THESIS: NFV AND VNF SERVICE CHAINING ON SAVI SDI

by

Seyed-Pouya Yasrebi

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science and Engineering Graduate Department of Electrical and Computer Engineering University of Toronto

© Copyright 2016 by Seyed-Pouya Yasrebi
Network Function Virtualization, a flexible networking architecture virtualizing network nodes, allows modular development of network functionalities and quick deployment on cloud infrastructures. Each network module individually deploys a virtualized network function (VNF). A network application might demand a set of inter-connected VNFs to work together. The logic and process of connecting VNFs together are called VNF service chaining.

In this thesis, two contributions were presented. First, a framework for VNF service chaining for software defined infrastructure was designed and tested using security attacks. The shortcomings of deploying security as a service with VNFs was examined, and a combined NFV/SDN solution was proposed. A parallelism technique was employed to make the NFV/SDN security solution scalable. Second, the real-time content delivery network of a stadium, equipped with virtualized customer premises Edges (vCPEs), accommodated a higher number of users employing our service chaining algorithm.
To my grandpa
Contents

1 Introduction 1

2 Background 3
2.1 Software Defined Networking ........................................ 3
  2.1.1 Networking Control Plane ......................................... 4
  2.1.2 OpenFlow Switch .................................................. 4
  2.1.3 OpenFlow Controllers .............................................. 5
2.2 Virtualization .......................................................... 6
  2.2.1 Virtual Machines .................................................. 7
  2.2.2 Containers ......................................................... 8
  2.2.3 OpenStack .......................................................... 9
2.3 Software Defined Infrastructure ..................................... 9
  2.3.1 SDI Architecture .................................................. 10
2.4 Smart Application on Virtualized Infrastructure ................... 10
  2.4.1 SAVI architecture ................................................ 11
  2.4.2 Multi-Tier Cloud ................................................ 13

3 VNFs, a means to implement heterogeneous security 15
3.1 Network Function Virtualization .................................... 16
  3.1.1 NFV and VNF ...................................................... 16
  3.1.2 NFV Architecture ................................................ 17
3.2 Service Chaining in SAVI .............................................. 19
  3.2.1 Integration with Orchestration .................................... 19
  3.2.2 Chaining mechanism on SAVI ..................................... 20
3.3 Network Security ........................................................ 22
  3.3.1 Network Attacks .................................................. 22
  3.3.2 SDN and Security .................................................. 22
  3.3.3 NFV and Security ................................................ 23
  3.3.4 SDN (Flow-based) Vs. NFV (DPI) Security ....................... 23
3.4 Security and SAVI SDI's architecture ............................... 27
  3.4.1 SAVI Testbed and security ....................................... 27
  3.4.2 Security Module Manager ........................................ 28
  3.4.3 Intrusion Detection System ...................................... 30
  3.4.4 Intrusion Prevention .............................................. 31
4 Scalability of Security Function Virtualization in SDI

4.1 Motivation

4.1.1 Previous work on scalability issue

4.1.2 Issues with basic IDS/IPS

4.2 IDS bottle-neck and scalability

4.2.1 case 1: One security vendor, no modularity

4.2.2 case 2: one security vendor, modular design

4.2.3 case 3: one security vendor, modular design, reduced port mirroring

4.2.4 case 4: multiple vendors

4.3 IPS bottle-neck and Scalability

4.3.1 case1:Same switches

4.3.2 case2:different rates

4.3.3 case3:different rates,different table size

4.3.4 case4:different rates,different table size,multiple paths

4.4 Analysis of IPS solutions

4.4.1 Scalability of network wide mitigation

4.4.2 Efficient use of MILP solver

5 Realtime content delivery and SAVI

5.1 Enhanced Real Time Content Delivery using vCPE and NFV Service Chaining

5.1.1 System description

5.1.2 Collocated chain model

5.1.3 Service chaining on constrained heterogeneous resources

5.1.4 Problem formulation

5.1.5 Heuristics

5.1.6 Related work

5.1.7 NP-Hardness Proof

5.2 Fast Network Flow Resumption for Live Virtual Machine Migration on SDN

6 Future work and Conclusion

6.1 Conclusion

6.2 Future work
List of Figures

2.1 Software defined networking APIs and layers ........................................ 4
2.2 Open flow and multiple controllers ...................................................... 5
2.3 OpenFlow table’s type of rules, actions, and statistics .......................... 6
2.4 Native Hypervisor sits between Hardware and multiple operating systems .... 7
2.5 Hosted Hypervisor sits between Original operating system and new operating system ... 7
2.6 Operating System level virtualization separates applications .................. 8
2.7 Architecture of Software Defined Infrastructure ...................................... 11
2.8 Architecture of Smart Applications on Virtual Infrastructure .................. 12
3.1 Comparison of resource types between Hardware and cloud solution ........... 17
3.2 Requests made to a web page during a day ........................................... 18
3.3 NFV architecture according to ETSI ................................................... 18
3.4 List of OpenFlow counters used for implementing security in SDN ............. 24
3.5 Implementing security using flow-based mechanism (SDN) ....................... 25
3.6 Implementing security using DPI (NFV) .............................................. 26
3.7 SDI Manager contains many modules. Each of these modules manage a functionality on SAVI Testbed. ................................................................. 28
3.8 Adding Security Module Manager to modules of SDI manager ................. 29
3.9 Security module manager should obtain information from Topology Manager and Network Control Manager. This cooperation of modules shall improve on the effectiveness of decisions made by the Optimal Decision maker. ........................................... 29
3.10 Detection using inline mode of IDS ..................................................... 30
3.11 Detect and attack using tapping mode of IDS ....................................... 30
3.12 Three possible points to mitigate an attack ......................................... 31
3.13 As number of attackers increase the ping time to the webserver increases. Also some packets will be lost in the network ................................................. 33
3.14 As the number of attackers to the web server increases, the more delay is observed for a regular user obtaining content ........................................ 34
3.15 CPU and RAM utilization did not undergo any stress after DOS attack. No significant information was extracted from these utilization to detect an attack. .................. 35
3.16 Attack gets blocked after getting detected by the NIDS. ......................... 35
3.17 Webservers traffic gets shifted to the Honey Pot to study attackers behavior ro provide attacker with false information. ................................. 36
4.1 OVS contains multiple levels of tables to minimize forwarding time.

4.2 Inserting same priority rules will take around 4 ms/rule on a vendor OpenFlow switch.

4.3 IDS scalability techniques.

4.4 All switches have the same rate for blocking attacks on a path.

4.5 Testing scalable IPS with four OVS switches.

4.6 Switches have different rate for blocking attacks on a path.

4.7 Switches have different rate, and different effective table sizes for blocking attacks on a path.

4.8 Switches have different rate, and different effective table sizes for blocking attacks in a network.

4.9 The time to block attackers seems to have an exponential form.

4.10 The time to block each attackers has a minimum in our use case around 320 attackers.

4.11 The processing time with this iterative approach becomes 2500 seconds for 1 million attackers.

4.12 This method estimates the resources left using a rapid random assignment technique.

4.13 Processes (deployed on VMs) will be connected in a circular fashion.

5.1 Stadium physical topology considered in basic cases.

5.2 Flow strands of stream and replay requests.

5.3 Simplified chains of stream and replay requests.

5.4 Lending chains to global branches.

5.5 Borrowing chains from global branches.

5.6 A test topology created with VINO for network flow resumption measurements.
Chapter 1

Introduction

The surge in content volume managed by data centers has resulted in an abundance of server communications. The shift of content nature from peer to peer model (in Email, and file transfer) to interrelated databases (widely used for social media content) has required infrastructure providers to manage their data efficiently. Furthermore, the emergence of applications with various content management requirements has demanded the infrastructure providers to provide various quality of service measures. For instance, the nature of content on applications such as Netflix (a movie streaming service), Pinterest (an online photo sharing platform) and massively multiplayer online games (MMOGs) are remarkably diverse.

In Netflix, accounting for 37 percent of web traffic in North America, movies have to be streamed from streamer servers to users. The content streamed from users vary based on location, time of year, and user profiles. For example, a recently added French movie is less likely to be played in Ontario compared to Quebec.

In Pinterest, a user based image sharing application, graphical contents have to be distributed among users. An image of Toronto’s Asian new year festival is more probable to be browsed in Toronto, rather than Vancouver.

In MMOGs, computational resources on data centers are connected. Users in a game consistently report their actions to other users. This data exchange creates a lot of data center traffic that needs to be managed by the infrastructure providers. The quality of service requirements to run such games are crucial. A delay experienced in a match can lower interest of players and ratings of the service. These delays could be the result of basic reasons such as lack of bandwidth to transfer data among cloud services or server overutilization causing a delay in one of the services.

The static nature of conventional networking prevents business applications from controlling the underlying networking infrastructure directly and stops the applications from reacting to the rapid changes in today’s internet traffic. For instance, high demand for playing Pokemon Go in Australia [24] caused service interruption, that could have been prevented, if application developers had control over the infrastructure. Providing these business applications with infrastructure access (networking and computing) can significantly improve their customers’ service quality.

A networking architecture that can provide networking access to mentioned business applications is called Software Defined Networking (SDN). SDN (SDN) [51] is an adaptable, programmable, and manageable approach to computer networking. Applying SDN to cloud infrastructure creates the notion
Chapter 1. Introduction

of Software Defined Infrastructure (SDI). SDI allows for the integrated control and administration of heterogeneous resources.

In an SDI environment, the business applications have access to interfaces enabling programmatic management of the resources. They can define their service graphs (SGs) determining the path that traffic must take through various network functions. The ability to dynamically realize the SG is what is referred to by service chaining. Use cases of service chaining include adding a firewall in front of web servers and providing real-time content delivery applications. Our goal was to deploy and evaluate these two use-cases (security and real-time content delivery applications) on an SDI.

A suitable platform that provides a complete control on SDI is Smart Application on Virtualized Infrastructure (SAVI) Testbed. SAVI testbed, having control on both computing and networking sides of SDI, has been deployed on nine universities across Canada and has been recently federated with Global Environment for Network Innovations (GENI).

In this thesis, the service chaining was realized on SAVI testbed to test Security features on SAVI SDI. The current implementation of service chaining on SAVI SDI can create chains in and between SAVI SDI regions.

Despite current network security solutions that employ Software Defined Networking on SDI, we introduced using Network Function Virtualization and used service chaining to implement security as a service in the form of Network Intrusion Detection and Prevention System. My contribution allowed deploying heterogeneous security system in cloud services rather than monitoring same qualities regarding all flows passing through network switches.

To test the security module as a Virtualized Network Function (VNF) on SAVI testbed, a service chaining feature was deployed. The NIDPS VNF used deep packet inspection to detect a denial of service attack. Using chained VNF was determined to have a set of draw backs for the throughput.

Soon after the realization and study of service chaining of VNFs, the benefit of combining NFV and SDN to create a better security solution was discovered. The benefits of NFV for detection and dynamic nature of SDN for mitigation of attack was used. Each DPI VNF was connected to a security module manager, deciding on placement of blocking rules and reporting them using SDN to the proper switches.

It was realized that our proposed solution in its raw form has some scalability issues. The scalability was only explored in terms of load scalability, and functional scalabilities. The goal was to minimize full attack time in a case with millions of attackers. The time for decision making by Security Module Manager waqs determined as one of the bottle necks. A parallelization scheme (pipe lining) was employed to address the scalability challenges.

The scalability solution that was proposed is not limited to the security applications and can be directly applied to other business applications with high-bandwidth, and delay-sensitive properties. It is a perfect example of how SDI architecture can significantly improve QOS for business applications.

The rest of the thesis depicts the benefits of SDI and service chaining on efficient utilization of physical resources in locally crowded scenarios. A stadium using a newly introduced devices in SAVI so-called virtualized Customer Premises Edge (vCPEs) was used as the use-case. Our goal was to enable streaming and playback features on users connecting to the wireless network of the stadium. It was demonstrated that the presence of vCPEs could significantly increase users with video service coverage. Further, an algorithm for service chaining of content delivery VNFs to add to the previously obtained number of users was employed.
Chapter 2

Background

To explain the content in the following chapters, a set of background material has to be described. I start by explaining two concepts that make SDI, an architecture for central management of networking and computing, possible. Software-defined networking (SDN) is employed to facilitate centralized networking control, and Virtualization is used to enable centralized computing management on SDI. Two examples of OpenFlow and OpenStack that are used on SAVI SDI to provide a better understanding of both SDN and Virtualization is used. SDN and Virtualization are put together to explain the architecture of SDI. Finally, SAVI, as multi-tier cloud implementation of SDI, will be reviewed.

Next chapters demonstrate:

• Role of SDN in creation of effective centralized security module manager in SDI

• Necessity of containers, as a form of virtualization, in using vCPEs

• Role of SDI in providing scalability to the proposed security solution

• The importance of SAVI in understanding bottlenecks of our theoretical proposals.

2.1 Software Defined Networking

To support dynamic applications, SDN proposes a central management of the network switches to efficiently route information through a network [43]. The logically centralized management of SDN provides a global view of the network. To implement this management, SDN physically separates the control plane from the forwarding plane. The control plane in the SDN handles administration and routing protocols (i.e. SNMP, CLI, OSPF, BGP, and ...) while the forwarding plane handles per-packet actions (i.e. Lookup, switching, and buffering). This separation allows the simultaneous evolution of switching and routing. This separate development requires an abstraction between the switching and forwarding layer. SDN’s programmability permits network administrators to control the networking layer efficiently. This abstraction eases network-wide traffic adjustment to cope with varying objectives.

The change in network traffic patterns, the popularization of cloud services, and the emerging big data technologies are primary attributes leading to network change. SDN designers aim to address both networking requirements and business objectives with SDN’s abstraction model (seen in Figure 2.1).
2.1.1 Networking Control Plane

Due to the complexity of network services, an abstraction is needed to separate the application layer from controlling the network layer. The application layer communicates with the routing layer (control layer) APIs namely northbound APIs of the control layer [44]. The primary goal of these northbound APIs is to model the networking layer as a single “big Switch” for the business layer. Northbound APIs have many varieties supporting different demands from the application layer. These APIs provide access to automation stacks such as Ansible as well as orchestration platforms such as Heat (to be discussed later).

On the opposite side, routing layer communicates with the switching layer (infrastructure layer) through APIs namely southbound APIs of the control layer. Southbound APIs enable dynamic modifications of switch flow entries to manage the network efficiently.

There are many SDN protocols for southbound APIs, most famous of which is OpenFlow [52]. OpenFlow was developed by Open Networking Foundation (ONF) [19]. With OpenFlow, flow-tables of switches in underlying layer can be modified based on real-time network traffic demands. There are many other southbound APIs such as OpFlex from Cisco [68], Lisp [36] promoted by ONF, and a few others are among these protocols. OpenFlow has been supported by many Physical switch Vendors such as HP [26], Cisco [4], Juniper [21] and ... a testament to its popularity.

```
Application
Layer
    Business
    App
    Business
    App
    Business
    App

Northbound API

Routing
Layer
    Network
    Service
    Network
    Service
    Network
    Service

Southbound API

Switching
Layer
    Switch
    Switch
    Switch
```

Figure 2.1: Software defined networking APIs and layers

2.1.2 OpenFlow Switch

OpenFlow can be used to manage routing tables of both physical and virtual switches. Physical OpenFlow-based switches such as HP [34], NEC, and Pronto have comparably larger number of flow table entries and faster flow setup rate compared to the virtual (software based) switches such as OpenFlow Virtual Switch (OVS)[22]. Lower flow setup rate does not discredit usefulness of the OVS, as the OVS can be effortlessly deployed on any VM (to be discussed later).

In traditional SDN, (as seen in Figure 2.2) a slicing software, such as Flow-Visor [67], aggregates the OpenFlow traffic from the underlying switches and distributes them among OpenFlow Controllers. A slicer is a software that slices the network to multiple domains and offloads routing decisions to multiple
controllers. OpenFlow Controllers are software programs that decide on the path that a packet should take to get to its destination. Conventionally Switches get connected to the Flow-visor acting as the load-balancer of southbound APIs traffic. The controller receiving the packet will make a decision upon packet’s arrival. Afterward, the controller will send its decision to the Flow-visor. Conflict free decisions verified by the Flow-visor will be applied to the flow tables of the switches.

![OpenFlow Diagram](image)

**Figure 2.2: Open flow and multiple controllers**

OpenFlow Table

OpenFlow Tables (as seen in Figure. 2.3) provide detailed measures on forwarding network packets. Each Table consists of many flow entries each of which including three parts: rule, action, and statistics. Each packet entering the OpenFlow switch gets compared to the flow table entries’ rules. Action for the longest match entry will be applied to the packet. In case that there are no rules that match the packet’s entries, the packet will be sent to the controller for decision making. The decision containing an action will be passed to the OpenFlow Switch, and a new rule will be added to the table. The Flow table actions vary between sending to a port, changing the content of the packet, to even dropping the packet. Finally, statistics field can be used to monitor some networking features for each flow entry such as byte counts and last time used. These characteristics will be used later to establish complex routing in a network.

2.1.3 OpenFlow Controllers

Packets that fail to find their corresponding flow table rule get forwarded to the OpenFlow controller. The OpenFlow Controller acts as an operating system for the network. Due to the implementation of OpenFlow controller in software, a higher level control can be exerted on the network traffic. There are many OpenFlow Controllers such as RYU [25], NOX [8], Beacon [16]. They vastly vary in their performance. Flows per second, Rules per flow-table, and multi-layer tables. RYU Controller is one of the promising SDN controllers. For Scalability purposes, sometimes multiple controllers get used in a network. As the separation of network domains and dedication of entirely orthogonal spaces to the controllers are difficult, a mechanism for conflict resolution is needed. Flow-visor is one of the slicers that claims to take care of the conflict resolution.
Chapter 2. Background

2.2 Virtualization

With growth in demand for computers in businesses, a new set of shareable machines were needed. Consequently, a set of mainframes were designed to separate user space information while allowing simultaneous access. The idea was called Compute virtualization. In precise, Compute virtualization [27] refers to creating a notion of physical compute machine. The primary goal of virtualization of compute devices is to reduce operating costs by sharing the computer among multiple users.

In the early stages, mainframes were being divided among users by time sharing. Time sharing was ultimately splitting resources of the mainframe. Division of critical resources such as RAM was resulting in significantly lower performance for each user. The notion of virtualization has evolved since then, and no longer mainframe resources get split. There are many types of virtualization, some of which are of interest to us in this work: Hardware level, Hosted level, Operating system level, Network level.

Hardware level virtualization refers to creating an abstraction between the operating system and the Hardware. This concept allows running multiple operating systems of the same or different kinds on the same physical hardware. Native or bare-metal Hypervisor is a software program that enables this abstraction. Native Hypervisor fully isolates different operating systems that each might run an untrusted application.

Hosted virtualization, similar to virtual-box [23], places a hosted hypervisor on top of the existing operating system of the computer. The hosted hypervisor sits between existing operating system and new operating systems.

Operating system virtualization refers to an abstraction between host operating system and applications running on a machine. This level of virtualization isolates applications from each other in the form of containers. Containers are similar to processes and share a base host OS while having different user spaces. This level of virtualization can be very useful for small sized computers as will be used in
Finally, network-level virtualization refers to allowing multiple Softwares/tenants to control single/multiple networks. External network-level virtualization allows system administrators to combine multiple Local Area Networks (LANs) into a single virtual LAN. Internal network-level virtualization, using software, emulates a physical network to containers. Network level virtualization are fundamentals of Software Defined Networking. Usually, network function virtualization does not require networking hardware virtualization. Encapsulation of packets on existing infrastructure creates network-level virtualization.

### 2.2.1 Virtual Machines

A Virtual Machine (VM) is an emulation of a physical computer that is sharing the underlying computing system. VMs are the direct result of Native and Hosted virtualization. The main benefit of the VMs is their isolation. If an application running on a VM causes its supporting operating system to crash, this event is completely hidden from other VMs. Isolation between VM conceals most of the resource usage characteristics between the VMs. A VM believes that it fully controls the resources assigned to it. Although this is merely an illusion, it is available via advanced resource sharing technologies [29].

Aside from the isolation property that VMs provide for their running applications, VMs can be easily maintained, conveniently recovered, and directly provisioned. These three features make using VMs easy
Chapter 2. Background

OS Virtualization

Operating System

App1 App2

Hardware

Figure 2.6: Operating System level virtualization separates applications during their lifecycle: creation, deployment, and destruction.

2.2.2 Containers

Containers are a lighter version of VMs. Unlike VMs, which run a full copy of the operating system on top of the virtualized layer, Containers only require a subset of an entire operating system with a few libraries to run their particular program. This feature makes them extremely light compared to the VMs. In a system using four VMs and a hypervisor, five operating systems are running. While in a similar system with Containers, only one operating system is running. Reduction in the number of the operating systems results in efficient use of computing resources allowing operation of more applications.

Containers, as discussed, are more efficient in packing applications into physical machines, but this does not make VMs obsolete. VMs still provide higher security measures to the applications running on them compared to Containers. The monetary savings with Containers is one of the main reasons for their popularity in today’s computation world. It is expected to have both VMs and Containers in cloud computing in the future despite the monetary savings of the Containers.

One set of issues with Containers are the privilege escalation attacks. An attacker with root permission in a Container has root access to the host. Containers in Linux access five standard namespaces: Network, Mount, Hostname, Process and Shared memory. There are many other Linux kernel subsystems located outside of these Containers under /sys. A Container with a super user access, violating the privacy and security measures, has full access to other Containers. This access can be avoided by restricting the Containers to access specific namespaces solely. It is also recommended to make common subsystems read only. However, this customization and the man-time effort undermines the main benefit of Containers. A few set of precautions can be helpful when dealing with Containers.

First, privileges of the Container shall be dropped by the Operating system as quickly as the Container completes an operation. Privilege dropping is to avoid keeping a channel of vulnerability open for a possibly malicious Container. This view of the Containers is consistent with the view to the program applications. Second, all Container services whenever possible shall run as non-root. If a Container panics the kernel, this will halt operation of other Containers as the host will be taken down. Hence to lower chance of panicking the kernel, Containers should avoid accessing the kernel. Third, It is a good practice to treat Containers, which have root access, similar to applications that have root access. Not handling the Container as a malicious app becomes an issue while Using developer-released Containers.
Using simply-found Container images on the web can quickly introduce malware to the underlying server. These are a few reasons that Containers have not fully replaced VMs.

Next, an advanced open source implementation of virtualization is discussed.

### 2.2.3 OpenStack

OpenStack is an open-source cloud computing platform. OpenStack as an Infrastructure as a Service (IaaS) provides its users with a set of controls to manage and monitor their hardware resource pools. The virtualization used in OpenStack is more complicated than basic virtualization. Cloud computing delivers the services (data and software) on demand to the users over the internet. OpenStack fully virtualizes the underlying hardware and provides command line tools, and a dashboard for users to manage their resources. Next, some of the essential components of the OpenStack used in this thesis are reviewed.

#### Compute-NOVA

Nova [12] is a Python-based cloud computing resource controller. Nova, using different hypervisor and virtualization technologies (such as KVM, VMware, Xen, Hyper-V, and LXC), automatically manages and configures computer resource pools. Nova commands are used to create and manage networks, keep backups, manage images.

#### Networking-Neutron


#### Glance

Glance [10] enables interacting with VM images. This interaction entails image creation, update, destruction, and retrieval. Glance can store the images in local or external storage. Glance takes advantage of an SQL-based database to manage the components in the system.

#### Ceilometer

Ceilometer [9] is the monitoring module on OpenStack. It provides a utilization data collection for both physical and virtual resources. The utilization is stored for analysis and management purposes. Alarms and actions can be specified to be activated as soon as the alarm defined criteria gets satisfied.

Next, a concept combining computing and networking virtualization is reviewed.

### 2.3 Software Defined Infrastructure

The main idea of the Software Defined Infrastructure [41] is to manage the functionalities of cloud computing, SDN, and other resource platforms as one single platform. This management enhances the platform to have the merits of cloud computing (i.e. virtualization) and SDN (i.e. greater network
programmability) together and to be thus able to make decisions that are based on the common state of computing, networking, and other resources.

Holding mutual view of the network resources (e.g. switches, controllers, NetFPGAs), and computing resources (e.g. VMs and physical resources without virtualization feature), SDI can enhance centralized security, mobility and content delivery management in many ways. This centralized view is especially useful for the case of heterogeneous clouds. SDI monitors the topological information, the state of the resources, and the overall bandwidth usage of all flows within a network.

Next, the architecture of SDI and its components are explained.

2.3.1 SDI Architecture

Software-Defined Infrastructure (SDI) manages computation and network resources in a converged manner. In SDI, separate controllers handle each of the resources. To create a converged control, a centralized controller namely SDI manager has to obtain information from different controllers to make a decision. The decision made by the SDI manager is more accurate than a decision made by other controllers individually.

A Topology Manager is required to report interaction of virtual and physical resources to the SDI manager. The Topology Manager is initialized using static topologies provided by administrators. The dynamic nature of the SDI shall be carefully recorded using Monitoring and Measurement (M&M) module. Both Topology Manager and M&M Manager provide the SDI Manager with the necessary information to manage computation, and route network traffic. The Topology Manager also checks with M&M manager to detect anomalies in the system[47].

A set of open interfaces has to be designed to provide the available information to the external entities such as service providers, infrastructure providers, and even applications demanding visibility to the data. These open interfaces create a semi-standard interface for users with different APIs to interact with the SDI. The system that sits between external entities and resources is called resource management system (RMS) (Figure. 2.7). SDI RMS shall control access, management, and configuration of the Infrastructure.

Next, a specific implementation of SDI that has realized such extensive features is discussed.

2.4 Smart Application on Virtualized Infrastructure

Smart Application on Virtual Infrastructure (SAVI) is a project that is an outcome of industry and academia partnership in Canada. The core idea of SAVI is to combine Cloud computing and Software Defined Networking to provide a single abstracted module, namely SDI Resource Management System (RMS).

SAVI studies applications that can improve their quality by accessing the underlying networking infrastructure. Some of these applications are real-time content delivery to geographically distributed nodes, database management of data for concentrated crowds, and management of green application located in different time zones.

The autonomic management system created by this applications ensures efficient use of the platform. SAVI also explores resource management to satisfy high-level objectives such as reliability of applications, cost reduction of the resources, and enhanced use of green energy. Resource virtualization has enabled employment of distributed applications on virtual networks.
By predicting the future workload of each of these virtual networks, it is possible to modify the allocated resources dynamically for each application. The performance received from each application will be used as a feedback to train this automatic resource management further.

Finally, SAVI enables researchers to deploy and test innovative network applications. SAVI provides virtual resources to the researchers to run their experiments and creates a set of APIs to be used by researchers. Orchestration, overlay networks, virtualized FPGAs, big data analysis, virtualized security, service chaining, and real-time content delivery are among some of the works done by this large group.

Next is a discuss on the architecture of SAVI Testbed.

2.4.1 SAVI architecture

SAVI RMS [42] can be thought of as Infrastructure as a Service (IaaS) as well as Network as a Service (NaaS) since it provides all the services provided in each of these. SAVI Testbed is the platform realizing SAVI research. SAVI Testbed is designed based on the concept of SDI, and it utilizes open-source cloud computing controller OpenStack [64] and OpenFlow [52] controllers together with a novel SDI Manager code-named Janus and a Topology Manager code-named Whale [57]. Eight Canadian Universities have deployed SAVI Testbed.

SAVI Testbed encompasses both physical and virtualized OpenFlow switches. Implementing switches using OVS [20] on VMs adds yet another dimension for scalability within SAVI. It can be certainly useful for security management, as discussed in more detail in the next chapters 2.8.

Resources

SAVI-RMS controls heterogeneous resources in a converged sense. In the beginning, only compute resources (CPU, RAM, Storage) and Networking resources were available on SAVI SDI. Both networking and computational resources are of the same importance to the SAVI-RMS in a way that the SDI manager...
uses information from all of these resources to conclude a decision. Currently, Field Programmable Gate Arrays (FPGAs) have been added to the SAVI-RMS’ pool of resources [32].

**Ryu controller**

RYU OpenFlow controllers command the SDN side of the SDI [48]. Each controller receives events from underlying OpenFlow switches and makes a decision correspondingly. The RYU controller has both southbound and northbound APIs. The southbound is used to communicate with hardware/software OpenFlow switches, and northbound is for sending events and receiving commands from the SDI manager. In SAVI-RMS many RYU controllers can run simultaneously.

Each RYU can only make the decision locally. Hence events that can not be handled by one RYU controller will be pushed to the SDI manager. The frequency of unhandled events by the RYU controller is much less than local events. Local events, which are more frequent, have been delegated to the local RYU controllers rather than the SDI manager. This delegation makes the networking control environment highly scalable.

**SDI manager-Janus**

SDI Manager, code-named Janus [57] consists of many modules and a module manager. Scheduling module, networking control module, and fault tolerant module are a few to be named here. The SDI module manager will overlook all suggested modules. Janus that encompasses modules and module manager is accountable for management and control tasks. The networking control module will be employed to implement security.
Topology Manager-Whale

Topology Manager, code-named Whale stores network node and link status in the form of a graph. This network topology contains connectivity of nodes, existing flows between nodes, and other general measurements. Whale is directly connected to Openow controller to obtain and update physical and virtual network properties. It also acquires physical and virtual node properties from the cloud controller (OpenStack based on SAVI). Whale transmits required information directly to Janus for control purposes.

Next, the SAVI testbed’s Multi-Tier nature allowing implementation of delay-sensitive applications is discussed.

2.4.2 Multi-Tier Cloud

Here, multi-tier nature of the SAVI SDI cloud including Core, Smart Edge, vCPEs, sensors, and actuators is discussed.

Core

The core provides massive computational resources connected via high-bandwidth links. These massive pools of resources are well-suited for large scale time insensitive applications as Core resources are usually located far from the applications.

Smart Edge

Edge nodes are placed closer to the applications with fewer computational resources about Core. Their location makes Edge nodes crucial in time sensitive applications. Factors such as shortage of resources, lower access latency to processing resources, and ability to cache content to reduce required bandwidth for applications result in higher deployment cost of Smart Edge nodes compared to Core nodes. Despite higher resource deployment costs, the number of Edges is greater than Core nodes to improve geographical accessibility.

vCPE

In comparison to Edge and Core nodes, vCPEs are devices with the smallest computational and networking resources. vCPEs can range from small computing units built into modern switches to micro-computers deployed on WiFi-access points to provide computational power in the proximity of users. vCPEs allow applications to save on bandwidth and to reduce latency. The scarcity of vCPEs and their higher deployment cost compels an application developer to incorporate these resources wisely in their systems.

vCPEs do not have enough resources to handle computation for large volumes of web traffics, so a cloud-based solution with Core and Edge nodes is required to satisfy end-to-end requirements for an application. By attaching vCPEs to a network [70], they get automatically connected to the SDI and can be easily used to deploy computing or networking functions. For instance, this feature becomes handy in stadiums. vCPEs attached to Access points can be used to provide real-time content to viewers outside of stadiums. The benefits of vCPEs in RTCD will be discussed in the incoming chapters.
Sensors, actuators, and Internet of Things

Internet of things (IOT) refers to connecting things to the web. IOT requires measuring features of the objects/qualities (OQs), making a decision regarding the obtained information, and enforcing actions to modify the current state of the system. Another layer of equipment is needed to enable access to the OQs. Sensors provide an acumen into the status of the OQs. Sensors in the current world can take measurements from the air temperature to the pulse shape of a human’s heart. The range of sensors is a testament to the complexity of the sensor network.

In a recent work [71], sensors were used to monitor the air quality of classrooms in a university setting. The sensors transmitted data to the vCPEs that forwarded the raw info to a Smart Edge node for processing. In the loss of connectivity, the vCPE stored the data and passed the data to the Smart Edge as soon as the connection was re-established. This feature would not have been possible without the vCPE in place.

Actuators, enablers of the sensor networks, apply decisions made by the control system to modify or maintain the current state of the OQs. It might seem trivial to manage a system using monitor-decision-act model, but control systems with this model usually have a feedback loop making it difficult to administer. The feedback loop has to be ensured to have a negative feedback or else the control system will not converge. This automaticity becomes extremely complicated when there are multiple correlated control variables to affect OQs.

In some delay-sensitive applications, the time between taking measurements to applying the action is minimal. For instance, in an active vibration cancellation system mounted on an assembly lines machines, the processing can not happen on the core sitting far (network wise) from the column. The decisions have to be made instantaneously and applied by actuators to cancel the vibration. Current implementations try solving this by creating a single box that connects sensors to vibrating actuators with a microcontroller in the middle. This distributed solution is ineffective when two devices are attached to each other since the action of one affects the performance of the other. Therefore a global view and a central management with vCPE are needed to obtain better results.
Chapter 3

VNFs, a means to implement heterogeneous security

Virtualization enables sharing of a physical machine’s resources among its users. The virtualization is required to provide remote access to users accessing campus networks. The extent of this resource sharing becomes apparent on data centers where servers are being pushed to be utilized to their maximum limit. In the previous chapter we discussed how:

- OpenFlow, an SDN protocol, virtualizes network by separating control plane and data plane.
- OpenStack virtualizes cloud resources and provides services using its modules.
- SAVI testbed combines OpenStack and OpenFlow to manage an SDI environment efficiently.
- SAVI's heterogeneous resources communicate with each other to provide SDI features.
- SAVI's multi-tier nature allows SDI users to take computation closer to the sensors in connection-sensitive applications.

The discussed applications consist of separate functions that could be distributed among these multi-layered SDI. Deploying these functions as software permits dynamic, fast, and reconfigurable designs. In this chapter the following questions are answered:

1. What are network function virtualization and virtualized network functions? What is the benefit of using VNFs compared to hardware solutions in networking?

2. How can SAVI SDI use VNFs to implement security?

3. What is the benefit of using NFV compared to SDN solutions for deploying security as a service?

4. What are the scalability issues of VNF security system on the SDI? What are the possible solutions? How do they compare to each other?

5. How is Service chaining implemented on of VNFs implemented on SAVI testbed.

To answer the above questions, we present an approach providing security services in software that runs on Virtual Machines(VMs). The method takes advantage of Software Defined Infrastructure (SDI) to enhance the security services provided by the network.
Chapter 3. VNFs, a means to implement heterogeneous security

3.1 Network Function Virtualization

3.1.1 NFV and VNF

Traditional service provisioning in the telecommunication industry was carried out by Network operators. Each service would have been provisioned by connecting multiple of its functions deployed on priority boxes. Strict chaining requirements, stability standards and the increasing number of short-lived services enforced the telecommunication service providers to upgrade their physical equipment constantly.

Network Function Virtualization (NFV) was introduced by European Telecommunications Standards Institute (ETSI) in 2012 [18]. NFV [37][38] is a network architecture model that aims for virtualizing network services. NFV proposes that all functions performed by network nodes should be defined in a virtualizable fashion such that all network operations could be classified into separate building blocks that can be chained together. Each such building block would represent a Virtualized Network Function (VNF) [15].

VNFs provide specific network functionalities, such as encryption/decryption, Virtual Private Network (VPN), Load Balancing, Firewall, Wide Area Network (WAN) Accelerator, and Intrusion Detection System (IDS) based on Deep Packet Inspection (DPI)[14]. The industry specification group in charge of NFV has argued that NFV’s provide quicker network deployments resulting in faster innovations. Other objectives of NFV include reduced Operational Expenses (OPEX) by greater use of automation, reduced power usage by turning off underutilized resources, and improved adaptation to new market models. The latter of the three would be due to NFV providing the opportunity to divide network implementation between various small and big players.

In comparison to the NFV, hardware boxes require part maintenance, underlying software update, and manual installations. One apparent benefit of VNFs is in their time to deployment. The time to boot a virtual machine and run a few lines of code to setup a VNF is not comparable with ordering a similar box from Vendors. This difference becomes significant when dealing with a connected set of VNFs. Connecting multiple boxes using wires and switches is comparably more time consuming than running code to connect VNFs in the desired fashion (as seen in Figure. 3.1).

VNFs running in a cloud environment can be a suitable candidate for highly dynamic applications. Consider an example where hardware security boxes have to be purchased to accommodate the maximum throughput of web traffic for a website. Figure 3.2 shows the varying nature of web traffic for a Web page. The web page has two traffic peaks in the shown day. One around the noon and another around the evening time. Considering an almost similar behavior of users on other days, the host of the web page must purchase a hardware security device to monitor the incoming packets with a throughput of 14000 users per hour. This security box is only 34 percent utilized on average. This utilization should be one of the main decision parameters when dealing with purchasing hardware rather than a cloud-based security system.

Contrary to the above argument, it is also claimed that owning the hardware boxes in a long run occurs a lower cost compared to renting cost of cloud services for some small sized companies. Consider an example where an application wants to process air pollution information. The air quality measurements constantly get transferred to the application irrespective of daytime. The static nature of this application makes it a suitable candidate for purchasing hardware and connecting them in a fixed topology. In this case, the company needs a firewall and a computer to process the data. Renting a VM for firewall and computational purposes (from Amazon) could cost near a thousand dollars annually [6] while purchasing
a firewall enabled router and a computer may cost nearly the same as a one-time cost.

### 3.1.2 NFV Architecture

There is more to running VNFs than booting up VMs. The overall architecture of NFV’s reference architecture (as shown in Fig. 3.3) to depict a glimpse of the complexity of running VNFs. Operating support systems manage many services composed of different components. OSS needs to communicate their needs through a management system to the infrastructure provider. This management service obtains requests from the OSS and translates them into infrastructure requirements. This translation happens using APIs of the orchestration service. The orchestration service defines the relationship between the computing resources and the networking topologies (to be discussed in more detail in next section). The translation generated by the orchestrator will be provided to the Virtual Infrastructure Manager to process and deploy the required VMs. The VNF managers oversee the operation of VNFs by controlling the lifecycle of each VNF such as instantiation, updating, sleeping, and even termination of the VNFs. It is worth noting that a VNF manager could be associated to one or more VNFs.
Chapter 3. VNFs, a means to implement heterogeneous security

Figure 3.2: Requests made to a web page during a day

Figure 3.3: NFV architecture according to ETSI
3.2 Service Chaining in SAVI

In this section, we provide a concrete definition of service chaining. We then review the features provided by the SDI RMS in the facilitation of service chaining. Finally, we describe design and evaluation of service chaining on SAVI Testbed.

An application deployment consists of services that the end-user interacts with, e.g. a web server, and other NFs, i.e. transparent components like a load balancer. NFV refers to the virtualization of arbitrary NFs, such as deep packet inspection (DPI) firewalls, load balancers. Individual NFs can be composed into an SG that specifies a list of services and the order of traversal. For instance, consider a web application (app) deployment. This app may consist of a firewall that filters the traffic and a load balancer that distributes the load across horizontally scaled web servers. The SG is an abstract object that corresponds to a set of SLAs (connectivity of NFs, throughput between each connection, maximum delay in a total service, availability of service). The realization of an SG is service chaining. Service chaining consists of two parts: creating the VNFs specified in the SG, and chaining them together. These can be done through the orchestration engine and the SDI manager, respectively. Specifically, the SDI manager has state information of the infrastructure and can direct the resource controllers to execute certain operations. These combined, allow the SDI Manager to perform functions such as fault tolerance and dynamic installation of network flows.

To facilitate service chaining, the SDI manager exposes many primitive functions, such as tapping and blocking. Tapping refers to sending a copy of the incoming traffic to a host that was not the intended destination. Blocking function handles dropping packets according to system requirements in switches. As an example of chaining, consider a WordPress deployment consisting of two VMs- one operating a web server and the other running a database. It is desirable to allow the application to change their SG dynamically. For instance, the application may want to insert an inline deep packet inspection (DPI) VNF in front of the web server. The applications requirement can be satisfied using service chaining. Now consider how service chaining in the web server example could be realized. First, the application would request the SDI manager to perform service chaining by inserting a DPI VNF in front of the web server. The SDI manager would direct the network controller to install specific flows in the switches of the underlying infrastructure to ensure all traffic headed for the web server goes to the DPI VNF. In this case, we would have to configure the DPI unit to forward the traffic to the web server. Since resource controllers can be directed to execute commands at any time, service chaining can be performed on a live system without service disruption.

Multicasting is another example of an application that can leverage VNF chaining. Consider the sequences of events when deploying a multicasting application. First, the orchestration engine creates casting modules such as virtualized transceivers and load balancers. Migrating the chained casting modules could reduce total bandwidth usage by reconstructing a more efficient multicast tree. In [76], Zhang et. al demonstrated cost reductions in deployments of multicast trees with a newly proposed routing algorithm that used dynamic chaining of VNFs.

3.2.1 Integration with Orchestration

Orchestration is the first step for service chaining. The challenge in service chaining consists of integrated management of multiple resource types such as computing and networking resources. First, we have to create the VNFs (these are typically VMs configured to perform the specific function). Second, we
need to connect the nodes to allow the required communication. As described above, the SDI manager applies network policies to ensure that the traffic traverses the required VNF(s). One of the orchestration engines running on SAVI is Heat.

**Heat**

Heat from OpenStack [45] facilitates the creation, modification, and deletion of cloud infrastructure resources over the applications life cycle. Applications specify what resources they need and how they should be configured in descriptive template files. The orchestration engine then parses these templates to provision the required resources. The applications can also modify and delete resources by providing new or modified templates to the orchestration engine. Therefore, the orchestration service allows management of complex topologies without increasing the cost of managing that complexity. Furthermore, template files for Heat are compatible with Amazon Web Services (AWS) CloudFormation [3] (another orchestration service running on Amazon). To extend orchestration service of SAVI to the overlay networks, running on multiple regions, VINO was developed [31].

**VINO**

Virtualized Infrastructure using Network Overlay (VINO) was an attempt to create network overlays between user specified VMs. Despite Heat, VINO automatically connects these VMs using VXLANs and OVS switches. VINO takes a configuration file and a template file as inputs from the user. The configuration file asks for the user’s access codes, user’s default preferences for region and tenant for booting the VMs. The template file has both information that NOVA requires to create the VMs and their connectivity in the overlay. The template would set the missing specifications according to the default values set in the configuration file. After obtaining images from Glance and booting the VMs, VINO enters each VM using ssh connection to create VXlans and verify their connectivity. VINO has recently added modification feature that permits the topology to be altered after creation.

Next, service chaining mechanism and the role of Heat and VINO in facilitating the mechanism are discussed.

### 3.2.2 Chaining mechanism on SAVI

The chaining mechanism of SAVI SDI receives three sets of IP addresses from the user. Two of these addresses are the endpoint IPs, and the third is the middle VNF’s (in the form of VM) IP address. These IP addresses could be in the class of internal IP addresses belonging to a SAVI region, or in the form of overlay IP addresses.

Initially, the chaining service verifies whether two paths can be created between each endpoint and the middle VNF. Feasibility of these two paths is confirmed using Janus incorporating topology information of the full network.

Next, the two paths between these two endpoints and the middle VNF gets setup using Janus APIs. To create these two paths, the switches on each path shall be modified. In the case of the internal network, the switches in the paths are OpenFlow Switches, while for the case of the overlay, the switches are OpenFlow Virtual Switches.

In the case of the overlay, an OVS is running on each of the three VMs. Two new VXlans connect the OVS running on the endpoints to the mid-VNF’s OVS. The mid-VNF gets placed between two
internal ports of the OVS. The mid-VNF, receiving the packets, takes some actions, modifies the packet (destination mac address) and sends it out of the virtual port of the OVS.

These paths could also get easily deleted (to delete chains) as Janus uses higher priority rules on the OpenFlow to set up chains. Therefore to remove the chain only rules with the higher priority gets withdrawn from the discussed OpenFlow switches.

The implemented service chaining application on the internal network was tested by creating two virtual machines on the Toronto Edge: a VM as an endpoint (web server) and another VM as middle-VNF using Heat orchestration service. The goal was to redirect all incoming traffic (from the internet heading to the end-point) to be passed through the middle-VNF. The gateway to the Toronto region was chosen as the first IP, the mid-VNF as the middle-IP, and the VM(web server) as the final IP for the chaining mechanism to create this redirection. In the same way, the service chaining was tested for the overlay networks. Using VINO, the mid-VNF was added by modifying VINO's template. Then service chaining was applied to place the mid-VNF between the two endpoints.

Using Snort in inline mode (as will be discussed in detail in the next section) DOS and SQL-injection attacks were mitigated.

In the next sections, running a security application on the SDI is discussed. Use of static service chaining and then use dynamic nature of SAVI SDI to create dynamic chains (by enabling decision making) are explained.

Before diving into the use of service chaining in security applications, we look at network security and current SDN security deployment solutions.
3.3 Network Security

Conventional network security systems consist of proprietary hardware boxes, usually including Application Specific Integrated Circuits (ASIC) to perform Deep Packet Inspection (DPI). These devices can detect malicious traffic, and prevent it from passing by blocking it. However, as hardware implementations, they provide limited flexibility to be customized based on specific customer needs. At the same time, these systems are costly and not easily deployed in a cloud. Their installation involves all the tasks associated with the physical installation of a hardware system. In most cases, the operation of such security devices may not be downsized when traffic drops and consequently the equipment goes underutilized. Also, the capacity upgrades require an additional installation causing operation disruptions. In contrast to hardware systems, a software-based approach based on sharable hardware can provide CapEx savings as well as scalability, flexibility, and customizability.

3.3.1 Network Attacks

Due to the rapid increase of internet speed and computational power in regular users, attackers find Denial of Service (DOS) attacks desirable. The attackers may use Personal Computers (PCs) to initiate an attack to bring down target servers. These servers will not be able to handle their incoming traffic. Therefore they will fail to provide service to ordinary users.

One of the possible attacks that a server can go down is a Distributed Denial Of Service (DDOS) Attack. In a DDOS attack, a bot-herder (malware producer) slaves a set of location wise distributed computers to send fake requests to target servers [55]. Since these requests are not directly coming from the bot-herder, it is usually challenging to detect the bot-herder.

On-premise physical security devices are capable of determining that an attack has happened. They collect the compromised IP addresses and use one of the following methods: block attackers locally to hinder them from accessing the server (sink-holed), route them to a non-existing server (blackholed), block malicious packets on the front end in the case that a user consumes the whole bandwidth (Bandwidth management) [54].

3.3.2 SDN and Security

Software Defined Networking (SDN) can be a powerful paradigm to implement centralized security management [63]. SDN aims towards a software realization of the network, which is well fit for a software realization of network security. As discussed earlier, in OpenFlow, the most developed example of SDN, new packets from sources, not known to the switches, generate an event inside the switch causing it to ask the controller for an action on the packet from the unknown source. This feature is useful when dealing with potentially malicious traffic coming from outside of the network. Also, OpenFlow allows for customized implementation of specific matches on flows, providing matching options such as protocol, IP and MAC destination or source, UDP or TCP port number. This aggregation of layers permits for a more extensive security solution, as discussed later.

SDN Security projects

Many cloud computing management systems, such as Open-Stack, do not have a dedicated security component/project. Similarly, most security projects tend to be targeted for SDN than NFV. Nonetheless,
as discussed, such attempts could be implemented in NFV, too. Examples for SDN security projects [1] include FortNOX [60], Security Enhanced (SE) Floodlight [7], Cloud Police [59], Flowvisor [67], and FRESCO [58].

In FortNOX, Openflow Controller is extended to include a security mediation service, which is to ensure policies/flows in a given network slice comply with a broader security policy defined at a higher level. Security Enhanced (SE) Floodlight follows a similar approach by having role-based authentication for flows submitted, flow conflict resolution (new vs. existing flows), and hierarchical vetting. Started in UC Berkeley, Cloud Police is a project that uses OpenFlow to customize security policies for virtual operating systems within a given host. This approach is essentially focused on setting the hypervisor in charge of security than the network infrastructure. Then, Flowvisor introduces another enhanced OpenFlow Controller, functioning as a proxy between OpenFlow switches and various OpenFlow controllers. This proxy is to create network resource slices, assigning the control of each slice to a separate controller, enforcing isolation between the slices. As we shall see later, this contrasts the somewhat distributed approach used in this chapter. Finally, introduced by Stanford Research Institute (SRI), FRESCO is an architecture for modular security services for SDN-based networks. FRESCO aims to provide a development and testing environment for security related applications and security policy managers for dynamical deployment of security policies. FRESCO assumes FortNOX is used for the controller and provides orchestration for security.

### 3.3.3 NFV and Security

Several security functionalities have been suggested to be implemented as VNFs. Such VNFs may be put together as a package, providing Security as a Service (SECaaS). Companies could outsource their security packages to the service providers or other third party companies via subscription [1]. This outsourcing could help companies to reduce the ownership costs by obtaining security as a service from the cloud (i.e. no need for the customer hardware). Services provided by such SECaaS may include Authentication, Anti-Malware, Intrusion Detection and Prevention, and security event management. In most cases, a combination of NFV and SDN are used while NFV could be implemented without SDN. NFV and SDN are complimentary since one is a model for dividing network operations into functional blocks while the other enables certain network services to be realized in software. Combined use of NFV and SDN is the approach employed in this chapter.

Next, we compare security solutions between NFV and SDN. We will explain why NFV is a better solution for achieving a heterogeneous security implementation.

### 3.3.4 SDN (Flow-based) Vs. NFV (DPI) Security

There are two levels for analyzing the security of the network: Deep packet inspection, and Flow-based Analysis. Deep packet inspection refers to looking at each and every packet header and embedded contents inside the packets. DPI can infer some information from the header of the packet and do further analysis to understand the contents. Deep analysis of the packets is a huge burden on the processing resources. It is evident that to this day Full-DPI IDS can not be done on all internet passing traffics by network or service providers. Switching these packets is itself a complicated task much less full analysis of packets. Aside from that if a service provider decides to analyze each packet before forwarding it to the destination it would introduce massive delays on the switching depending on the incoming web
traffic of the web-server. The DPI technology has been used only for basic analysis of headers of packets to block some IP addresses.

Another level for analysis of network traffic is analyzing the flows obtained from switches. Each flow in the open flow table can provide a set of basic measurements regarding that flow. These basic measurements that are referred to as counters are divided into Per flow-table, per flow-entry, per port, per queue, per group, and per meter. Each is an attribute that can be added to the OpenFlow table on the switches (as seen in Figure 3.4).

<table>
<thead>
<tr>
<th>Counter</th>
<th>Bits</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Per Flow Table</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference count (active entries)</td>
<td>32</td>
<td>Required</td>
</tr>
<tr>
<td>Packet Lookups</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Packet Matches</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Per Flow Entry</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Received Packets</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Received Bytes</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Duration (seconds)</td>
<td>32</td>
<td>Required</td>
</tr>
<tr>
<td>Duration (nanoseconds)</td>
<td>32</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Per Port</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Received Packets</td>
<td>64</td>
<td>Required</td>
</tr>
<tr>
<td>Transmitted Packets</td>
<td>64</td>
<td>Required</td>
</tr>
<tr>
<td>Received Bytes</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Transmitted Bytes</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Receive Drops</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Transmit Drops</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Receive Errors</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Transmit Errors</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Receive Frame Alignment Errors</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Receive Overrun Errors</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Receive CRC Errors</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Collisions</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Duration (seconds)</td>
<td>32</td>
<td>Required</td>
</tr>
<tr>
<td>Duration (nanoseconds)</td>
<td>32</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Per Queue</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmit Packets</td>
<td>64</td>
<td>Required</td>
</tr>
<tr>
<td>Transmit Bytes</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Transmit Overrun Errors</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Duration (seconds)</td>
<td>32</td>
<td>Required</td>
</tr>
<tr>
<td>Duration (nanoseconds)</td>
<td>32</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Per Group</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference Count (flow entries)</td>
<td>32</td>
<td>Optional</td>
</tr>
<tr>
<td>Packet Count</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Byte Count</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Duration (seconds)</td>
<td>32</td>
<td>Required</td>
</tr>
<tr>
<td>Duration (nanoseconds)</td>
<td>32</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Per Group</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bucket Packet Count</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Byte Count</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Per Meter</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow Count</td>
<td>32</td>
<td>Optional</td>
</tr>
<tr>
<td>Input Packet Count</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Input Byte Count</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>Duration (seconds)</td>
<td>32</td>
<td>Required</td>
</tr>
<tr>
<td>Duration (nanoseconds)</td>
<td>32</td>
<td>Optional</td>
</tr>
<tr>
<td><strong>Per Meter Band</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In Band Packet Count</td>
<td>64</td>
<td>Optional</td>
</tr>
<tr>
<td>In Band Byte Count</td>
<td>64</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Figure 3.4: List of OpenFlow counters used for implementing security in SDN

Although each counter provides a measurement regarding the state of the switch and the passing
Chapter 3. VNFs, a means to implement heterogeneous security

packets, it is possible to extract more useful information by combining some of these counters. For instance, by using Received packets and Duration (seconds) in Per-Flow Entry, it is possible to find the average rate of the flow and apply or measure QOS parameters. Another application could be using Packet Lookups and PacketMatches to identify whether the rate of new users accessing the server is in expected range or not.

Heterogeneous Security

Following discussion between DPI and Flow-based, the question becomes their applicability to different scenarios. We believe that Flow-based is useful where a similar set of rules and criteria are to be checked for all flows of a switch. By adding a column to the OpenFlow Table, we can observe that the measurement is occurring for all flows in the table (as seen in Figure 3.5). Analysis of all entries can drastically put a load on the underlying processing, affecting the performance of OpenFlow switch in other criteria. Also by increasing the flow size table (adding counters) the speed of querying might get a hit on the OpenFlow Switch.

![Figure 3.5: Implementing security using flow-based mechanism (SDN)](image)

Due to performance hit that a software switch gets by applying homogenous security measures to a network, a heterogeneous approach for implementing security at the high level is suggested. For instance, three web-servers one with high-security standards and minimal downtime requirements and the others with very loose security measures should not receive the same network packet processing from the network provider (as seen in Figure 3.6). It is apparent that the sensitive web server requires more security check than the other one.

Next, we discuss our implementation of security on SAVI Testbed, the advantages of SAVI in implementing security, and evaluate the performance of our centralized security module manager.
Figure 3.6: Implementing security using DPI (NFV)
3.4 Security and SAVI SDI’s architecture

We propose SAVI SDI to solve the issues of current security implementations. Our solution is to be scalable, dynamic, and secure for traffic management. Since SAVI SDI is connected to Whale, it is aware of every network route that exists on any of the OpenFlow switches. SDI’s awareness of the full system allows it to re-route or manage security hazards.

There are some features in SAVI SDI that make it a unique platform regarding security implementations: dynamical placement of VMs, allocation of network resources via cloud computing, smart functionality based module associator, and intelligent traffic and routing controller.

Since SDI Manager has control over both network resources, cloud computing resources, as well as Whale, it is capable of being programmed to place security modules optimally according to network topology. This placement can be enforced by Service Providers or network admins according to their quality of service.

SDI allows dynamic allocation of network and computational resources for security modules. Such modules can change their processing power (due to change of incoming traffic) or even their functionalities (expansion of security policy in security modules).

SAVI/SDI is capable of implementing layers of security modules for different network attacks. Therefore it is capable of blocking attacks based on their severity in a different part of the network architecture. This feature takes advantage of SDI Manager accessing Whale and Network Manager. Whale may identify the critical points for layer implementations, and SDI will take the order and put software-defined functions in place.

SAVI/SDI has access to SDN controllers and shall act upon a malicious traffic either by blocking it or forwarding to a different server for study purposes. These fake servers are called Honey Pots that are used to analyze security attacks. SDI uses Whale’s knowledge of network to determine a proper network point for applying preventing measure [61].

3.4.1 SAVI Testbed and security

The full view of the SDI to the current state of the system is useful in determining anomalies that might arise from malicious attacks. Monitoring bandwidth and computational resources and their correlation help to draw both the big and the detailed picture required for a network security management module.

SAVI RMS (as seen in Figure 3.7) permits to customize the location of VMs and routing of flows. In particular, the security VMs described above may need to be carefully located, be migrated, or downsized. SAVI RMS provides open interfaces to deal with all these, with no need for the programmer to know the lower level implementation.

Unlike the hardware boxes described above, SAVI RMS can launch security VMs for users on the fly. This rapid ability to boot VMs can create a dynamic security mechanism and save a significant portion of the computation for rest of the apps that are more bandwidth hungry.

Using OpenFlow, SAVI RMS is capable of installing flows for various layers. Among the available OpenFlow controllers, SAVI employs RYU [25] to perform flow-based network functions, including VPN, Firewall, and as we discuss here, security as a VNF. Another significant benefit in using SAVI SDI is to achieve environmental sustainability and reduced carbon footprint, by providing the opportunity to manage the computing energy in a much smarter fashion.
SAVI-SDI’s basic security feature

It is usual for an attacker to forge an IP address of a user, meaning an attacker sends packets with a different IP than hers to avoid making her IP evident. In other cases, the attacker floods a server with random different IP addresses. Janus inherently prohibits this random IP attack. Janus prevents this by using security measures on its OpenFlow switches. Hence, SDI sets forwarding-rules according to connected internal IP addresses that are issued and authenticated. Any divergence between source IP address and the IP rules that are in place will cause the packets to be dropped. Moreover, flooding with different source IP addresses will be rejected in the closest OpenFlow switch. This synchronicity will prevent attackers to forge IP addresses through SAVI SDI.

As previously mentioned, some attackers use malware to initiate attacks from other insecure devices known as bots [74]. These compromised machines behave as puppets for the attacker and launch an attack. One type of these security attacks is a DDOS attack. This attack will be detected via Intrusion Detection Systems (IDS).

3.4.2 Security Module Manager

Here, we review SAVI SDI RMS and its components. The aim is to create an image of SAVI SDI to aid the reader in having a better intuition of how some of the functionalities used in the security VNF design are performed in the lower layers.

Consider one VNF security module that gets managed by an SDI security module manager (shown in Figure. 3.8 and 3.9). SDI Security Module Manager requires a few information sources to customize its
decision regarding detection and mitigation. Some of these sources are SDI Monitoring and Measurement [46], Whale and SDI Network Control Module.

![Figure 3.8: Adding Security Module Manager to modules of SDI manager](image1)

![Figure 3.9: Security module manager should obtain information from Topology Manager and Network Control Manager. This cooperation of modules shall improve on the effectiveness of decisions made by the Optimal Decision maker.](image2)

In our work, we have studied a security module that detects certain types of DOS attack. For the victim, we use a web server containing a web page as a point of interest for attackers. The attackers will try to flood the web server using their bandwidth and high processing power. A typical result would be the exhaustion of web servers resources to respond back to the requests sent by the attackers. Consequently, the web server will be unable to provide service to regular users. The proposed security module employs a statistical anomaly based IDS as a deep packet inspector service to study and monitor the behavior of packets moving toward the targeted web server. Right after the attack occurs, the deep packet inspector will identify the attack and raises a flag.
3.4.3 Intrusion Detection System

IDS is to identify malicious system activities and violation of system policies. Usually, a detection is followed by some way of reporting the intrusion to system administrators. Some Network IDSs have the capability of preventing intrusion themselves. One way to detect an intrusion is to use deep packet inspection (DPI).

There are two methods for using the IDS (DPI) (as seen in Figures. 3.10 3.11. One is the case that the IDS only monitors the data by tapping from the line, and the other could be placing the IDS on the line as a serial module. We tested both of these scenarios. In comparison, the former had a near line speed performance since the computation is completely separate from the forwarding switch. Thus, the performance of switch does not degrade in the tapping case. In the latter, IDS system may be enhanced to perform the job of a Network Intrusion Prevention System as well. The issue is that the inspection part is CPU intensive. Since the CPU allocated for the IDS gets occupied by packet inspection, the performance of the forwarding section of the VM gets affected. Therefore if the DPI was used as an inline mode, the line speed becomes limited. In tapping method, two sets of rules will be installed on a tapping switch via Janus. One for copying the incoming traffic of the web server to IDS, and the other for copying the outgoing traffic of the web server to the security module.

![Figure 3.10: Detection using inline mode of IDS](image)

![Figure 3.11: Detect and attack using tapping mode of IDS](image)

It is true that all of these depend on the resources of underlying VM, and one can argue that by increasing the computation power on the inline mode, the forwarding will not be affected. The counter argument is that for the scalability purposes, tapping seems a better-suited version for this task. Since the number of security rules that needs to be processed might increase over time, tapping solution is more scalable.

Note: We choose to use the IDS in its tapping mode to avoid loss of line speed. Using a combination of IDS in the tap mode and SDI’s security module manager as the prevention system will be a good match to secure the web server.

Deep packet inspector also has the capability of identifying the attackers (in this case compromised users). Right after the attack was detected and the bots were flagged, the attack will be sent to SDI Security module manager as a report. This report allows Janus take a global decision on the network. In here, SDI Security manager contacts Whale to acquire the location of the reported IP addresses. Topology refers to its graph-based database to locate IP addresses. At this point, Janus makes a decision based on its security module manager to either study the attack or to block the attack.
3.4.4 Intrusion Prevention

Intrusion prevention system refers to acting on an attacker, either by modifying an IDS placed on an inline mode to not forward related packets or by manipulating OpenFlow Switches on different parts of the network to block or redirect the attacker. There are three possible Prevention Scenarios: Blocking the attack in the back end, blocking the attack in the front end, and study the attack in a different server (Fig.3.12).

![Diagram of Intrusion Prevention System](image)

Figure 3.12: Three possible points to mitigate an attack

In the case of an attack, the first measure as a short-term solution is to block packets associated with intrusion from reaching the original server. This action plan is initiated by sending a message to modify a flow in a switch near the IDS security module. Here the packets are blocked near the victim web server. If SDI determines the attack not to be fatal, SAVI/SDI can use an already created honeypot (a copy of the original server) and direct the attack to the honey pot. The location of honeypot will be determined via a decision from Whale as well as the SDI security module manager.

In case that the SDI does not find the attack interesting, it will contact Whale to map and locate the compromised IP address at its nearest Open-Flow switch. Then SDI will directly block the IP address at its closest OpenFlow switch (front end). In the other case that the security module manager decides to study the attack, SDI will determine a suitable point in the network to instantiate a Honeypot server or use an already created one. This Honeypot server could usually be set up on the same path as the attack to minimize the number of rule modifications on other switches. For instance, if an attacker starts scanning ports of the server to see which ones are open, this could be a start of an attack in the future. Therefore, Janus might see fit to forward this attacker to a honeypot before an attack is launched to be on the conservative side. The forwarding rules also have to act as a network address translator for source and destination addresses to hide the fact that the attacker is accessing the Honey Pot server.

3.4.5 Implementation and Evaluation of basic IDS/IPS

In this section, we discuss the testing and verification of the proposed methods. Parameters of interest include attack detection and response time, resource utilization and the transitional time from attack detection to mitigation. Here we have initiated a DDOS attack from some VMs to a web page. The
number of attackers for our experiment has changed from 1 to 4. At this moment, SNORT has been employed as an IDS.

**Snort, an open source IDS/IPS**

Snort is a free open source software for network security. It implements intrusion detection and prevention using Deep Packet Inspection (DPI). Snort performs real-time network traffic analysis, logging any suspicious packets according to certain defined security rules that are customizable. Having been around for about two decades, Snort has been praised as one of the best open source software ever written [13].

Snort has three modes of operation: Sniffer mode to display network packets on the computer console, Packet Logger mode to saves network packets to the disk, and Network Intrusion Detection System mode that compares captured packets to a set of rules to detect anomalies. The last mode of operation is of interest to us in this work. It is possible to filter network packets based on IP address and protocol in Snort. For instance, snort package used on SAVI TB was filtering out ssh packets used to access VMs. This filtration can significantly reduce the processing required by a security VM.

The NIDS modes benefit a configuration file as described above. In the configuration files, rules will be specified according to the snort configuration manuals. For instance, a rule can create an alarm if a TCP protocol is used to access a port of a monitored IP address. These alarms can be generated in multiple formats. The alarms could be sent to the console, file, or a socket of UNIX for another program. Here we have captured the alarms into a file. The file gets parsed, and useful alarm information gets extracted. This information will be processed by the Security module manager, and the decision will be passed to the OpenFlow switches.

In our basic setting, a rule was set up to count the number of connection requests to detect a DOS attack using Snort. Snort by default makes a decision every 5 seconds. Meanwhile, it keeps the state for each flow. The number of TCP requests for an IP address gets captured in the default period. An alarm will be raised if the number of requests becomes higher than the specified threshold. The threshold gets set by the administrator. Snort has three levels of threshold that is its recommended values. The administrator can change this values to the desired numbers learned from nature of the network. The default 5 seconds may also be modified. In the case of a DOS attack, smaller sample time might not be adequate to capture DOS attack. For instance, a period of 0.5 seconds could be error prone to false positives.

Our experiments have been developed and evaluated on SAVI Testbed, which as mentioned, is an implementation of SDI. The IDS sits on a VM that has been created via SAVI SDI. First, the behavior of the system without the security module in place has been studied. Second, the system was examined under a set of attack prevention methods, and the following system performance was extracted.

**DOS attack and web page response**

In the case that there are no security modules in place, as the number of attackers increase, the web page loading time increases as well. Ping time and web page load time have been tested. The time that it takes for the ping request to come back from a server under attack slowly increases (Fig.3.13). This change is drastically visible after adding a third attacker’s packets to the web server. The ping time does not strictly increase for every instance as the number of attackers grows and some ping requests might have a smaller number but on average, growth is visible. The web page loading time from the regular user to the web server is an almost exponential function of the number of attackers. In Figure
3.14 the time to load a web page from an attacked web-server via a regular user is demonstrated under a different number of attackers. This result is very intuitive. As the number of attackers increases the buffer will get more utilized with the attack packets. Therefore the delay for a response will increase significantly.

![Graph showing ping time vs number of attackers](image)

Figure 3.13: As number of attackers increase the ping time to the webserver increases. Also some packets will be lost in the network.

It was observed that the fourth attacker completely brought down the web page and the web page timed out (the time to load became bigger than 20 seconds). To fully study the attack, the resources such as CPU usage, memory usage, and bandwidth utilization on a VM was monitored. It was quite surprising to observe that with the number of attackers that we studied, the only depleted resource was the incoming/outgoing bandwidth of the web server. The CPU and memory usage fluctuated only by nearly 2% (as seen in Fig. 3.15), while the bandwidth of the traffic had a boost as the flood of traffic came in. The fluctuation of these two graphs indicates that our DOS attack did not significantly modify the CPU and the RAM usage.

### Attack mitigation evaluation

To assure that the system is properly responding to attack scenarios, two mitigation have been tested: Front end blockage, and Honey-pot. In Fig.3.16. The bandwidth resources on the web server have been monitored. As soon as an attack is initiated, a spike in the bandwidth utilization is visible.

This attack is followed by an automatic detection from IDS and a report to SDI security manager to block the attackers. Hence the bandwidth utilization will go back to normal after the attack. The bandwidth utilization has been looked at for the real web server as well as the HoneyPot server for the same period. In Figure. 3.17 we can observe that the traffic has been shifted from the Real Server to the
Figure 3.14: As the number of attackers to the web server increases, the more delay is observed for a regular user obtaining content.

Honey Pot after the attack is detected. The next important factor for our purpose is a response time from the initiation of attack to time of blockage. Our intention is to demonstrate the reasonable performance of the system. This end to end number varies according to the number of rules that are in place for the IDS module. But for our test purposes, the detection and re-routing numbers are demonstrated in the Table 3.1.

Table 3.1: Timing for detection and different mitigations.

<table>
<thead>
<tr>
<th>Action</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to detect</td>
<td>&lt; 5</td>
</tr>
<tr>
<td>Honey Pot transfer</td>
<td>&lt; 6</td>
</tr>
<tr>
<td>Block the IP</td>
<td>&lt; 2</td>
</tr>
<tr>
<td>Detection + Block</td>
<td>&lt; 7</td>
</tr>
<tr>
<td>Detection + Honey Pot</td>
<td>&lt; 11</td>
</tr>
</tbody>
</table>

Finally, to check whether our system identifies a legitimate traffic as an attack or not, large files were located on the web server. These data were to be requested by multiple (10 in our test case) VMs. In this case, the VMs were supposed to download a file. Although there was a spike in the bandwidth utilization of the web server, no attack was detected via the IDS.
Figure 3.15: CPU and RAM utilization did not undergo any stress after DOS attack. No significant information was extracted from these utilization to detect an attack.

Figure 3.16: Attack gets blocked after getting detected by the NIDS.
Figure 3.17: Webservers traffic gets shifted to the Honey Pot to study attackers behavior to provide attacker with false information.
Chapter 4

Scalability of Security Function
Virtualization in SDI

In this chapter scalability of security modules for Intrusion detection and prevention systems in a Software Defined Infrastructure will be discussed. The primary goal is to consider scalable architectures for a full security system. Our proposed distributed design is a more efficient method of attack mitigation compared to the solution proposed in the previous chapter. Finally, a general mitigation approach problem is formulated as an Integer Linear Programming (ILP) problem. We propose a parallelization (pipe lining) mechanism to make NP-hard ILP tractable.

4.1 Motivation

We are living in a time that typical applications have millions of downloaders from Google play as well as Apple-Store. Assume a virus, infecting ten of these applications, targets a web server. That would make it 10 million users trying to access and flood a web server. Assuming the upload of each user being 7 Mbps then the volume of attack would be almost 10 TBps. Therefore the question is how to block this number of IPs in a short period.

There are two main bottlenecks while dealing with these many attackers on the NIPS side: Flow setup rate and flow table capacity. There are new switching technologies that have tried to increase flow setup rate as we speak. But the rate of technology advancement is no match for the speed of attacks growth. The volume of attack was 400 Gbps according to [2] from 4529 NTP servers with an average speed of 87 Mbps in 2014. New Switch vendors have tried to employ different technologies to enhance their switch performances. Here we will study improvements on various switch properties and how they have affected limiting features of the switches, this study would be followed by comparing different current switches and justify our solution for real life implementations in security applications.

4.1.1 Previous work on scalability issue

Using programmable tables has allowed flexibility in the design of switches. These tables allow classification of flows into different tables. One instance of OpenFlow distributed table is OVS tables as seen in Figure. 4.1 [66]. In OVS there is a table for Exact match cache and another table for Mega Flow cache.
Packets from more frequent flows, passing through OVS, have an exact match and will be processed efficiently. The rest of packets will be handled by the MEGA FLOW table aggregating multiple flows together. Detecting correct forwarding rule in Mega flow table, which is significantly slower than the exact match table, requires more time. This sole property has allowed the OVS to increase the speed of flow processing that leads to higher flow handling rate.

For our security application purpose, attackers’ IPs will be found by the DPI in IDS. We can assume that the set of attackers will have a randomly distributed IP addresses. This randomness hinders use of Mega flows to block attackers. Therefore we only can use Exact matches for blocking attacks that do not provide many benefits of OVS.

Figure 4.1: OVS contains multiple levels of tables to minimize forwarding time.

Some literature has discussed usage of multiple kinds of tables to increase the number of rules stored on a switch. The rules are categorized and located on different tables according to their frequency of usage. The limitation imposed by the TCAM tables has stopped storage of all rules on the TCAM tables. Therefore usage of different levels of caches, as well as different kinds of memories, have been exploited to control the cost of switches while maximizing the number of stored rules on a switch [72].

In some approaches, a frequency of forwarding rules have been determined by an entropy measure and rules get swapped around according to their entropy [66]. Although it is sometimes difficult to predict the future of arriving packets, this method gives a better performance compared to nonprioritized tables.

An almost same argument as above holds for security applications as well. Without loss of generality, it is viable to assume that the nature of attackers would be of same entropy. This assumption is valid since the attackers will have an almost same locality in time due to the nature of attacks. Therefore the blocked attacks have to be usually stored with the same priority and in the same TCAM table for fast access. In conclusion, packet rate prediction property will not be useful when dealing with almost same attackers.

To increase flow setup rate in switches, a set of very powerful processors have been installed on top of switches. Mounting powerful processors on switches is completely against the direction of the original
idea in OpenFlow since switches were supposed to become simpler rather than being equipped with powerful processors. These processors have removed local flow modification bottlenecks. One instance of such commercial-grade switches are NOVI-flow [17] which has used an Intel-Core i7 processor on top of its network switches to deliver a relatively high flow modification rate up to 3200 flow mod/sec. Also, it benefits from a 125K to 1M TCAM entries, 8 GB DDR3 memory, and 16 GB SSD flash storage. The 3200 rules per second are still insignificant for a security problem of discussed scope.

In a more advanced technology, a server has been fully mounted on top of a switch to provide service by employing both software and hardware. A company called Corsa [5] has applied this architecture to produce fast flow modifications up to 10,000 per seconds. By looking deeper into this approach, we realize that not all these alterations are happening over the TCAM table, and some of the rule updates are happening on other cache or memories. As DOS attackers are actively attacking the system, their corresponding security rules are required to remain over TCAM. Hence even this setup can not meet the requirement of our solution.

In [40] a measurement over a vendor switch was taken. They observed that inserting rules with the same priority would take 3.3 ms while adding multiple priority rules will take as much as 18 ms on average as system stabilizes. The rule-packing feature of switches is the main reason for this time difference. They demonstrated that a linear relation exists between the number of rule installations and the time of installation. In our problem, we will consider different flow setup rates over the network switches. The Figure 4.2 was obtained from [40].

![Figure 4.2: Inserting same priority rules will take around 4 ms/rule on a vendor OpenFlow switch](image)

4.1.2 Issues with basic IDS/IPS

As discussed in previous sections, we demonstrated that security could be implemented via a network security module. In there, it was explained that security is provided as a service. Hence every authenticated user or ISP can contact SDI manager and acquire the security level that she desires.

The first method was to block the attack in the switch near the IDS. This decision could be pre-programmed into security modules so that the attack could be blocked right away after detection. Near IDS blockage is an efficient solution for fast-paced attack prevention since the decision is made locally and the attack will not reach the server. The catch is that this security method will not prevent the
attack from the base so the attack will still consume bandwidth resources on the network. This solution is similar to current industry hardware solutions.

Second, blocking the attack at the front end. This mitigation requires security module manager to acquire the location of the attackers IP from Topology manager. Then SDI manager will block the IP address of the attacker at a close switch to the attacker. The switch will prevent the bandwidth waste that existed in the previous case from not blocking the attacker at the front end. The attacker will only be blocked if the attack is not of interest for the security module manager for studying purposes.

The third method sends the attack packets to a fake server to allow the attacker to harm the fake server for the sake of studying that attack. Sometimes a user might be sent to a fake server if an IDS gets suspicious of the user’s intention for accessing the server.
Chapter 4. Scalability of Security Function Virtualization in SDI

4.2 IDS bottle-neck and scalability

The IDS contains an open source anomaly based packet inspector namely Snort, a single threaded DPI. This IDS monitors the packets and compares cumulative packet properties with the rules that were initially installed on it. Programmability of this open source software makes it a suitable IDS to be employed in a security module. It is worthwhile to mention that these rule comparisons depend on the processing power provided by their VMs. The number of rules that can be checked for a constant processing power on a security module is inversely proportional to the bandwidth passing through an IDS [65]. This bottleneck would be one area that we will focus during this paper.

Due to this bottleneck, a central security manager should take control of the system to wisely distribute the security modules and allocate processing power for them. This security module manager will access the topology manager to make topology based optimized decisions. As explained earlier in this chapter, security module manager is a part of SAVI RMS. It oversees the type of rules that are installed on top of each security module, assigns multiple functionalities for different security modules in the system, locates the security modules near intended monitored web-server, and decides on the number of security modules and processing power of each of them.

It is quite essential to look into details of that design. Every switch is connected to an OpenFlow controller (RYU in SAVI Testbed). Since every switch is part of SAVI testbed, Whale is aware of the location of each switch. With the information that Janus receives from Whale, Janus can notify its security module manager to allocate security modules in locations close to that OpenFlow switches to tap traffics.

Processing power on the security module varies according to the size of traffic that needs to be tapped and the number of rules required to be checked. Here we discuss four methods that the IDS could be scaled, and each method’s pros and cons will be considered. In all the below cases it is assumed that a fixed bandwidth will be monitored for a large set of rules.

4.2.1 case 1: One security vendor, no modularity

When VMs get created, the user will be asked for the number of Virtual CPUs (VCPUs) on the VM. For a security module, the number of VCPUs is correlated to the number of rules that could be checked and the size of passing traffic. It is possible that an IDS gets multiple CPUs to control network traffic for a switch (as seen in first part of Figure. (4.3a), followed by division of traffic inside the VM. The divided traffic would be sent to multiple CPUs (using the multi-core feature of Snort 3.0), and information gets aggregated for analysis in one VM. This aggregation is a very desirable approach since traffic is not duplicated at all and it could be efficiently used.

**case 1: SAVI Testbed Implementation:** The size of a security module can be automatically controlled using the orchestration service of Janus. The vCPU utilization of the VM shall be controlled using the monitoring and the measurement of Janus (or by using ceilometer) to adjust the VM size in the case of a rise in traffic.

The one issue might be that the designer might want to use more modular features to verify whether incoming traffic obeys the normal behavior defined by the IDS or not. This modularity is to avoid having a single point of failure in a VM. In this design, if the IDS VM goes down then nothing will be monitored on the web server. This would lead to the next approach.
4.2.2 case 2: one security vendor, modular design

An ISP might want to have separate modules for an IDS to verify different rules (i.e. buy two separate functionalities from different security providers). For instance, one module could check for DDOS attacks while the other may monitor web attacks, etc. To do so, the ISP should be able to route traffic into each of these modules. The ISP could connect multiple of these modules to a switch’s different virtual ports and monitor the traffic. This topology is shown in Figure (4.3b).

**case 2: SAVI Testbed Implementation:** Deploying this architecture is easily obtainable using VINO. VINO allows service chaining on a single region using VXlans as was discussed. The tapping API can be used for port mirroring of the OVS’ internal ports. Rest becomes the basic IDS module implementation transferring the anomalies to the security module manager.

One issue using port mirroring is that the traffic has to be replicated for different modules. Port mirroring would put a lot of overhead on the main switch. Also, the outgoing bandwidth of a Switch might not be enough to forward the incoming traffic to many IDSs. The Third approach could be implemented to reduce port mirroring effect.

4.2.3 case 3: one security vendor, modular design, reduced port mirroring

In this case, a tree of VMs will be connected to the main switch to acquire the incoming traffic. This architecture can significantly reduce the port mirroring since only the root of the tree will obtain the traffic from the main switch. Then the traffic will be copied to other security modules that exist in the system. This approach could be seen in Fig (4.3c).

The deeper the security module is embedded in the tree; the more delay is introduced for the traffic to reach that IDS. Hence depending on the delay sensitivity of different rules, the ISP might want to rearrange security modules on the tree.

**case 3: SAVI Testbed Implementation:** To create the tree nature of the IDS, the tapping (port mirroring) and the chaining features are required in combination. For nodes with a single child, the basic chaining is enough to put the node in the middle of two modules. For nodes with children, the tapping is required to mirror ports to multiple modules.

The below model will be needed in the case that the user (designer) decides to use multiple security vendors to run a security check on traffic

4.2.4 case 4: multiple vendors

In this case, the bandwidth will be sent to multiple vendors (each with a black-box IDS) where each of them will check for a different set of rules (as in Figure. ??). For instance, one could detect an attack using arrival of packets, and the other could identify the same attack using nature and content of the packets.

In general, each module can report to the Module Manager to block its malicious users. Wild card determination, as well as weighted voting mechanism, could be offloaded from Security Module manager to the Voting mechanism. Each of these vendors can implement their security system using one of the three methods described before.
Figure 4.3: IDS scalability techniques
4.3 IPS bottle-neck and Scalability

As previously described an NFV security function was used to detect IP addresses which were responsible for an attack. Two possible methods could have been blocking the attack at the server or blocking the attack at the front end of attack initiation.

OpenFlow switches have different vendor specific properties. Some features vary according to supplier’s use of equipment and technologies. One simple difference is TCAM technology among the various suppliers. This technology difference affects the size of forwarding tables. The more complex rules stored in the TCAM, the fewer rules can be pushed into it [28]. Another difference between different switches is the rate of rule installation on the various TCAM tables. But one thing that all of them have in common is that they all have limitations.

This following sections look at a simple mathematical proof of each proposed IPS implementation and then moves to more complex scenarios.

Consider a topology where attackers initiate an attack from Front-End, and all traffic goes through a so called Direct Path. As soon as the security module manager gets notified of detected attackers, a decision will be made according to the sensitivity of the application to block the attackers.

In the case that the security module manager decides to block the attack due to its severity, there is a time associated with its blockage. Assume that $R_{front}$ is the rate of installing rules on a front end OpenFlow switch. $N$ is the number of attackers. $Time_{IDS}$ is the time associated with detecting the attacks. $Time_{Report\ Attack}$ is for reporting the attack to the central attack manager. $Time_{Process}$ is for determining the proper location of attack blockage. $Time_{install}$ is for installation of rules on the OpenFlow switches. $Time_{Report\ Decision}$ is for sending the attacker to the blocking switch. Finally, $Time_{total}$ is the total time from attack initiation to its full mitigation. Therefore:

$$Time_{total} = Time_{IDS} + Time_{Report\ Attack} + Time_{Process} + Time_{Report\ Decision} + Time_{Install\ Rules} \quad (4.1)$$

where

$$Time_{Install\ Rules} = \frac{N}{R_{front}} \quad (4.2)$$

In this formula, the $N/rate_{front}$ represents the time that it takes to install rules on the Front-End switch. We call this installation time. It is easy to observe that the total time is a linearly dependent on detecting attacker time. We have not considered the complexity of $Time_{report}$ for now since a batch of IP addresses can be forwarded very quickly to the SDI module manager.

In the following cases (as seen in Fig. 4.4), it is tried to reduce the installation time with a heuristic algorithm that does not guarantee any time performance for the blockage. The approach is to block the attack over the path that all these attackers employ to reach the server.

4.3.1 case1:Same switches

Say there are K switches over that path all of which have $R$ rule setup rate (dedicated to security application). Therefore the total time will change to

$$Time_{Install\ Rules} = \frac{N}{R \times K} \quad (4.3)$$
Here we observe that the Block-Time has been divided by K since there are multiple switches cooperatively blocking the attacks on the same path. The calculated Block-Time is the total time that takes to stop packets fully from reaching the destination. We need to consider the necessity of migrating the rules on the path back to the gateway switch.

![Diagram of switches](image)

Figure 4.4: All switches have the same rate for blocking attacks on a path

**Feasibility Analysis of Basic IPS on SAVI SDI:** We created a basic topology consisting of four OVS switches running on multiple Virtual machines connected together using VINO. To measure the flow setup rate of our operating OVS switch in Gateway, we sent ten thousand flow setup requests to the OVS in a second and examined the installed rules of the OVS table. It was observed that only 5256 rules were fully installed on the OVS. Taking this number as a ballpark of the flow setup rate we used the same RYU controller to send forty thousand rules to the four OpenFlow switches created by VINO. After printing the existing rules, 21 thousands of the rules were setup. It was difficult to generate these many VMs to generate the 21 thousand attackers. therefore this test was used to prove that this could be an efficient method of blocking the attackers. As RYU and OVS are implemented on the SAVI Testbed, it is theoretically possible to obtain similar performance from the security module manager of SAVI infrastructure.

### 4.3.2 case2: different rates

This case (as seen in Fig. 4.6) assumes that switches on the same path have different rule-setup rates. Considering k switches with rates $R_i$ the total time that it takes to block the attack on that path will be:

$$Time_{Install\ Rules} = \frac{N}{\sum_{i=1}^{k} R_i}$$

(4.4)

As it is observed the total time to block an attack is a function of the number of switches located over the Direct path as well as the portion of rule setup rate that is associated with the different switches located on that path.
The number of attackers blocked on switch $j$ is $Task_j$ where:

$$Task_j = \frac{R_j}{\sum_{i=1}^{k} R_i} \times N$$ (4.5)

Figure 4.6: Switches have different rate for blocking attacks on a path

4.3.3 case3: different rates, different table size

Here (as seen in Fig. 4.7), we assume that there is a table size limit (dedicated to network security apps) on all switches $T = [T_1, ..., T_k]$. Note that in here, the table size limits are effective-table-sizes, which are only a portion of each switch’s flow table capacity. Assumption is that Number of attackers is less than total flow setup limit of all switches on that forwarding path.

Figure 4.7: Switches have different rate, and different effective table sizes for blocking attacks on a path

Now the goal is to minimize the total time that the attacks are in effect. Therefore the problem becomes:

$$\text{minimize } \max\left(\frac{1}{R_1}Task_1, \frac{1}{R_2}Task_2, \ldots, \frac{1}{R_k}Task_k\right)$$

subj.to  

$$0 \leq Task_j \leq T_j$$

$$\sum_{j=1}^{k} Task_j = N$$
this problem can be transformed to

\[
\text{minimize } t \\
\text{subj. to } \frac{1}{R_j} T_{\text{task}_j} \leq t \\
T_{\text{task}_j} \leq T_j \\
T_{\text{task}_j} \leq R_j * t \\
\sum_{j=1}^{k} T_{\text{task}_j} = N
\]

since \(T_{\text{task}_j}\) is a large integer, we can assume it to be non-integer and use LP to solve the problem.

### 4.3.4 case4: different rates, different table size, multiple paths

Assume that there are \(N\) attackers each with a different path. The path of each attacker represents the possible attack blocking switches for the attacker. For instance, if we assume that there are five switches and the fourth attacker is passing through second and third switches, then the vector associated with possible blocking switches for that attacker would be \(p_4 = [0, 1, 1, 0, 0]\). In the same way \(p_i = [p_{i,1}, p_{i,2}, \ldots, p_{i,K}]\) represents possible blocking switches for attacker \(i\). \(K\) is the total number of switches in the network. This information is available from Whale (Fig. 4.8).

![Figure 4.8: Switches have different rate, and different effective table sizes for blocking attacks in a network](image)

We define a selection variable \(x_{i,j}\) where \(x_{i,j}\) is equal to 1 if attacker \(i\) gets blocked on switch \(j\). It is apparent that

\[
\sum_{j=1}^{K} x_{i,j} = 1 \tag{4.6}
\]

as each attacker has to get blocked on a switch. Also

\[
x_{i,j} \leq p_{i,j} \tag{4.7}
\]

as each attacker can be only blocked on switches that are on her path. We are not considering re-routing an attacker here.
Again, the goal is total attack time minimization:

\[
\begin{align*}
\text{minimize} & \quad \max \left( \frac{1}{R_1} \sum_{i=1}^{N} x_{i,1}, \frac{1}{R_2} \sum_{i=1}^{N} x_{i,2}, \ldots, \frac{1}{R_K} \sum_{i=1}^{N} x_{i,K} \right) \\
\text{subj. to} & \quad \sum_{i=1}^{N} x_{i,j} \leq T_j \\
& \quad x_{i,j} = [0,1] \\
& \quad \sum_{j=1}^{K} x_{i,j} = 1
\end{align*}
\]

which can be transformed to

\[
\begin{align*}
\text{minimize} & \quad t \\
\text{subj. to} & \quad \frac{1}{R_j} \sum_{i=1}^{N} x_{i,j} \leq t \\
& \quad \sum_{i=1}^{N} x_{i,j} \leq T_j \\
& \quad x_{i,j} = [0,1] \\
& \quad x_{i,j} \leq p_{i,j} \\
& \quad \sum_{j=1}^{K} x_{i,j} = 1
\end{align*}
\]

Knowing this problem is NP-HARD due to its binary linear programming nature (0−1 integer programming), we start solving this problem by using a bisection on \( t \). The problem becomes a feasibility problem (i.e.: \( t = t_0 \))

\[
\begin{align*}
\text{find} & \quad x_{i,j} \\
\text{subj. to} & \quad \sum_{i=1}^{N} x_{i,j} \leq R_j t_0 \\
& \quad \sum_{i=1}^{N} x_{i,j} \leq T_j \\
& \quad x_{i,j} = [0,1] \\
& \quad x_{i,j} \leq p_{i,j} \\
& \quad \sum_{j=1}^{K} x_{i,j} = 1
\end{align*}
\]

The solution can be found using MILP solvers (for small size of data).
4.4 Analysis of IPS solutions

It is worth looking at this solution more precisely. We mentioned that the total time that it takes for attacks to be mitigated is

\[ Time_{total} = Time_{IDS} + Time_{Report\, Attack} + Time_{Process} + Time_{Report\, Decision} + Time_{Install\, Rules} \]  

(4.8)

Consider using this supposedly scalable method of blocking attackers. Here we want to compare Inline blocking, with Back-End Switch blocking (blocking attacker at the location of IDS), with basic IDS/IPS solution (front-End mitigation) blocking the attacker at its earliest switch visiting, and with the scalable attack mitigation using switches on attacker’s path. The comparison can be seen in Table 4.1.
Table 4.1: Comparison of components Mitigation techniques for DDOS. Comparisons are normalized with the size of the network (Low, Average, High).

<table>
<thead>
<tr>
<th>Comparison Criteria</th>
<th>Inline NFV</th>
<th>Back-End Switch</th>
<th>Front-End Switch</th>
<th>Scalable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>Inefficient</td>
<td>Inefficient</td>
<td>Efficient</td>
<td>Average</td>
</tr>
<tr>
<td></td>
<td>-The bandwidth of attack reaches the VNF</td>
<td>-The bandwidth of attack reaches the switch that the IDS is tapping</td>
<td>-The bandwidth of attack no longer reaches the network</td>
<td>-The bandwidth of attack still reaches the network but gets blocked on many switches</td>
</tr>
<tr>
<td>Time to Report Attack</td>
<td>Zero</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>The inline VNF does not report the attack to the Security Module Manager and directly drops the packets reaching it</td>
<td>A local Security module manager can be provisioned to enable fast communication to security module and OpenFlow Controller</td>
<td>A local Security module manager can be provisioned to enable fast communication to security module and OpenFlow Controller</td>
<td>Contacting the Central Security Module Manager takes a relatively large time depending on the size of the network.</td>
</tr>
<tr>
<td>Time to Process</td>
<td>Zero</td>
<td>Zero</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>The inline VNF simply blocks the attack, and no decision is to be made</td>
<td>The OpenFlow Switch can be reported by the VNF to block the attack at the switch simply without making any decision</td>
<td>The Security Module manager has to ask Whale for the origin of the attacker. The processing time is approximately the time to query from the Whale. This can vary based on the size of graph stored on Whale’s database.</td>
<td>The Security Module Manager has to ask Whale for the full path of the attacker. After obtaining the path, it should make a proper decision regarding the mitigation point of the attacker. Solving an MILP with a large data set could be very challenging.</td>
</tr>
<tr>
<td>Time to Report Decision</td>
<td>Zero</td>
<td>Low</td>
<td>High</td>
<td>Average</td>
</tr>
<tr>
<td></td>
<td>The inline VNF does not receive a decision from the Security Module Manager</td>
<td>A local Security module manager can be provisioned to enable fast communication from security module to the OpenFlow Controller</td>
<td>Even with presence of a local security module manager, it might take a while to contact the front-end Switch</td>
<td>Mitigating switches are dispersed with different time to reach (for Security module manager)</td>
</tr>
<tr>
<td>Time to Install Rules</td>
<td>Does not Apply</td>
<td>High</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>querying the forbidden IPs becomes time-consuming</td>
<td>Installing all flows on one switch takes a long time and might be unfeasible</td>
<td>Installing all flows on one switch takes a long time and might be unfeasible</td>
<td>Installing flow is distributed</td>
</tr>
</tbody>
</table>
4.4.1 Scalability of network wide mitigation

The supposedly scalable solution seems to be challenged when it comes to solving the problem for a size of million attackers. Even in a small problem with 1000 attackers and 20 Switches, the MILP solver took almost six seconds to solve the problem. This indicates that using the MILP solver is not sufficient in finding the solution for distributed scalable IPS. On the other hand, with simple Front-End blocking all 1000 attackers on a switch (with flow setup rate of 5000 per seconds) can be blocked in only 0.2 seconds.

Assuming that the Front-End Switch can block all one million attackers in its flow table, this action takes 200 seconds. We intend to see whether we can use the idea of the scalable solution to decrease this time. Before modifying the scalable solution, we want to study the effect of the number of attackers on the duration of solving the MILP.

For a network with 20 switches, we observed the following times to solve the MILP problem for a different set of users (as seen in Fig.4.9).

![Figure 4.9: The time to block attackers seems to have an exponential form](image)

We can not precisely calculate the duration of blocking one million attackers using MILP solver. First Issue is the size of data. One of the constraints in the optimization problem \((x_{i,j} \leq p_{i,j})\) requires a matrix with a very large number of entries. For the problem with a million entries the matrix size becomes \(1 \text{ trillion} \times \text{number of switches}^2\). This matrix size is not accepted by solvers.

To avoid using such data, we consider solving the problem in batches using MILP solver.

4.4.2 Efficient use of MILP solver

Figure 4.10 demonstrates the average processing time dedicated to each attacker with respect to the size of attackers considered in the each subproblem.
Chapter 4. Scalability of Security Function Virtualization in SDI

Figure 4.10: The time to block each attackers has a minimum in our use case around 320 attackers

We can compute the allocation of attackers in batches of 320 attackers, then update the resources consumed and repeat this process (as seen in Figure 4.11)

Figure 4.11: The processing time with this iterative approach becomes 2500 seconds for 1 million attackers

The processing time of this approach takes longer than directly blocking them on the Front End Switch.

In the mentioned approach, Iterations are done in a serial manner. Meaning that iteration $m + 1$ waits for the iteration $m$ to report the resources left.

We wanted to make this calculation faster by making this process parallel. Here, We propose a basic parallelism computation distribution among multiple VMs to make solving the problem faster. As seen in Fig. 4.12, after assigning the 320 attackers (found via simulation previously), the process estimates the resources that might be used after computation. The goal of this work was not to propose the best estimation, rather it was to find an estimation to enable pipe lining. We used weighted round robin as a mean to estimate possible allocation of attackers to the switches.

The computed estimation will be passed to the next process to calculate its placement of attackers. This way the next process (previously called iteration) does not need to wait for outcome of the results from this process.

These processes will be connected to each other as seen in Fig. 4.13 with one of the nodes as the head. Each process (running on a separate VM) receives an estimate from the its previous process. The
Figure 4.12: This method estimates the resources left using a rapid random assignment technique. The inputted estimate of resources gets corrected by previously consumed resource offset and the process iterates again.

Figure 4.13: Processes (deployed on VMs) will be connected in a circular fashion.

The processing time is inversely proportional to the number of VMs connected in a circular fashion. It is worth noting that increasing number of VMs results in an error prone estimation, hence a trade-off between the estimation error and the number of VMs exists.

Using 100 VMs in a circular fashion can reduce the processing time well below 30 seconds. More detailed discussion on the estimation technique as well as other scalability aspects of proposed security architecture are to be explored in future works.
Chapter 5

Realtime content delivery and SAVI

This chapter is mostly dedicated to studying the effect of vCPEs and Service chaining on an RTCD application. Section 5.1 discusses an RTCD application on SAVI and considers basic cases with and without vCPE to demonstrate the necessity of vCPEs on SDI. Next, this section formulates the service chaining to reach the optimal solution of service chaining capacity maximization problem. A heuristic solution has been proposed and evaluated in the section to make service chaining (an NP-hard problem) tractable. At the end of this section, related works on RTCD application concludes the section. Section 5.2 discusses evaluation of fast network flow resumption of VM migration as in [77]. Fast VM migration with less service interruption is a necessity in having a dynamic Service Chaining platform.

5.1 Enhanced Real Time Content Delivery using vCPE and NFV Service Chaining

Real-time content delivery (RTCD) systems support applications and new business models in sectors such as finance, security, and telecommunication. In security, face recognition algorithms determine identities and report flagged criminals to police departments. In telecommunication, the live content of sports events is streamed to video subscribers with low delay. RTCD systems are required to avoid content delivery delays for different geographical locations, minimize system inaccessibility in dynamic applications, and provide applications to a higher number of users.

Computing and networking resources need to be provisioned to avoid excessive content delivery delays and enable buffering required by RTCD networks. These resources have become an essential part of content upload, distribution, and streaming in settings where live content needs to be processed rapidly to meet customer needs for superior quality and lower delay.

Availability of dynamic RTCD applications can be improved by converged management of heterogeneous network and compute resources via a software-defined infrastructure (SDI). To cope with dynamic nature of RTCD applications, SDI efficiently handles configuration modifications for the application by rerouting network traffic, scaling computational resources or even fully migrating computational resources [49]. SDI also employs a unified interface that facilitates deployment and management of RTCD technologies such as monitoring and live broadcasting [75].

Network and resource re-configurations usually cause short duration system disturbances affecting system availability. Therefore minimizing system reconfiguration is essential in maximizing systems avail-
ability. The disturbances oblige content delivery providers to migrate resources minimally and deploy a mechanism to release and acquire computing elements rapidly. SDI provides such features by converged control of both networking and computing resources. The architecture of an SDI implementation shall be used to enhance RTCD in this work. The Smart Application on Virtual Infrastructure (SAVI) testbed is a fully deployed SDI resource management system (RMS). SAVI aims to support low-latency and high data volume applications based on a multi-tier cloud including Smart Edges. The notion of multi-tier cloud on SAVI extends to sensor networks with a third tier that contains virtualized Customer Premises Edges (vCPEs). vCPE’s definition in SAVI is different from virtual customer premises equipment, running fixed set of network functions. vCPEs in SAVI are referred to container hosting platforms located at the extreme Edge.

This section aims to create a scalable algorithm that processes submitted requests to the VNF chaining platform [73][35][33][62] of an RTCD application running on SAVI. The objective of this algorithm is to place and connect the VNFs in a way to maximize the capacity of submitted requests. The results of this algorithm note the importance of having vCPEs in increasing service coverage of the RTCD application.

5.1.1 System description

Kaleidoscope [75] is an application that involves gathering and distribution of video in live events to and from massive groups of users, and as such, it is an ideal use case for an RTCD application on the SAVI testbed. Kaleidoscope can leverage dynamic cloud resource allocation and network configuration in real time to achieve efficient resource management and better service performance. Our work can be used for resource allocation and service chaining in Kaleidoscope.

We examine a stadium using Kaleidoscope (seen in Fig. 5.1). The stadium is equipped with high-quality cameras placed in different angles of the stadium. These cameras are connected to Wi-Fi access points and send video feeds to a streamer server. In this work, we solely focus on content distribution from the streamer server to the fans in the stadium.

Cloud services are necessary for conventional stadium content delivery systems. Conventional systems mostly provide a video stream on a single stadium screen. Therefore conventional computing resources are simply not powerful enough to accommodate processing of tens and even hundreds of thousands of smart devices. Also, stadiums debate between renting resources from Infrastructure providers versus purchasing a full private cloud to handle their usage.

![Stadium physical topology considered in basic cases](image)

**Figure 5.1:** Stadium physical topology considered in basic cases

In the setting under consideration, all users should be able to stream content captured by the cameras in the field at all times. Besides, a playback feature has to be available for a percentage of these users. A set of traffic requests should be submitted to the VNF chaining platform of the SAVI testbed to
enable stream and playback features. These traffic requests contain service chain information for content distribution originated from a streamer-server. Each service chain traffic request includes deep packet inspection (DPI), HTTP-streamer, cache, and the streamer server. Aside from the streamer server, the locations of all VNFs are unknown. The streamer server is located on the Smart Edge as a single shared instance. Therefore, most of the requests will be directed from the vCPE nodes to the Smart Edge.

In our case an ordering for placement of all VNFs is necessary. Each user request is in the form of a strand, a small set of connected chains that form a graph. The VNF strands are shown in Fig.5.2. In the stream case, the DPI has to be the first module that a request from a user logically visits. This placement of DPI prevents harms that a request can cause on the SDI. It is important to note that the DPI is only used on the uplink (having 5 percent of the downlink traffic) to avoid any contention. Following the DPI, a request should face an HTTP-streamer module. In this configuration, a separate HTTP-streamer VNF (for stream and replay) and Cache VNF (for replay) is needed for each request. In general, VNFs might change the nature and size of the bandwidth. DPIs forward the data without changing the bandwidth of the flows.

Local caches reduce bandwidth consumption of the playback feature. In the rest of this section, we demonstrate that deployment of local caches on vCPEs saves on total bandwidth and allows more users to stream. To efficiently use scarce resources on vCPE, we propose the use of the containers instead of VMs for deploying VNFs. In other words, VNFs will be deployed on containers that are running on vCPEs.[69].

Here we define multiple basic cases and find the maximum number of users (streaming and replaying with $b = 5Mbps$) that can be supported by the SDI. As seen in Fig.5.1, $\beta_0$ is the bandwidth between the Smart Edge and the top-tier switch, $\beta_1$ is the bandwidth between the top-tier switch and each low-tier switch, $\beta_2$ is the bandwidth between low tier switches and each of their connected access points, and $\beta_3$ is the bandwidth of wireless link connecting each access point to its user group. $K_1$ is the number of low-tier switches and $K_2$ is the number of the access points per low-tier switch according to the figure.

Consider following numbers as example to ease understanding of first few cases: $\beta_0 = 10\text{GigE}$, $\beta_1 = 1\text{GigE}$, $\beta_2 = 600\text{Mbps}$, $\beta_3 = 200\text{Mbps}$, $K_1 = 300$, $K_2 = 5$. These numbers and all following example quantities are obtained from existing industry hardware and software.

5.1.2 Collocated chain model

First, we assume all VNF modules of a strand have been collocated on the same compute module. The strands have been transformed to chains by ignoring the up-link due to its smaller bandwidth, and video has been multi-casted from streamer server to all possible locations for the HTTP-streamers (seen in
**Smart Edge compute only**

In this case, the bottleneck is the link between Smart Edge and the top-tier switch with bandwidth $\beta_0$. The SE can support up to $\beta_0/b$ users (2000 fans in our example).

**vCPE enabled access point only**

Despite the previous case, all VNFs have to be placed on the APs. Each user needs $b$ bandwidth per stream or replay from the wireless bandwidths. Hence $(\beta_3/b)$ users can be supported by an AP accounting for $K_1K_2(\beta_3/b)$ in total (i.e. 72000 fans in our example).

**limited vCPEs only**

Consider a case that only $K'_2$ (i.e. 1) out of $K_2$ APs are equipped with vCPEs. Therefore these vCPEs could be shared by their neighbor APs. The multi-cast used to distribute $V$ (i.e. $V = 10$) camera feeds only consumes a bandwidth $Vb$ from the link connecting low-tier switch to the vCPEs. $\beta_2 - Vb$ is available to service neighbor access points. Therefore number of users supported by the SDI (here $\approx 42000$ fans) is limited to the capacity of wireless links ($\text{WIFI}_{\text{cap}}$) and capacity of outgoing links from the APs with vCPEs ($\text{vCPE}_{\text{link-cap}}$)

- $\min \{ \text{WIFI}_{\text{cap}}, \text{vCPE}_{\text{link-cap}} \}$
- $\text{WIFI}_{\text{cap}} = K_1K_2(\beta_3/b)$
- $\text{vCPE}_{\text{link-cap}} = K_1K'_2((\beta_2 - Vb)/b + \beta_3/b)$

**Smart Edge and limited vCPEs**

In this case the bandwidth left from multicast on the main link shall be used to service the users. This modifies the previous formula to: (43980 users)

- $\min \{ \text{WIFI}_{\text{cap}}, \text{vCPE}_{\text{link-cap}} + \text{SE}_{\text{link-cap}} \}$
- $\text{SE}_{\text{link-cap}} = ((\beta_0 - Vb)/b)$
Processing limit on the vCPEs

Assume that collocated VNFs from a stream request chains require res1 amount of vCPU and each vCPE has vCPU capacity of RES1. The problem becomes

\[
\text{minimum} \quad \text{WIFI}_{\text{cap}}, \text{vCPE}_{\text{cap}} + \text{SE}_{\text{link-cap}} \\
\text{vCPE}_{\text{cap}} \quad \min(\text{vCPE}_{\text{link-cap}}, \text{vCPE}_{\text{res-cap}})
\]

where vCPE_{\text{cap}} is the number of users supported by resource and link capacity of vCPEs (i.e. RES1/res1 is the users supported by resources). To increase the vCPE_{\text{cap}}, it seems reasonable to distribute VNFs of the strands instead of collocating them.

5.1.3 Service chaining on constrained heterogeneous resources

We have realized by now that reducing computation on the vCPEs of the AP can lead to higher number of users. To efficiently use computation on the APs the location of DPIs and HTTP-streamers, as the main computation hungry VNFs, are of great importance. So far the effect of adding vCPEs on the number of users was observed. The vCPEs and Switches with processing are becoming very common in today’s networking research. It was notable that vCPEs do not need to be installed on each and every AP to provide a reasonable service to the users.

In this case, we are assuming APs and lower-tier switches are equipped with heterogeneous resources. Computing the maximum number of users that can be supported by the infrastructure may not be as straightforward as it was in previous cases.

Next, a mathematical formulation is developed to explore the effect of service chaining on the maximizing of user counts.

5.1.4 Problem formulation

Connectivity of servers (compute modules) S with direct physical links L is represented by a directional graph \( G(S, L) \) in which server \( S_i \) is connected by link \( L_{ij} \) with effective bandwidth (chosen below 70% of actual rate to avoid queuing delay) of \( \beta_{ij} \) (Gbps) and propagation delay of \( \delta_{ij} \) (ms) to server \( S_j \). Delay on the links (header processing delay) is comparably smaller than computational delays (delays involved in using RAM). This makes the total delay the sum of all processing delays and a constant number. The constant delay eliminates the need to discuss the notion of delay bound in our problem formulation. Each server has a set of resources \( R = \{r_1, r_2, ...\} \) i.e: CPU (GHz), RAM (GB), Disk (GB), Disk Read Speed (Mbps) and a capacity associated with each resource \( C_r \). Since abundant storage is available, the storage capacity is not viewed as a bottleneck and is removed from the calculations.

In our use-case, a group of requests \( T \) is submitted to the RTCD application. Each requests \( t \) contains an ingress point \( u_t \), an order of VNF chain \( \zeta_t \), a vector of bandwidths \( \beta^t \) that could change in each VNF process. Every VNF \( m \in \zeta_t \) has a set of required resources \( \rho^t_m \) that need to be fulfilled with the SDI. Also a vector of virtual links \( \Gamma^t \) is associated with each request. A physical link from a server to itself, modeling inter-server traffic forwarding, has been regarded for collocated VNFs on servers.

As link delays were determined negligible with respect to the VNF delays, the total VNF delays in a chain are irrelevant of where the VNFs are placed and therefore constant.
Table 5.1: Decision variables for VNF placement and link assignments

<table>
<thead>
<tr>
<th>Var.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{tms}$</td>
<td>VNF m from traffic t on server s placement decision variable</td>
</tr>
<tr>
<td>$y_{tγl}$</td>
<td>virtual link $γ$ from traffic t on physical link l chaining decision variable</td>
</tr>
</tbody>
</table>

To be fair between users, our goal is to equally maximize number of users that can stream among different branches. A branch is a sub-tree of the SDI tree (as seen in Fig. 5.1 with a low-tier switch as its root. Our topology has $K_1$ branches (equal to the number of lower-tier switches in the SDI). By defining $T_{str,i}$ as the number of users that can stream from an AP of branch $i$, our goal becomes maximizing the minimum of the $T_{str,i}$s among different branches. This problem is equivalent to maximizing number of chains per AP when all access points have similar number of chains:

\[
\text{maximize} \quad \text{stream} \\
\text{s.t} \quad \text{stream} = α \text{ replay} \\
\quad \text{stream} = T_{str,1} = ... = T_{str,k} \\
\quad \text{replay} = T_{rep,1} = ... = T_{rep,k} \\
\quad α \quad \text{Replay percentage} \\
\quad \text{constraints} \quad \text{bandwidth and compute limitations}
\]

The maximum number of chains depends on the allocation of VNFs onto the physical servers. To formulate this allocation, a parameter $x_{tms}$ is needed to indicate whether VNF $m$ from traffic $t$ is assigned to server $s$ or not. On the other hand, physical links that are used to connect VNFs together have limited capacity. To determine physical link $l$ usage for virtual link $γ$, another parameter $y_{tγl}$ is necessary.

To ensure that traffic requests $T$ are assigned and connected as submitted to the chaining service, additional constraints need to be applied to the optimization problem as described next.

To guarantee that all VNFs for the stadium are placed on the network, each VNF has to be allocated to one and only one server:

\[
\forall t \in T, m \in ζ^t \\
\sum_s x_{tms} = 1
\]

Allocated containers to a server should require less resources than what is available on that server:

\[
\forall s \in S, r \in R \\
\sum_t \sum_{m \in ζ^t} x_{tms} * ρ_m \leq C^r_S
\]

Each virtual link connects a VNF to another. To avoid redundant traverses, only a single direction
of a physical link should be associated to a virtual link ($\bar{l}$ is the reverse direction of physical link $l$).

$$\forall l \in L, t \in T, \gamma \in \Gamma^t$$

$$y_{\gamma l}^t + \gamma_{\gamma \bar{l}}^t \leq 1$$

Virtual links that are mapped to a physical link should require less bandwidth than what that link provides:

$$\forall l \in L$$

$$\sum_{t \in T} \sum_{\gamma \in \Gamma^t} y_{\gamma l}^t \beta_t^t \leq \beta_l$$

The following constraint assures that at least a physical link is used for each virtual link:

$$\forall t \in T, \gamma \in \Gamma^t$$

$$\sum_{l \in L} y_{\gamma l}^t \geq 1$$

Next one needs to assure that every link of the traffic requests is fully mapped to the physical network. This condition is sufficient only if two consecutive VNFs on a chain do not collocate. We also require full path connectivity by equivalent input/outputs for flows between nodes. The reason for separating sums in the below expression is to emphasize the inbound and outbound traffic. The source and destination of each traffic are considered in above constraint.

$$\forall t \in T, \gamma \in \Gamma^t$$

$$\{(i,j)\mid \gamma = (i,j)\}, \{(s,d)\mid (s,d) = l \in L, s \neq d\}$$

$$\sum_{d \in S} y_{(i,j)(s,d)}^t - \sum_{d \in S} y_{(i,j)(d,s)}^t = x_{is}^t - x_{js}$$

To consider inter-server links, the possibility of incorrect collocations are eliminated as follows:

$$\forall t \in T, \gamma \in \Gamma^t$$

$$\{(i,j)\mid \gamma = (i,j)\}, \{(s,d)\mid (s,d) = l \in L, s = d\}$$

$$y_{(i,j)(s,d)}^t \leq x_{is}^t$$

$$y_{(i,j)(s,d)}^t \leq x_{js}^t$$

### 5.1.5 Heuristics

Due to the binary nature of variables and integer format of constraints, the mixed binary linear problem above is an NP-Hard problem (proof in the appendix). Since each VNF can be placed on $n$ servers, the solution space grows exponentially with respect to the number of servers in the SDI service chaining problem. Due to the tree topology of the graph, the paths between two VNFs in a chain can be uniquely determined. This tree feature keeps the complexity at the node selection level. $n^T$ possibilities for the ILP problem makes finding the optimal solution quite time-consuming and possibly unfeasible.

Based on the problem formulation we believed that considering a possibility of all servers for each VNF, and the possibility of all physical links for each virtual link were the main reasons for increasing the size of the problem. It was possible to benefit from the tree-shaped nature of the SDI to control the
size of the problem. Our goal was to abstract global branches as black-boxes (resource pool) for chains in a local branch. In this approach, each chain will use its branch if possible before borrowing resources from rest of the branches.

We propose a two-step heuristic process for solving the problem: Move DPIs to the Smart Edge to efficiently utilize vCPEs, and next find a right balance between inner and outer chain capacities for each branch to maximize the number of users per AP.

The DPI is on the upstream with minimal bandwidth passing through it. To avoid waste of resource on the vCPEs, it seems legitimate to offload the DPIs to the Smart EDGE. Placing DPIs on the vCPEs could drastically decrease the number of users benefiting the vCPEs. The placement of the Http-streamers and the caches is still a question.

We categorize the chains into two types: Inner chains and outer chains. Inner chains are the chains located in a branch to support the users from the same branch. With this definition, outer chains become the chains in a branch supporting users from other branches. A trade-off between inner and outer chains is apparent. The more that a branch supports inner chains, the less it can deploy outer chains. Now the balance between inner and outer chains per branch must be found.

With this categorization, users in a branch can have their chains assigned to use resources from the local branch, to be borrowed from global (other) branches, or even Smart Edge.

On the side of resources, a branch can either support global users using local resources or local users using global resources but not both at the same time since a branch either has excess or lack of resources (to lend to other branches or borrow resources from others).

The inner and outer chain view to the problem is valid since having HTTP-streamer and caches if placed in different branches, consumes networking resources in the SDI drastically.

To obtain a heuristic maximum number of the users, a similar set of figures as 5.4 and 5.5 has to be generated for each branch. These charts demonstrate the relationship between the number of users served per access point in a branch and the number of local users supported locally, global users supported locally, local users supported globally, and local users that cannot be supported.

To generate this graph (figure 5.4), we calculated the maximum number of global users supported locally for each number of users-per-AP. In the cases that the branch is supporting a few number of its users (local users supported locally), a high number of users from other branches can be supported (global users supported locally). On data 14, the branch reaches a point that it no longer can lend resources to other branches. It starts borrowing resources from global resources to support some of its local users (local users supported globally).

A VNF service chaining in SDI is valid where a number of total possible global users supported locally is higher than total borrowed chains (local users supported globally). This can be easily determined by combining all branch figures. Note that Smart Edge itself has a separate figure that only supports global users with local resources.

\[
\max_{\text{total users per AP}} \sum_{i \in \text{Branches, Smart Edge}} GL_i \\
\text{where} \\
GL_i \leq LG_i \\
GL_i \quad \text{global users supported locally}_i \\
LG_i \quad \text{local users supported globally}_i
\]
5.1.6 Related work

[50] by Gaspary et.al. divides the service chaining problem into three steps of placement, assignment, and chaining. This work solely considers reducing the number of VNFs on the infrastructure, and it does not address service chaining’s bandwidth consumption. [56] by De Turk et.al. considers a more general hybrid infrastructure with legacy hardware and cloud resources. It simply offloads work from
legacy to the cloud as the capacity of the legacy is reached. [53] by Karl et.al. considers traffic requests containing non-ordered VNFs. This work suggests a heuristic, selecting one of the possibilities, to deal with the exponential growth of possibilities for non-ordered chains. Maximizing remaining bandwidth for chain placement is one of the prominent problems that it considers. [39] by Wen et.al. takes the same objective and proposes inter and intra-chain congestion control as the cause of running out of remaining bandwidth. Their service chain selection algorithm maximizes the remaining bandwidth. Finally, [30] by Boutaba et.al. defines an ILP for the cost parameters in service chaining and takes an exhaustive approach to finding the minimum cost of service chaining in their network.

5.1.7 NP-Hardness Proof

Following is another formulation of the problem. By showing that the new formulation is an NP-Hard problem, we conclude that our original problem is also NP-Hard.

We intend to utilize the underlying SDI to maximally contain Kaleidoscope VNF-graphs. The total number of VNF-graphs can be translated to maximization of the total number of users served by APs. Assuming $ns_i, nr_i$ are the total number of users that can stream and replay in user group i, the objective would become:

$$\max (ns + nr)$$

$$ns = ns_i, nr = nr_i$$

Consider a user group $i$ with our specific VNF-graph shown in Fig.5.2. $ns_{ij}$ is the number of streams from $ns_i$ that have their DPIs located on node $j$. This would consume a bandwidth correlated with $ns_{ij}$ from node $i$ to node $j$, and a set of resources consumed on node $j$.

In a similar way $ns_{ijk}$ is the number of streamers from $ns_{ij}$ with their Http-Streamers on node $k$. An
additional term \(nr_{ijkh}\) is used for the replay’s cache component

\[
ns_i = \sum_i (ns_{ij}) ,
ns_{ij} = \sum_k (ns_{ijk})
\]

\[
nr_i = \sum_i (nr_{ij}) ,
nr_{ij} = \sum_k (nr_{ijk}) ,
nr_{ijk} = \sum_h n_{ijkh}
\]

\[(\alpha - \epsilon)ns_i \leq nr_i \leq (\alpha + \epsilon)ns_i\]

Due to the tree form of the topology there is a unique way to connect a node to another. \(W_{ij}\) represents paths used to connect node \(i\) to node \(j\). Now we can construct formulation of bandwidth usage in our case.

Consider \(\beta_s\) to be the bandwidth used for stream \(i\), \(B_{SUD}\) the bandwidth from user to the DPI, \(B_{SDH}\) the bandwidth from DPI to the Http-Streamer, \(B_{SHU}\) the bandwidth from Http-Streamer to the User-groups. \(\beta_s\) can be formulated as follows:

\[
\beta_s = \sum_j ns_{ij}W_{ij}B_{U,D} + \sum_j \sum_k ns_{ijk}(W_{jk}B_{D,H} + W_{ki}B_{H,U})
\]

\[
\beta_r = \sum_j nr_{ij}W_{ij}B_{U,D} + \sum_j \sum_k nr_{ijk}(W_{jk}B_{D,H} + W_{ki}B_{H,U})
\]

\[
+ \sum_j \sum_k \sum_h nr_{ijkh}W_{hk}B_{C,H}
\]

\[
Cs^t_q = \rho_D(ns_{iq}) + \sum_j \rho_H(ns_{ijq})
\]

\[
Cr^t_q = \rho_D(nr_{iq}) + \sum_j \rho_H(nr_{ijq}) + \sum_j \sum_k \rho_C(nr_{ijkq})
\]

With physical resource constraint of:

\[
\text{Phys res.} \quad \sum_i Cs^t_q + Cr^t_q \leq C_q
\]

\[
\text{Netw res.} \quad \sum_i \beta_s + \beta_r \leq B
\]

As the above problem is ILP (therefore NP-Hard) and equivalent with our original problem, the original problem is NP-Hard as well.
5.2 Fast Network Flow Resumption for Live Virtual Machine Migration on SDN

As discussed before, containers have not fully replaced VMs due to their security shortcomings. Use of VMs in service chaining applications is an active project. Despite containers which can be easily migrated from a host to another, VMs do not have the same privilege. The VMs carrying a large image of the operating system require a longer time to be migrated. In this last part of this chapter, live migration of VMs will be studied [78]. The goal is to evaluate a scheme that migrates networking resources of virtual machines to resume network flows rapidly on SDN.

Migration of VMs has become a vital part of today’s data center management. Some of these VMs have to be migrated while running (live migration). Pre-Copy and post-copy are the two popular methods of performing the live migration. In the pre-copy scheme, the memory pages get copied to the new host. Memory pages that get used after this copy process have to be re-copied. Next, the operation of VM gets halted, and the rest of the memory pages which were in use get transferred. After copying the full content, the VM sends an activation message to establish a connection back. In the Post-copy scheme, the services get interrupted immediately at the start of the migration process. The required content including in-use memory pages get copied to the destination and service gets started. After transferring the in-use part of the VM, the other pages get moved for a full migration. Lastly, the old VM gets terminated.

In the Pre-Copy procedure, service interruption happens well after copying major part of the memory, creating a large time window to establish a network connection to the new host. While in Post-Copy, Service interruption happens at the beginning of the procedure, making connection establishment a part of the service interruption. Hence fast network resumption becomes a critical bottleneck in the Post-Copy procedure. In this work, [78] an ILP problem was formulated, and its NP-completeness was proven. In this part of the chapter we briefly talk about the implementation of the work to reduce the service interruption time.

The goal was to measure the time saving that can be achieved using Janus compared to simple RYU controller. To test this VINO was used to create the required topology. Using VINO template, a topology shown in Figure. 5.6 was implemented. The content of Host one ($h_1$) is to be migrated to Host two ($h_2$). In the Post-Copy method, a path has to be established between $h_3$ and $h_1$. Using a basic RYU switch, establishing a path requires a packet to be sent from $h_3$ to $s_1$. Next, $s_1$ has to contact the controller to ask for a forwarding rule to be pushed to the switch. Then the packet traverses through the switch to the $s_3$ and the similar scheme goes on till the packet reaches $h_2$. This traversal using basic RYU requires contacting the RYU controller four times in this scenario. Also, the time that it takes for each switch to push the rule has to be considered in the total evaluation of the delay.

The following measurements were done to estimate possible savings that can be obtained using a single path setup feature of Janus. The average time for $h_3$ to ping $h_2$ while the path is setup was 89.8 ms. This number is the total time of traversing five links in both directions and getting forwarded by four switches in both directions.

\[ 5 \times T_{\text{traverse}} + 4 \times T_{\text{forward}} = 89.8/2(\text{ms}) \]
The time for $h_3$ to ping switch $s_3$, while no path was setup was also measured to be 58.2 (ms).

$$2 * T_{traverse} + T_{forward} + T_{controller} = 58.2/2 (ms)$$

The time for $h_3$ to ping switch $s_3$, with the path in place was measured to be 35.3 (ms).

$$2 * T_{traverse} + T_{forward} = 35.3/2 (ms)$$

The Forwarding to controller time according to the above measurements becomes 11 ms (forwarding and decision making).

Janus could provide significant savings by avoiding multiple requests to the controller that exists in the basic RYU scenario. The savings in this case (not considering Janus’ path evaluation) would be 44 ms (out of 89.8 ms).
Chapter 6

Future work and Conclusion

6.1 Conclusion

In this thesis, we presented benefits of combined use of NFV and SDN. A framework for Service chaining was implemented to enable the study of VNFs on SAVI SDI. I believe that Service chaining as a problem should be discussed in context. We studied service chaining in two of many aspects of SDI: Security and Content delivery.

Use of pure NFV solutions for security was investigated using the SAVI framework. A few bottlenecks of VNF placement on network traffic was discussed. Security as a Service was implemented using open source software on SAVI SDI, and a Security Module Manager was proposed to direct security modules and networking switches. The Security Module Manager implemented security on SDI benefiting from NFV for detection and SDN for mitigation.

A comparison between different Mitigation techniques was made, and the scalability of the Security Module Manager on blocking network level distributed attacks was challenged. A parallelism technique was proposed to address the issues of Security Module Manager.

In another use-case of the service chaining, the benefit of vCPEs and service chaining on an RTCD application was examined by a few use-cases and an algorithm to efficiently place, and chain VNFs for a stadium was proposed and evaluated.

6.2 Future work

For future work, there are three directions for improvement:

For IDS/IPS, a working platform should be built using the proposed problem. We developed a generic problem which could be considered under a specific topology to obtain a detailed solution. Whale’s query times should be significantly reduced.

To improve the performance of scalable security system, and IDS tree can be designed according to a set of objectives with respect to the constraints such as throughput and redundancy. Also, the approximation techniques shall be designed in a way to require less number of VMs to achieve allocation task. Other scalability directions can also be considered to expand on this work.

For VNF service chaining, allocation of physical vCPEs to branches could be an interesting extension. The problem of user maximization inherently is dependent on both physical resource allocation and
virtual resource chaining. In our work, we solely considered the chaining part. An iterative solution to find a trade-off between the presence of vCPEs and number of users shall be an interesting problem.

Finally, container service chaining implementation could be an exciting addition to the SAVI SDI. As containers are dominating the third tier of SAVI, having service chaining in the presence of containers will truly present benefits of end-to-end orchestration.
Bibliography


[73] P. Yasrebi, S. Bembey, H. Bannazadeh, and A. Leon-Garcia. Virtual network function service
chaining on savi sdi. In 1st EAI International Conference on Future access enablers of ubiquitous
and intelligent infrastructures (Fabulous), September 2015.

[74] Takeshi Yatagai, Takamasa Isohara, and Iwao Sasase. Detection of http-get flood attack based
on analysis of page access behavior. In 2007 IEEE Pacific Rim Conference on Communications,

[75] Qi Zhang, Sai Qian Zhang, Jieyu Lin, H. Bannazadeh, and A. Leon-Garcia. Kaleidoscope: Real-
time content delivery in software defined infrastructures. In Integrated Network Management (IM),

[76] S. Zhang, Q. Zhang, H. Bannazadeh, and A. Leon-Garcia. Routing algorithms for network func-
tion virtualization enabled multicast topology on sdn. Network and Service Management, IEEE

resumption for live virtual machine migration on sdn. In 2015 IEEE 23rd International Conference

network flow resumption for live virtual machine migration on sdn. In 2015 IEEE 23rd International