crossing them per unit time. We faithfully believe that an SDN controller can tolerate such traffic load and categorization processing due to the following reasons:

1. The MACM module analyses the ingress/egress traffic flowing solely into/from the maliciously detected switch(es) rather than inspecting the whole network traffic as is the case in traditional intrusion detection systems in the literature.

2. The number of mc switches in modern data center topologies is minimal compared to the total number of network switches. For instance, in the k-ary FatTree topology, which is widely deployed in today's data centers, the total number of switch nodes is $O(k^2 + k)$. Each edge switch is connected to $O(k/2)$ neighbors and each aggregation or core switch to $O(k)$ neighbours.

3. The MACM algorithm operates merely on the packet headers, thus eliminating the need to transmit the entire flow of packets to the controller. This drastically reduces the size of the ingress/egress flows transmitted to the SDN controller and the resources needed to process them.

4. Most modern SDN architectures are relying on replicated controller instances, which aids in balancing and distributing the processing load of the collected network data for analysis and mining.

5. A feasible approach that can be easily employed in the VISKA implementation to reduce the load on the SDN controller is to employ a central dedicated host for traffic collection and analysis. In this sense the MACM attack categorization

module can be efficiently deployed on an external host connected to the controller to further enhance the VISKA efficiency and performance.

MACM malfunction and attack categorization cost: This module is invoked when the malicious switches, with the predetermined granularity size m, are detected. As demonstrated in Section 4.2.2.3 (MACM phases 1 and 2), the algorithm captures packets at the malicious switch(es) ingress and egress ports and stores necessary header information of the real network traffic captured packets (this collection process of phase1 has a constant runtime complexity C). The collected information tables from the egress and ingress ports of the malicious data element(s) are analyzed, and certain features are extracted for the attack categorization. The basic operation in the MACM phase 2 complexity is the hashing operation (refer to Figure 4.5) which is executed on the total number of packets in the ingress and egress traffic flowing into the *m* maliciously-detected switch(es). let f_{em} and f_{im} respectively represent the number of egress and ingress packets in the collected traffic. This renders the complexity of phase 1 as follows:

$$\text{Cost of MACM}_{phase\ 1,2} = \theta(C + 2(f_{em} + f_{im})) \qquad (12)$$

The factor 2 in Equation 12 is the result of applying the hashing operations on the source as well as on the destination IP packet addresses in the collected flows as demonstrated in Figure 4.5.

The MACM phase 3 (Malfunction and Attack Categorization Module), presented in Algorithm 4, analyzes the traffic information tables collected and stored in phases 1 and 2.

The complexity of this phase depends on the specific attack type which is summarized in Equation 13 below:

$$\text{cost of } \mathrm{MACM}_{\text{phase 3}} = \text{cost (DoS detection)} + \text{cost(Interruption detection)} + \text{cost(blocking on dest IP)} + \text{cost (blocking on srcIP)} + \text{cost(MITM attack at destIP)} \quad (13)$$

Analyzing equation 13 based on the MACM phase 2 code in Algorithm 4, we get equation 14 which depends on the number of egress and ingress packets in the collected traffic, f_{em} and f_{im} respectively. This is formulated as follows:

$$\text{Cost\_ } \mathrm{MACM}_{\text{phase 3}} = O(f_{em}) + O(f_{em}) + O(f_{im}) + O(f_{im}) + O(f_{em}) = O(f_{em} + f_{im}) \quad (14)$$

Based on equation 8 and 10, the complexity cost of the MACM module is designated as follows:

$$T_{MACM}(f_{em}, f_{im}) = \text{Cost of } \mathrm{MACM}_{\text{phase 1,2}} + \text{Cost of } \mathrm{MACM}_{\text{phase 3}}$$

$$T_{MACM}(f_{em}, f_{im}) = \theta(C + 2(f_{em} + f_{im})) + O(f_{em} + f_{im}) \quad (15)$$

The MACM module is therefore effectively order of $O(f_{em} + f_{im})$.

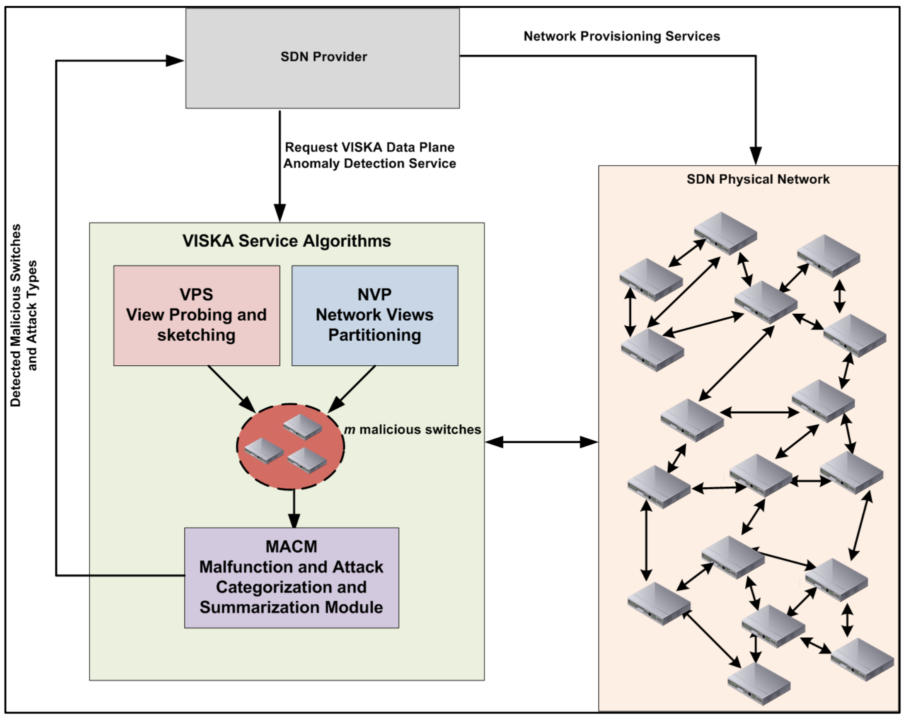

SDN Provider
Network Provisioning Services
Request VISKA Data Plane Anomaly Detection Service
SDN Physical Network
VISKA Service Algorithms
VPS View Probing and sketching
NVP Network Views Partitioning
m malicious switches
MACM Malfunction and Attack Categorization and Summarization Module
Detected Malicious Switches and Attack Types

Fig. 4.6 - : VISKA general architecture

A summary of the complexity analysis of the MACM module is presented in Table 4.4.

In comparison with the literature of intrusion detection mechanisms, VISKA's worst case complexitiy of $O\,(n^2 e\,(\log n)^2)$ in detecting the source of malicious attacks is considered highly efficient. In the most prominent IDS backtracking plugin to the SNORT system implemented in [86], the intrusion detection worst case complexity was found to be $O\,(p^4)$. The input parameter p in [86] represents the number of malicious packets generating the attack. This is usually much higher than the number of nodes n that we followed in VISKA's algorithmic complexity analysis. In [87] the system maintains a set of Tunable

Finite Automaton (TFA) states for categorizing malicious attack sources. The number of active states *b* is linked to the total number of states by the below worst case complexity cost:

$$P = \sum_{i=1}^{b} \binom{N_T}{i} = O({N_T}^{b})$$

Where P is the number of all possible statuses and N_T is the number of TFA states. The runtime worst case complexity is directly proportional to *P* in TFA setups. Comparing to VISKA, complexity-wise $N_T^b \gg \mathrm{n}^2\mathrm{e}\,(\log \mathrm{n})^2$. This renders the VISKA worst-case runtime much more efficient than pattern matching intrusion detection implementations in TFA setups.

Non-Deterministic Finite Automata (NFAs) for deep packet inspection using pattern matching currently exhibits a very poor exponential runtime algorithmic cost in its various implementations using: (1) backtracking algorithms (pcre), (2) Table lookup (lex) by converting the NFA to a Deterministic Finite Automata (DFA), and (3) On-the-fly determinization by employing the grep command implementation. A similar exponential IDS in SDN environments in presented in [88].

Sketch Size Analysis

The sketch data structures are created on the controller probing module and are incrementally updated based on the probing data packets. The size of the sketch allocated counters has a great influence on the error ϵ and the confidence level δ of the sketch computations.

Sketch number of counters W: Based on the analysis in [20, 21], using two four-wise independent hashing functions for the index $\{0, .. \mathrm{W}-1\}$ and the value $\{-1, +1\}$ computations, the second normal difference $|\mathrm{s}(\mathrm{D}_1) - \mathrm{s}(\mathrm{D}_2)|^2 = \Delta$ will correctly estimate $|D_1 - D_2|^2$ with error ϵ and confidence level $(1 - \delta)$ given that the depth of the sketch W (number of counters) is directly proportional to the number of computations required by the four-wise independent hashing function in the worst case complexity analysis: $W \in O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$. This is described in section 4.1.2.1. Therefore, W for each sketch is dependent on the error ϵ and the confidence level δ and not on the number of the probing data packets k. Having W independent of the size of the input data is of great significance in the VISKA algorithm implying that the probing size can be dynamically increased with limited space requirements on the controller.

Sketch counter size z: Each counter in the sketch data structure can hold values between $[-b, +b]$ as a result of the increments/decrements of the hash function on the probing data packets. Range b depends on the size of memory allocated for each counter. Evidently, b is dependent on the size of the probing input k, the depth of the sketch, and the computations' confidence level. Based on the following equation, b is $O(\log k)$:

$$z = \log(2b) = 1 + \frac{1}{2}\log\left(4\frac{k}{W}\ln\left(\frac{200W}{\delta}\right)\right) \tag{16}$$

Equation 16 is based on the following analysis and computations [20]. Based on the union bound [2], no counter overflows in the counter array with a maximum probability of $(1 - \frac{\delta}{100})$ since the confidence level is $(1 - \delta)$ for the correct functioning of the sketch with error ϵ. This suffices that a counter would overflow with a maximum probability of $(\frac{1}{W}\frac{\delta}{100})$. Consider variable X_i to represent the sketch resulting values in $\{-1, +1\}$ to be stored in the sketch counters at index i. X_i will be equal to 1 with ptobability W/2 and -1 with probability

W/2 and 0 otherwise. Variable $X = \sum_{i=1}^{k} X_i$ is the result count in each bin after applying all the input probing packets. Applying the Chernoff bound [36] for the size of each counter $|X|$ to exceed the allocated size b, we get the following:

$$P(|X| \geq b) \leq 2e^{-b^2/2k\mathrm{Var}[X_i]} < \frac{\delta}{100W} \qquad (17)$$

knowing that $Var\ [X_i] = 1/W$ and solving 17, results in equation 16.

Table 4.4 VISKA complexity analysis summary of the macm attack categorization module

Term	Meaning	Value
$T_{MACM}(f_{em}, f_{im})$	The cost of the MACM attack categorization module as a function of the number of egress packets f_{em} and the number of ingress packets f_{im}.	Cost of $\mathrm{MACM}_{\mathrm{phase\ 1,2}}$ + Cost of $\mathrm{MACM}_{\mathrm{phase\ 3}}$
Cost of $\mathrm{MACM}_{phase\ 1,2}$	Represents the cost of data collection and applying the hash operation on the destination IP and source IP addresses of each egress and ingress packet.	$\theta(C + 2(f_{em} + f_{im}))$
Cost_ $\mathrm{MACM}_{\mathrm{phase\ 3}}$	The worst case runtime cost of detecting the different types of DoS, Interruption/Blocking, and MITM attacks.	$O(f_{em} + f_{im})$
$T_{MACM}(f_{em}, f_{im})$	Adding the costs of phases 1, 2, and 3 and simplifying the complexity terms, we get the worst case runtime cost of the MACM module to be in the order of $f_{em} + f_{im}$.	$\theta(C + 2(f_{em} + f_{im})) + O(f_{em} + f_{im}) = O(f_{em} + f_{im})$

4.1.3 System Implementation

The VISKA system design is implemented on top of the Mininet network emulator. We created two main testbed network topologies in Mininet represented in the theoretical linear topology and the popular data center FatTree [52] network topology. This choice is adopted to test the VISKA algorithm behaviour on diverse network topologies in order to empirically analyze the performance efficiency of the algorithms on linearly- and hierarchicallyconnected network switch infrastructures. The technical specification of the simulation environment is described in Table 4.5.

Table 4.5 Simulation environment technical details

SDN controller	FloodLight 0.91 [32]
Network topology	Linear and FatTree
Network size	For each topology, we implemented five network sizes comprising 10, 50, 100, 150, 200, and 250 SDN switching elements. These parameters are chosen to cover a wide range of data center sizes for viably testing the scalability of the algorithms on real SDN networks.
Remote cloud service	The cloud algorithms are developed using the Java platform and deployed on an Amazon EC2 [16] VM.
Cloud VM instance	the t2.micro with 1 vCPU and 1 GB RAM
Physical machine running Mininet	MacBook Pro Mid 2015 laptop running OSX10.12 and supported with 2.5 GHz Intel Core i7 and 16 GB of DDR3 RAM

A probing module is developed on the FloodLight SDN controller for the purpose of (1) exchanging a set of k probing packets, (2) calculating the corresponding sketches and timestamp accumulators, and (3) sending their sketch and timestamp data structure results to the VISKA cloud service.

The VISKA cloud service calculates the sketch's second norm difference estimation as well as the valid timestamp differences of the probing packets and recursively executes the NVP graph partitioning algorithm based on the detection of malicious operation in the tested partitions.

To simulate the Category II attack (refer to Section 4.2.2.3), we used the Mininet delay property on all the links connecting the malicious switch SW6 to the neighbouring switches

SW3, SW4, SW7, and SW10 (refer to Figure 4.4). The implementation value set for the delay property to initiate a delay attack is 40 ms.

To simulate the various Category I attacks presented in Section 4.2.2.3, we used the Floodlight REST API's. Table 4.6 describes the list of attacks supported:

Table 4.6 Attack types supported by the MACM module

Attack Type	Description
DoS attack on host with IP address 10.0.0.4	To simulate this attack, we crafted a static flow action rule that commands the malicious switch SW6 to forward all traffic received on any of its ingress ports to the IP address 10.0.0.4. The following is an example of a StaticEntryPusher command that simulates this attack: `curl -X POST -d '{"switch" : "00:00:00:00:00:00:00:06" , "name" : "DOS_on_10.0.0.4" , "priority" : "32768" , "in_port" : "any" , "active" : "true" , "eth_type" : "0x0800" , "actions" : "set_field = eth_dst → a6:b8:34:23:8e:42 , set_field = ipv4_dst → 10.0.0.4, output = all" }' http://localhost:8080/wm/staticentrypusher/json`
DoS attack on controller	Analogous to the DoS attack on a specific IP, we command SW6 to send all traffic received on any of its ingress ports to the controller.
Interruption of traffic to host with IP address 10.0.0.4	To simulate this attack scenario, we craft a static flow action rule that commands the switch SW6 to drop a portion of the packets destined to host 10.0.0.4. For instance, switch SW6 only drops the packets with destination IP 10.0.0.4 and received on ingress port 1. Packets received on ingress ports 2 and 3 and destined to 10.0.0.4 are forwarded normally.
Blocking traffic destined to host: IP address 10.0.0.4	To simulate this attack, we crafted a rule similar to the one above but which commands the malicious switch SW6 to drop all packets destined to host 10.0.0.4. That is SW6 drops packets destined to 10.0.0.4 received on any of its ingress ports. This is achieved using the StaticEntryPusher commands.
Blocking traffic from host with IP address 10.0.0.3	In this attack scenario the malicious switch SW6 is commanded to drop all packets with source IP 10.0.0.3 received on any of its ingress ports.
Man-in-the-Middle attack via host 10.0.0.2	In this attack scenario, the malicious switch SW6 modifies the destination IP and Ethernet addresses to those of the host 10.0.0.2.

We simulated the attacks described in Table 4.6 using the FloodLight REST APIs. This resulted in generating a dataset for testing the performance (accuracy) of the attack categorization algorithms in the testbed implementation as well as in further real-world deployments on target SDN networks. The generated dataset is comprised of TCPDump raw network packets collected in the SDN network topology presented in Figure 4 and generated using the IPerf tool [59], web browsing, email messaging, video streaming over a time period of 4 hours. The dataset consists of 130547 packets with a total size of 978 MBs simulating 30 DoS attacks, 30 Interruption/Blocking attacks, and 30 MITM attacks. The main purpose of the generated data set is to tune the attack detection thresholds employed in the MACM algorithm to optimal values based on the ROC curves described later in this section.

Table 4.7 Summary of terms used in the simulation

Network Size n	10, 50, 100, 150, 200, 250
$\tau\ (\geq \epsilon)$	0.1
$1-\delta$	99%
Δ	≥ 0.1
k	10^8 packets
z	18 bits
$W > \left(\frac{1}{\epsilon^2} log \frac{1}{\delta}\right)$, size_of (TSA), size_of(Val)	700 counters
Maximum malicious partition size m	3
Percent malicious switches Ms (resulting in $\Delta \geq \tau$)	5%, 10%, and 15% of network size

The attacks introduced in Table 4.6 change in the probing streams that resulted in a second norm difference in the calculated sketches $\Delta \geq \tau$, and in a timestamp difference $\partial > \Gamma_d$ in the delay attack simulation. Subsequently, the attacks were successfully detected to the granularity of m switches, which was set to as low as one switch in the experiments. A highly significant parameter that is considered in the experiments is the percent of malicious switches introduced (Ms). Therefore, we tested a malicious switch number consisting of 5%, 10%, and 15% of the network size for the two topologies. The parameters used in the experiments are summarized in Table 4.7. It is worth mentioning here that we employed a sketch data structure of 1575 bytes (W = 700 and z = 18 bits), which is typically the size of a single IP packet, to summarize a network traffic flow consisting of 10^8 probing packets.

In this work we present the analysis of the DoS, Interruption/Blocking, and MITM attacks for each of the previously mentioned topologies and sizes. In the analysis of the VISKA attack detection and localization part, for each tested configuration, we calculate the average number of recursive steps and the total average convergence time needed by the VISKA VPS and NVP algorithms to localize all malicious nodes in the different SDN network topologies. The experiments on each configuration are replicated 5 times. The average number of recursive steps and the convergence time results are plotted in Figures 4.7 and 4.8, respectively, for the linear topology, and Figures 4.9 and 4.10 for the FatTree topology.

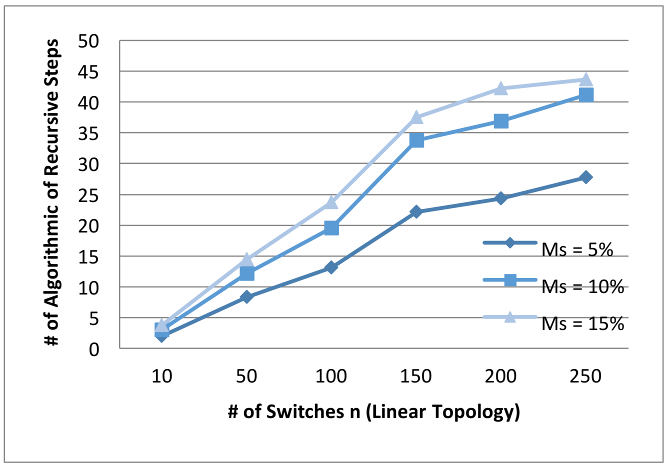


Fig. 4.7 - : Average number of recursive steps required to detect the malicious switches in the linear network topology.

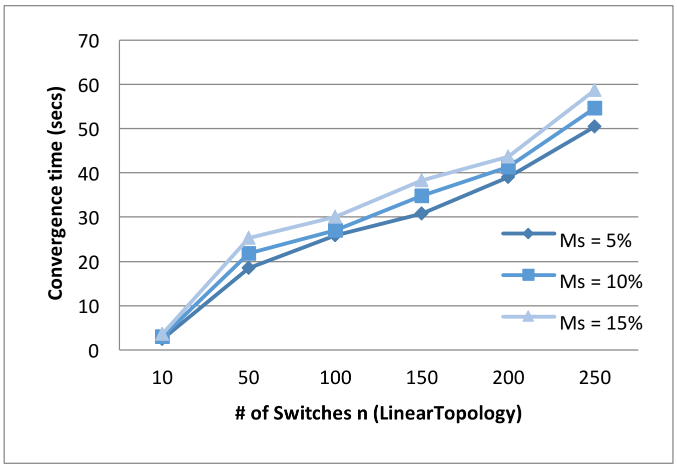


Fig. 4.8 - : Convergence time required by VISKA to detect the malicious switches in the linear network topology.

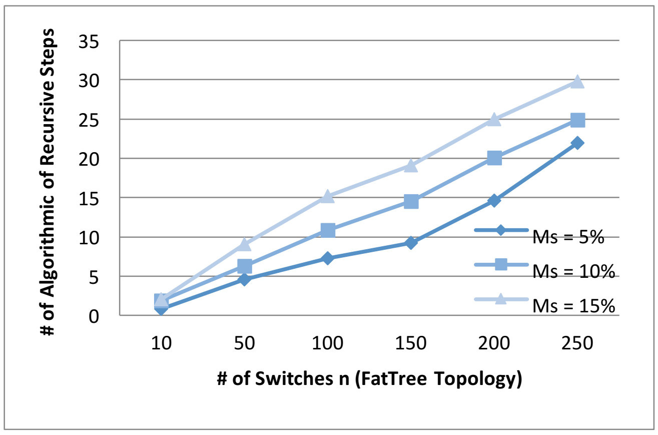


Fig. 4.9 - : Average number of recursive steps required to detect the malicious switches in the FatTree network topology .

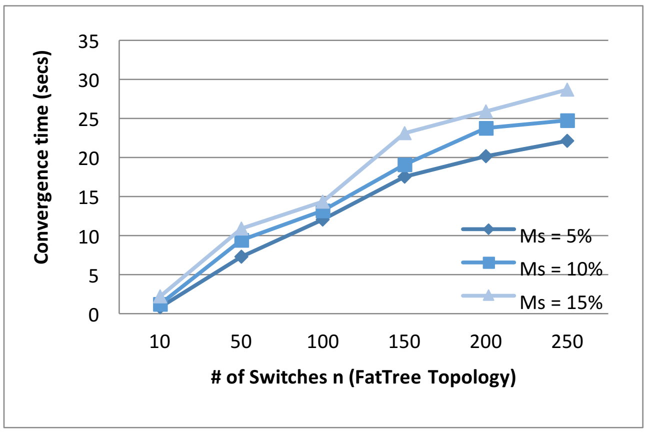


Fig. 4.10 - : Convergence time required by VISKA to detect the malicious switches in the FatTree network topology.

The VISKA algorithms successfully converged to detecting the sources of malicious forwarding in the SDN data plane in all the executed experiments. The presented results demonstrate relatively better performance of VISKA on the FatTree topology compared to the linear one. The improvement reached an average of 42% in the number of recursive steps and 49.6% in the convergence time over the different network sizes and degrees of maliciousness tested. This renders the VISKA algorithms better suited for operation on a real hierarchical data center topology represented by the FatTree topology. Evidently, the convergence time of VISKA algorithms is proportional to the SDN network size and the degree of switch maliciousness. The proposed VISKA service is thus demonstrated to provide the network with a scalable network security solution with the flexibility and dynamism of software. The convergence time results support the scalability of the algorithms on linear and hierarchical data center topologies.

In the linear topology, the increase in network size from 10 to 50 resulted in a convergence time increase of 16 sec (Ms=5%), 18.75 sec (Ms=10%), and 21.66 sec (Ms=15%), while the increase from 50 to 250 resulted an increase of 32 sec (Ms=5%), 32.84 sec (Ms=10%), and 33.5 sec (Ms=15%).

Similarly for the *FatTree* topology, the increase in netwok size from 10 to 50 resulted in a convergence time increase of 6.5 sec (Ms=5%), 8.3 sec (Ms=10%), and 8.7 sec (Ms=15%), while the increase from 50 to 250 resulted an increase of 14.85 sec (Ms=5%), 15.3 sec (Ms=10%), and 17.8 sec (Ms=15%).

The uniform variations in the convergence time as the network size and degree of maliciousness increase show that the algorithms scale well as the network size and degree of maliciousness increase because of the recursive polylogarithmic nature of the VISKA partitioning algorithm, which focuses the security probing, solely, on isolated parts of the network.

The analysis of the MACM attack categorization algorithm is realized on the topology presented in Figure 4.4, for simplifying the analysis. This starts by implementing a mechanism to collect the ingress and egress traffic flowing into/from the malicious switches, respectively. To achieve this, we use the *Floodlight* REST APIs to push static flow action rules into the mc switches that are sending and receiving traffic from/to the malicious switch(es). The static flows pushed by the controller will transparently result in sending all traffic destined to and received from a maliciously-identified switch into the controller. Crafting such flows in *OpenFlow* is pretty simple: first, we create a JSON message indicating (1) the name of the flow, (2) the DatapathID (DPID) of the switch we want to insert this flow on, (3) the set of criteria matching the traffic to be transparently redirected, and (4) the actions to be executed by the switch on the traffic matching the flow criteria. For instance, transparently sending a copy of all traffic flowing from switch SW6 to switch SW10 in the topology demonstrated in Figure 4.4, we leverage the curl tool [60] to push the corresponding action rule as follows:

```
curl -X POST -d '{"switch": " 00:00:00:00:00:00:00:0a", "name": "flow-SW6
              -SW10", "priority": "32768", "in_port": "4", "active": "true",
   "eth_type": "0x0800", "actions": "set_field = output
                 = normal, controller"}' http://localhost:8080/wm/staticentrypusher
                 /json
```

Where

- *"switch":" 00:00:00:00:00:00:00:0a"* is the DPID of switch SW10
- *"in_port":"4"* designates the SW10 interface connected SW10 to SW6.
- *"actions":"set_field= output=normal, controller"* commands SW10 to send all traffic matching the specified criteria to the controller and to the normal interface specified by the switch's L2 pipeline.

Analogous static flow rules are injected into switches SW3, SW4, and SW7. Moreover, similar JSON messages are crafted to configure the *Floodlight* controller to send action rules to switches SW3, SW4, SW7, and SW10 enforcing the transfer of a copy of the ingress traffic destined to SW6 into the controller.

After collecting the ingress/egress traffic, we applied the *hashs* and *hashd* aggregation functions to categorize the packets based on the source and destination IPs, respectively. The list of MACM phase 2 features specified in Section 4.2.2.3 are extracted from the aggregated data and analyzed for the purpose of attack categorization. The hashing and analysis procedures are implemented in Python in the Floodlight controller's address space.

The most significant step in the accurate detection of the various attack types in the MACM attack categorization module is the tuning of the threshold parameters to achieve an optimal true positives/false positives attack detection rate. The MACM attack categorization module relies on:

1. The ΓEd, Γdos, Γp threshold parameters for detection DoS attacks.

2. The Γbh threshold parameter for detecting Interruption/Blocking attacks.

3. The Γmitm threshold parameter for detecting MITM attacks.

Table 4.8 Threshold values used in the MACM implementation

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
ΓEd	3.3	3.19	3.08	2.97	2.86	2.75	2.64	2.53	2.42	2.31	2.2
Γdos	1069977	1019450.3	968923.6	918396.9	867870.2	817343.5	766816.8	716290.1	665763.4	615236.7	564710
Γp	1857	1769.3	1681.6	1593.9	1506.2	1418.5	1330.8	1243.1	1155.4	1067.7	980
Γbh	722	688.45	654.9	621.35	587.8	554.25	520.7	487.15	453.6	420.05	386.5
Γmitm	516	492.75	469.5	446.25	423	399.75	376.5	353.25	330	306.75	283.5
	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	
ΓEd	2.09	1.98	1.87	1.76	1.65	1.54	1.43	1.32	1.21	1.1	
Γdos	514183.3	463656.6	413129.9	362603.2	312076.5	261549.8	211023.1	160496.4	109969.7	59443	
Γp	892.3	804.6	716.9	629.2	541.5	453.8	366.1	278.4	190.7	103	
Γbh	352.95	319.4	285.85	252.3	218.75	185.2	151.65	118.1	84.55	51	
Γmitm	260.25	237	213.75	190.5	167.25	144	120.75	97.5	74.25	51	

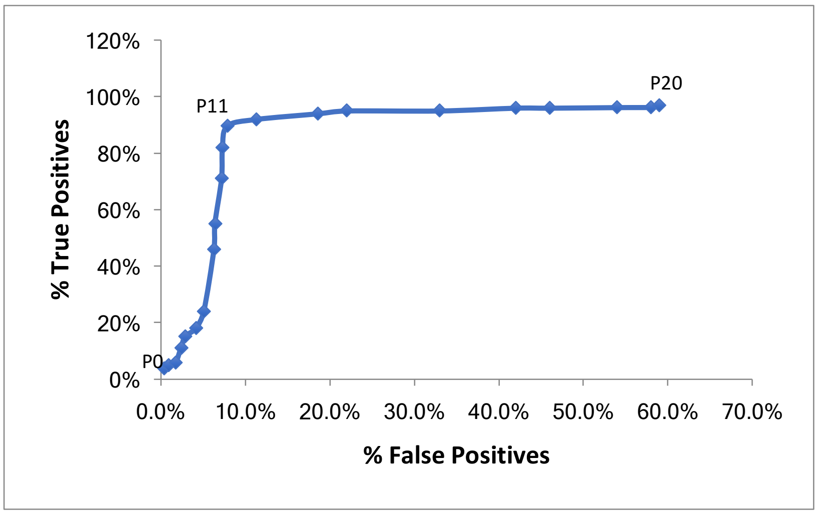


Fig. 4.11 - : ROC of the true positives versus the false positives rates for the different DoS attack threshold parameter triplets ΓEd, Γdos, and Γp.

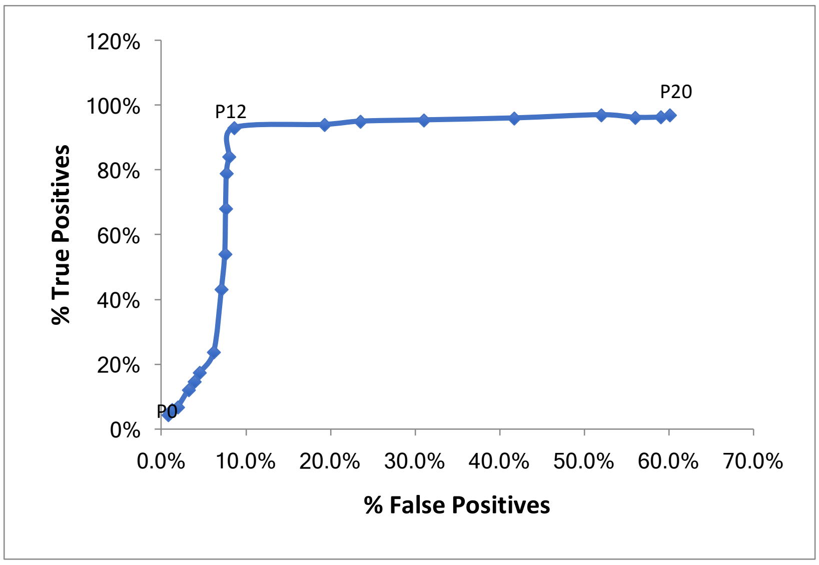


Fig. 4.12 - : ROC of the true positives versus the false positives rates for the different values for the Interruption/Blocking attack threshold parameter Γbℎ.

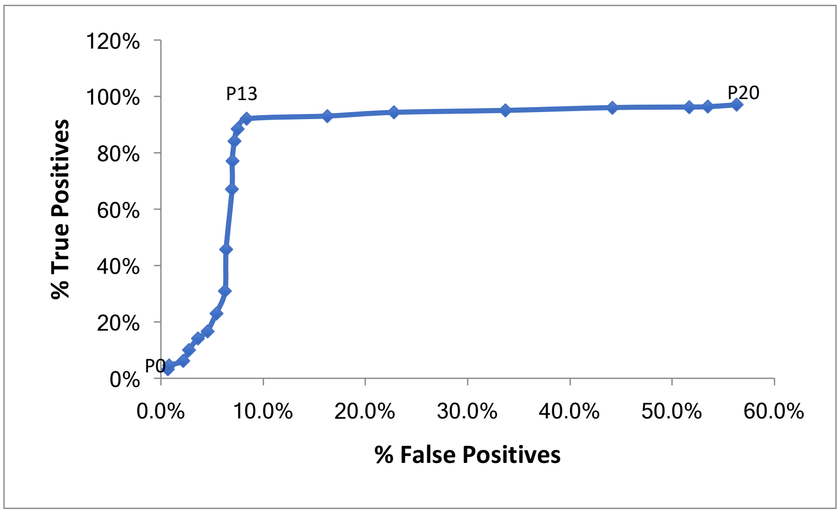


Fig. 4.13 - : ROC of the true positives versus the false positives rates for the different values for the Man-in-the-Middle attack threshold parameter Γmitm.

In the testbed MACM implementation, we utilized 21 threshold values for each attack type in the ranges specified in Table 4.1. The following threshold range parameters are used:

$X_{max} = 1.72\ Mpps$, $P_{avg} = 576\ bytes$, $t_i = 60\ secs$, $\alpha_l = 10\%$, $\alpha_h = 90\%$, $\beta_l = 5\%$, $\beta_h = 70\%$, $\gamma_l = 5\%$, γ_h =50%. The 21 threshold values are presented in Table VIII and are generated in increments of 5% between the low threshold range and the high threshold range. The attack scenarios included in the dataset described earlier in this section are adopted on each of the 21 threshold values presented in Table 4.8 in order to empirically find the optimum value for the DoS thresholds $\Gamma Ed, \Gamma dos, \Gamma p$, the Interruption/Blocking threshold $\Gamma b h$, and the MITM threshold $\Gamma mitm$. Based on the true positives and false positives rates of the attack detection system observed, we generate the ROC curve for each attack type (refer to Figures 4.11, 4.12, and 4.13). The resulting ROC curves for each attack type shows that at higher values of thresholds, the system does not detect the attack. As we gradually increment the threshold values, an optimum point is reached representing the best true positives/false positives detection rate. This is represented in point P12 in the DoS ROC (refer to Figure 4.11) where $\Gamma Ed = 1.98, \Gamma dos = 463.6 \times 10^3, \Gamma p$ = 804.6 at which the system resulted in almost 90% true positives rate and around 8% false positives rate. After this point, for lower threshold values, the attack is detected, however the false positives rate increases rapidly. The optimum threshold value for the Interruption/Blocking attack detection is located at point P12 (refer to Figure 4.12) with $\Gamma b h$=319.4 resulting in a 93% true positives rate and an 8.6% false positives rate. The optimum $\Gamma mitm$ (213.75) for the MITM attack detection is located at point P13 (refer to Figure 4.13) with a true positives rate of 92% and a false positives rate of 8.4%.

4.2 Secure Authentication for the Edge Using SDN and Wireless Virtualization

This section presents a secure authentication protocol for edge computing using the

Software-Defined Networking platform and wireless network virtualization. The main contribution of the presented protocol lies in: (1) designing and implementing a practical solution to the security problem of rogue Fog nodes, and (2) providing the cloud service provider with exclusive control over the security configuration and specification of its leased virtual networks independent of the security mechanisms implemented by the underlying infrastructure provider. The cloud provider dynamically enforces its security policies on the communication at the network edge, therefore circumventing any possible security vulnerabilities in the underlying physical wireless infrastructure, such as the recently discovered key reinstallation vulnerability in WPA2. The authentication protocol manages the trusted communication among the mobile client, the infrastructure provider, and the cloud service provider, yet confines the PKI deployment to solely the infrastructure and cloud providers. This enhances the scalability of the system and reduces the complexity of its security management and configuration. A system testbed is simulated using the Mininet emulator and the Amazon EC2 Cloud.

4.2.1 System Design

The SDN architecture is targeted by all the members of the networking industry namely: Internet service providers, network infrastructure providers, network vendors, enterprises, and end users [62]. The innovations of the SDN architecture are mainly represented in: (1) the separation of the control and management plane from the data plane, (2) the centralization of the control plane, (3) the availability of control plane programmability APIs, and (4) the support of dedicated flow-based protocol control. SDN is utilized in the MEC/Fog architecture in which the Cloud is the central controller of the Fog servers where selective applications are localized and are timely synchronized and managed by the controller.

Next, we present the system and threat models and then, move on to describe the security protocol steps executed by the mobile client, Cloud service provider, and the network infrastructure provider to ensure the security objectives described in Section 4.1.1.

4.2.2 System Model

The system model we present in this work is a MEC architecture based on the virtualization of computing, storage, and networking infrastructure and more specifically, virtualizing the wireless access points and base stations. An infrastructure provider (InP) offers infrastructure services that are leased by Cloud service providers to offer their clients localized services, in order to enhance the QoS delivered using wireless access points. Mobile users connect to virtual wireless access points to gain access to the Cloud servers locally at the network edge. The system architecture components are described as follows (refer to Figure 4.14):

- *Infrastructure Provider (InP):* The InP manges and offers wide varieties of physical network resources and substrates that meet the requirements and needs of tenants. This entity lies at the lowest system level and provides the computing, storage and network resources for the MEC system. This provider can range from a resource-lucrative Internet Service Provider (ISP) to a relatively small Internet cafe to provide a wireless Internet access point as well as network infrastructure leased by Cloud service providers.
- *Cloud Service Provider (CSP):* The CSP provides clients with a set of Cloud services based on the Cloud computing model. To enhance the mobile client experience when

operating on their Cloud services, CSPs utilize MEC computing by leasing computing, storage, and networking infrastructure at the InP site.

- *Virtual Network Layer:* The InP in the proposed system, provides CSPs with full control over their leased MEC networks by utilizing network virtualization and SDN. Network virtualization facilitates the feasibility of creating multiple heterogeneous networks on the same physical infrastructure (Figure 4.14). Sharing the physical resources results in "elastic" network topologies composed of virtual nodes connected via virtual links to provide dynamic end-to-end connectivity services. The network virtualization property, imposed by a network virtualization hypervisor layer, provides the spawned virtual networks with full isolation among each other to achieve relatively high levels of privacy and security. In such virtualization environments, virtual networks are characterised by elevated degrees of flexibility and ease of management, elasticity and dynamism, scalability, isolation, and heterogeneity. Wireless virtualization is moreover employed by utilizing the capabilities of modern wireless base stations to support mulitple SSIDs, which map to the various virtual base stations at the virtual network layer. Each registered CSP, in the InP network infrastructure, is provided with virtual network and access point and is supported with a full access control over the leased resources. In other words, the CSP is the only entity that specifies which mobile clients are capable of accessing its virtual SSID at the network edge. This is enforced by utilitzing trusted computing units such as TPM modules by the Trusted Computing Group (TCG) [63] to physically and logically secure the access to the SSID key database.

- *Mobile Clients:* mobile clients are the main consumers of the edge Cloud services via their wireless networking interfaces. Mobile clients subscribe to the InP site to gain access to the high QoS MEC Cloud services at the provider site. Mobile clients are granted access to the MEC services by the corresponding CSP.

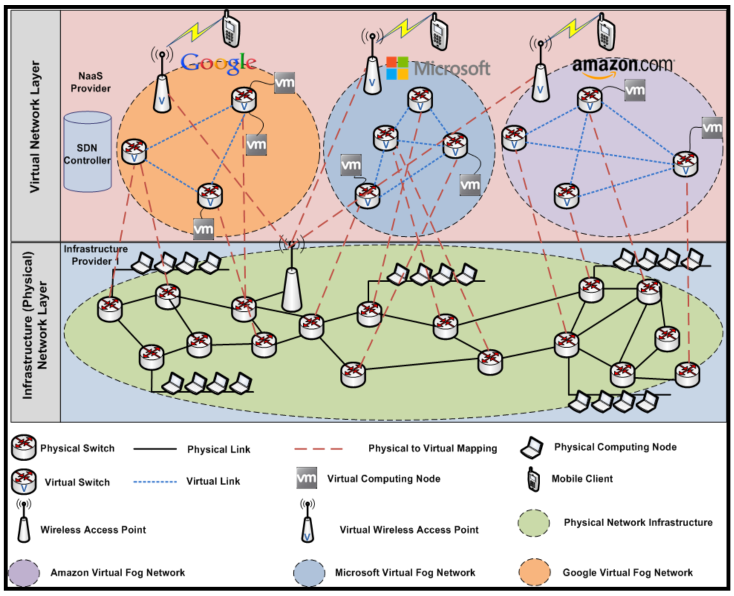
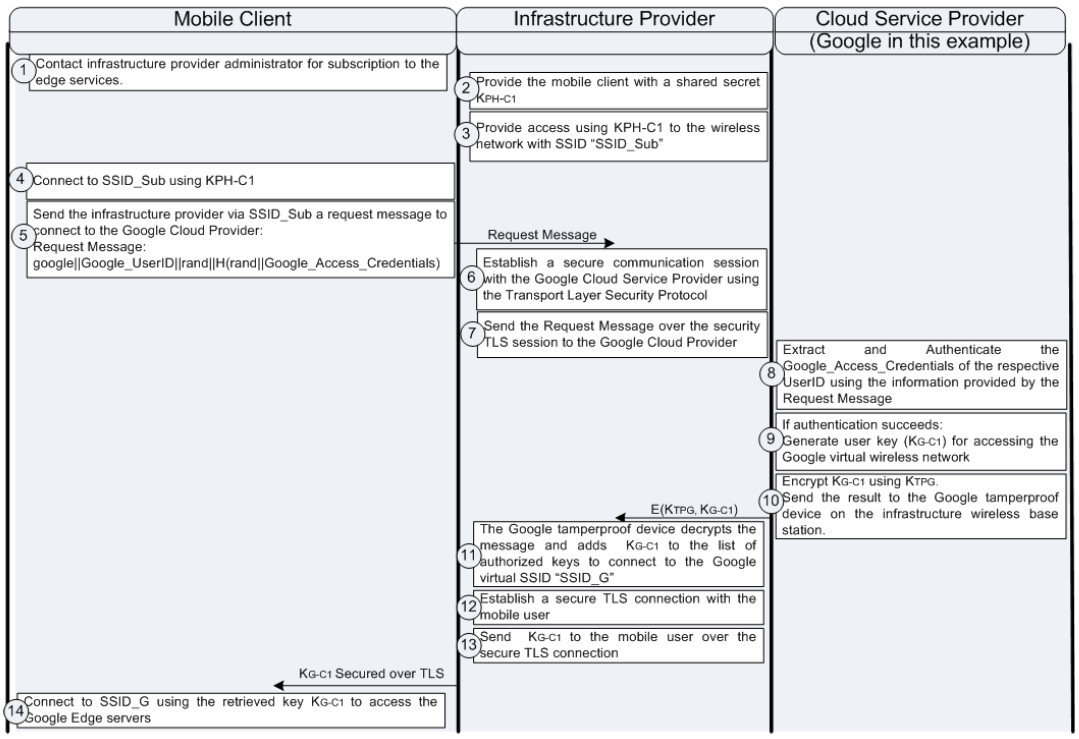


Fig. 4.14 - : The virtualized MEC architecture proposed in this work

4.2.3 Threat Model

The main threat model we assume in this work is represented in the rogue Fog node attack, which results when a certain attacker maliciously injects a false server node in the network and claims its ownership to a particular legitimate CSP. This attack incurs disastrous consequences since it can trick clients to interact with a rogue server node as if it were a legitimate one and thus, it puts huge amounts of client communication data, some of which is considered sensitive, in the hands of the attacker. The attacker may leverage the sensitive data to further execute other forms of active attacks on the underlying network infrastructure. For this reason, exceptional attention should be directed towards mitigating such kinds of attacks. The approach we follow in this work targets this problem by providing the CSP with a full security control over its leased virtual networks by leveraging SDN, wireless virtualization, and trusted computing. All the authentication mechanisms of the Fog servers, as well as the mobile clients are totally under the control of the CSP. Moreover, since the client communication with the Fog nodes is mainly wireless and based on the 802.11i Wi-Fi standards, concerns arise regarding the vulnerabilities in the underlying wireless network security implementations. This is considered highly crucial after the recently discovered key reinstallation vulnerabilities in the WPA2 security standard currently securing 802.11i wireless networks. The security protocol presented mitigates such forms of vulnerabilities by a set of design features that will be described in the security analysis presented in Section 4.2.5.

The system design proposes a novel MEC architecture that delegates the authentication and confidentiality of the edge servers directly to the Cloud service provider to ensure high security levels and mitigate the possibility of rogue Fog node attacks. We assume that the infrastructure and Cloud service providers belong to a public-key infrastructure (PKI) where each entity is equipped with a public/private key pairs. The various entities' public key is signed by a trusted certification authority (CA) and enlcosed in a public-key certificate. It should be noted here that the mobile client may not be part of the PKI. This enhances the scalability of the system and reduces to great degree the complexity of its security management and configuration.



1 Contact infrastructure provider administrator for subscription to the edge services.

2 Provide the mobile client with a shared secret K_{PH-C1}

3 Provide access using KPH-C1 to the wireless network with SSID "SSID_Sub"

4 Connect to SSID_Sub using KPH-C1

5 Send the infrastructure provider via SSID_Sub a request message to connect to the Google Cloud Provider:
Request Message:
google||Google_UserID||rand||H(rand||Google_Access_Credentials)

6 Establish a secure communication session with the Google Cloud Service Provider using the Transport Layer Security Protocol

7 Send the Request Message over the security TLS session to the Google Cloud Provider

8 Extract and Authenticate the Google_Access_Credentials of the respective UserID using the information provided by the Request Message

9 If authentication succeeds:
Generate user key (K_{G-C1}) for accessing the Google virtual wireless network

10 Encrypt K_{G-C1} using K_{TPG}.
Send the result to the Google tamperproof device on the infrastructure wireless base station.

11 The Google tamperproof device decrypts the message and adds K_{G-C1} to the list of authorized keys to connect to the Google virtual SSID "SSID_G"

12 Establish a secure TLS connection with the mobile user

13 Send K_{G-C1} to the mobile user over the secure TLS connection

14 Connect to SSID_G using the retrieved key K_{G-C1} to access the Google Edge servers

Fig. 4.15 - : The authentication protocol execution steps.

4.2.4 The Mobile Client-InP-CSP Security Protocol

The main objective of the mobile client-InP-CSP security protocol is to provide a secure authentication mechanism of the mobile clients and the Fog servers to the CSP. This is done by delegating the security configuration and specification to the Cloud provider itself and providing it with an isolated wireless virtual network whose access is entirely under the control of the CSP. After the authentication step, the mobile clients and the Google VMs can communicate securely using Transport Layer Security (TLS) encrypted sessions.

The protocol steps are described in Figure 4.15:

Firstly, the mobile client (C1) contacts the InP administrator to request access to the Internet access point at the InP site. The InP physically shares a key K_{PH-C1} with the client C1 that enables it to connect to the subscriber wireless network (SSID_Sub). After this step, the client C1 is connected to the InP site using the shared $K_{PH\text{-}C1}$ via the SSID_Sub wireless network. C1 uses the SSID_Sub channel to request access to the MEC services (in this example Google services) at the InP network edge.

In step 4, C1 sends to the InP the following request message:

$$Google||Google\_UserID||rand||H(rand||Google\_Access\_cridentials)$$

The InP learns from the message body the identity of the CSP that the client wants to connect to (Google in this example), and then establishes a secure communication session with it using the secure TLS protocol. The InP forwards the request message to that corresponding CSP (Google) via the secure TLS session. From this point on, the CSP is the entity responsible for the access control to its servers at the network edge (InP site). The CSP extracts from the request message the CSP User-Access-credentials to authenticate the client and ensure his/her eligibility to access its MEC servers. This step prevents any malicious user from accessing the CSP servers at the network edge. Consequently, after authenticating the mobile client, the CSP generates a user key $K_{\mathrm{G-C1}}$ for accessing the CSP virtual wireless network at the edge. To send this key safely to the InP site, the CSP encrypts it using a key K_{TPG} that is previously shared with a tamperproof device installed at the infrastructure wireless base station. The tamperproof device could be a TPM module that is totally configured and controlled by the CSP. The CSP sends the key-access-message: $(E_{K_{\mathrm{TPG}}}(K_{\mathrm{G-C1}}))$ to the tamperproof module. At the InP site, the tamperproof module decrypts the key-access-message and appends $K_{\mathrm{G-C1}}$ to the list of authorized keys database that can connect to the CSP virtual SSID (in this example Google virtual SSID “SSID_G”). The tamper proof device establishes a secure TLS connection with the mobile client C1 over the SSID_Sub channel in order to send the SSID_G access key $\mathrm{K}_{\mathrm{G-C1}}$. The mobile user C1 can now connect using this key to the virtual base station SSID_G using the retrieved key $K_{\mathrm{G-C1}}$ and thereby access the CSP MEC servers at the network edge with high QoS.

Further data communication between the mobile client and the edge servers is exclusively managed by the CSP. One option that the CSP would implement is TLS-based HTTPs sessions for enforcing the confidentiality of data without relying on the underlying security mechanisms provided by the wireless infrastructure.

4.2.5 *Security Analysis*

We formally verified the safety of the proposed security protocol using the SPAN/AVISPA cryptographic verification tool [64]. SPAN/AVISPA analyzes each step in the protocol execution and checks its security against a set of specified safety goals and attack scenarios. The main attacks considered are represented in masquerade, replay, and man-in-the-middle attacks. SPAN/AVISPA is a role-based security verifier that implements the set of acting nodes or agents as roles and specifies the set of messages exchanged among these roles. The protocol specification language employed in SPAN/AVISPA is the High-Level Protocol Specification Language (HLPSL) [65], which consists of a set of formal constructs for defining the roles (agents), the keys and parameters used in the different protocol phases, the messages exchanged among the designated roles, the knowledge assumed by the agents and the attacker, and the set of safety goals and security properties that AVISPA should verify.

The verification scenario we implemented is comprised of 4 main roles: (1) the mobile client, (2) the infrastructure edge provider, (3) the cloud service provider, and (4) the tamperproof edge device. The security properties that AVISPA verified are represented in:

(1) The security of the Google_Access_Credentials in the request message sent from the mobile client to the infrastructure provider in step 5 and the freshness of this request message.
(2) The security of the request message sent from the infrastructure provider to the cloud service provider in steps 6 and 7.
(3) The security of the KG-C1 symmetric encryption key exchanged between the cloud service provider and the Google tamperproof device in step 10.
(4) The security of the KG-C1 symmetric encryption key exchanged between the Google tamperproof device and the mobile client.
(5) The safety against the various forms of masquerading, replay, and man-in-the-middle attacks.
SPAN/AVISPA produced a “safe” output for all the above security properties. A detailed analysis of the presented authentication protocol steps is presented in the following set of points:

(1) In protocol steps 1, 2, 3, and 4 the mobile client gets access to a subscription WiFi network (SSID_Sub). The access to SSID_Sub is provided by physically sharing an access key K_{PH-C1}. This network allows the client to communicate with the InP. In protocol steps 5, 6, 7, and 8 the mobile client authenticates to the CSP to get access to its MEC services.
(2) The access is granted by sending an authentication request message via the InP to the CSP using a secure TLS session. The main components in the authentication request message

are the mobile client user ID and a hash of the access credentials to the MEC services. A random salt value is appended to the access credentials before hashing to prevent any brute force guessing attacks on the access credentials.

If the mobile client is authenticated successfully to the CSP, the latter generates a secret key K_{G-C1}. This is the key that the mobile client uses afterwards to access the CSP virtual wireless network provided by the InP.

(3) To give the CSP exclusive control over the security of its leased virtual WiFi network, a CSP tamper-proof TPM module is installed on the infrastructure wireless base station. K_{G-C1} is securely transmitted to the TPM module by encrypting it with a CSP key K_{TPG} to get $E_{K_{TPG}}(K_{G-C1})$. K_{TPG} can be loaded into the TPM module in the initial deployment phase mainly by burning it into the module's ROM. It is worth mentioning here that using tamperproof modules is not anymore an expensive design choice in current computing and network architectures. This fact is corroborated by a set of indicators in successful network and Cloud implementations that rely on trusted computing modules and tamper-proof coprocessors for providing their security services and trust commitments [66, 67].

(4) In protocol step 11, the TPM module decrypts $E_{K_{TPG}}(K_{G-C1})$

and adds K_{G-C1} to the database of authorized access keys to access the CSP wireless virtual network SSID_G. It is worth mentioning here that the CSP is exclusively controlling the access to SSID_G by generating K_{G-C1} and maintaining a tamper-proof key database for storing it securely on the infrastructure wireless base station using a TPM module.

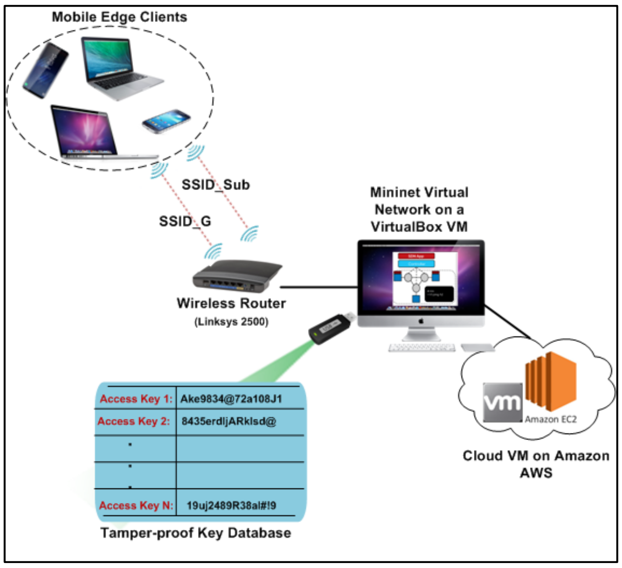
(5) In protocol steps 12 and 13, the TPM module sends K_{G-C1} securely to the mobile client using an encrypted TLS session.

(6) In protocol step 14, the mobile client uses K_{G-C1} to access the virtual CSP wireless network SSID_G.

All further data communication between the mobile client and the CSP edge servers takes place over SSID_G. The CSP dynamically implements its own security policies that governs the communication with the mobile clients independent of the infrastructure security policies. This is highly vital to circumvent any security vulnerabilities in the underlying wireless network [68]. A paragon example on that is the recently discovered key reinstallation attack on WPA2 protocol implementations [69]. Note that WPA2 key reinstallation vulnerability does not jeopardize the authentication protocol steps described in Section 4.2.1.3 since the protocol messages are either innately not confidential or are secured using custom TLS encrypted sessions.

4.2.6 *System Implementation*

In this section, we present a brief proof-of-concept implementation of the proposed MEC authentication protocol. The implementation model is illustrated in Figure 4.16. We execute the functionality of the CSP using an Amazon EC2 VM. The VM instance type used is the t2.micro with 1 vCPU and 1 GB RAM. The InP network infrastructure is simulated using the Mininet network emulator running on a VirtualBox Linux VM. The physical machine employed to run Mininet is an iMac Late 2013 desktop running OSX El Capitan 10.11 and supported with 2.7 GHz quad-core Intel Core i5 processor and 8 GB of 1600 MHz DDR3 RAM. The mobile edge device set utilized consists of two Android mobile devices and two Mac OS X laptops as specified below. We use these devices to emulate the concurrent authe*n*tication protocol execution of up to 100 mobile edge devices.


Mobile Edge Clients
SSID_Sub
SSID_G
Mininet Virtual Network on a VirtualBox VM
Wireless Router (Linksys 2500)
Access Key 1: Ake9834@72a108J1
Access Key 2: 8435erdljARklsd@
Access Key N: 19uj2489R38al#!9
Tamper-proof Key Database
Amazon EC2
Cloud VM on Amazon AWS
Fig. 4.16 - : The system proof of concept implementation.emulate the concurrent authentication

To achieve this, we develop a multi-threaded client application to run on the Android phones and another one to run on the MAC laptops.

Each application thread emulates a mobile edge device. Along with the specs of the devices, we show the maximum number of threads to be executed on each device, which typically represents the maximum number of concurrent authentication protocol executions (MCPE) that are initiated on that particular device. Moreover, we utilize the VMware Fusion server virtualization software to respectively partition the Mac laptops into two guest VMs running Windows and Ubuntu in addition to the Mac OSX main host machine.

- Samsung Galaxy S4 mini running Android KitKat 4.4.4, with 1.7 GHz dual-core Krait 300 and 1.5GB RAM; MCPE=5.
- Samsung Galaxy S8 running Android Nougat 7.0, with 4x2.3 GHz octa-core M2 and 4GB RAM; MCPE=5.

- Apple MacBook Pro Retina (2012) running Mac OSX Sierra 10.12, with 2.3 GHz quad-core Intel Core i7 and 8GB 1600 MHz DDR3 RAM;
    - Main host: 15; Windows 7 VM (2 GB RAM): 15; Ubuntu 14.04 VM (2 GB RAM): 15
- Apple MacBook Pro Retina (2015) running Mac OSX High Sierra 10.13, with 2.5 GHz quad-core Intel Core i7 and 16 GB 1600 MHz DDR3 RAM;
    - Main host:15; Windows 7 VM (2 GB RAM): 15; Ubuntu 14.04 VM (2 GB RAM): 15

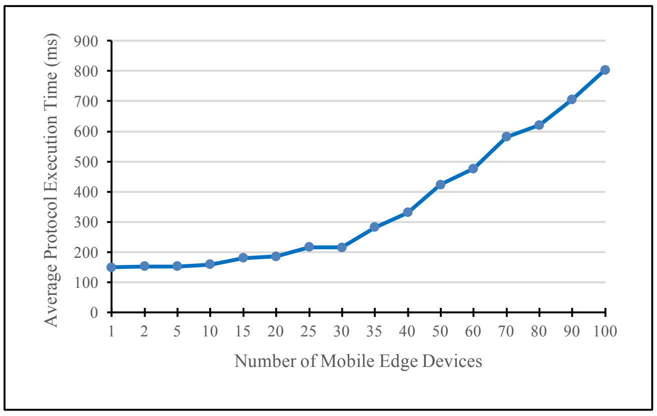


Fig. 4.17 - : The average protocol execution time vs. the number of concurrent protocol executions for authenticating mobile edge clients.

We followed this mobile edge device emulation using light-weight application threads to reduce the wireless channel allocation load on the wireless base station and to better reflect the performance overhead imposed by the authentication protocol operations. The wireless base station employed is the Linksys E2500 (N600) dual-band wireless-n router. The TLS implementation is based on the Java Secure Socket Extension (JSSE) and the encryption and hashing implementations are respectively developed using the Java Cipher and

MessageDigest classes. The tamperproof device is simulatedusing a USB stick encrypted with a CSP key and containing the keys authorized to access the SSID_G Wi-Fi network. We assume that the logic for accessing the SSID_G network channel is totally under the control of the CSP, i.e. executed in the tamperproof device and cannot be logically or physically accessed by the InP. This implementational model is considered a viable proof-of-concept that demonstrates the correct operation of the authentication protocol. A set of 10 experiments was conducted with and without authentication during different times of the day. The MEC server application tested is a variant of the Google Drive file sharing service. The respective response time of the application with authentication on the mobile client experienced an end-to-end time delay of 149 ms (refer to Figure 4.17) from the initiation of the authentication request to receiving the respective WiFi credentials to connect to the Fog server(s). To emulate the MEC authentication protocol operation in a real-life network scenario, we executed the protocol on increasing number of mobile edge devices (using application-level threads) ranging from a single device up to a maximum of 100 concurrent nodes. Figure 4.17 demonstrates the average end-to-end time delay experienced at each node when connecting the respective concurrent devices. As expected, the average time delay increases as the number of connected devices increases to reach 800 ms when all the 100 edge nodes are concurrenlty executing the authentication protocol. This delay is experienced only once and is mainly due to the Internet communication with the main cloud service provider (Amazon EC2 in this implementation). Another important performance parameter that we measured is related to the network overhead imposed by the application of the proposed authentication protocol. This was found to be 6.1 KB, which is mainly due to the overhead of the TLS session between the InP and CSP. This TLS session is only executed once and it should be noted that smart TLS tuning techniques such as session reuse and caching would reduce this overhead to less than 1 KB.

CHAPTER 5

CONCLUSION AND FUTURE DIRECTIONS

In this thesis, we present the research accomplishments that has been achieved which satisfies the thesis goals and objectives. We present the design and implementation of a set of network services and protocols for dynamically and flexibly provisioning, configuring, and securing networks in programmable network environments. This thesis work focuses on the utilization of network virtualization and programmability through the separation of network hardware and software in order to provision novel dynamic cloud services for network tenants. The set of services delivered create a major breakthrough in the current networking platform and secure communication services in order to pave the way for a more dynamic, agile and flexible networking infrastructure. The following is a list that summarizes the research work that has been achieved and the corresponding fulfilled network services.

5.1 Dynamic Vnets Provisioning in Virtualized SDN-Based Cloud Architectures

The main contribution behind this work is to offer an integrated service that handles the different aspects of VNet creation and management on behalf of cloud tenants while satisfying their requirements and constraints. The work employed a flexible network partitioning mechanism that allows the hosting of the network parts on multiple cloud providers to provide a minimum VNet deployment cost. The presented centralized cloud service creates VNets in SDN-based cloud architectures. NCaaS relieves tenants from the burdens and complexities of VNet creation and management by supporting dynamic provisioning operations based on QoS, pricing, privacy, reliability, and energy requirements. NCaaS provides a unified interface through which tenants network specifications and constraints are fed into the service provisioning algorithms. These algorithms, in turn, handle the negotiations with the different SDN NaaS providers and dynamically apply the necessary VNet creation, partitioning, and migration mechanisms to ensure the satisfaction of the tenants' preferences. A mathematical formulation of the cost optimization problem is provided along with a set of approximation algorithms for carrying out the partitioning and topology designs. A test bed proof of concept implementation of the NCaaS algorithms is developed on top of the OpenVirteX network virtualization platform and tested using the Mininet network emulator.

5.2 An SDN-Based Virtualized Network Framework For Linking Geographically Separated Enterprise Branches

Towards the fulfillment of the thesis objective, another main challenge was presented in VNCS, a network connectivity service for linking enterprise branches using virtualized SDN-based provider networks. The VNCS service creates and manages Virtual SDNs (vSDN) for the tenant's sites that are logically decoupled from the network infrastructure, providing configurable network topology and isolation on the SDN network virtualization platform. VNCS service algorithms ensure the dynamic spawning of the tenant specified vSDN that satisfies its constraints with minimum connection cost based on the SDN providers' offers and tenant's specifications and preferences that are inputs to the VNCS algorithms interface. VNCS enables tenants to specify how their remote sites are to be connected along with a list of network constraints independently from the service provider and the infrastructure provider by dynamically configuring, managing and maintaining interconnecting networks that link tenant's remote sites. The work mathematically formulated the network connectivity problem and described an approximation algorithm for solving it. The VNCS service algorithms receive a set of input specifications from the enterprise tenant and SDN providers and ultimately creates a configurable network topology while coordinating the encapsulation/decapsulation mechanisms for unifying the address space among the various branches to appear as seamlessly operating on a single network. VNCS is implemented on top of the OVX network virtualization platform and tested with a wide set of input configurations and network sizes. The implementation analysis verified the correctness of the resulting network topology and the feasibility of the convergence time needed to construct this topology.

5.3 Secure Sketching On Programmable SDN Network Views For Security Service Provisioning

A novel approach in localizing malicious nodes in the SDN data plane and categorizing any present attacks by utilizing network programming and probabilistic sketching is presented. The VISKA security algorithms are designed to run in real time with minimal convergence time for isolating malicious forwarding elements in the data plane. This is the main contribution of the work where malicious switch detection is achieved by an efficient logarithmic divide-and-conquer approach that divides the network view in half in each recursive iteration. The network programming functions in SDN allow the system to autonomously isolate network partitions that may be experiencing malicious activity. This is done flexibly with pure software operations. The attacks detected include: 1) network time delay insertion, 2) MITM, 3) DoS on a certain server, 4) block on a certain source IP, 5) block on a certain destination, 6) miscellaneous blocks to induce network malfunction, and

7) DoS on the controller. The algorithms were tested for convergence using a variety of SDN network sizes and number of malicious switching elements. The various attacks were experimented and the detection thresholds were identified. The system was capable of achieving over 90% detection accuracy. It is worth mentioning here that a very appealing application to the VISKA model is in supporting net neutrality in modern SDN-based NaaS provider networks. VISKA attack categorization mechanisms can provide a valuable feedback on probable breaches that violate net neutrality exertion in an SDN-based network. This is demonstrated by the following points:

1. VISKA detects malicious traffic shaping violations that induce delay attacks on network packets by leveraging the Timestamp Accumulator data structure presented in Section 4.1.2.1.

2. VISKA detects DoS attacks that interfere with the "freedom of speech" approach pushed by the Open Internet [61] standards. The Open Internet approach indicates that the full network resources should be accessible by clients transparently and easily.

3. VISKA prevents any discrimination by IP address by detecting blocking attacks on a certain destination or source network address.

4. VISKA aids in preventing malicious over provisioning of network bandwidth by detecting delay attacks resulting from unfair bandwidth distribution.

5.4 MEC Secure Authentication Protocols using SDN and Wireless Virtualization

We presented a secure and scalable authentication protocol for edge computing using the SDN platform and wireless network virtualization. The edge computing system is mainly based on virtualizing networking resources including the wireless access point where an ISP offers infrastructure services that are leased by Cloud service providers to offer their clients localized services using the ISP access point. In such architecture, it was feasible to devise a security protocol that assigns the confidentiality and the authenticity of the mobile user-edge services to the Cloud service provider. As a result, the Cloud service provider can fully control and customize the security policies of the edge service communication. The main contribution of this work is to target the security problem of rogue Fog nodes and to provide a secure and flexible network creation and configuration mechanisms for Cloud service providers at the network edge. A system testbed was simulated using the Mininet emulator and the Amazon EC2 Cloud. Future extensions include: (1) extending the authentication architecture to cover cellular MEC platforms, (2) leveraging the tamperproof device to deploy more sophisticated authentication, confidentiality, and privacy modules to provide customized security levels depending on the mobile users' preferences and the services requested and (3) utilizing a real tamperproof cryptographic coprocessor for truly safeguarding the SSID_G key database and operation from the InP.

THESIS RELATED PUBLICATIONS

Conference Papers:

Maha Shamseddine, Wassim Itani, Ali Chehab, and Ayman Kayssi, "VISKA: Secure Sketching on Virtualized SDN Network Views", *in proc. of the IEEE International Conference on Communications (ICC'17), 21-25 May 2017, Paris, France*

Maha Shamseddine, Imad Elhajj, Ali Chehab, Ayman Kayssi, and Wassim Itani, "NCaaS: Network Configuration as a Service in SDN-Driven Cloud Architectures", *in Proc. of the 31st ACM/SIGAPP Symposium on Applied Computing (SAC 2016), April 4 – 8, 2016, Pisa, Italy.*

Maha Shamseddine, Imad Elhajj, Ali Chehab, Ayman Kayssi, "VNCS: Virtual Network Connectivity as a Service – A Software-Defined Networking Approach", *in 3rd IEEE Symposium on Software Defined Systems SDS'2016, in Conjunction with the IEEE international Conference on Cloud Engineering, April 4 – 8, 2016, Berlin, Germany.*

Maha Shamseddine, Wassim Itani, Ali Chehab, Ayman Kayssi, "Secure Authentication for the Edge using SDN and Wireless Virtualization", submitted for publication in IEEE Globecom 2018.

Journal Papers:

Maha Shamseddine, Wassim Itani, Ali Chehab, Ayman Kayssi, "Network Programming and Probabilistic Sketching for Securing the Data Plane", in Security and Communication Network, Volume 2018, https://doi.org/10.1155/2018/2905730.

Book Chapters:

Maha Shamseddine, Ali Chehab, Ayman Kayssi, and Wassim Itani, "On-Demand Network Virtualization and Provisioning Services in SDN-Driven Cloud Platforms", *accepted for* *publication in the IET book "Network as a Service for Next Generation Internet" edited by Qiang Duan and Shangguang Wang.*

REFERENCES

[1] Costa, Paolo, Matteo Migliavacca, Peter Pietzuch, and Alexander L. Wolf. "NaaS: Network-as-a-Service in the Cloud." *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. 2012.

[2] Nunes, Bruno Astuto A., Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. "A survey of software-defined networking: Past, present, and future of programmable networks." *IEEE Communications Surveys & Tutorials* 16, no. 3 (2014): 1617-1634.

[3] Al-Shabibi, Ali, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, Guru Parulkar, Elio Salvadori, and Bill Snow. "OpenVirteX: Make your virtual SDNs programmable." In *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 25-30. ACM, 2014.

[4] Das, Saurav, Guru Parulkar, and Nick McKeown. "Rethinking IP core networks." *Journal of Optical Communications and Networking 5*, no. 12 (2013): 1431-1442.

[5] Hao, Fang, T. V. Lakshman, Sarit Mukherjee, and Haoyu Song. "Enhancing dynamic cloud-based services using network virtualization." In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pp. 37-44. ACM, 2009.

[6] Chowdhury, NM Mosharaf Kabir, and Raouf Boutaba. "Network virtualization: state of the art and research challenges." *IEEE Communications magazine*47, no. 7 (2009): 20-26.

[7] Drutskoy, Dmitry, Eric Keller, and Jennifer Rexford. "Scalable network virtualization in software-defined networks." *IEEE Internet Computing* 17, no. 2 (2013): 20-27.

[8] Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. "Xen and the art of virtualization." In *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164-177. ACM, 2003.

[9] Armbrust, Michael, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee et al. "A view of cloud computing."*Communications of the ACM* 53, no. 4 (2010): 50-58.

[10] Mijumbi, Rashid, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. "Network function virtualization: State-of-the-art and research challenges." *IEEE Communications Surveys & Tutorials* 18, no. 1 (2015): 236-262.

[11] Hipp, Emily L., Yuh-yen Yeh, and Burton A. Hipp. "Virtual network environment." *U.S. Patent 7,146,431, issued* December 5, 2006.

[12] Jain, Raj, and Subharthi Paul. "Network virtualization and software defined networking for cloud computing: a survey." *IEEE Communications Magazine*51, no. 11 (2013): 24-31.

[13] Sezer, Sakir, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. "Are we ready for SDN? Implementation challenges for software-defined networks." *IEEE Communications Magazine* 51, no. 7 (2013): 36-43.

[14] Medved, Jan, Robert Varga, Anton Tkacik, and Ken Gray. "Opendaylight: Towards a model-driven sdn controller architecture." In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. 2014.

[15] McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. "OpenFlow: enabling innovation in campus networks." *ACM SIGCOMM Computer Communication Review* 38, no. 2 (2008): 69-74.

[16] Wen, Heming, Prabhat Kumar Tiwary, and Tho Le-Ngoc. "Wireless virtualization." In *Wireless Virtualization*, pp. 41-81. Springer International Publishing, 2013.

[17] Turner, Sean. "Transport Layer Security." *IEEE Internet Computing* 18, no. 6 (2014).

[18] Drutskoy, Dmitry, Eric Keller, and Jennifer Rexford. "Scalable network virtualization in software-defined networks." *IEEE Internet Computing* 17, no. 2 (2013): 20-27.

[19] Sherwood, Rob, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. "Flowvisor: A network virtualization layer." *OpenFlow Switch Consortium, Tech. Rep* (2009): 1-13.

[20] Jin, Xin, Jennifer Gossels, Jennifer Rexford, and David Walker. "Covisor: A compositional hypervisor for software-defined networks." In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 87-101. 2015.

[21] OpenStack Neutron home page at: http://docs.openstack.org/developer/neutron/

[22] Sefraoui, Omar, Mohammed Aissaoui, and Mohsine Eleuldj. "OpenStack: toward an open-source solution for cloud computing." *International Journal of Computer Applications* 55, no. 3 (2012).

[23] Itani, W., Kayssi, A., & Chehab, A. (2009). Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures. In *DASC'09. Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, 2009.* (pp. 711-716). IEEE.

[24] Itani, W., Ghali, C., Kayssi, A., & Chehab, A. (2014). Reputation as a Service: A System for Ranking Service Providers in Cloud Systems. In *Security, Privacy and Trust in Cloud Systems* (pp. 375-406). Springer Berlin Heidelberg.

[25] Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Introduction to NP-Completeness of knapsack problems* (pp. 483-493). Springer Berlin Heidelberg.

[26] Martello, S., & Toth, P. (1981). A bound and bound algorithm for the zero-one multiple knapsack problem. *Discrete Applied Mathematics*, *3*(4), 275-288.

[27] Matlab Mathworks home page at: www.mathworks.com/products/matlab/

[28] VirtualBox homepage: http://www.virtualbox.org

[29] Doraswamy, Naganand, and Dan Harkins. *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall Professional, 2003.

[30] Hamzeh, Kory, Grueep Pall, William Verthein, Jeff Taarud, W. Little, and Glen Zorn. *Point-to-point tunneling protocol (PPTP)*. No. RFC 2637. 1999.

[31] Townsley, W., A. Valencia, Allan Rubens, G. Pall, Glen Zorn, and Bill Palter. *Layer Two Tunneling Protocol "L2TP"*. No. RFC 2661. 1999.

[32] Kellerer, H., Pferschy, U., & Pisinger, D.(2004). Introduction to NP-Completeness of knapsackproblems (pp. 483-493). *Springer Berlin Heidelberg*.

[33] 20 Top Internet Service Providers, retrieved from Web: http://www.practicalecommerce.com/articles/3225-20-Top-Internet-Service-Providers, retrieved on Nov. 23, 2015

[34] Yu, Minlan, Lavanya Jose, and Rui Miao. "Software Defined Traffic Measurement with OpenSketch." Presented as part of *the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13),* pp. 29 - 42. 2013.

[35] Y. G. Desmedt. "A Threshold cryptography". In: *European Trans. on Telecommunications* 5.4 (1994).

[36] Betge-Brezetz, Stephane, Guy-Bertrand Kamga, and Monsef Tazi. "Trust support for SDN controllers and virtualized network applications." *IEEE Conference on Network Softwarization (NetSoft), London,* pp. 1-5, 2015.

[37] Kreutz, Diego, Fernando Ramos, and Paulo Verissimo. "Towards secure and dependable software-defined networks." Proceedings of *the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM*, 2013.

[38] Skowyra, Richard, Andrei Lapets, Azer Bestavros, and Assaf Kfoury. "A verification platform for sdn-enabled applications." *IEEE Int. Conference on Cloud Engineering (IC2E)*, pp. 337-342, 2014.

[39] Shin, Seungwon, Phillip A. Porras, Vinod Yegneswaran, Martin W. Fong, Guofei Gu, and Mabry Tyson. "FRESCO: Modular Composable Security Services for Software-Defined Networks." NDSS, 2013.

[40] Yazici, Volkan, M. Oguz Sunay, and Ali O. Ercan. "Controlling a software-defined network via distributed controllers." arXiv preprint arXiv:1401.7651 (2014).

[41] Scott-Hayward, Sandra, Gemma O'Callaghan, and Sakir Sezer. "SDN security: A survey." *In IEEE SDN For Future Networks and Services (SDN4FNS)*, pp. 1-7, 2013.

[42] McKeown, Nick. "Software-defined networking." I*NFOCOM keynote talk 17.2* (2009): 30-32.

[43] Matsumoto, Stephanos, Samuel Hitz, and Adrian Perrig. "Fleet: Defending SDNs from malicious administrators." *Third workshop on Hot topics in software defined networking.* ACM, 2014.

[44] Zaalouk, Adel, Rahamatullah Khondoker, Ronald Marx, and Kpatcha Bayarou. "OrchSec: An orchestrator-based architecture for enhancing network-security using Network Monitoring and SDN Control functions.*" IEEE Network Operations and Management Symposium (NOMS)*, pp. 1-9, 2014.

[45] Betge-Brezetz, Stephane, Guy-Bertrand Kamga, and Monsef Tazi. "Trust support for SDN controllers and virtualized network applications." *IEEE Conference on Network Softwarization* , 2015.

[46] Barman, Dhiman, Piyush Satapathy, and Gianfranco Ciardo. "Detecting attacks in routers using sketches." *IEEE Workshop on High Performance Switching and Routing,* pp. 1-6, 2007.

[47] Cormode, Graham, and Minos Garofalakis. "Sketching streams through the net: Distributed approximate query tracking", *International Conference on Very large data bases, VLDB Endowment, 2005*.

[48] Karger, David,"Global Min-cuts in RNC and Other Ramifications of a Simple Mincut Algorithm.", *4th Annual ACM-SIAM Symposium on Discrete Algorithms,* 1993.

[49] Goldberg, Sharon, et al. "Path-quality monitoring in the presence of adversaries." *In: ACM SIGMETRICS Performance Evaluation Review.* Vol. 36. No. 1. ACM, 2008.

[50] M. Thorup and Y. Zhang, "Tabulation based 4-universal hashing with applications to second moment estimation." *In the proc. of SODA, 2004, pp. 615–624.*

[51] Dhawan, Mohan, et al. "SPHINX: Detecting Security Attacks in Software-Defined Networks." NDSS. 2015.

[52] Al-Fares, Mohammad, Alexander Loukissas, and Amin Vahdat. "A scalable, commodity data center network architecture." *ACM SIGCOMM Computer Communication Review* 38, no. 4 (2008): 63-74.

[53] De Oliveira, Rogério Leão Santos, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. "Using mininet for emulation and prototyping software-defined networks." In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pp. 1-6. IEEE, 2014.

[54] Mininet homepage: http://www.mininet.org

[55] Creating Multiple Redundant Controllers in Mininet, https://github.com/mininet/mininet/blob/master/examples/controllers.py

[56] Microsoft Windows Azure home page: http://www.microsoft.com/windowsazure/

[57] Jankowski, Damian, and Marek Amanowicz. "On efficiency of selected machine learning algorithms for intrusion detection in software defined networks." *International Journal of Electronics and Telecommunications* 62, no. 3 (2016): 247-252.

[58] Mills, David L. "Network time protocol (NTP)." *Network* (1985).

[59] Iperf tool, https://iperf.fr/en/

[60] Conquering The Command Line Unix And Linux Commands For Developers, Chapter 3, CURL.

[61] Ku, Raymond Shih Ray. "Open Internet Access and Freedom of Speech: A First Amendment Catch-22." Tul. L. Rev. 75 (2000): 87.

[62] Jain, Raj, and Subharthi Paul. "Network virtualization and software defined networking for cloud computing: a survey." *IEEE Communications Magazine* 51, no. 11 (2013): 24-31.

[63] Reineh, Ahmad-Atamli, Giuseppe Petracca, Janne Uusilehto, and Andrew Martin. "Enabling Secure and Usable Mobile Application: Revealing the Nuts and Bolts of software TPM in todays Mobile Devices." *arXiv preprint arXiv*:1606.02995 (2016).

[64] Armando, Alessandro, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P. Hankes Drielsma et al. "The AVISPA tool for the automated validation of internet security protocols and applications." *in International conference on computer aided verification,* pp. 281-285. Springer, Berlin, Heidelberg, 2005.

[65] Von Oheimb, David. "The high-level protocol specification language HLPSL developed in the EU project AVISPA." *In Proceedings of APPSEM 2005 workshop,* pp. 1-17. 2005.

[66] Itani, W., A. Kayssi, and A. Chehab. "Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures." In the Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, 2009. DASC'09., pp. 711-716. IEEE, 2009.

[67] Reineh, Ahmad-Atamli, Giuseppe Petracca, Janne Uusilehto, and Andrew Martin. "Enabling Secure and Usable Mobile Application: Revealing the Nuts and Bolts of software TPM in todays Mobile Devices." *arXiv preprint arXiv:1606.02995* (2016).

[68] Alblwi, S., and K. Shujaee. "A Survey on Wireless Security Protocol WPA2." *In the Int. Conf. on security and management,* pp. 12-17. 2017.

[69] M. Vanhoef and F. Piessens. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. *In CCS'17*, Oct. 30–Nov. 3, 2017, Dallas, TX, USA.

[70] Niyaz, Quamar, Weiqing Sun, and Ahmad Y. Javaid. "A deep learning based DDoS detection system in software-defined networking (SDN)." *arXiv preprint arXiv:1611.07400* (2016).

[71] Tang, Tuan A., Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. "Deep learning approach for network intrusion detection in software defined networking." In *Wireless Networks and Mobile Communications (WINCOM), 2016* International *Conference on*, pp. 258-263. IEEE, 2016.

[72] Da Silva, Anderson Santos, Juliano Araujo Wickboldt, Lisandro Zambenedetti Granville, and Alberto Schaeffer-Filho. "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn." In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pp. 27-35. IEEE, 2016.

[73] Panchen, Sonia, Peter Phaal, and Neil McKee. "InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks." *No. RFC 3176* (2001).

[74] Ye, Jin, Xiangyang Cheng, Jian Zhu, Luting Feng, and Ling Song. "A DDoS Attack Detection Method Based on SVM in Software Defined Network." *Security and Communication Networks 2018* (2018).

[75] Braga, Rodrigo, Edjard Mota, and Alexandre Passito. "Lightweight DDoS flooding attack detection using NOX/OpenFlow." *In 35th IEEE Local Computer Networks Conference (LCN)*, pp. 408-415. IEEE, 2010.

[76] Bonomi, Flavio, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things." In *Proc. of the first MCC workshop on Mobile cloud computing*, pp. 13-16. ACM, 2012.

[77] Luan, Tom H., Longxiang Gao, Zhi Li, Yang Xiang, and Limin Sun. "Fog computing: Focusing on mobile users at the edge." *arXiv preprint arXiv:1502.01815* (2015).

[78] Hu, Yun Chao, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. "Mobile Edge Computing—A Key Technology Towards 5G." *ETSI White Paper 11 (2015).*

[79] Beck, Michael Till, Martin Werner, Sebastian Feld, and S. Schimper. "Mobile edge computing: A taxonomy." In *Proc. of the Sixth International Conference on Advances in Future Internet*. 2014.

[80] Intel and Nokia Siemens Networks, "Increasing mobile operators' value proposition with edge computing. "

[81] Stojmenovic, Ivan, Sheng Wen, Xinyi Huang, and Hao Luan. "An overview of Fog computing and its security issues." *Concurrency and Computation: Practice and Experience* (2015).

[82] Stojmenovic, I., and S. Wen. "The Fog computing paradigm: Scenarios and security issues." *In proc. of the Federated Conf. on Computer Science and Information Systems (FedCSIS)*, 2014, pp. 1-8. IEEE, 2014.

[83] Yang, Mao, Yong Li, Depeng Jin, Li Su, Shaowu Ma, and Lieguang Zeng. "OpenRAN: a software-defined ran architecture via virtualization.*" In ACM SIGCOMM computer communication review,* vol. 43, no. 4, pp. 549-550. ACM, 2013.

[84] M. Vanhoef and F. Piessens. "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2." *In CCS'17*, Oct. 30–Nov. 3, 2017, Dallas, TX, USA.

[85] Alblwi, S., and K. Shujaee. "A Survey on Wireless Security Protocol WPA2." In the *Int. Conf. on security and management*, pp. 12-17. 2017.

[86] Smith, Randy, Cristian Estan, and Somesh Jha. "Backtracking algorithmic complexity attacks against a NIDS." *In the 22nd Annual Computer Security Applications Conference (ACSAC'06)*, Miami Beach, FL, USA, 2006.

[87] Xu, Yang, Junchen Jiang, Rihua Wei, Yang Song, and H. Jonathan Chao. "TFA: A Tunable Finite Automaton for Pattern Matching in Network Intrusion Detection Systems." *IEEE journal on selected areas in communications* 32, no. 10 (2014): 1810-1821.

[88] Zhao, Qi, Chuanhao Zhang, and Zheng Zhao. "A decoy chain deployment method based on SDN and NFV against penetration attack." *PloS one* 12, no. 12 (2017): e0189095.