Smart and secure network softwarization

Abdelhadi Azzouni

THÈSE DE DOCTORAT DE
LA SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique de Paris

Présentée par

Abdelhadi AZZOUNI

Pour obtenir le grade de

DOCTEUR de la SORBONNE UNIVERSITE

Sujet de la thèse :

Smart and Secure Network Softwarization

Soutenue le 13 Avril 2018
devant le jury composé de :

M. Nadjib AIT SAADI	Rapporteur, Professeur - ESIEE Paris
M. Olivier FESTOR	Rapporteur, Professeur - Directeur de Télécom Nancy
M. Prosper CHEMOUIL	Examinateur, Directeur de recherche - Orange Labs
M. Raouf BOUTABA	Examinateur, Professeur - Université de Waterloo - Canada
M. Igor. M MORAES	Examinateur, Professeur - Universidade Federal Fluminense, Brésil
M. Rami LANGAR	Examinateur, Professeur - Université Paris-Est
M. Guy PUJOLLE	Directeur de thèse, Professeur - Sorbonne Université
Mme. T-M-Trang NGUYEN	Co-directrice de thèse, HDR - Sorbonne Université

Remerciements

Je souhaite remercier en premier lieu mon directeur de thèse, M. Guy PUJOLLE, Professeur des Universités et ma co-directrice de thèse, Mme. Thi-Mai-Trang NGUYEN, HDR-Sorbonne Université, pour m'avoir accueilli au sein de leur équipe. Je lui suis également reconnaissant pour le temps conséquent qu'ils m'ont accordé, leurs qualités pédagogiques et scientifiques, leur franchise et sa sympathie. J'ai beaucoup appris à leur côtés et je leur adresse ma gratitude pour tout cela.

Je tiens à remercier tous les membres de l'équipe PHARE pour leur aide, leur soutien et pour leur sympathie durant mes trois années de thèse. Les moments que j'ai passé avec les thésards de l'équipe resteront gravés à jamais dans ma mémoire.

Je tiens aussi à remercier Professeur Raouf BOUTABA et son équipe de recherche à l'université de Waterloo au Canada, avec qui j'ai passé une grande partie de ma thèse.

Enfin, je remercie toute ma famille, à commencer par mes parents qui m'ont continuellement soutenu durant ma thèse, je remercie également mes frères et sœurs ainsi que mes amis pour leurs encouragements.

Abstract

The recent trend toward Network Softwarization is driving an unprecedented techno-economic shift in the Telecom and ICT (Information and Communication Technologies) industries. By separating the hardware on which network functions/services run and the software that realizes and controls the network functions/services, Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are creating an open ecosystem that drastically reduces the cost of building networks and changes the way operators operate their networks. SDN and NFV paradigms add more flexibility and enable more control over networks, thus, related technologies are expected to dominate a large part of the networking market in the next few years (estimated at USD 3.68B in 2017 and forecasted by some to reach $54B by 2022 at a Compound Annual Growth Rate (CAGR) of 71.4% [1]).

However, one of the major operators' concerns about Network Softwarization is security. In this thesis, we have first designed and implemented a pentesting (penetration testing) framework for SDN controllers. We have proposed a set of algorithms to fingerprint a remote SDN controller without having direct connection to it. Using our framework, network operators can evaluate the security of their SDN deployments (including Opendaylight, Floodlight and Cisco Open SDN Controller) before putting them into production. Second, we have studied the Topology Discovery problem in SDN controllers and discovered major security (as well as performance) issues around the current de-facto OpenFlow Topology Discovery Protocol (OFDP). In order to fix these major issues, we have designed and implemented a new secure and efficient OpenFlow Topology Discovery Protocol (called sOFTDP). sOFTDP requires minimal changes to the OpenFlow switch design and is shown to be more secure than previous workarounds on traditional OFDP. Also, sOFTDP outperforms OFDP by several orders of magnitude which we confirmed by extensive experiments.

The second axis of our research in this thesis is smart management in softwarized networks. Inspired by the recent breakthroughs in machine learning techniques, notably, Deep Neural Networks (DNNs), we have built a traffic engineering engine for SDN

[1] This is a very optimistic forecast. A conservative one would still predict a double digit number http://www.reportsnreports.com/reports/166733-software-defined-networking-sdn-and-network-virtualization-market-global-advancements-business-models-technology-roadmap-forecasts-analysis-2012-2017.html

called NeuRoute, entirely based on DNNs. Current SDN/OpenFlow controllers use a default routing based on Dijkstra's algorithm for shortest paths, and provide APIs to develop custom routing applications. NeuRoute is a controller-agnostic dynamic routing framework that (i) predicts traffic matrix in real time, (ii) uses a neural network to learn traffic characteristics and (iii) generates forwarding rules accordingly to optimize the network throughput. NeuRoute is composed of two main components: NeuTM and NeuRoute-TRU. NeuTM is a traffic matrix (TM) prediction framework that uses Long-Short Term Memory (LSTM) Neural Network architecture to learn long-range traffic dependencies and characteristics then accurately predicts future TMs. NeuRoute-TRU is a path selection engine that computes optimal paths for traffic matrices predicted by NeuTM. NeuRoute-TRU achieves the same results as the most efficient dynamic routing heuristic but in much less execution time.

Keywords

Network Softwarization, Software Defined Networking, Network Function Virtualization, Virtual Networks, Virtualization, Cloud, Network Security, Traffic Engineerig, Routing, Traffic Matrix, Network Management, Machine Learning, Neural Networks.

Table of Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Summary

1.1 Context and Motivations

Computer networks are complex and can be very difficult to manage. In a typical network, one can find many kinds of equipment, ranging from forwarding elements such as routers and switches to middleboxes, which are equipments that perform a wide range of networking tasks, such as firewalls, network address translators (NATs), load balancers, intrusion detection/prevention systems, etc.

For the past few decades, network operators have been relying on a handful of equipment vendors that provide proprietary and vertically integrated hardware running complex, closed and proprietary control software. The software implements network protocols that undergo years of standardization and interoperability testing. Because of the lack of network programmability and flexible management interfaces, network administrators typically configure individual network devices adapting tedious and error-prone manual configuration methods. This mode of operation has slowed innovation, increased complexity, and inflated both the capital and operational costs of running a network.

The recent trend toward Network Softwarization is driving an unprecedented techno-economic shift in the Telecom and ICT (Information and Communication Technologies) industries. By separating the hardware on which network functions/services run and the software that realizes and controls the network functions/services, Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are creating an open ecosystem that drastically reduces the cost of building networks and changes the way operators manage their networks. SDN and NFV paradigms enable the design, deployment and management of networking services with much lower costs and higher flexibility than traditional networks. In particular, they are contributing to the deployment of 5G infrastructures, from high data rate fixed-mobile services to the Internet of Things. As a result, new value chains and service models are emerging, creating novel business models and significant socio-economic impact [39, 40].

SDN and NFV are two sides of the same trend toward network softwarization. SDN involves three principles: separation of the control logic (control plane) from packet forwarding (data plane), centralization of the control logic and programmability of the data plane through well defined control plane-data plane interfaces.

Unlike traditional networks where control is distributed and embedded into network devices (switches and routers), SDN logically centralizes the control plane in one entity called the SDN controller. The SDN controller runs on a single or cluster of servers, has a global view of the network, and translates high level operational policies into switch/flow level traffic management decisions. This separation allows employing much simpler forwarding hardware (generic switching equipment that is built using cheap merchant silicon) that provides much faster packet forwarding. OpenFlow [15] is the standard communications interface defined between the control and forwarding planes of an SDN. This programmability enables a great flexibility in network management, and leads to faster innovation in network traffic engineering, security and efficiency.

On the other hand, NFV softwarizes network functions (NFs) such as load balancers, firewalls and intrusion detection systems that were previously provided by special-purpose, generally closed and proprietary hardware. NFs will now be implemented by software that could run on virtual machines running on commodity hardware. They also can be provisioned as virtual NFs (VNFs) in a cloud service to leverage the economies of scale provided by cloud computing, and consequently, reducing network capital and operational expenditures [40].

While either SDN or NFV can be used by itself, the two technologies are complementary and there is big synergy in combining both of them. However, the new features brought by either SDN or NFV are also the source of new security challenges, and the combination of both technologies may increase the impact of the related security threats. For NFV and SDN to achieve widespread adoption by the industry, security remains a top concern and has been a hurdle to the adoption of network softwarization. Based on recent surveys, security is one of the biggest concerns impacting the broad adoption of SDN and NFV [9]. In this thesis, we discuss thoroughly the security concerns in softwarized environments and contribute by proposing concepts and designing tools at different security levels.

Another research question that we tackle in this thesis is How to leverage the centralized control in SDN to advance traffic routing, which is arguably the most fundamental networking task. For decades, innovation in routing did not get a fair attention from the industry for various historical, financial and practical reasons. However, the unprecedented growth in network traffic and application requirements witnessed by today's networks is driving a huge need for automation. Network operators are paying attention to innovative routing and traffic engineering solutions. We argue that the combination of network sofwarization and the recent breakthroughs in machine learning techniques offer an ideal framework for network automation. For example, thanks to new powerful deep learning techniques, operators are able to model very complex networks and find patterns in large amounts of network data, which offers great opportunities toward automation of network control.

1.2 Contributions

In this thesis we address two major challenges in network softwarization, namely, security and traffic engineering.

First, we present a thorough state of the art study of security in SDN and NFV followed by Three contributions aiming to define and fix major security issues. The first contribution addresses the security of SDN controllers. The second contribution exposes major security issues found in the de-facto OpenFlow Topology Discovery Protocol (OFDP) then proposes a secure and efficient Topology Discovery Protocol for SDN called sOFTDP. Finally, we contribute to the implementation of a secure, light weight orchestration system for virtual network functions called nf.io [1].

[1] In collaboration with the networking research group at David R. Cheriton School of Computer Science, University of Waterloo, Canada. Paper here: http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p361.pdf. Code here:

In the second part of this thesis, we combine state of the art machine learning techniques with the centralized control offered by SDN to build a Predictive Dynamic Routing System called NeuRoute. We build NeuRoute in two steps, with a separate contribution at each step. In the first contribution, we study the traffic matrix prediction problem and propose a novel machine learning-based framework (called NeuTM) to efficiently predict traffic matrix in real time. In the second contribution, we combine NeuTM with a route selection engine, that is entirely based on Deep Neural Networks, to optimize routing in SDN.

1.2.1 Fingerprinting OpenFlow controllers: The first step to attack an SDN control plane

As a first contribution, we demonstrate the feasibility of the fingerpirinting attack on SDN controllers. We propose techniques allowing an attacker placed in the data plane, which is supposed to be physically separate from the control plane, to detect which controller is managing the network. The fingerpirinting techniques are categorized into two classes: Timing Analysis based techniques and Packet Analysis based techniques. Timing Analysis based techniques use precise time measurements to infer key parameters of the controller. Packet Analysis based techniques allows an attackers to identify the controller by observing the content of LLDP packets broadcasted by the OpenFlow Topology Discovery Protocol during the topology discovery process. Note that this work has as primary goal to emphasize the necessity to highly secure the controller and provide a penetration testing framework for network administrators. It also led to a conference publication in the the Global Communication conference (Globecom) [151]

1.2.2 sOFTDP: Secure and Efficient OPenFlow Topology Discovery Protocol

As a second contribution, we demonstrate that the de-facto protocol for topology discovery in SDN, named OFDP (OpenFlow Topology Discovery Protocol), has serious security and performance problems. Then, we introduce a novel protocol that we call sOFTDP to replace OFDP. sOFTDP is more secure and outperforms OFDP by several orders of magnitude, which we confirm with extensive evaluation. Note that this contribution led to two conference publications: one in the the Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net) and the second in the IEEE/IFIP Network Operations and Management Symposium (IEEE/IFIP NOMS).

1.2.3 NeuTM: A Neural Network-based Framework for Traffic Matrix Prediction in SDN

As a third contribution, we study the traffic matrix prediction problem and propose a novel framework (called NeuTM) based on Long Short Term Memory Neural Networks, which is a

https://github.com/abdelhadi-azouni/nf.io

machine learning technique proven to be very powerful in predicting time series, and has been successfully utilized in many areas including video frame prediction and speech recognition. The goal of NeuTM is to precisely estimate historic traffic matrices in an SDN then accurately predict the future one. The benefits of traffic matrix prediction range from security to traffic engineering. For example, to detect DDoS attacks in their early stage, it is necessary to be able to detect high-volume traffic clusters in real-time, which is not possible relying only on current monitoring tools. We have implemented NeuTM as an application on top of an SDN controller. Through extensive testing, we show that NeuTM achieves outstanding prediction results compared to previous methods. Note that this contribution is the object of a conference publication in the IEEE/IFIP Network Operations and Management Symposium (IEEE/IFIP NOMS).

1.2.4 NeuRoute: Predictive Dynamic Routing for Software-Defined Networks

As a fourth contribution, we combine NeuTM (previous subsection) with a route selection engine, that is entirely based on Deep Neural Networks, to optimize routing in SDN. The final routing system is called NeuRoute. So far, the common practice to ensure a good QoS is to over-provision network resources. Operators over-provision the network so that capacity is based on peak traffic load estimates. Although this approach is simple for networks with predictable peak loads, it is not economically justified in the long-term. NeuRoute predicts traffic matrix in real time and optimizes routes for the predicted future traffic matrix. We achieve this by "teaching" routing to the system using a dynamic routing heuristic and real network data. Finally, experiments show that NeuRoute outperforms the most efficient dynamic routing heuristic by four orders of magnitude on real network data. This contribution is the object of a conference publication in the International Conference on Network and Service Management (CNSM).

1.3 Outline

The rest of this thesis is organized as follows. The second part introduces the network softwarization paradigm. The third part addresses security issues in network softwarization. The fourth part addresses routing and traffic engineering in SDN. The fifth part concludes this thesis and draws a plan for future work.

Part II

Overview of Network Softwarization

Chapter 2

Overview of Network Softwarization

Summary

In this chapter, we give an overview of network softwarization paradigms including Software Defined Networking (SDN) and Network Function Virtualization (NFV) and a brief history of these technologies. We also discuss the integration of NFV and SDN in the same network. Finally, we present some use cases of network softwarization technologies.

2.1 Introduction

In modern days, with the increasing diversity and data rates from users, Telecommunications Service Providers (TSPs) must correspondingly and continuously purchase, store and operate new physical equipment, and manually manage the complex network with very limited tools. It leads to high expenditure and operation costs. Network softwarization technologies including SDN and NFV [41, 42] were proposed as new networking paradigms allowing the design, deployment and management of networking services with much lower costs. SDN and NFV decouples physical network equipments from the functions that run on them. For example, a commodity server can be used to run multiple Virtual Network Functions (VNFs) on it, and SDN offers an abstraction to speed up innovation in the networks through the separation between the data plane and the control plane. Network softwarization improves the operating performance and operational efficiency.

2.2 An Overview of Network Softwarization

SDN is an architecture that decouples the control logic from the packet forwarding hardware for a dynamic, manageable, cost-effective, and adaptable network system. To promote the adoption of SDN through open standards development, the Open Networking Foundation (ONF) [43] was formed in 2011. As defined by ONF, the high-level architecture of SDN consists of three planes: data plane, control plane, and application plane [44]. SDN brings a great flexibility in network programming which accelerates the rate of innovation in network architectures and operations.

On the other hand, NFV utilizes virtualization technologies to provide network functions (NFs) through running software on standard commodity servers [45]. For example, a virtualized intrusion detection system can be set up to protect the network security without deploying dedicated physical units. Compared to traditional network architectures, NFV greatly reduces the equipment expenditure through using common-purpose commodity servers for NFs. Furthermore, NFV enables flexible network function deployment and dynamic operation [46, 47].

While either SDN or NFV can be used by itself, there is synergy in combining the two technologies. While NFV replaces specific types of networking gear such as routers, switches, firewalls, and load balancers with virtual ones, SDN offers an abstraction of the forwarding elements (switches, routers) and standard interfaces between these elements and management applications. SDN is thus considered to be a natural platform to enable NFV. In this thesis, we use the term "Network Softwarization" to denote the use of either or both SDN and NFV technologies in building networks. In addition to being cost effective solution, the move towards network softwarization will enable more systematic, industrial and abstracted methods to manage large networks and to meet quality of service requirements.

In the following sections, we present a brief history of Network Softwarization, some detail about SDN and NFV, and a few use cases of network softwarization.

2.2.1 A Brief History Of Network Softwarization

Network softwarization was inspired by the early work on programmable networks[48] started in the middle of 1990's. The programmable networks promote the separation of hardware and control software. The Open Signaling (OPENSIG) [49] and the Active Networking [50], [51] were two leading projects on programmable networks [52]. The OPENSIG project targeted on creating more open, extensible, and programmable ATM and mobile networks [53], while the Active Networking [54], [55] aimed at building programmable networks to encourage greater innovations [56]. However, both projects encountered security problems since they ran the user code at the infrastructure level. This blocked the further researches and developments in these areas [57].

After the programmable networks, separating the data plane and the control plane attracted more and more attention. More effort was directed to build open and standard interface between the control plane and the data plane [58], [59], [60], [61], [62]. ForCES [60] was a framework to separate the forwarding element and the control element. RCP [61] and PCE [62] were developed to provide a logically centralized control over the network. These works had the constraints of scalability and compatibility, which hampered their further development and deployment. The 4D [63], SANE [64], and Ethane [65] were designed later to fully separate control plane and data plane furthermore. In particular, the switch design in Ethane became the basis of the original OpenFlow API [66], which directly led to the arrival of SDN.

A Stanford University research group first introduced the concept of OpenFlow and used it in the campus networks [15]. OpenFlow provided a programming interface for programming the network elements, which was later standardized by ONF [66], [67]. Soon after many controllers designs appeared such as NOX [68], Onix [69], ONOS [70], Cisco IOS [71], Junos OS [72], ExtremeXOS [73], and Service Router Operating System (SR OS) [74]. These designs led to more convenient deployment, management and control of network protocols and applications [75].

The Network Function Virtualization (NFV) is another dimension of network softwarization, which uses software-based network components to replace the traditional dedicated hardware. The term was proposed for the first time in June 2012, followed by the publication of the first NFV white paper [45] in October 2012 by the Network Functions Virtualization Industry Specification Group (NFV ISG) of the European Telecommunications Standards Institute (ETSI) [76]. The organization soon released the second [77] in 2013 and the third NFV white paper [78] in 2014. Nowadays the membership of ISG NFV has grown to over 270 individual companies including 38 of the world's major service providers from both telecoms and IT vendors [76].

2.2.2 A Brief Overview of SDN

Software-Defined Networking has emerged as an efficient, flexible and cost effective network paradigm, capable of supporting the dynamic nature of today's network applications while

lowering operating costs through simplified hardware and improving management through centralized control. SDN focuses on three key features:

- Decoupling the control plane from the data plane (the underlying network),
- Centralization (one physical machine or a cluster of physical machines) of the controller, and
- Programmability of the network and enabling third party applications to operate on the network through management APIs.

SDN architecture: As shown in Figure 2.1, the SDN architecture design consists of three planes:

- *Data Plane*: It comprises physical switches and virtual switches that act as the forwarding elements.
- *Control Plane*: It includes different software-based SDN controllers. These SDN controllers are able to supervise and control the network behaviors. The controllers communicate through three interfaces: the south-bound interface, the north-bound interface, and sometimes the east/west-bound interface [79].
- *Application Plane*: It consists of SDN applications for users, such as network virtualization, server load balancing, and security application [80].

The control plane manages the underlying network through the control channel using a SDN control protocol. As stated before, OpenFlow is the first standardized open protocol to manage communication between the control plane and the data plane. The controller implements network stack abstraction and offer it as an API so that developers can build management applications on top of it.

The logical centralization of control offers the following benefits: First, modifying network policies through software is simpler and less error-prone than via low-level device configurations. Second, the centralization of control logic with global knowledge of the network state simplifies the development of more sophisticated network applications. This ability to dynamically program the network to control the underlying data plane is the crucial value of SDN.

Flow handling: The control/forwarding layers separation (figure 2.1) reduces switches to basic packet forwarding devices containing flow tables populated with localized flow rules. These rules are managed by the control plane to describe how incoming packets will be handled based on matching fields (such as packet header content, incoming port, etc.). When a corresponding flow rule is not found in the flow tables, the switch requests the control plane to compute a routing pathfor the new flow and send back corresponding flow rule updates. The communication between control plane and data plane is done via a southbound interface such as OpenFlow protocol. Note that openFlow messages must be exchanged through a secured channel. To exchange with the controller, the switch needs to implement a separate OpenFlow module [66]. On the other side, the controller provides an abstraction of the network stack

and the underlying instrument as a network API (Northbound API). Hence, developers are able to create third-party business applications on top of it.

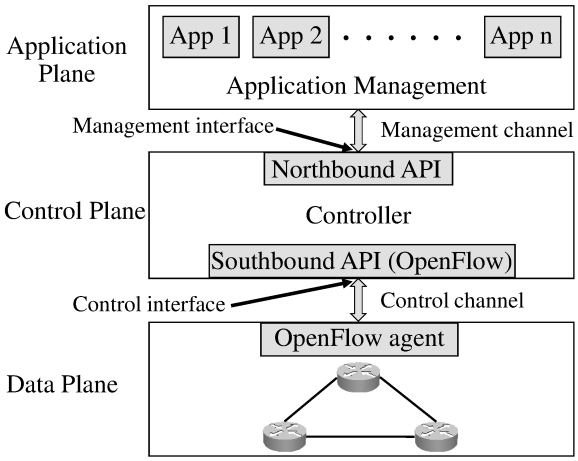


Figure 2.1: SDN architecture and interfaces.

OpenFlow protocol: The controller can add, update, and delete flow entries, both proactively and reactively (in response to packets) through exchanging OpenFlow messages with switches. An OpenFlow message is either a switch-to-controller message or controller-to-switch message [66]. Some typical OpenFlow messages are listed as follows:

- *Hello messages*: typically exchanged between the controller and the switch when the connection is first established.
- *Echo request/reply messages*: used to exchange information about latency, bandwidth and liveness. Echo request timeout indicates disconnection.
- *Packet-In messages*: used by the switch to send a packet to the controller when it has no flow-table matching the packet.
- *Packet-Out messages*: used by the controller to inject packets into the data plane of a particular switch.
- *Flow-mod messages*: used by the controller to modify the state of an OpenFlow switch.
- *Stats request messages*: used by the controller to request information about individual flows.

2.2.3 A Brief Overview of NFV

The emerging of Network Function Virtualization (NFV) has gained much attention from both the academia and industry due to its potential of cost reduction and operational efficiency.

The main idea of NFV is to replace dedicated network appliances, such as routers and firewalls, with software running on commercial off-the-shelf servers.

NFV architecture: As shown in Figure 2.2, the NFV Architecture is composed of three key elements [43]:

- *Network Function Virtualization Infrastructure (NFVI)*: NFVI is composed of the commercial-off-the-shelf hardware and the abstractions of the computing, storage and network resources. The abstraction is achieved through a virtualization layer based on hyperviser, which decouples the virtual resources from the underlying physical resources.
- *Virtual Network Functions and Services (VNF)*: A VNF is a virtualized functional block within a network infrastructure that has well defined external interfaces and well-defined functional behavior. Examples of VNFs include virtualized Residential Gateway, virtualized firewall, and virtualized load balancer. VNFs can be realized through virtual machines.
- *NFV Management and Orchestration (NFV MANO)*: NFV MANO performs the orchestration and lifecycle management of NFVI resources and VNFs [81]. It provides the functionality required for the operations of the VNFs such as the configuration of the VNFs and the infrastructure these functions run on. It covers three functional blocks: NFV Orchestrator, VNF Managers, and Virtualized Infrastructure Manager.

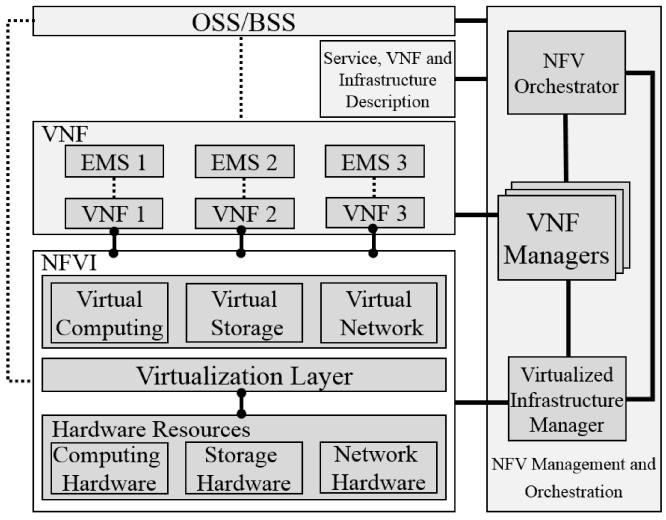


Figure 2.2: NFV architectural framework (Adapted from [81]).

NFV implementation: The implementation of NFV faces a few major challenges. For example, how to manage and orchestrate all virtual resources? how to integrate the virtual resources so they are compatible with existing platforms? and how to ensure a secure, resilient, and available platform? The implementation requirements are addressed by the NFV virtualization requirements document [82]. To guarantee the service availability and maintain

resiliency in NFV, automated recovery from failures should be enabled [83]. Some related open source projects have made contributions to the growth of NFV, for example, OpenDaylight [84] and OpenStack [85, 86]. In September 2014, Open Platform for NFV (OPNFV) was created to develop open source projects to overcome the implementation challenges faced by NFV [87].

2.2.4 The Integration of NFV and SDN

NFV and SDN are highly complementary in building software-based solutions for more scalable, agile, and innovative networks, but they are different from each other. For example, SDN enables automated operations, flexible policy control and effective security management on all NFV resources, while NFV generates reliability and elasticity in SDN by implementing SDN controller as a VNF running on a virtual machine. Greater benefits can be achieved by combining NFV and SDN [88, 89].

NFV and SDN have a common goal to make the networks more flexible, dynamic and secure through Softwarization. NFV is able to manage dynamic software-based network functions and SDN can coordinate network flows among VNFs. The integration of NFV and SDN is a ongoing research topic [90, 91]. For example, Omnes et al. [92] proposed a multi-layered architecture (Figure 2.3) integrating both NFV and SDN technologies to overcome the security challenges faced by the Internet of Things (IoT). A modular-based NFV architecture allowing policy-based management and orchestration for VNFs is presented in [93] to provide secure NFs in SDN. The designed architecture combines NFV and SDN, aiming at deploying diverse VNFs in SDN flexibly and guarantee the security of network services.

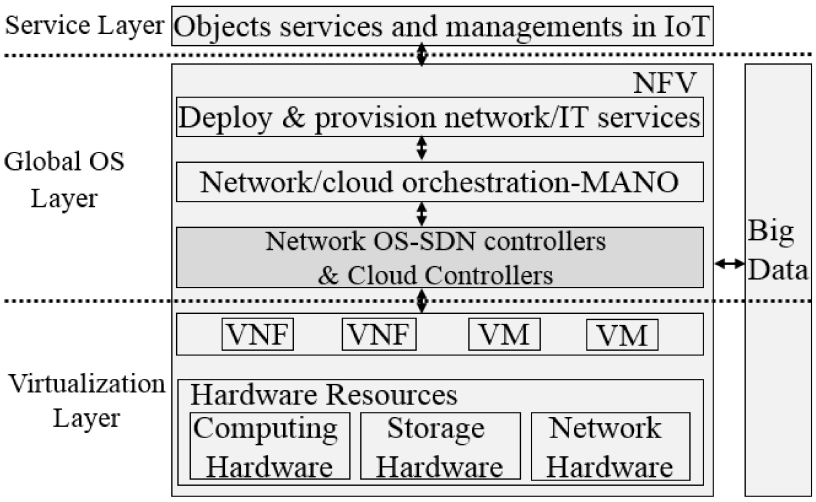


Figure 2.3: Combined NFV and SDN architecture (Adapted from [92]).

2.2.5 Use Cases of Network Softwarization

There are many use cases for Network softwarization technologies. In this subsection, we list a few that are commonly cited in the literature.

Enterprise Networks: Enterprise networks are usually in large scale and have strong demands for security and performances [65]. SDN is able to satisfy these requirements by programmatically implementing network policies, monitoring network behaviors and adjusting network performances, while NFV can bring down the cost of enforcing security and keeping up the performance. For example, An enterprise network can use SDN for network flow handling and build network function such as switches, firewall, IDS upon virtualized infrastructure.

Data Centers Managing traffics and enforcing policies are very important in large scale data centers since a service disruption could result in a huge profit loss. Google applied the concept of SDN into data centers and designed B4, a software defined wide area network (WAN) to connect Google's world-wide data centers [94], [95]. As an OpenFlow enabled SDN architecture, B4 [95] controls different switches and splits application flows among various paths to balance the network capacity. CloudNFV [96] is an open project that implements both NFV and SDN technologies into the framework. The project was launched in 2013 and its goal was to make every application that can run in the cloud into a potential virtual function [97].

Home Networks The current home networking model includes relatively low-cost, error-prone devices and becomes increasingly complicated by involving different network operators and services. Security and privacy become an issue in such environment. SDN and NFV technologies offer a solution to the problem, such as moving the home gateway to the cloud [98]. Many home virtualization solutions have been proposed [99, 100, 101]. Fig. 2.4 illustrates the virtualization of services and functions in home networks. However, virtualization of home networks moves the responsibility from end users to the operator, thus generating new security issues to network service providers. How to ensure safety and secure communication among home NFV components remains an open problem [99].

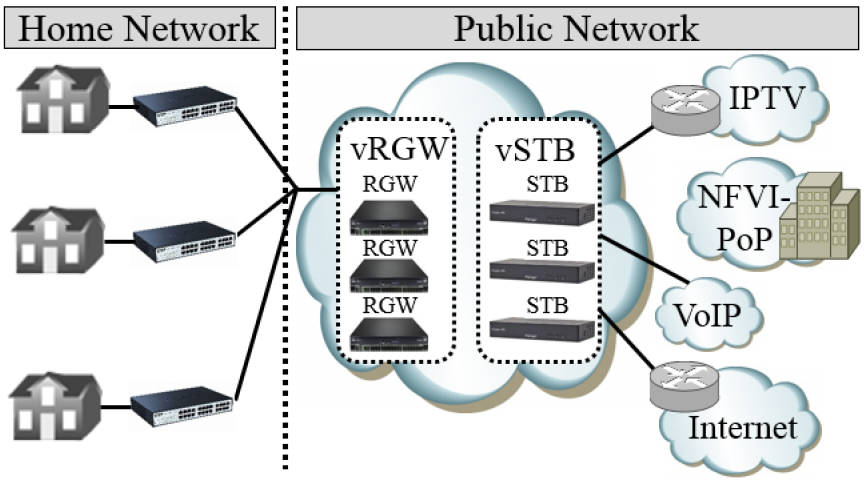


Figure 2.4: Network functions virtualization of home environment (Adapted from [102]).

Mobile Core Networks Nowadays, mobile core networks are equipped with a huge variety of expensive and proprietary hardware appliances. The introduction of NFV is expected to significantly reduce the network complexity and improve operational efficiency in mobile core networks by using IT virtualization technology. Evolved Packet Core (EPC) is the newest core network architecture for a cellular system, it includes NFs like mobility management entity (MME), serving gateway (SGW), and packet data network gateway (PGW). Fig. 2.5 depicts the virtualization of EPC. Many research and industry projects has been conducted to use network softwarization techniques for mobile core networks [103, 104]. For example, [105] proposes an information-centric architecture to integrate wireless network virtualization with information-centric networking (ICN) in 5G mobile wireless networks, and Cloud4NFV platform [104] is designed to manage service functions in a telco cloud environment.

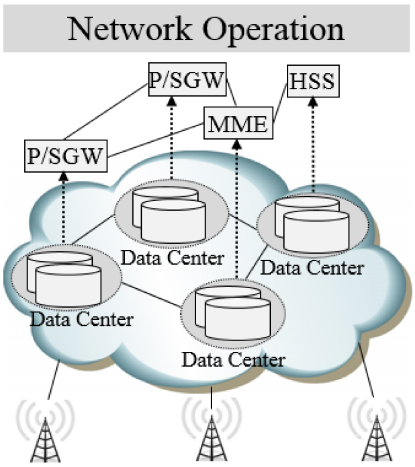


Figure 2.5: Network functions virtualization of EPC (Adapted from [102]).

Part III

Security in Softwarized Networks

Chapter 3

Security in Softwarized Networks: State of the Art

Summary

As an emerging network technologies, network softwarization encounters many security threats from both SDN and NFV sides. In this chapter, we focus on the security issues in network softwarization. More specifically, we provide detailed information about and categorization of security concerns and solutions in softwarized networks including SDN and NFV. We particularly highlight the overlap between the two paradigms. We also introduce the latest development trends and research work in network softwarization security and provide methods and guidance on how to assure their security. We overview the existing security platforms for network softwarization platforms that have been implemented and used in practice.

3.1 Introduction

Although network softwarization technologies reduce costs, offer great flexibility in network management, and accelerate innovation, they also bring new potential security threats [106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120]. The centralized nature of SDN controllers, the shared resources of the virtualized infrastructure and the multivendor software stack in a softwarized environment open a new set of attack points and security challenges for network operators. For example, a new vulnerability of SDN would be the fact that the SDN centralized controller could be a single point of failure. Controller scalability issues, resilience to failure and how to maintain the network availability when the controllers happen to fail should be seriously taken into account. More importantly, security problems including Distributed Denial of Service (DDoS) attacks, forged or faked traffic, unauthorized access and so on, remain to be overcome.

NFV also brings various security concerns. For example, components in NFV architectural frameworks, such as hypervisors and orchestrators, may be vulnerable to potential security threats. The shared storage and networking may introduce new security vulnerabilities. Application programming interfaces (APIs) are used to support programmable orchestration in NFV creating additional security risks to virtual network functions. Furthermore, hypervisors, hardware and VNFs are likely to be offered by different vendors, thus resulting in integration complexity and generating security loop-holes.

As identified by the European Telecommunications Standards Institute (ETSI)-NFV [10], the investigation into security for NFV/SDN covers several different domains including:

- Topology validation and enforcement
- Availability of management support infrastructure
- Secured boot
- Attestation
- Secure crash
- Performance isolation
- User/Tenant authentication, authorization and accounting (AAA)
- Authenticated time service
- Private keys within cloned images
- Back-doors via virtualized test and monitoring functions
- Multi-Administrator isolation
- Security monitoring
- Regulatory (lawful intercept and retained data)

- Privacy

On the solution side, various works [126, 127, 128, 134, 135] have emerged to address the security concerns in network softwarization and pointed out possible solutions. ONF proposed a set of security principles [107] that provide criteria and instructions for designing and developing ONF specifications for OpenFlow networks. NFV ISG also provided guidances [124, 107] to ensure security in NFV's external operational environment and related technologies for security and trust in NFV. In the rest of this chapter we will give a systematic overview and categorization of existing works on the security issues on network softwarization.

3.2 Security Challenges in Network Softwarization: State of the Art

In addition to the threats specific to SDN or NFV, the combination of the two technologies in one single network also increases the security risks and open doors to more security attacks [121], [122]. Considering a SDN network with virtual OpenFlow switches (vSwitches are considered VNFs and run on top of a hypervisor), a flooding attack on a vSwitch exhausts its resources much easily compared to a physical switch. Another example is when a SDN controller is running as a VNF on top of a hypervisor, in this case an attack from a compromised application can spread to other co-existing controllers leveraging eventual weaknesses in the hypervisor.

In this section, we present the state of the art of security challenges in network softwarization technologies including SDN and NFV, classified by the nature of addressed issues, and we highlight eventual overlaps that might amplify the impact of security threats.

3.2.1 Lack of Authenticated/Authorized Access

Underlying areas of concern in network softwarization security include user/tenant authentication and authorization. In SDN, controllers run many applications that need to access the network resources. Both authentication and authorization of the applications is required to ensure that only trusted applications are connected to the network. Also, because of the logically centralized control in SDN and the possibility to run multiple controllers to manage the same underlying network, it is possible that a fake controller gains access to the network resources and compromise a part of or the entire network [123]. Therefore, there is a strong demand for authentication and authorization of applications and controllers in SDN.

On the other hand, the introduction of NFV brings new security issues when it comes to Authentication and authorization [124] [125]. Unauthorized access threats are expected to increase when implementing network functions in a virtualized environment. That is, in addition to being target to traditional unauthorized access threats that affect VMs (or containers), VNF-specific aspects like chaining and placement requirements may bring additional severity to this kind of threats.

3.2.2 Data Leakage

Timing-analysis based attacks can be used to get important information from the SDN components [127]. [126] demonstrates the possibility that an attacker discovers a remote SDN network and hence launch some SDN-specific attacks on it. An inference attack model for flow table capacity and usage is described in [128].

Data leakages impact is amplified in SDN+NFV environments. All traditional cross-VM side channel attacks are also applicable on VNFs. Furthermore, In a virtualized SDN environment, multiple OpenFlow vSwitches are instantiated on top of an OpenFlow capable switch hypervisor (e.g., Openvswitch) [129]. Assume that different vSwitches are assigned to different tenants, in this case, if the logical networks and their associated credentials (e.g., keys, certificates) are not securely isolated or containerized, this can lead to data leakages that can compromise the functionality of SDN vSwitches and affect the network functionality. This scenario is more disastrous in the case of co-existing virtual controllers on top of a vulnerable hypervisor.

3.2.3 Data Modification

In a softwarized environment, data modification attacks often come after data leakage attacks by exploiting vulnerabilities in the virtualization infrastructure or the lack of access control in SDN controllers. SDN has a split-plane architecture, and security threats arise when intermediate components are designed between the control plane and data plane [130], [131], [132]. For example, FlowVisor [130] is proposed as a hypervisor for OpenFlow, but it can not provide an appropriate isolation mechanism. Therefore, an attacker has the chance to modify the data in the network. Also, a vulnerable control channel is subject to Man-In-The-Middle attacks that affects the consistency of control-switch communications. Knowing that the use of TLS to secure the control channel is optional in the OpenFlow switch specification [66], this problem seems to be not seriously addressed yet, and thus, the data modification is a security challenge to be overcome in SDN.

Similarly, access to the virtualization infrastructure or to VNFs by an attacker, generally results in data modification that affects the compromised VNF, the whole VNF chain or the whole infrastructure. In the case of VNF chains controlled within SDN network, data modification on SDN elements (vSwitches or controllers running as VNFs) can be more disastrous [133]. For example, changing forwarding data in vSwitches can easily affect the whole chains of VNFs.

3.2.4 Malicious Applications

Both SDN and NFV have brought the idea of running third party applications on top of the networking infrastructure, which makes them both be the target of attacks from malicious applications. A malicious application can attack the SDN controller and affect the entire

network. A flawed third party application also can bring vulnerabilities and security flaws to the entire system. On the other hand, a malicious VNF (e.g., a malicious vSwitch instance or a malicious middlebox installed by a tenant) can affect the security of co-exsiting vSwitches and middleboxes [136].

3.2.5 Denial of Service Attacks

Denial of Service (DoS) attacks and Distributed Denial of Service (DDoS) attacks are the major security concerns in SDN. This is due to its nature of centralized control plane. A successful DoS attack can prevent legitimate users from getting access to the network resources and services. DoS attacks and DDoS attacks can cause detrimental effects on SDN controllers [137]. For example, an attacker can assign random headers to IP packets and keep transmitting these IP packets in order to drive the SDN controller out of normal working state [138]. DOS attacks are also considered to be the most common threat on virtual environments and thus VNFs, exploiting many hypervisor platforms and ranging from flooding a network with traffic to sophisticated leveraging of a host's own resources [139]. In NFV+SDN environments, the limited computing power of OpenFlow vSwitches makes the DOS attacks more effective since it exhausts the vSwitch's resources much easier compared to a physical switch.

3.2.6 Compromised Infrastructure

As a software-based system the components in SDN and NFV may contain vulnerabilities which may be exploited by attackers. The results may be the compromised or malfunctioning components in the network. For example, the controller or switches [140] in a SDN may be compromised, which may cause partial or complete network dysfunctional. How to detect and isolate compromised network components can be a challenge for SDN.

On the NFV side, compute domain, hypervisor domain, and network domain constitute the NFV infrastructure (NFVI). The compute domain includes the bare-bone servers and storage, the hypervisor domain moves the resources from the hardware to the virtual machines, and the network domain manages the VNFs. NFVI suffers from both internal and external security threats. Internal threats result from inappropriate operations of people and it can be avoided by following strict operational procedures. External threats exist due to the design or implementation vulnerabilities. To solve this problem, NFVI should adopt standard security mechanisms for authentication, authorization, encryption and validation [141], [142], [143].

Table 3.1 summerizes mentionned attacks, classified by nature, on both SDN and NFV and highlights the eventual overlaps. The symbol ☑means that the attack originated from either NFV or SDN can affect (and the impact may be amplified) in case both technologies are used together. Example, VNF instance hijacking attack is more disastrous when the hijacked VNF is actually a controller VM for example. Data plane vSwitches are also concerned by this attack since they are considered as VNFs.

Table 3.1: Security threats on SDN+NFV environments

Security threats	**SDN**			**NFV**	
	Application layer	Control layer	Data plane	NFVI	VNF
Unauthorized access					
Unauthorized application	✓	✓	✓		
Controller hijacking		✓	✓		
VNF instance hijacking		☑	☑		✓
Unauthorized access to NFVI		☑	☑	✓	✓
Data leakage					
Inference attacks			✓		
Credential theft		☑	☑	✓	✓
Data modification					
Man In The Middle		✓	✓		
Malicious applications					
Malicious rule injection	✓	✓	✓		
DOS					
DOS on Controller		✓			
DOS on SDN swicth			✓		
DOS on VNFs (including vSwitches)		☑	☑		✓

3.3 Conclusion

Network softwarization technologies including SDN and NFV reduce network equipment costs and improves operational efficiency. However, For NFV and SDN to achieve widespread adoption by the industry, security remains a top concern and has been a hurdle to the adoption of network softwarization. Based on recent surveys, security is one of the biggest concerns impacting the broad adoption of SDN/NFV [9]. In this chapter, we highlighted security concerns in network softwarization technologies. We summarized the security challenges in SDN and NFV and provided existing security solutions from the literature to address those security concerns. Some proposed security architectures for SDN and NFV are presented. Finally we discussed future challenges that remain unsolved in software-based networks.

Chapter 4

Fingerprinting OpenFlow Controllers

Summary

SDN controllers are considered as Network Operating Systems (NOSs) and often viewed as a single point of failure. Detecting which SDN controller is managing a target network is a big step for an attacker to launch specific/effective attacks against it. In this chapter, we demonstrate the feasibility of fingerpirinting SDN controllers. We propose techniques allowing an attacker placed in the data plane, which is supposed to be physically separate from the control plane, to detect which controller is managing the network. Our primary goal is to emphasize the necessity to highly secure the controller. We focus on OpenFlow-based SDN networks since OpenFlow is currently the most deployed SDN technology by hardware and software vendors.

4.1 Introduction and Motivation

The centralized control provided by SDN is expected to facilitate the deployment and hardening of network security [137, 123]. However, SDN controllers can be subject to new threats compared to conventional network architectures. For example, an attacker can change the whole underpinning of the network traffic behavior by modifying the controller. The Open Networking Foundation (ONF) identifies a number of SDN security issues that the community must address [144]:

- The centralized controller emerges as a potential single point of attack that must be protected.
- The southbound interface between the controller and underlying networking devices (OpenFlow) is vulnerable to threats that could degrade the availability, performance, and integrity of the network. Using TLS or UDP/DTLS is recommended to secure the OpenFlow channel.
- The underlying network must be capable of enduring occasional periods where the SDN controller is unavailable.

In the most common schemes of attacking a remote system, the first step is to determine the set of possible attacks by collecting information about the target. In this chapter, we demonstrate some techniques that allow an attacker to fingerprint the OpenFlow controller of the network. Once the attacker knows which controller is used, he/she can launch tailored attacks exploiting its known vulnerabilities. We study the common case where the attacker is placed in the underlying network managed by the target SDN controller, and does not have access to either the controller or the control channel.

This work aims to demonstrate the feasibility of fingerpirinting OpenFlow controllers, with the ultimate goal of building a Penetration Testing framework that can be used by network administrators to test their SDN networks. Many frameworks have been created for the same purpose in traditional networks, NMAP [145], for instance, is a widely used scanner that can fingerprint remote systems among other capabilities. OWASP Zed Attack Proxy Project (ZAP) [146] is another security testing framework that includes fingerprinting remote web servers and web applications. As in NMAP and ZAP, our proposed techniques are not to be used separately, that is, one may get non-accurate results when only using the first Timing-Analysis based technique (section 4.4.1.1) for example, but the combination of all proposed techniques, generally gives accurate results. The key contributions in this chapter are as follows:

- We demonstrate the feasibility of fingerpirinting attack on OpenFlow controllers by designing, implementing and testing several fingerpirinting techniques and
- We highlight the need for building a Penetration Testing framework for SDN networks.

This chapter is organized as follows. Section 4.2 provides OpenFlow background information. Related works are discussed in section 4.3. Our proposed fingerpirinting techniques are

presented in section 4.4, our experimental testbed is described in section 4.5 and the results are given in section 4.6. Section 4.7 concludes the chapter and discusses some future directions.

4.2 Background Information

We recall main OpenFlow messages. An OpenFlow message is either a switch-to-controller or a controller-to-switch message. OpenFlow messages are detailed in [149] of which the most important ones are:

- Hello messages: exchanged between the controller and the switch when the connection is first established.
- Echo request/reply messages: used to exchange information about latency, bandwidth and liveness.
- Packet-In messages: used by the switch to send a packet to the controller when it has no flow-table matching the packet.
- Packet-Out messages: used by the controller to inject packets into the data plane of a particular switch.
- Flow-mod messages: used by the controller to modify the state of an OpenFlow switch.
- Stats request messages: used by the controller to request information about individual flows.

4.3 Related Work

S. Shi and G. Gun developed SDN Scanner [126] which exploits the network header field change. If a client sends packets to an SDN network, this client will observe different response times, because the flow setup time can be added in the case of non-matching flow (i.e., there is no corresponding flow rule in the data plane: response time $T1$) compared to the case when the corresponding flow rule exists (response time $T2$). SDN Scanner collects the response times then uses statistical tests to compare them. Thus, if an attacker can clearly differentiate $T1$ from $T2$ then he/she can detect the SDN network (the presence of an SDN controller). The evaluations conducted in the paper showed that SDN Scanner can fingerprint 24 networks out of 28 (i.e., a fingerprinting rate of 85.7%). However, SDN Scanner does not detect the controller type. In addition, collecting accurate values of $T1$ and $T2$ is extremely hard in real-world WANs because of the many variables that affect the response time. As such this method may not be efficient in WANs. [150] leverages information from the RTT and packet-pair dispersion to fingerprint controller-switch interactions (i.e. whether an interaction between the controller and the switches has been triggered by a given packet) in a remote SDN network.

L. Junyuan et. al [128] propose techniques to infer key network parameters like flow table capacity and flow table usage. For example, when the flow table is full, extra interactions between controller and switch are needed to remove some of the existing flow entries to make room for new ones, which may result in a performance decrease of the network. An attacker can take advantage of the perceived performance change to launch more effective attacks. More specifically, knowing the flow table size and usage, the attacker can estimate with high accuracy how many packets he/she needs to generate per second to flood the flow table and the required time to fill it up. hence, he/she could choose and correctly configure their attacking tools. Contrary to [128], our methods aim to infer control plane parameters to fingerprint controllers, which is more critical and of higher impact.

4.4 Fingerprinting OpenFlow Controllers

The main approach developed in this chapter is to combine several techniques to fingerprint an SDN controller from its underlying data forwarding plane. Although our proposed techniques can be used separately, the accuracy of the results is much higher when combining them. Also, using only one technique may not give any result in some situations. In other words, each method has its success probability, and combining several techniques intuitively increases the probability of identifying the type of SDN controller used.

The following subsections present our techniques categorized into two classes: Timing-Analysis based techniques and Packet-Analysis based techniques.

4.4.1 Timing-Analysis based techniques

These techniques are based on time measurement to infer some indicative parameters of the controller.

4.4.1.1 Timeout Values Inference

Each flow entry has an idle_timeout and a hard_timeout field values associated with it. They indicate respectively the time in seconds after which the entry will be removed from the switch if no packet matches it, and the time after which to remove the entry anyway. These timeout values can be set and modified by application developers or network administrators. But, in most cases when the network or parts of the network only need a basic flow forwarding without additional traffic engineering logic, the network admins tend to use the forwarding applications that come with the controllers (typically L2-Switches) and the probability that they change these applications' parameters is fairly low. Note that in recent controllers, those forwarding elements even include some advanced features [152].

The idea is to infer flow-entry timeout values and compare them to known timeout values of different controllers (timeout database). The timeout database is constructed as follows:

for open source controllers, default timeout values can be gathered from their code source or configuration files. For proprietary controllers, the default timeout values can easily be figured out by simply using the controller and directly measuring the values. This method can be fairly accurate because of the low probability for default values to be modified by administrators.

To measure timeout values from an end-host in the underlying network, we propose the two following algorithms (algorithm 1 and algorithm 2). These algorithms consider network disruptions that may affect communication channels between end-hosts and the switch, and between the switch and the controller. Both algorithms require the ability to connect to another end-host in the same data plane (a pingable end-host). Algorithm 1 measures idle_timeout in two steps: first, it calculates RTT_avg (average Round-Trip Time using ping) in case when corresponding flow entries are installed in the switch. Measurements may be made for a configurable duration and/or number of probing packets n. Second, it measures RTT every $wait$ seconds. $wait$ value will be incremented by $step$ seconds until a significant difference between measured RTT and calculated RTT_avg is encountered. This difference means that the flow entry expired and the switch needed to call the controller asking how to handle the new ping. Final value of $wait$ matches the flow-entry $idle\_timeout$ value. A more accurate version of the algorithm is conceivable by using a binary search around the final $wait$ value, but by using $step$ of $5ms$, the algorithm remains very accurate even without binary search.

Note that in some controllers, the default $idle\_timeout$ value is set to 0 which means infinite, so the flow entry will never be removed. We found this in the *Ryu* controller and *Hydrogen*, an old version of *OpenDaylight* [153]. In this case, after a number of iterations, the algorithm will decide that the $idle\_timeout$ value is infinite and the controller may be *Ryu* or *Hydrogen* version of *OpenDaylight*. The search space has been limited to two controllers in this case, but we need to apply more techniques to decide which one of them.

Algorithm 1 idle_timeout measurement

1: Send first ping to install flow entry;
2: Send n pings and calculate the average ping time RTT_avg;
3: Wait $wait$ seconds;
4: Send one ping and calculate ping time T_{ping}
5: **if** $T_{ping} \approx RTT\_avg$ **then** //the flow entry still exists
6: $wait \leftarrow wait + step$;
7: Go to 3;
8: **else**//idle_timeout expired and the flow entry removed
9: $idle\_timeout = wait$
10: **end if**

To measure $hard\_timeout$ value, we first calculate the average of RTT time (RTT_avg) and $idle\_timeout$ values as in algorithm 1. Second, we send one ping to install the flow entry in the switch. Then, we send a ping every $wait$ seconds such as $wait$ value is less then $idle\_timeout$. As long as the RTT value is close to the average (RTT_avg), we continue to add $wait$ seconds to the $hard\_timeout$ value initialized to zero. We stop when we find a RTT

value which is significantly greater than (RTT_avg).

Algorithm 2 hard_timeout calculation

1: $hard\_timeout \leftarrow 0$ seconds;
2: Calculate RTT_avg as in algorithm 1;
3: Calculate $idle\_timeout$ as in algorithm 1;
4: Send one ping to make the controller install flow entry;
5: Wait $wait$ seconds, $wait$ must be less then $idle\_timeout$;
6: Send one ping and calculate ping time T_{ping};
7: **if** $T_{ping} \approx RTT\_avg$ **then** //the flow entry still exists
8: $hard\_timeout \leftarrow hard\_timeout + wait$
9: Go to 5;
10: **else**//hard_timeout expired and the flow entry removed
11: print $hard\_timeout$
12: **end if**

The attacker then compares the measured values ($idle\_timeout, hard\_timeout$) to known timeout values of controllers to guess which controller is used.

4.4.1.2 Processing-Time Inference

Each SDN controller is programmed differently using different tools, libraries and frameworks, so that each controller has its own execution speed. In other words, when receiving packets from the data plane, each controller takes a different time to process those packets and reply back to the data plane. The idea of this technique is to use estimated packet-processing time to determine the controller. As we mentioned before, authors of [126] used timing to determine if a remote network is an SDN network based on the difference of RTT in two cases: presence and absence of flow entries. As it has been mentioned by the authors, it is very difficult to measure with high accuracy the RTT to a remote network in a WAN because of many potential sources of disruption that may result in random variations of RTT values. In our technique, these disruption sources are minimal since the attacker is placed in the data plane of the target controller. And unlike [126], our method uses some key parameters inferred from the network to estimate processing time with higher precision.

The main idea in our approach is to measure the response time of the target controller and compare it to the processing-time database created beforehand. The processing-time database is a table that associates each controller to its processing time. Like in the previous technique (timeout values inference), we need a pingable destination end-host in the same data plane (the best scenario is that the attacker controls the destination end-host as well to be sure that its processing time does not affect the measurements). To create the processing-time database, we use a simplified architecture (Fig. 4.1) where the propagation times (1) and (2)

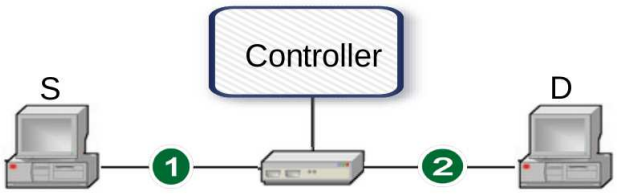


Figure 4.1: Simplified architecture to measure controllers' processing time

are minimal. we first measure $idle\_timeout$ and RTT_avg values as in algorithm 1. Then, we send n (100 for example) pings separated by $period$ seconds between every two pings, with $period$ greater then $idle\_timeout$. Every ping will cause the switch to send a Packet-In to the controller (by receiving the Packet-In message, the controller processes it to extract field values and installs the corresponding flow rule into the switch). Finally we calculate the average ping time T_{pavg} of the n pings and we record $T_{pavg} - RTT\_avg$ value in a table. This is the processing time of the current controller. We repeat this process with all controllers and we create the processing-time database by inserting tuples $(controller, processing\_time(T_p))$ (algorithm 3).

Algorithm 3 Building the processing-time database

1: Calculate RTT_avg as in algorithm 1;
2: Calculate $idle\_timeout$ as in algorithm 1;
3: **for** $i \leftarrow 1..n$ **do**
4: Wait $period$ seconds, $period$ must be greater than $idle\_timeout$;
5: Send a ping and save ping time;
6: **end for**
7: Calculate the average of saved ping time values T_{pavg} and calculate controller processing time $T_p = T_{pavg} - RTT\_avg$;
8: Insert $(controller, T_p)$ in the processing-time database;

Note that, as propagation times (1) and (2) (Fig. 4.1) are minimal, measured RTT_avg is accurate and hence T_p values are accurate.

Algorithm 4 Fingerprinting $controller$

1: Calculate RTT_avg as in algorithm 1;
2: Calculate $idle\_timeout$ as in algorithm 1;
3: **for** $i \leftarrow 1..m$ **do** //m = 20 for example
4: Wait $period$ seconds, $period$ must be greater then $idle\_timeout$;
5: Send a ping and save ping time;
6: **end for**
7: Calculate the average of saved ping-time values RTT' and compare $RTT' - RTT\_avg$ to the processing-time entries;

Now that we have the processing-time database, to fingerprint the target controller that manages the real SDN network we are connected in, we first measure RTT_avg to a destination, then we ping the same destination with a spoofed IP address to ensure that no corresponding flow entry exists in the switch and we compare the value $RTT - RTT\_avg$ to the processing-time database entries. For accuracy, we do not rely on a single ping, disruptions can happen during the ping affecting the response time. Instead, we send many (20 for example) pings with *period* seconds between every two pings (such as *period* value is greater than *idle_timeout*), we calculate the average of these ping times RTT' and finally compare the value $RTT' - RTT\_avg$ to the processing-time database entries (algorithm 4).

In addition to the probability that the network admin somehow modifies the execution time of the controller, which we argue is very low, there is a fair chance that during the scan, the controller is overloaded resolving requests and installing rules, which may significantly change the response time. In this case, if the attacker has further knowledge about the network state then he/she can surpass this problem. For example, he/she can avoid peak hours, and only scan the controller when the network is in its normal state.

4.4.2 Packet-Analysis based techniques

4.4.2.1 LLDP message analysis

This is a passive method which consists of identifying the controller by sniffing and analyzing OpenFlow Discovery Protocol ($OFDP$) packets sent over the data plane.

SDN is based on maintaining a global network view at the level of the controller. To obtain the global network topology, discovery modules of the controllers use $OFDP$ to collect updated information from different elements of the network including end hosts. $OFDP$ leverages the packet format of Link Layer Discovery Protocol ($LLDP$) with subtle modifications to perform topology discovery in an OpenFlow network.

Unlike ordinary $LLDP$ enabled switches, an OpenFlow switch needs the controller to send and process $OFDP$ messages and cannot do this by itself. The following is a simple scenario of the topology discovery process using $OFDP$. First, the SDN controller creates an individual LLDP packet for each port on each switch. Then, the controller sends these packets to the switches via Packet-Out messages that include instructions to send them out on the corresponding ports. In each switch, all received $LLDP$ packets will be forwarded to neighbours. When a switch receives a new $LLDP$ packet from another switch, it forwards it to the controller via a Packet-In message. At the end of the process, the controller will get information about all the data-plane connections. The entire discovery process is repeated periodically with the time periods varying from one controller to another, which is can be leveraged to identify which controller is managing the network. Also, the content of the $LLDP$ packets differs from one controller to another, which can be used accurately identify the controller. Table 4.3 in section 4.6.3 shows $LLDP$ packets sent by different controllers.

4.4.2.2 ARP response analysis

This technique can only be used to determine if the controller is the Hydrogen version of OpenDaylight and cannot be generalized to other types of controllers. It builds on the observation of how the controller reacts to unknown Address Resolution Protocol (ARP) requests in the data plane. The attacker sends an unknown ARP request, which means that the destination IP address is not assigned to any host in the network. As the destination IP is not present in the network, the switch, in addition to broadcasting the request, sends it to the SDN controller via a Packet-In message asking how to handle it. The OpenFlow specifications indicate that the controller responds to the switch by a Packet-Out and/or a flow-mod message explaining how to handle the request. The controller's response message differs from one controller to another, but the only controller whose behavior can be captured from an end-host is Hydrogen. Hydrogen version of OpenDaylight instructs the switch to broadcast the request once again which duplicates it in the broadcast domain. This duplicated ARP request, with one of the switch's Media Access Control (MAC) addresses as source address, indicates that Hydrogen is used.

As we mentioned in the introduction, the techniques we presented in this section are not to be used in an exclusive manner. Each technique is able to identify the controller with a certain probability that we did not compute analytically in this chapter. The user can use a subset or all the techniques executing them one by one, or better combine them in some optimal order. The selection of the optimal combination of techniques is an interesting research question that we leave for future work.

4.5 Experiment Environment and Methodology

As shown in Figure 4.2, our experiment environment consists of four physical machines (only three are shown in Fig. 4.2) carrying 4 virtual machines each and connected via OpenFlow virtual bridges (Openvswitch) forming a small-size data-center where VMs generate random traffic (ping and iperf) to random destinations. Note that, since we are not exploiting any weaknesses in the switch, it does not make any difference using a virtual switch or a physical one in this context, we only need a switch that correctly implements OpenFlow specifications. Note also that we did not add hops (transit switches) between bridges and the controller because even in real-world networks, a very small number (0, 1 or 2) of transit switches is enough to build a fairly large Local Area SDN network, like a data-center or a campus network. Such a small number of hops does not affect timing measurements in previous algorithms, The attacker is on the red (or black) VM connected to $br0$, and can ping the orange (or dark grey) VM connected to $br2$ (it could be any other VM in the network).

We have performed our experiments on five open source still maintained, OpenFlow controllers among the most widely used: OpenDaylight [153], POX [68], Beacon [161], Floodlight[155], and Ryu [162].

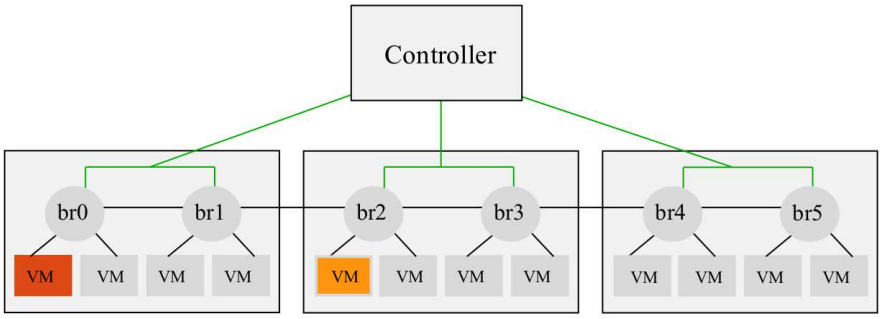


Figure 4.2: Test environment

4.6 Results

4.6.1 Timeout Values Inference technique

To evaluate the Timeout Values Inference method, we ran algorithms 1 and 2 ten times on each controller from the set of our target controllers. Default timeout values are given in table 4.1. Our algorithms did 2 errors in 50 measurements in both *idle_timeout* and *hard_timeout* and that is because algorithm 1 is used in algorithm 2.

Controller	idle_timeout (s)	hard_timeout (s)
OpenDaylight	0	0
Floodlight	5	0
POX	10	30
Ryu	0	0
Beacon	5	0

Table 4.1: Default Timeout Values

We also evaluated algorithms 1 and 2 separately by manually setting different values for *idle_timeout* and *hard_timeout* in POX source code, and running the algorithms from the attacker virtual machine (red VM in Fig. 4.2) to infer these values. We set the values $5, 10, 15, ..30ms$ for *idle_timeout* and the values $10, 20, ..60ms$ for *hard_timeout* respectively. For each algorithm, we repeated the execution 10 times on each value. *idle_timeout* calculation algorithm has an error rate of 0.03% (2 errors in 60 measurements) with a relative error of less than $1s$. The *hard_timeout* calculation algorithm has an error rate of 0% (no error) on 60 measurements.

4.6.2 Processing-Time Inference technique

First, we have built the processing-time database (table 4.2) of our set of target controllers by running algorithm 3 ($n = 100$) on a simplified testbed as described in Fig. 4.1.

Controller	T_p (ms)	T_p adjusted (ms)
OpenDaylight	1.004	0.177
Floodlight	3.454	2.627
POX	34.266	33.439
Ryu	5.216	4.389
Beacon	3.197	2.370

Table 4.2: Processing-time database (T_p: processing time).

Then, to evaluate this technique in our experimental environment (Fig. 4.2) we have run algorithm 4 ten times: measured T_p in Fig. 4.3 is the average value of the different executions. To get more precise comparisons, we calculate T_p adjusted: adjusted processing time = processing time - the average of RTT time in case the flow rule exists (RTT_{avg}).

For controllers Floodlight and Beacon which have very similar values of T_p, the use of only this technique, is not sufficient as it cannot decide between them.

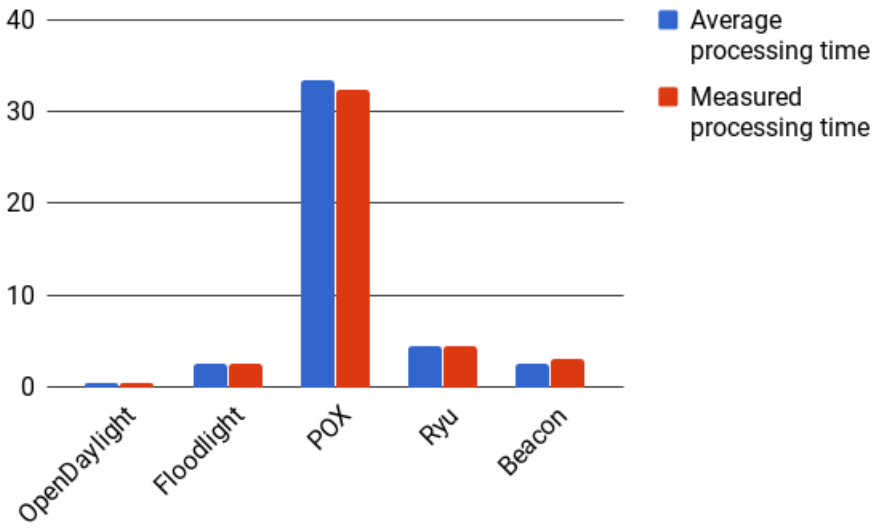


Figure 4.3: Measured processing times compared to average processing times (in ms)

4.6.3 *LLDP* message analysis technique

Figure 4.4 compares *LLDP*-packet reception intervals for different controllers. Table 4.3 shows the difference between controllers' *LLDP* packets. By receiving the *LLDP* packet, the attacker compares the different values against Fig. 4.4 and table 4.3 to identify the controller. Similar to technique 4.4.1.1, for proprietary controllers, the way to gather *LLDP* information is to simply use the controllers, analyze its *LLDP* packets, then use this information to fingerprint target controllers.

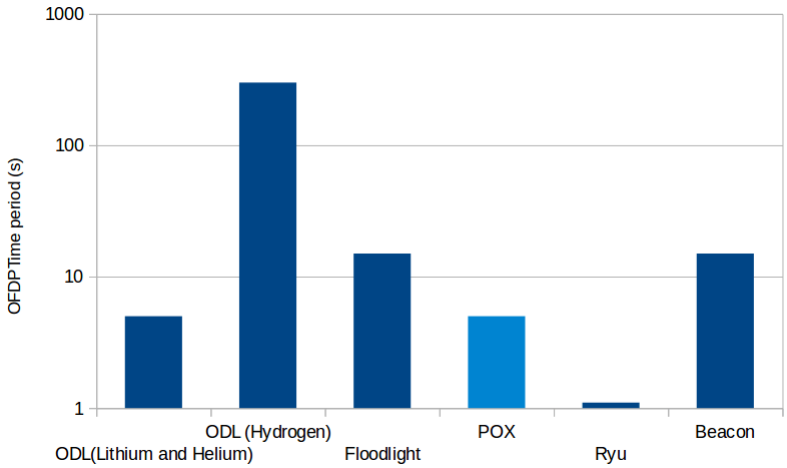


Figure 4.4: Controllers' LLDP-emission-interval comparison

4.7 Conclusion

In this work, we demonstrated the feasibility of fingerprinting attacks on OpenFlow controllers from the data plane by designing, implementing and testing practical techniques to identify the controller without access to the control plane. This is a critical step for a number of attack models since it provides the attacker with sufficient information about the controller to carry out more tailored attacks. Knowing the vulnerabilities of the target controller or one of its components, the attacker can indeed use known or design new attacks to take down the controller. In the future, we plan to expand the scope of this work by fingerprinting a larger set of controllers and by designing more techniques for fingerprinting controllers. We also plan to investigate formal methods for the evaluation of fingerprinting techniques and how they can be possibly combined to increase success rate. Finally, we plan to explore what countermeasures must be deployed to harden the security of SDN networks against controller fingerprinting and subsequent attacks.

Controller	$OFDP$ interval (s)	Remarks
OpenDaylight (Lithium & Helium)	5	LLDP packets include System Name field with value = "openflow" and no System Description field
OpenDaylight (Hydrogen)	300	LLDP packets include System Name field with value = "OF|[MAC address of the OF switch]" and no System Description field
Floodlight	15	Each LLDP packet is followed by an 0x8942 Ethernet packet sent in broadcast. This makes it easy to distinguish between Floodlight and Beacon
POX	variable (≈ 5)	LLDP packets include System Description field with value = "dpid:[MAC address of the OF switch]"
Ryu	1	Note that the Topology discovery module is still not stable and not included in the controller core.
Beacon	15	LLDP packets include two "unknown" fields and no System Name or Description feild

Table 4.3: Results of $LLDP$ message analysis

Chapter 5

Secure Topology Discovery Protocol for OpenFlow Networks

Summary

One of the SDN controller's duties is to perform an accurate, secure and near real time topology discovery to provide management applications with an up-to-date view of the network topology. However, all current SDN controllers perform topology discovery using OpenFlow Discovery Protocol (OFDP), which is far from being secure and efficient [156]. In this chapter, we show that $OFDP$ has serious security and performance problems, then we introduce a novel protocol $sOFTD$ (secure and efficient OpenFlow Topology Discovery), as an alternative that is more secure and more efficient than $OFDP$.

5.1 Introduction

The separation between the control plane and the data plane introduced by Software-Defined Networking (SDN) allows operators to employ quite damn, remarkably cheap but very fast hardware to forward packets, moving the control logic to a centralized and much smarter entity called controller. The controller plays the role of an operating system of the network. It abstracts the underlying forwarding hardware details and offers high level APIs that the network admins leverage to program their networks. One of the fundamental functions that a controller must offer is an accurate, near real time visibility of the network topology. This function is known as Topology Discovery. Topology discovery in SDN is more sensitive compared to traditional networks based on Link-State routing protocols like OSPF. In SDN, To discover the network topology, all current OpenFlow controllers implement the same protocol OFDP (OpenFlow Discovery Protocol).

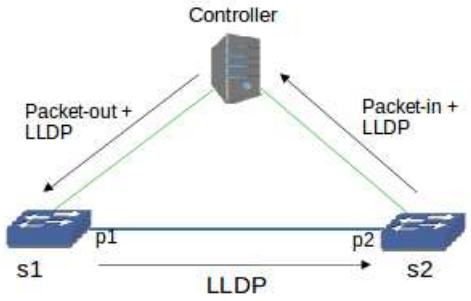


Figure 5.1: Discovering a unidirectional link in OFDP

Figure shows how OFDP works; To discover the unidirectional link s1 → s2, the controller encapsulates a LLDP packet in a Packet-out message and sends it to s1. The packet-out contains instruction for s1 to send the LLDP packet to s2 via port p1. By receiving the LLDP packet via port p2, s2 encapsulates it in a Packet-in message and sends it back to the controller. The controller receives the LLDP packet and concludes that there is a unidirectional link from s1 to s2. The same process is performed to discover the opposite direction s2 → s1 as well as all other links in the network. Note that, OFDP packets are sent to a "normal" multicast MAC (01:23:00:00:00:01) to avoid being swallowed by 802.1d compliant switches.

In dynamic networks like large data-centers and multi-tenant cloud networks, keeping an up-to-date visibility of the topology is a critical function; Switches leave and join the network dynamically creating changes in the topology which affects routing decisions that the controller has to make continuously. To remain up-to-date, the controller needs to repeat the process described in figure 5.1 periodically. The period separating two discovery rounds must be chosen carefully based on the dynamicity, size and capacity of the network; A 10 seconds period might not be suitable for a highly dynamic network as it may introduce a delay of up to 10 seconds. A short period (e.g. 3 seconds) also might not be suitable for a less-dynamic large size network as the large number of frequent discovery packets may exhaust controller's

resources. Put together, every discovery-round period T, the controller sends $\sum_{i=1}^{n} p_i$ (where n is the number of switches and p_i is the number of ports in switch i) Packet-out messages and receives $2L$ Packet-in messages. [160] proposes to reduce the number of Packet-out messages to n by rewriting LLDP packet-headers in the switch.

A non optimized or buggy topology discovery mechanism can affect routing logic and drastically reduce network performance. Our main goal in this paper is to demonstrate that $OFDP$ has serious, non-solved yet, security and performance problems, then we briefly introduce $sOFTD$ (secure and efficient OpenFlow Topology Discovery), a secure alternative that is more efficient than $OFDP$.

The remainder of this chapter is organized as follows: In section 5.2 we demonstrate why $OFDP$ shouldn't be implemented in production networks. We introduce our alternative protocol $sOFTD$ in section 5.3. Then we evaluate $sOFTD$ in section 5.4. Related work is discussed in section 8.5 and finally we conclude the paper in section 8.6.

5.2 Why OFDP shouldn't be implemented in production networks

5.2.1 OFDP is not secure

As implemented by all controllers we have tested (OpenDaylight [153], Floodlight [152], NOX [68], POX [68], Beacon [161], Ryu [162] and Cisco Open SDN Controller [163]), OFDP uses clear, non authenticated LLDP packets to detect links between switches which makes it vulnerable to a number of attacks:

Switch spoofing. As described in figure 8.8, e ach LLDP packet contains a version field, flags, TTL and TLVs (Type-length-value) for information advertisement. Mandatory TLVs in OFDP are $ChssisSubtype$ and $PortSubtype$ to track packets. In figure 5.1, the LLDP packet sent by switch s2 to the controller contains the tuple ($chassisSubtype = switch1ID, PortSubtype = p1$), hence the controller will detect that this is the same packet he sent to switch s1 with p1 as out-port.

The problem is that all controllers we have tested set $chassisSubtype$ value to the MAC address of the local port of the switch (figure 5.4), which makes it easy for an adversary to spoof that switch since controllers use that MAC address as a unique identifier of the switch. By intercepting clear LLDP packets containing MAC addresses, a malicious switch can spoof other switches to falsify the controller's topology graph. In the example shown in figure 5.3, s4 intercepts LLDP packets from s1 containing s1's local port MAC address. Now, s4 can use it as its own MAC and reconnect to the controller as s1 messing up the controller's topology graph (e.g. the controller adds nonexistent links $s1 \rightarrow s3$ and $s3 \rightarrow s1$). We have tested the switch spoofing attack successfully against Opendaylight and Floodlight.

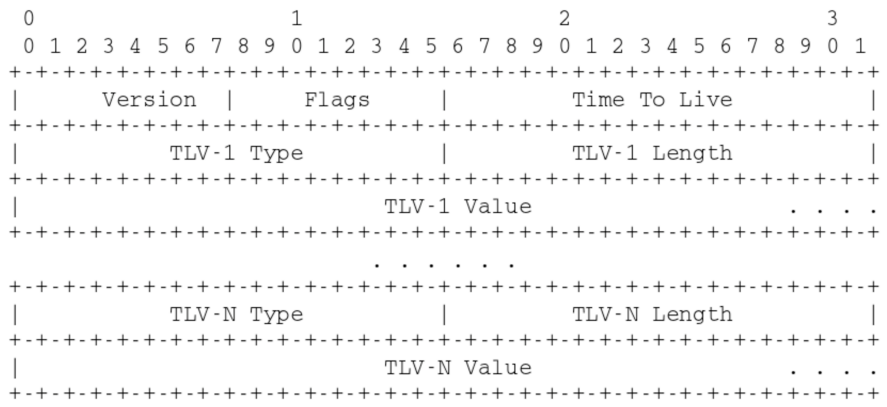


Figure 5.2: LLDP packet format [158]

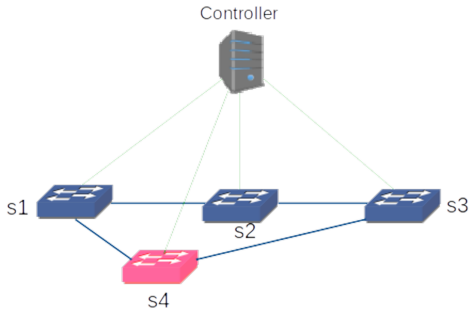


Figure 5.3: Switch spoofing attack

Link Fabrication. [157] and [159] pointed out that OFDP is vulnerable to link fabrication attacks; In figure 5.5, the adversary has control over two end-hosts h1 and h2 connected to switches s1 and s3. h1 sends the LLDP packets received from s1 to h2 through an out-of-band connection (could be a tunnel over s2 for example), and h2 replicates them to s3. The controller receives the LLDP packets from s3 and creates a link between s1 and s3. While not detected, the fake link pushes the controller into wrong routing decisions that affects all communications involving s1 and s3. If the attacker has control only over h1, but knows the DPID of s3 then he still can fabricate a unidirectional link s3 $\rightarrow$ s1 by injecting fake LLDP packets into s1.

Another form of link fabrication is by LLDP injection; By monitoring the traffic, the adversary gets the LLDP content used by the controller. Then, he/she injects the same LLDP packets into the network creating bogus links between switches or between the adversary hosts and switches.

[157] proposes to authenticate the LLDP packets by adding a key-Hash Message Authentication Code (HMAC) as an optional TLV in LLDP packets. As mentioned by the authors, this technique only works against fake LLDP injection but not against link fabrication by packet

```
Frame 9205: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface 0
Ethernet II, Src: 52:13:80:1f:e2:e1 (52:13:80:1f:e2:e1), Dst: NiciraNe_00:00:01 (01:23
    Destination: NiciraNe_00:00:01 (01:23:20:00:00:01)
    Source: 52:13:80:1f:e2:e1 (52:13:80:1f:e2:e1)
    Type: 802.1 Link Layer Discovery Protocol (LLDP) (0x88cc)
Link Layer Discovery Protocol
    Chassis Subtype = Locally assigned, Id: dpid:9a508e55184c
    Port Subtype = Port component, Id: 34
    Time To Live = 120 sec
    System Description = dpid:9a508e55184c
    End of LLDPDU
```

Figure 5.4: LLDP content used by POX controller

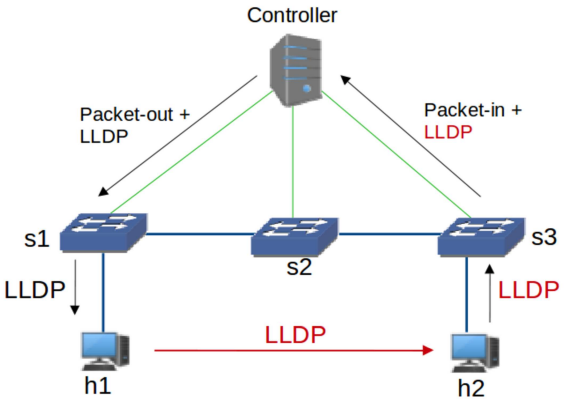


Figure 5.5: Link Fabrication attack

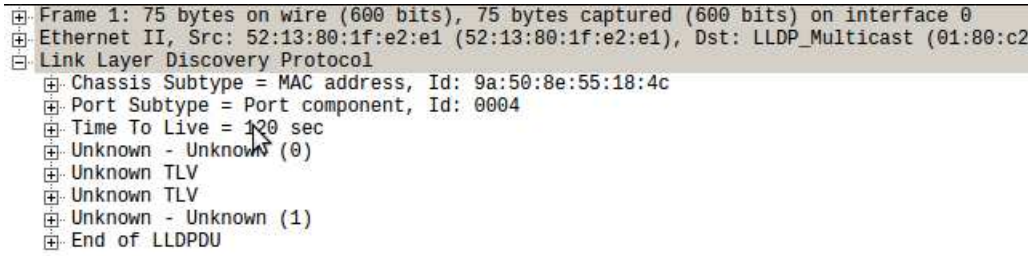


Figure 5.6: LLDP content used by Floodlight controller

duplication (Figure 5.5). [159] proposes a similar technique but using dynamic keys: a unique key for each LLDP packet assuming that h2, for example, uses only one example of LLDP packets received from h1 to generate future fake packets. However, an attacker controlling both hosts can permanently forward captured LLDP packets from h1 to h2 and from h2 to h1 and inject them back into switches without any modification.

Controller fingerprinting. As we explained in a previous work [164], the LLDP content is different from one controller to another which allows fingerprinting attacks on SDN controllers. An adversary (h1 in figure 5.5) matches the LLDP content he receives from s1 (LLDP packets originate from the controller) against a controller signature database to detect which controller is managing the network. Such information is very useful to launch specific and more efficient attacks on the controller. Figures 5.4 and 5.6 present the LLDP content of controllers POX and Floodlight respectively. Note also that the controllers use different default discovery-round periods which offers another way to differentiate between controllers. Although it is possible that network admins change both discovery-round period and LLDP content, it is more likely that the default values are kept unmodified.

LLDP Flood. This is a form of DoS attack where an adversary generates enough fake LLDP packets to exhaust the controller resources. In figure 5.7, host h1 generates large number of LLDP packets and send them to s1 which has a rule to forward every LLDP packet to the controller. Hence, a large number of LLDP packets can exhaust the link connecting the switch to the controller as well as the controller resources. Basic countermeasure methods like port blocking or packet filtering may not be effective, especially in the case of very dynamic environments (e.g. multi-tenant cloud) since connected hosts and switches change frequently, which may result in preventing legitimate LLDP packets from reaching the controller.

5.2.2 OFDP is not efficient

By using OFDP, the controller periodically sends many packets to every switch in the network, which could result in performance decrease of the data plane. Experiments made on different controllers [170] show that when the network size (i.e. number of switches) exceeds some threshold, running the discovery module alone results in significant increase of the controller's CPU usage and considerable decrease in network performance.

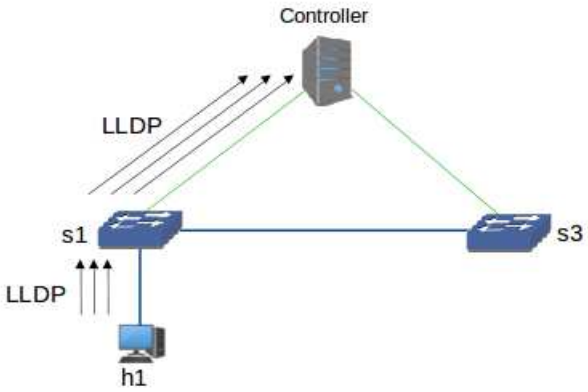


Figure 5.7: LLDP flood attack

5.2.3 Other issues

Other issues with OFDP include that it may not reliably work for heavily loaded links because discovery packets might get dropped or delayed. Moreover, when using OFDP in a multi-controller SDN network (e.g. running several guest controllers through FlowVisor), discovery cost increases linearly as more controllers are added.

5.3 Introducing sOFTDP: Secure OpenFlow Topology Discovery Protocol

In a dynamic data center SDN, the controller needs to be updated whenever a topology change occurs in order to make the suitable routing decisions for the new topology. Topology changes typically occur as a consequence of two events: *(i)* a new link is added to the network or *(ii)* an existing link is removed from the network. Both events are the result of either adding a new switch, removing an existing switch or adding/removing a link between two existing switches (the latter include link and switch failures). sOFTDP design assumes that the controller has **no prior knowledge** of the occurrence of such events and it is expected to dynamically update its topology map and adapt its routing decisions accordingly.

Network as a Service (NaaS) platforms are good examples where the controller has no prior knowledge of upcoming events. This is the case for example, of a public Cloud provider that offers customers the possibility to create their own networks, including hosts and SDN switches in virtual machines. The consumer (or tenant) can create hosts and switches and link them on the fly using a web interface, while the provider's controller manages the network and ensure the connectivity. In this way, the tenant can add, remove, place or move host and switch instances without worrying about the underlying configuration.

In the remaining of this section we first identify some fundamental requirements for topol-

ogy discovery in the context of dynamic virtualized data center networks, then we detail the sOFTDP design choices.

5.3.1 Fundamental requirements for topology discovery

Topology discovery is a critical process that is required to be:

- **Error free:** a topology error leads to wrong routing of flows. The impact can be very harmful if the error is in the routing core (core routers and links)
- **Secure**: a discovery protocol must be secure, preventing the introduction of fake links and information leakage (including topology information).
- **Efficient:** a discovery protocol must not flood the controller with redundant information and only transmit the topology events information when they occur.

5.3.2 sOFTDP design

sOFTDP [1] is designed to satisfy the above requirements. The main idea is to move a part of the discovery process from the controller to the switch. By introducing minimal changes to the OpenFlow switch design, sOFTDP enables the switch to autonomously detect link events and notify the controller. We also implement the necessary logic in the controller to handle switch notifications. The key ingredients of sOFTDP design are: Bidirectional Forwarding Detection (BFD) as port liveness detection mechanism, asynchronous notifications, topology memory, FAST-FAILOVER groups, "drop lldp" rules and hashed LLDP content. In the following we describe each of these mechanisms.

5.3.2.1 BFD as Port Liveness Detection mechanism

sOFTDP uses BFD (Bidirectional Forwarding Detection [166]) as port-liveness-detection mechanism to quickly detect link events. Instead of requesting topology information by sending periodic $LLDP$ frames, the controller just listens for link event notifications from switches to make topology updates. Hence, the switch needs a mechanism to autonomously and quickly detect link events and report them to the controller.

BFD is a protocol that provides fast routing-protocol-independent detection of layer-3 next hop failures. BFD establishes a session between two preconfigured endpoints over a particular link, and performs a control and echo message exchange to detect link liveliness. sOFTDP implements BFD in asynchronous mode: once a session is set up with a three-way handshake, neighbor switches exchange periodic control messages to confirm absence of a

[1] We interchangeably use the name sOFTDP for the topology discovery protocol and for the topology discovery application implementing it

failure (presence of link) between them. Note that sOFTDP only relies on BFD to detect link removal events. For link addition events, sOFTDP uses OFPT_PORT_STATUS messages to update the topology as we will detail in the next subsection. The reason for using BFD instead of OFPT_PORT_STATUS messages in detecting link removal is to include link failures that do not originate from administratively shutting down ports, e.g., failure of the underlying physical link or switch failure, etc.

BFD detection time of link events depends on the control packet transmission interval T_i and the detection multiplier M[166]. The former defines the frequency of control messages and the latter defines how many control packets can be lost before the neighbor end-point is considered unreachable. In the worst case, failure detection time is given by equation 5.3.1

$$T_{det} = M * T_i \tag{5.3.1}$$

The transmit interval T_i is lower-bounded by the RTT of the link. Note that a transmit interval of $T_i = 16.7ms$ and a detection multiplier of $M = 3$ are sufficient to achieve a detection time of $T_{det} = 50ms$. Also, $M = 3$ prevents small packet loss from triggering false positives. Furthermore, a such session generates only 60 packets per second.

5.3.2.2 Asynchronous notifications

sOFTDP enables the switch to inform the controller about port connectivity events. In case of administrative changes to port status (port turned up or down), the switch reports it via a OFPT_PORT_STATUS message defined in OpenFlow switch specifications. But, in the case of link failure or the remote port going down, OpenFlow doesn't provide any mechanism for the switch to inform the controller. sOFTDP adds this functionality to the switch by defining a new switch-to-controller message BFD_STATUS.

5.3.2.3 Topology memory

sOFTDP keeps track of topology events and builds a database of potential backup links besides the actual link database. When a new link is added, sOFTDP computes the local topology (relative to the added link). If the new link forms a shorter path between two switches and no traffic engineering application decides otherwise, the new path will be used for forwarding and the previous one will be saved as potential backup. sOFTDP installs OpenFlow FAST-FAILOVER groups [66] on the switches of the new link and marks the link as 'safe to remove' since it has at least one potential backup. Note that potential backup links are not considered backup links until all traffic engineering applications agree. Traffic engineering applications must communicate with sOFTDP to prevent interference in selecting primary paths

and backups.

5.3.2.4 FAST-FAILOVER groups

OpenFlow groups enable OpenFlow to abstract a set of ports as a single forwarding entity allowing advanced forwarding and monitoring at the switch level. The group table contains group entries; each group entry is composed of a set of action buckets with specific semantics dependent on the group type. When a packet is sent to a group, the actions in one or more action buckets are applied to it before forwarding to the egress port. Groups buckets can also forward to other groups, enabling to chain groups together.

The following four types of group tables are provided:

- **All:** used for multicast and flooding
- **Select:** used for multipath
- **Indirect:** simple indirection
- **Fast Failover:** use first live port

Different types of group tables are associated with different abstractions such as multicasting or multipathing. In particular, the Fast Failover Group Table monitors the status of ports and applies forwarding actions accordingly. When the monitored port goes down, the Fast Failover Group Table switches to the first port alive without consulting the controller [66].

sOFTDP enables seamless removal of switches and links while preserving connectivity. In order to accomplish that, when a link removal event occurs, sOFTDP uses OpenFlow FAST-FAILOVER groups (optional in OpenFlow 1.1+) to watch switch ports and perform fast switchover to backup links. Hence, switches concerned by the link removal start forwarding flows through the backup link and do not have to wait until the controller receives the topology event and installs new rules.

5.3.2.5 "drop lldp" rules

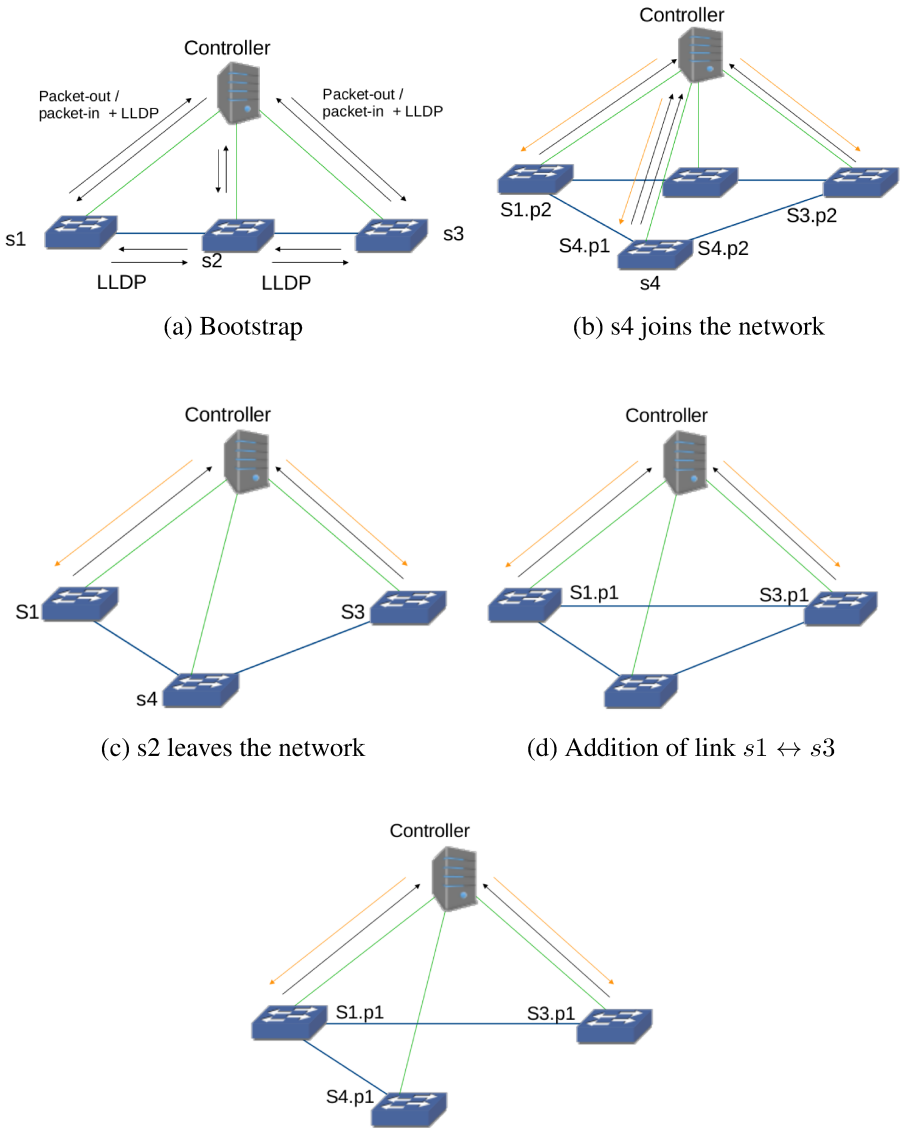
The switch has a rule "drop lldp" to drop every LLDP packet to prevent LLDP flooding attacks (see figure 5.7). In a SDN running OFDP, traditional Denial of Service (DoS) mitigating methods like placing firewalls or Intrusion Detection Systems (IDSs) to filter out LLDP packets are not effective because it is hard to distinguish between legitimate LLDP packets (generated by the controller and forwarded by switches) from the fake ones (generated by the attacker and also forwarded by switches to the controller). By removing periodically broadcasted LLDP packets, sOFTDP eliminates the possibility that malicious LLDP packets get forwarded to the controller and hence prevents it from being flooded.

5.3.2.6 Hashed LLDP content

The controller sends encrypted LLDP packets only when it receives a OFPT_PORT_STATUS with the flag $PORT\_UP$ set to 1 indicating the port went from down to up status. The LLDP packets are sent only to the concerned switches along with OpenFlow rules to forward them to the controller. These rules must have a higher priority than "drop lldp" rules and their *hardtimeout* values are set to $500ms$. The purpose of the LLDP packets here is to learn added links as shown in figure 5.8b and detailed in subsection 5.3.3. Finally, $500ms$ is a small arbitrary value to ensure that potential malicious LLDP packets generated exactly during this time window will not significantly affect the controller.

5.3.3 How sOFTDP works

Figure 5.8 shows how sOFTDP works. To bootstrap, the controller sends LLDP packets to all connected switches like in traditional OFDP (figure 5.8a). The main difference is that we do not use clear MAC addresses as switch DPIDs. Instead, we use hash values of them to prevent all information discloser and switch spoofing attacks. We also hash *system_description* field value to prevent controller fingerprinting [164].



(a) Bootstrap

(b) s4 joins the network

(c) s2 leaves the network

(d) Addition of link $s1 \leftrightarrow s3$

(e) Removal of link $s3 \leftrightarrow s4$

Figure 5.8: How sOFTDP works

When a new switch joins the network, it starts by establishing a connection with the controller: The switch and the controller exchange *hello* ($OFPT\_HELLO$) messages that specify the latest OpenFlow protocol version supported by the sender. Then, the controller sends the switch a $feature\ request$ message asking its capabilities. The switch responds with a $feature\ reply$ message, which includes the local MAC address (that corresponds to the internal port of the switch) in the *switch datapath ID* field and that's how the controller keeps track

of connected switches. The *feature reply* message also includes ports status which are all down initially. In figure 5.8b, when switch $s4$ joins the network and new links are established, switches s1, s3 and s4 send $PORT\_STATUS$ messages to inform the controller that involved ports s1.p2, s4.p1, s4.p2 and s3.p2 went up and are connected. The controller 'c' then sends LLDP packets to be forwarded only through those ports: $c \rightarrow s1.i \rightarrow s1.p2 \rightarrow s4.p1 \rightarrow s4.i \rightarrow c$, $c \rightarrow s4.i \rightarrow s4.p2 \rightarrow s3.p2 \rightarrow s3.i \rightarrow c$, $c \rightarrow s3.i \rightarrow s3.p2 \rightarrow s4.p2 \rightarrow s4.i \rightarrow c$ and $c \rightarrow s4.i \rightarrow s4.p1 \rightarrow s1.p2 \rightarrow s1.i \rightarrow c$. With i indicates internal port.

Once all LLDP packets are received, the controller identifies the new links and store them in the Topology Map. Note that by using $PORT\_STATUS$ messages as trigger to learn new links, the controller doesn't need to periodically send discovery packets and switches do not need to be too smart to determine the new links (as in [167]) or to store them locally.

Once the new topology is computed, the controller detects multiple paths between pairs of switches. Independently of traffic engineering applications running on the same controller, the sOFTDP topology module tags shortest paths as primary paths and longer paths as secondary paths or potential backups (e.g., $s1 \leftrightarrow s2$ is a primary path and $s1 \leftrightarrow s4 \leftrightarrow s3 \leftrightarrow s2$ is a secondary path). Then, if not specified otherwise by any traffic engineering application, the controller installs fast-failover group rules on the switches of the shortest path. This ensures continuity of connectivity in case of topology events. In the example shown in figure 5.8b, there are two similar paths, in term of number of hops, between $s1$ and $s3$. The controller arbitrary tags $s1 \leftrightarrow s4 \leftrightarrow s3$ as primary path and installs fast-failover group rules on switches $s1$ and $s3$ to watch ports $s1.p1$, $s1.p2$, $s3.p1$ and $s3.p2$.

When a switch leaves the network ($s2$ in figure 5.8c), neighbor switches detect and report link events to the controller: BFD session on $s1.p1$ detects the link $s1.p1 \leftrightarrow s2.p1$ failure and sends a BFD_STATUS message to the controller. In the case of link removal, the controller doesn't need to send LLDP packets and just removes the link from the topology map. The same process applies to link $s2.p2 \leftrightarrow s3.p1$. Switches s1 and s3 automatically switch traffic through the path s1 $\leftrightarrow$ s4 $\leftrightarrow$ s3 using the fast-failover group rules installed previously.

When a link is added between two existing switches ($s1 \leftrightarrow s3$ in figure 5.8d), the involved ports $s1.p1$ and $s3.p1$ send $PORT\_STATUS$ messages to the controller with "port up" flags set. The controller then sends LLDP packets to be forwarded only through $s1.p1$ and $s3.p1$: $controller \rightarrow s1.internal \rightarrow s1.p1 \rightarrow s3.p1 \rightarrow controller$ and $controller \rightarrow s3.internal \rightarrow s3.p1 \rightarrow s1.p1 \rightarrow controller$. After the new topology is computed, the controller tags $s1 \leftrightarrow s3$ as the shortest path and $s1 \leftrightarrow s4 \leftrightarrow s3$ as a potential backup path and installs fast-failover group rules on $s1$ and $s3$ (in this particular example, the same rules already exist)

When an existing link is removed ($s3 \leftrightarrow s4$ in figure 5.8e), the involved ports $s3.p2$ and $s4.p2$ detect loss of connectivity very quickly using BFD and report it to the controller via BFD_STATUS messages. The controller then drops the link $s3 \leftrightarrow s4$ from its topology map without the need to send LLDP packets. Finally, the controller removes the tag from the remaining path.

5.4 Evaluation

5.4.1 Emulation Testbed

To evaluate sOFTDP, we implemented sOFTDP topology module on Floodlight. We conducted experiments on an emulated testbed using Mininet [168]. The emulated testbed is composed of four virtual bridges based on Open vSwitch [169] and controlled by Floodlight controller from a different physical machine. We upgraded existing Open vSwitch (of mininet v2.2.1) to the newer version 2.3.1 that supports the BFD protocol and fast failover groups. Then we added a simple patch to Open vSwitch to send BFD_STATUS to the controller upon BFD events (see section 5.3.2.2). BFD detection time is set to $1ms$.

5.4.2 Experiments and results

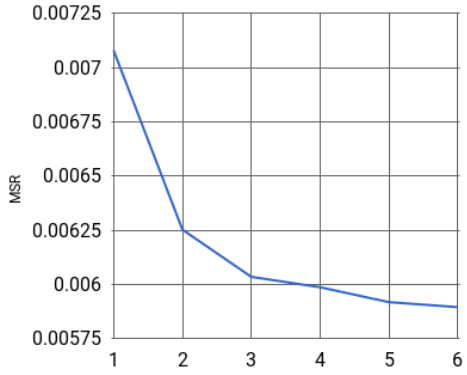


Figure 5.9: New link detection time in ms

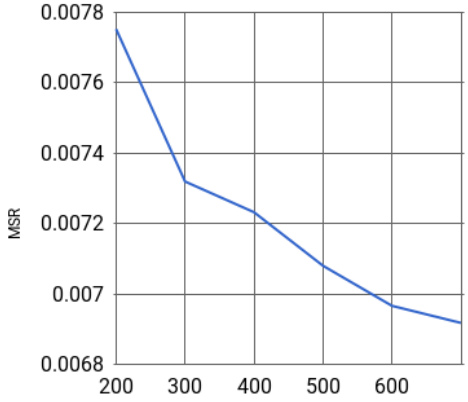


Figure 5.10: Adaptation time in ms

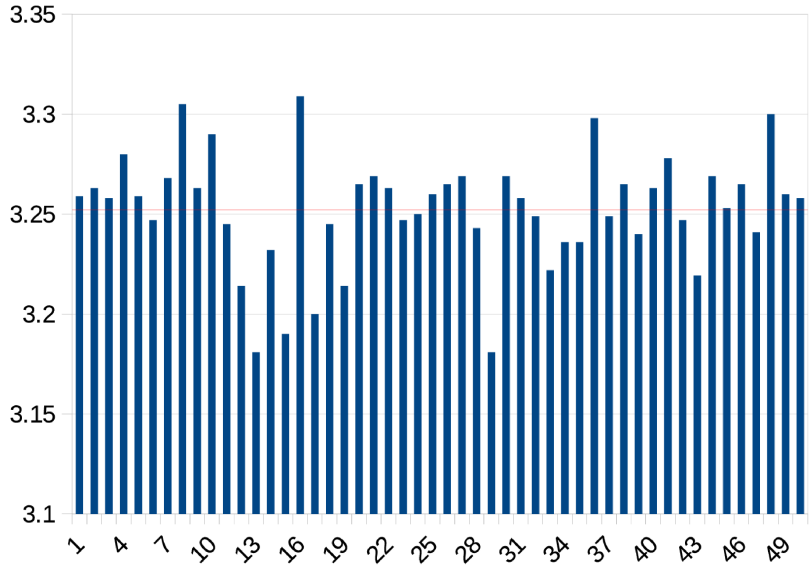


Figure 5.11: Link removal detection time in ms

As previously explained, to handle link removal, sOFTDP uses BFD protocol to detect port events at the switch level, then the switch triggers a notification the the controller (BFD_STATUS message) and finally the controller removes the link from the topology map. To handle link addition, sOFTDP listens for OFPT_PORT_STATUS messages triggered by the ports going up then the controller sends a LLDP message to the concerned switches to confirm the added link and finally adds it to the topology map. Accordingly, two scenarios are implemented and evaluated:

Scenario one: Link $s1.p1 \leftrightarrow s3.p1$ is added (figure 5.8d). We measure the *learning time* the controller takes to know about the added link as given in equation 5.4.2.

$$T_{learn}(i,j) = max_{d\in\{i,j\}}(T_{pstatus}(d)) + RTT_{LLDP}(i,j) \tag{5.4.2}$$

$$RTT_{LLDP}(i,j) = T_{delv}(c,i) + T_{delv}(i,j) + T_{delv}(j,c) \tag{5.4.3}$$

$$T_{learn}(\{i,j\}) = max(T_{learn}(i,j), T_{learn}(j,i)) \tag{5.4.4}$$

Where: $T_{learn}(i,j)$ is the time necessary to learn unidirectional link (i,j) $T_{pstatus}(i) = T_{trsm}(i,c)$ is the time OFPT_PORT_STATUS message takes from switch i to the controller. $RTT_{LLDP}(i,j)$ is the round trip time that a LLDP packet sent from the controller takes to go through switch i then switch j and back to the controller. $T_{learn}(\{i,j\})$ is the time necessary to learn bidirectional link $\{(i,j),(j,i)\}$ and finally $T_{delv}(x,y)$ is the packet delivery time from node x to node y. Figure 5.9 shows the average of 50 performed experiments and 95% confidence interval yielding learning times of $5.68 \pm 0.85ms$

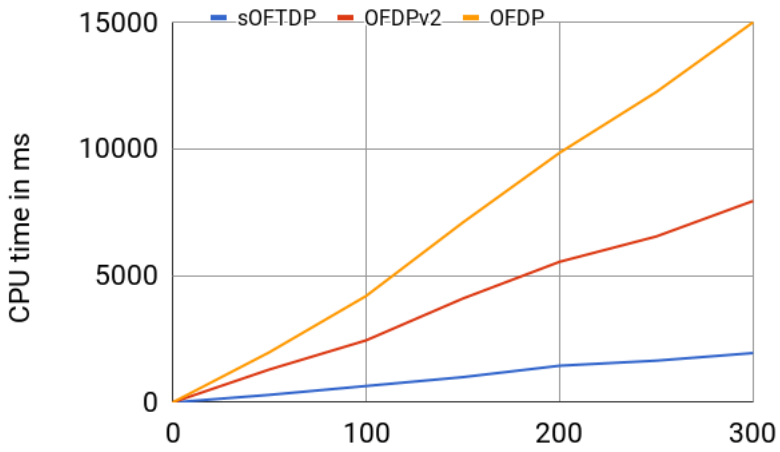


Figure 5.12: CPU time (y axis, in ms) over number of switches (x axis)

To further demonstrate sOFTDP performance, we also measure the overall *adaptation time* when a new link is added (see figure 5.8d) to the network. *adaptation time* includes learning the new link, installing fast-failover group rules in the switches and the actual switchover time. Figure 5.10 depicts the result averaged from 50 conducted experiments. The average and 95% confidence interval are of $6.12 \pm 0.7ms$.

Scenario two: Link $s3.p2 \leftrightarrow s4.p2$ is removed (figure 5.8e). We measure the *learning time* the controller takes to learn the change in the topology. The link is brought down upon the first BFD_STATUS message of either of its endpoints.

The *learning time* in this case is:

$$T_{learn}(i,j) = min(T_{bfd}(i), T_{bfd}(j)) \tag{5.4.5}$$

Where $T_{bfd}(x)$ is the BFD detection time on the involved port of node x. Computed as follows:

$$T_{bfd}(x) = T^x_{det} + T_{delv}(x,c) \tag{5.4.6}$$

Where T^x_{det} is BFD *detection time* given in equation 5.3.1 and $T_{delv}(x,c)$ is *packet delivery time* from node x to the controller.

Figure 5.11 shows sOFTDP learning time average (taken over 50 performed experiments) and the 95% confidence interval resulting in $3.25 \pm 0.008ms$

sOFTDP *learning time* is independent of the size of the network and depends only on the inter-switch RTT and the RTT between switches and the controller. Figure 5.12 compares

sOFTDP to OFDP and OFDPv2 [160] in term of CPU time. Each experiment is performed over $200s$ period during which one topology event is generated every second.

5.5 Related work

Unlike our proposal, most of previous work focus either on security problems or on performance problems in OFDP. In [157], authors identified *link fabrication* attacks on OFDP and proposed to authenticate LLDP packets by adding an optional TLV "HMAC" to ensure their origin and integrity.

The average overhead introduced by this approach differs between the first discovery round and the following rounds, because the HMAC value is computed once and cached for the future construction and validation of LLDP packets. The average overhead accounts for 80.4% of overall LLDP construction time in the first round and accounts for 2.92% in the following rounds. Although this approach prevents link fabrication by fake LLDP injection, it does not defend against link fabrication in a relay manner as discussed in section 5.2. The authors argue that a solution to the link fabrication by a relay could be that the controller monitors ports to detect whether the connected machine is a host or a switch. If the connected machine is a host generating LLDP packets then an alert is triggered. However, a host can easily behave like a switch making this solution unpractical.

A similar approach was proposed in [159] but using HMAC with a dynamic key which is randomly generated for every single LLDP packet. This approach adds an extra 8% of in CPU load.

OFDPv2 [160] reduces the number of OFDP-related packet-out messages by rewriting LLDP packet headers in the switch. In the traditional OFDP, the controller sends $\sum_{i=1}^{n} p_i$ packet-out messages every discovery round, where n is the number of switches and p_i the number of ports in switch i. The number of packet-out messages shrinks to n by sending only one packet per switch and rewriting copies for different ports at level of the switch. OFDPv2 achieves 50% reduction in CPU load compared to OFDP but obviously requires more logic to be added to the switch. Also, OFDPv2 does not reduce the number of packet-in messages that the controller periodically receives from switches.

In [167] authors implemented the ForCES [59] protocol to communicate the topology information between switches and the controller. Switches acquire neighbor topology information by exchanging LLDP packets as in traditional networks and store it in their device maps. The acquired information is updated periodically as LLDP frames are exchanged. Then, upon receiving a topology change notification from a switch, the controller needs to query the connected switches in order to learn their respective neighbors. The authors measured an average learning time of $12ms$ without considering the LLDP exchange time. In other words, the LLDP time exchange time is not included and it takes $12ms$ for the switch to detect the topology change, send a notification to the controller and then answer the controller's request for the topology information.

5.6 Conclusion

In this work, we extended our previous paper on OFDP limitations by introducing and detailing a novel topology discovery protocol for OpenFlow (sOFTDP). We argue that this is the first time major security and performance issues related to the topology discovery process in current SDN controllers, are tackled. Our proposal requires minimal changes to the OpenFlow switch design and is shown to be more secure (by design) than previous workarounds on traditional OFDP. Also, our proposal outperforms OFDP and OFDPv2 by several orders of magnitude which we confirmed by proof of concept experiments. Further experiments on larger physical testbeds are being conducted ad will be included in future work.

Part IV

Traffic Engineering in Softwarized Networks

Chapter 6

Traffic Engineering in Softwarized Networks: State of the Art

Summary

Traffic Engineering (TE) is a key element in network management. It studies measurement and management of network traffic and optimizes routing mechanisms in order to improve utilization of network resources and meet quality of service (QoS) requirements. Compared to traditional networks, SDN greatly facilitates the design and imeplementation of TE algorithms due to its programmability and global centralized control. In this chapter, we review SDN traffic engineering research and introduce our contributions in this field.

6.1 Introduction

Traffic Engineering (TE) generally means analyzing network traffic and optimizing traffic flow and resource allocation in order to enhance the performance of an operational network [11]. TE has been widely exploited in data networks, from past ATM (Asynchronous Transfer Mode) networks to current IP/MPLS networks. However, current networking paradigms and the TE solutions implemented for them are unfavorable for the future generations of networks due to two main reasons. First, they don't meet the requirements of today's Internet applications in term of real time reaction, and support of large amount of traffic. The underlying network should be able to classify a variety of traffic types and to provide a suitable service for each traffic type in real time (i.e., order of ms). Second, the massive growth of large scale data centers, with significant bandwidth requirements -as the needs for cloud computing, multimedia contents, and big data analysis are increasing- urgently requires new, intelligent and more efficient traffic engineering mechanisms to assure optimal resource utilization and performance.

In the past, traditional service provider networks employed technologies like X.25, Frame Relay or ATM (standardized in the late 1980s) to provide Layer 2 point-to-point virtual circuits with contractually predefined bandwidth. Traffic engineering (in the sense of optimal distribution of load across all available network links) was inherent in these services. The calculation of the optimum routing of virtual circuits was mostly done off-line by a network management platform. Real-time on-demand establishment of virtual circuits was offered by networks employing Frame Relay or ATM as follows [12]:

- The management platform examines the free network capacity.
- The management platform computes the end-to-end hop-by-hop path throughout the network that satisfied the contractual requirements.
- The management platform establishes a virtual circuit along the computed path.

On the other hand, most IP-based services, including VPNs implemented with MPLS VPN, IPsec or L2TP (Layer 2 transport protocol), follow a completely different service model:

- Every IP packet is independently routed through the network, and every router in the path makes independent next-hop decisions.
- All packets toward the same destination take the same path once merged.

Layer 2 switched networks employ complex circuit setup mechanisms and expensive switching methods. On the other hand, IP networks focus on low-cost, high-speed switching of a high volume of traffic. Hence, IP networks were adopted by nearly all service providers to build next-generation networks, even modern fiber-optics networks. However, there are many scenarios in IP networks where some links are underutilized and others are overloaded. Effectively, there is need for traffic engineering capabilities in routed IP networks, but they are simply not available in the traditional hop-by-hop, destination-only routing model that most IP networks use.

In order to bring traffic engineering to IP-based networks, various approaches have been proposed:

- Build on a layer 2 switched technology that has inherent traffic engineering capabilities. This approach was used extensively by service providers when they were introducing IP as an additional service on top of their layer 2 switched technology. This approach is no longer adopted and operators only use pure IP-based networks.
- Adjusting parameters of the IP routing protocols to influence the paths taken by packets. These techniques shift traffic to underutilized links by artificially lowering their cost and making them more attractive to routing protocols such as Open Shortest Path First (OSPF). Such approach is almost impossible in a complex network and works only in niche situations.
- Using more advanced techniques such as IP-over-IP tunneling and Multiprotocol Label Switching (MPLS) traffic engineering. While virtual circuits implemented with IP-over-IP tunnels are too complex and so are better avoided, MPLS, on the other hand, has a complete set of traffic engineering features similar to those available in advanced ATM or Frame Relay networks. For example: tracking available resources on each link using extensions to OSPF for example. When establishing an MPLS tunnel, the end-to-end path through the network is computed hop-by-hop based on the reported state of available resources.

Due to their unique features, SDNs provide huge incentive for new TE techniques that exploit the global network view, monitoring capabilities, and programmability available for better traffic control and management. In the rest of this chapter, we survey current SDN traffic engineering techniques and proposals. We pay a special attention to traffic measurement and cognitive routing. We refer to [13] for a survey on non-cognitive routing methods in SDN.

6.2 SDN Traffic Engineering

Traffic engineering mechanisms in SDN can be more efficient and more intelligent compared to the conventional approaches such as ATM, IP, and MPLS because of the major advantages of the SDN architecture. More specifically, SDN provides:

- Centralized view that enable deploying a scalable measurement and monitoring mechanisms. The SDN controller collects and stores the entire network information, including the network topology, the network status, and global application requirements including QoS and security requirements.
- Global programmability. The network operator leverages the SDN controller to dynamically program the forwarding layer devices to optimize the allocation of network resources.

- Openness. The interface between the controller and forwarding equipment has is unified across vendors [13, 14].

There are four axes of traffic engineering in SDN: flow management, fault tolerance, topology update, and traffic analysis.

First, according to the basic operation of flow management in SDNs, when a flow arriving at the switch does not match any rules in the flow table, it will be processed as follows:

- The first packet of the flow is sent by the ingress switch to the controller.
- The controller computes the forwarding path for the flow.
- The controller sends the corresponding forwarding rules to install in the flow tables at each switch along the computed path.
- Finally, all subsequent packets in the flow are forwarded in the data plane along the installed path.

To balance the load across multiple paths, the SDN controller can leverage OpenFlow to program multiple switches to carry the same flow. However, in this operation, if the aggregated traffic consists of high number of new flows, the resulting latency overhead from installing large numbers of forwarding rules can be yielded at both the control plane and data plane. Hence, the traffic engineering mechanisms must address the tradeoffs between the latency and load-balance.

Second, whenever a failure occurs (switch, link or controller failure), the SDN should be able to recover rapidly, transparently and gracefully to ensure the network reliability. A single failure should be recovered within 50 ms in carrier grade networks [16]. As we covered in chapter 5, in OF v1.1+, a fast failover mechanism is introduced in which an alternative port and path can be specified, enabling the switch to switch to an alternate forwarding path without requiring a round trip to the controller. However, the failure recovery still needs to consider the limited memory and flow table resources at switches.

Third, as we covered in chapter 5, the key challenge in topology update is how the SDN controller can efficiently update the network with required consistency in (near) real time. The topology discovery and update process is more complex in large SDN network, where switches are controlled by clusters of controllers and mostly through in-band control traffic.

Finally, the traffic analysis mechanisms should include:

- Traffic/network monitoring tools which are the most important prerequisite for all other traffic analysis tasks.
- network invariant checking mechanisms
- programming error debugging software
- flow/state data collection
- analytics/ mining of patterns/characteristics

- etc.

Finally, many SDN architectures use the existing flow based monitoring tools from traditional IP networks, which can lead to high monitoring overhead for the switch [17]. With the release of OpenFlow v1.3, more advanced flow metering mechanisms were introduced, hence the need to design more advance monitoring tools to take advantage of it, while keeping a low complexity design, low measurement overhead and a high measurement accuracy.

6.2.1 Traffic Monitoring/Measurement in SDN

Traffic measurement is a crucial task in traffic engineering. It includes three main subtasks: network topology measurement, network traffic measurement, and network performance measurement. We have extensively studied the network topology measurement in chapter 5, and here we focus on the traffic and performance part.

In order to react to traffic changes, the management applications require accurate and timely statistics on network resources at different aggregation levels (such as flow, packet and port) [13]. Hence the SDN controller must continuously monitor traffic counters and performance metrics to quickly adapt forwarding rules in switches.

However, many SDN architectures use traditional monitoring solutions that either require complex additional modules at the switch or impose significant measurement overhead. For instance, Cisco's NetFlow [18] installs probes at switches as special modules to collect either complete or sampled traffic statistics, and send them to a central collector [13]. Another flow sampling tool is sFlow [19], which samples traffic in a time-based fashion. Another flow sampling tool is Juniper's JFlow [20] which is quite similar to NetFlow.

These approaches introduce significant overhead incurred by statistics collection from the whole network. To cope with this problem, The following solutions was proposed to provide more efficient monitoring with higher accuracy and lower overhead.

PayLess [21] is a query-based monitoring framework designed as a component of the OpenFlow controller. PayLess provides a flexible RESTful API, translating request commands from applications into flow statistics collection at different aggregation levels (such as flow, packet and port), where it performs highly accurate information gathering in real-time without incurring significant network overhead. Instead of making the controller continuously polling switches, PayLess uses an adaptive scheduling algorithm that achieves the same level of accuracy as continuous polling with much less communication overhead. The evaluation results show that PayLess sends only 6.6 monitoring messages per second on average, compared to the controller's periodic polling, which has an overhead of 13:5 monitoring messages per second on average.

OpenTM [22] is a query-based monitoring system that aims to measure the traffic matrix (TM) for OpenFlow networks. It keeps track of all active flows in the network. First, it gets the routing information, including routing paths, from the routing applications, then it periodically polls flow byte and packet-count counters from switches on each flow path. Using

the polled statistics, OpenTM constructs the traffic matrix which represents the added amounts of traffic measured from each source to each destination in the network. In general, the number of network flows is very big, but available measurement resources, namely Ternary Content Addressable Memory (TCAM), are expensive, power hungry and hence limited, so in practice, it is impossible to obtain the traffic matrix by measuring the size of each flow directly. To solve this problem, selective and random switch polling approaches were proposed. Usually, random switch polling is not effective because selected switches might not be of significant importance in term of traffic volume. Selective switch polling however are more efficient.

iSTAMP [23] is an intelligent Traffic (de)Aggregation and Measurement Paradigm. iSTAMP partitions TCAM entries of switches into two parts (1) wildcard rules for aggregate measurements, and (2) fine grained rules to de-aggregate and directly measure the most informative flows for per-flow measurements. iSTAMP then processes these aggregate and per- flow measurements to estimate individual network flows using a variety of optimization techniques. iSTAMP seems to find balance between limitations of network resources and measurement accuracy, however, it does not consider routing and flow aggregation feasibility when designing optimal flow aggregates and only focuses on single-switch scenario. [24] extended iSTAMP framework to multi-switch scenario.

OpenMeasure [25]: assumes that the aggregation matrix is given based on the underlying routing and flow aggregation rules. OpenMeasure leverages the global view of SDN controller to identify the available monitoring resources, employs an online learning algorithm to determine the most informative flows for sampling and places flow sampling rules in selected SDN switches. Furthermore, OpenMeasure is light-weight and compatible with hybrid SDN networks.

In contrast to the on-demand query-based approaches, passive push-based monitoring methods have been proposed to analyze control messages between the controller and switches. These methods use the controller messages to monitor and measure network utilization, such as bandwidth consumption, without inducing additional overhead.

FlowSense [26] analyzes dynamic changes in network flows using messages received by the controller. For example, FlowSense uses PacketIn and FlowRemoved messages in OpenFlows networks to estimate per flow link utilization. The evaluation results show that FlowSense has higher accuracy compared to the request-based methods.

OpenSketch [27] is a software defined traffic measurement architecture, which separates the measurement data plane from the control plane. OpenSketch is generic and designed to allow more customized operations. It can perform efficient data collection with respect to flow selection by using both hashing and wildcard rules. In the data plane, OpenSketch provides a three-stage pipeline (hashing, filtering, and counting), which can be implemented with commodity switch components and support many measurement tasks. OpenSketch provides a library that contains a list of sketches, the sketch manager, and the resource allocator. Sketches can be used for different measurement programs such as heavy hitter detection and fine-grained delay measurement. The skecth library makes measurement programming easier by freeing operators from understanding the complex switch implementations and parame-

ter tuning in diverse sketches. The measurement library automatically configures the data plane pipeline for different sketches and allocates resources (switch memory) across tasks to maximize accuracy.

OpenSample [28] proposed by IBM, is a sampling-based SDN measurement system. Rather than using the expensive OpenFlow rules which, OpenSample leverages sFlow [19], which is present in most switches, to capture packet header samples from the network with low overhead and uses TCP sequence numbers from the captured headers to measure accurate flow statistics. From random samples, OpenSample can infer a variety of information about the network including the elephant flows and link utilization.

6.2.2 Cognitive Routing in SDN

Recently, machine learning (ML) techniques have made breakthrough progress in a variety of application fields, such as computer vision, speech recognition and bioinformatics. Machine learning is the applied science of constructing algorithms and models that can learn to make decisions directly from data without following predefined rules. Machine learning generally deals with complex problems where the notion of probability and uncertainty plays an important role, and usually require classification, regression and decision making. Machine learning techniques may perform close to or even better than humans in such problems (eg. the go game [29], chess and facial recognition).

Although applying machine learning techniques in networking, particularly traffic engineering, became an active and promising research area recently, it has not been used widely in computer networks. For example, various machine learning-based approaches proposed for routing in computer networks during the 1990's and early 2000's [30, 31, 32, 33] remained isolated and failed to get much attention from the networking research and industry communities [34].

Why apply machine learning to routing? Routing is one of the most fundamental networking tasks and, consequently, has been extensively researched in a various contexts such as data centers, WANs, ISP networks, interdomain routing with BGP, wireless networks, and more) [38].

There is a big deal of uncertainty in route-optimization when trying to optimize routing configurations with respect to previously observed traffic, with the hope that these configurations fare well with the future traffic. Unfortunately, optimizing with respect to past traffic conditions or optimizing for the worst-case conditions across a broad range of considered traffic scenarios might fail miserably in achieving good performance even if the actual traffic conditions are not too different from the optimized ones.

Machine Learning provides a new option: leveraging information about past traffic conditions to learn good routing configurations for future traffic conditions. Although the exact future traffic demands are unknown, on can realistically assume that some information about the future are embedded in the historical traffic demands (e.g., changes in traffic across times

of day, day of week, the skewness of traffic, etc.). Hence, a valid approach is to continuously observe traffic demands and adapt routing with respect to learned traffic characteristics.

Traditionally, network intelligence was implemented in the network nodes that were very limited in term of computational power and memory, which made them incapable of processing computationally intense machine learning algorithms. Now, with SDN and NFV centralizing the network intelligence and decoupling it from the network devices, it is possible to implement global machine learning based control algorithms on powerful control servers.

In the following, we present the few recent applications of machine learning to routing in SDN.

[207] proposes a supervised machine learning based meta-layer to solve the dynamic routing problem in real time. The authors use a heuristic algorithm's results as input to train their framework. The meta-layer architecture is quite extensible and can accommodate a variety of traffic engineering scenarios in the network. The results show that the meta-layer gives heuristic-like results. However, their proposal considers origin-destination pairs independently, hence the routes are optimized without considering the global condition of the network. As we detail later in chapter 8, this proposed scheme is not practical since the number of OD pairs (hence the number of neural networks associated) explodes in large networks.

[36] proposes a reinforcement learning based QoS-aware adaptive routing (QAR) implemented on a multi-layer hierarchical SDN control plane. QAR algorithm optimizes routing policy with regard to long-term revenue. Specifically, the softmax action selection policy, state-action-reward-state-action (SARSA) method for quality update, and Markov decision process (MDP) with QoS-aware reward function are used to realize an adaptive, QoS-provisioning routing Simulation results show that QAR provides fast convergence with QoS provisioning.

[37] proposes a machine learning based application-aware multi-path flow routing framework for SDN called AMPS. Applications generated by the devices have different bandwidth and delay requirements and compete for a constrained resource such as bandwidth or low latency path. AMPS controller prioritize flows based on application type and assign a path based on its classified priority. AMPS controller supports multipath routing using Yen-K-shortest path algorithm. The authors implemented AMPS on OpenvSwitch as a proof of concept and observed a significant improvement in comparison to traditional routing techniques.

6.3 Conclusion

In this chapter, we have reviewed research work on traffic statistics collection in SDN and applying machine learning to traffic engineering, specifically routing in SDN. It is clear that applying machine learning techniques in networking, particularly traffic engineering, is rapidly getting attention from the networking community although the number of contributions is still very limited. In the following two chapters, we present our contributions in this field: we first introduce a machine learning based traffic matrix prediction framework then we follow it by a

machine learning based predictive dynamic routing framework for SDN.

Chapter 7

Real Time Traffic Matrix Prediction for OpenFlow Networks

Summary

A Traffic Matrix (TM) is a matrix giving the traffic volumes between origin and destination nodes in a network. It has a tremendous utility for IP networks planning and management. It is widely used in network planning, resource management and network security. One of the most interesting problems revolving around traffic matrices is that these matrices are often hard to measure in real time in large operational IP networks or data centers. TM prediction is defined as the problem of estimating future network traffic matrix from the previous and

achieved network traffic data. In this chapter, we present NeuTM, a novel machine learning-based framework for predicting TM in large networks. By validating our framework on real-world data from GÉANT network, we show that our model converges quickly and gives state of the art TM prediction performance.

7.1 Introduction

Having an accurate and timely network TM is essential for most network operation/management tasks such as traffic accounting, short-time traffic scheduling or re-routing, long-term capacity planning, network design, and network anomaly detection. For example, to detect DDoS attacks in their early stage, it is necessary to be able to detect high-volume traffic clusters in real-time, which is not possible relying only on current monitoring tools. Another example is, upon congestion occurrence in the network, traditional routing protocols cannot react immediately to adjust traffic distribution, resulting in high delay, packet loss and jitter. Thanks to the early warnings, a proactive prediction-based approach would be faster, in terms of high-volume traffic detection and DDoS prevention. Similarly, predicting network congestion is more effective than reactive methods that detect congestion through measurements, only after it has significantly influenced the network operation.

Network traffic is characterized by: self-similarity, multiscalarity, long-range dependence and a highly nonlinear nature (insufficiently modeled by Poisson and Gaussian models for example). These statistical characteristics determine the traffic's predictability [171].

Several methods have been proposed for network traffic prediction and can be classified into two categories: linear prediction and nonlinear prediction. The ARMA/ARIMA model [173], [176], [178] and the Holt–Winters algorithm [173] are the most widely used traditional linear prediction methods. Nonlinear forecasting methods commonly involve neural networks (NN) [173], [179], [180]. The experimental results from [184] show that nonlinear traffic prediction based on NNs outperforms linear forecasting models (e.g. ARMA, ARAR, HW). [184] suggests that if we take into account both precision and complexity, the best results are obtained by a Feed Forward Neural Network predictor with multiresolution learning approach. However, most of the research using neural networks for network traffic prediction aims to predict the aggregate traffic value. In this work, our goal is to predict the traffic matrix which is a far more challenging task.

Unlike feed forward neural networks (FFNN), Recurrent Neural Network (RNNs) have cyclic connections over time. The activations from each time step are stored in the internal state of the network to provide a temporal memory. This capability makes RNNs better suited for sequence modeling tasks such as time series prediction and sequence labeling tasks. Particularly, Long Short-Term Memory (LSTM) is a powerful RNN architecture that was recently designed by Hochreiter and Schmidhuber [186] to address the vanishing and exploding gradient problems [177] that conventional RNNs suffer from. RNNs (including LSTMs) have been successfully used for handwriting recognition [172], language modeling, phonetic labeling of acoustic frames [181].

Our contribution in this chapter is threefold.

- First, we present, for the first time, a LSTM based framework for large scale TM prediction.
- Second, we implement our framework and deploy it on a Software Defined Network

(SDN) and train it on real world data using GÉANT data set.

- Finally, we evaluate our LSTM models at different configurations. We also compare our model to traditional models and show that LSTM models converge quickly and give state of the art TM prediction performance.

Note that we do not address the problem of TM estimation in this chapter and we suppose that historical TM data is already accurately obtained.

The remainder of this chapter is organized as follows: Section 7.2 summarizes time-series prediction techniques. LSTM architecture and equations are detailed in section 7.3. The process of feeding the LSTM model and predicting TM is described in section 7.4. Evaluation and results are presented in section 7.5. Related work is discussed in section 8.5 and the chapter is concluded by section 8.6.

7.2 Time Series Prediction

For completeness sake, we give a brief summary of various linear predictors based on traditional statistical techniques. We use the same notation and definitions as in [184] and we refer to the original paper and to [192] for a thorough background. Then we discuss NNs usage for time series prediction.

7.2.0.1 Linear Prediction

7.2.0.1.1 ARMA model The time series $\{X_t\}$ is called an ARMA(p, q) process if $\{X_t\}$ is stationary and

$$X_t - \phi_1 X_{t-1} - ... - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + ... + \theta_q Z_{t-q} \tag{7.2.1}$$

where $\{Z_t\} \approx WN(0, \sigma^2)$ is white noise with zero mean and variance σ^2 and the polynomials $\phi(z) = 1 - \phi_1 z - ... - \phi_p z^p$ and $\theta(z) = 1 + \theta_1 z + ... + \theta_q z^q$ have no common factors. Predictions can be made recursively using: $\widehat{X}_{n+1} = \begin{cases} \sum_{j=1}^{n} \theta_{nj}(X_{n+1-j} - \widehat{X}_{n+1-j}) & if 1 \leq n \leq m) \\ \sum_{j=1}^{q} \theta_{nj}(X_{n+1-j} - \widehat{X}_{n+1-j}) & \\ +\phi_1 X_n + .. + \phi_p X_{n+1-p} & if n \geq m \end{cases}$ where $m = max(p, q)$ and θ_{nj} is determined using the innovations algorithm.

7.2.0.1.2 ARAR algorithm The ARAR algorithm applies memory-shortening transformations, followed by modeling the dataset as an AR(p) process: $X_t = \phi_1 X_{t-1} + .. + \phi_p X_{t-p} + Z_t$. The time series $\{Y_t\}$ of long-memory or moderately long- memory is processed until the transformed series can be declared to be short-memory and stationary:

$$S_t = \psi(B)Y_t = Y_t + \psi_1 Y_{t-1} + ... + \psi_k Y_{t-k} \tag{7.2.2}$$

The autoregressive model fitted to the mean-corrected series

$X_t = S_t - \overline{S}, t = \overline{k+1, n}$, where $\overline{S}$

represents the sample mean for $S_{k+1}, ..., S_n$, is given by $\phi(B)X_t = Z_t$, where $\phi(B) = 1 - \phi_1 B - \phi_{l_1} B^{l_1} - \phi_{l_2} B^{l_2} - \phi_{l_3} B^{l_3}, \{Z_t\} \approx WN(0, \sigma^2)$, while the coefficients ϕ_j and the variance σ^2 are calculated using the Yule–Walker equations described in [192]. We obtain the relationship:

$$\xi(B)Y_t = \phi(1)\overline{S} + Z_t \tag{7.2.3}$$

where $\xi(B)Y_t = \psi(B)\varphi(B) = 1 + \xi_1 B + ... + \xi_{k+l_3} B^{k+l_3}$ From the following recursion relation we can determine the linear predictors

$$P_n Y_{n+h} = -\sum_{j=1}^{k+l_3} \xi P_n Y_{n+h-j} + \phi(1)\overline{S} \quad h \geq 1 \tag{7.2.4}$$

with the initial condition $P_n Y_{n+h} = Y_{n+h}$ for $h \leq 0$.

7.2.0.1.3 Holt–Winters algorithm The Holt–Winters forecasting algorithm is an exponential smoothing method that uses recursions to predict the future value of series containing a trend. If the time series has a trend, then the forecast function is:

$$\widehat{Y}_{n+h} = P_n Y_{n+h} = \widehat{a}_n + \widehat{b}_n h \tag{7.2.5}$$

where $\widehat{a}_n$ and $\widehat{b}_n$ are the estimates of the level of the trend function and the slope respectively. These are calculated using the following recursive equations:

$$\begin{cases} \widehat{a}_{n+1} = \alpha Y_{n+1} + (1-\alpha)(\widehat{a}_n + \widehat{b}_n) \\ \widehat{b}_{n+1} = \beta(\widehat{a}_{n+1} - \widehat{a}_n) + (1-\beta)\widehat{b}_n \end{cases} \tag{7.2.6}$$

Where $\widehat{Y}_{n+1} = P_n Y_{n+1} = \widehat{a}_n + \widehat{b}_n$ represents the one-step forecast. The initial conditions are: $\widehat{a}_2 = Y_2$ and $\widehat{b}_2 = Y_2 - Y_1$. The smoothing parameters α and β can be chosen either randomly (between 0 and 1), or by minimizing the sum of squared one-step errors $\sum_{i=3}^{n} (Y_i - P_{i-1} Y_i)^2$ [192].

7.2.0.2 Neural Networks for Time Series Prediction

Thanks to their strong self-learning and their ability to learn complex non-linear patterns, Neural Networks (NNs) are widely used for modeling and predicting time-series. NNs are capable of estimating almost any linear or non-linear function in an efficient and stable manner, when the underlying data relationships very complex. Unlike the techniques presented above, NNs rely on the observed data rather than on an analytical model. Furthermore, The architecture and the parameters of a NN are determined solely by the dataset.

A neural network consists of interconnected nodes, called neurons. The interconnections are weighted and the weights are also called parameters. Neurons are organized in layers: a) an input layer, b) one or more hidden layers and c) an output layer. The most popular NN architecture is feed-forward in which the information goes through the network only in the forward direction, i.e. from the input layer towards the output layer, as illustrated in figure 8.3.

7.3 Long Short Term Memory Neural Networks

FFNNs can provide only limited temporal modeling by operating on a fixed-size window of TM sequence. They can only model the data within the window and are unsuited to handle historical dependencies. By contrast, recurrent neural networks or deep recurrent neural networks (figure 7.2) contain cycles that feed back the network activations from a previous time step as inputs to influence predictions at the current time step (figure 7.3). These activations are stored in the internal states of the network as temporal contextual information [181].

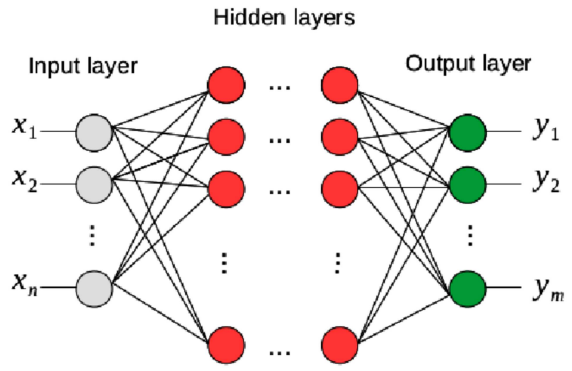


Figure 7.1: Feed Forward Deep Neural Network

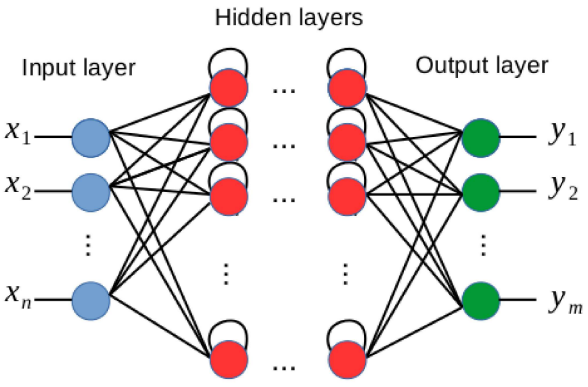


Figure 7.2: Deep Recurrent Neural Network

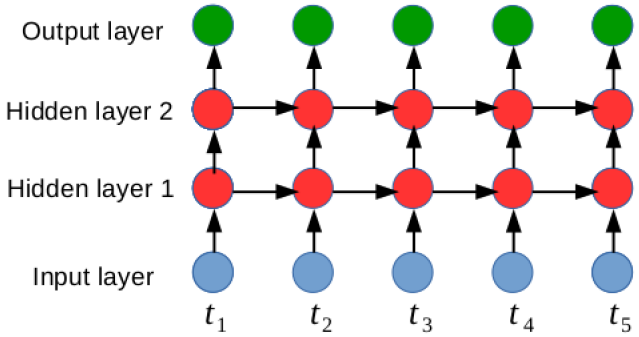


Figure 7.3: DRNN learning over time

s

However, training conventional RNNs with the gradient-based back-propagation through time (BPTT) technique is difficult due to the vanishing gradient and exploding gradient problems. The influence of a given input on the hidden layers, and therefore on the network output, either decays or blows up exponentially when cycling around the network's recurrent connections. These problems limit the capability of RNNs to model the long range context dependencies to 5-10 discrete time steps between relevant input signals and output [182].

To address these problems, an elegant RNN architecture named Long Short-Term Memory (LSTM) has been designed [186]. LSTMs and conventional RNNs have been successfully applied to sequence prediction and sequence labeling tasks. LSTM models have been shown to perform better than conventional RNNs on learning context-free and context-sensitive languages for example [175].

7.3.1 LSTM Architecture

An LSTM RNN is composed of units called memory blocks. Each memory block contains memory cells with self-connections storing (remembering) the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. Each memory block contains an input gate to control the flow of input activations into the memory cell, an output gate to control the output flow of cell activations into the rest of the network and a forget gate (figure 7.4).

The forget gate scales the internal state of the cell before adding it back to the cell as input through self recurrent connection, therefore adaptively forgetting or resetting the cell's memory. The modern LSTM architecture also contains peephole connections from its internal cells to the gates in the same cell to learn precise timing of the outputs [174].

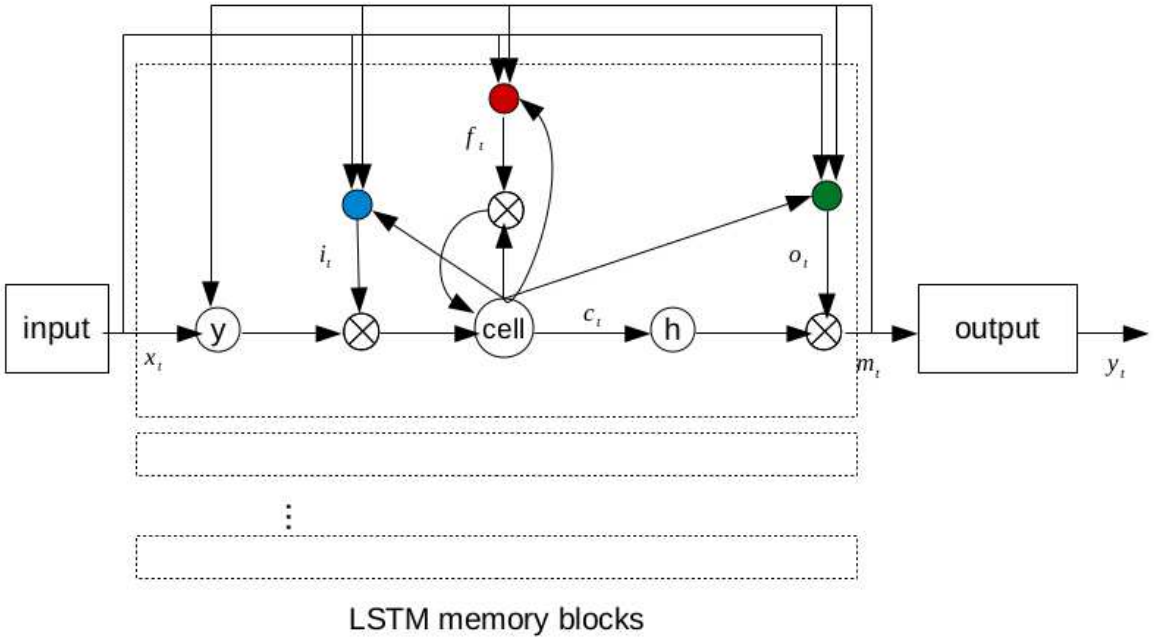


Figure 7.4: LSTM architecture

7.3.2 LSTM Equations

An LSTM network maps an input sequence $x = (x_1, ..., x_T)$ to an output sequence $y = (y_1, ..., y_T)$ by computing the network unit activations using the following equations iteratively from t = 1 to T.

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i) \tag{7.3.7}$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}m_{t-1} + W_{fc}c_{t-1} + b_f) \tag{7.3.8}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c) \tag{7.3.9}$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + b_o) \tag{7.3.10}$$

$$m_t = o_t \odot h(c_t) \tag{7.3.11}$$

$$y_t = \varphi(W_{ym}m_t + b_y) \tag{7.3.12}$$

Where i, f, o and c are respectively the input gate, forget gate, output gate and cell activation vectors. m is the output activation vector. $\odot$ is the element-wise product of the vectors. g and h are the cell input and cell output activation functions. tanh and φ are the network output activation function. The b terms denote bias vectors and the W terms denote weight matrices. and σ is the logistic sigmoid function [181].

7.4 Traffic Matrix Prediction Using LSTM RNN

We train a deep LSTM architecture with a deep learning method (backpropagation through time algorithm) to learn the traffic characteristics from historical traffic data and predict the future TM.

7.4.1 Problem Statement

Let N be the number of nodes in the network. The N-by-N traffic matrix is denoted by Y such as an entry y_{ij} represents the traffic volume flowing from node i to node j. We add the time dimension to obtain a structure of N-by-N-by-T tensor (vector of matrices) S such as an entry s_{ij}^t represents the volume of traffic flowing from node i to node j at time t, and T is the total number of time-slots. The traffic matrix prediction problem is defined as solving the predictor of Y^t (denoted by $\widehat{Y}^t$) via a series of historical and measured traffic data set (Y^{t-1}, Y^{t-2}, Y^{t-3}, ..., Y^{t-T}). The main challenge here is how to model the inherent relationships among the traffic data set so that one can exactly predict Y^t.

7.4.2 Feeding The LSTM RNN

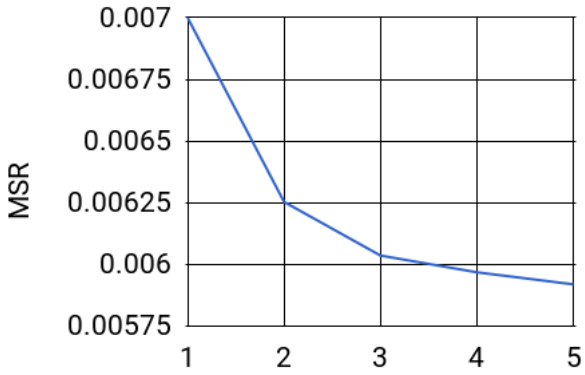


Figure 7.5: MSE over number of hidden layers (500 nodes each)

To effectively feed the LSTM RNN, we transform each matrix Y^t to a vector X^t (of size $N \times N$) by concatenating its N rows from top to bottom. X^t is called traffic vector (TV). Note that x_n entries can be mapped to the original y_{ij} using the relation $n = i \times N + j$. Now the traffic matrix prediction problem is defined as solving the predictor of X^t (denoted by $\widehat{X}^t$) via a series of historical measured traffic vectors (X^{t-1}, X^{t-2}, X^{t-3}, ..., X^{t-T}).

One possible way to predict the traffic vector X^t is to predict one component x_n^t at a time by feeding the LSTM RNN one vector $(x_0^t, x_1^t, ..., x_{N^2}^t)$ at a time. This is based on the assumption that each OD traffic is independent from all other ODs which was shown to be

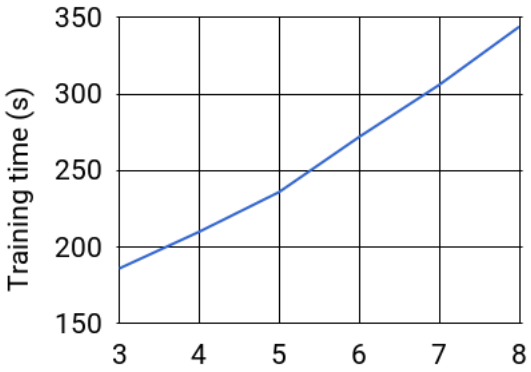


Figure 7.6: Training time over network depth (20 epochs)

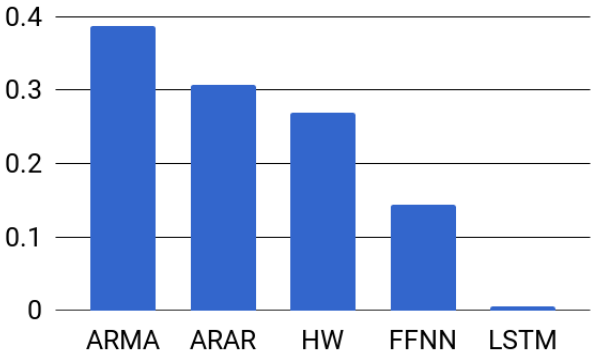


Figure 7.7: Comparison of prediction methods

wrong by [193]. Hence, considering the previous traffic of all ODs is necessary to obtain a more correct and accurate prediction of the traffic vector. **Continuous Prediction Over Time**: Real-time prediction of traffic matrix requires continuous feeding and learning. Over time, the total number of time-slots become too big resulting in high computational complexity. To cope with this problem, we introduce the notion of learning window (denoted by W) which indicates a fixed number of previous time-slots to learn from in order to predict the current traffic vector X^t (Fig. 8.2). We construct the W-by-N^2 traffic-over-time matrix (that we denote by M) by putting together W vectors (X^{t-1}, X^{t-2}, X^{t-3}, ..., X^{t-W}) ordered in time. Note that $T \geq W$ (T being the total number of historical matrices) and the number of matrices M is equal to T/W.

7.4.3 Performance Metric

To quantitatively assess the overall performance of our LSTM model, Mean Square Error (MSE) is used to estimate the prediction accuracy. MSE is a scale dependent metric which

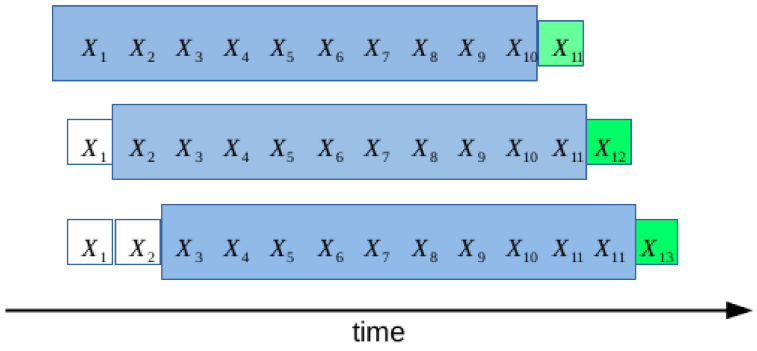


Figure 7.8: Sliding learning window

quantifies the difference between the forecasted values and the actual values of the quantity being predicted by computing the average sum of squared errors:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y}_i)^2 \tag{7.4.13}$$

where y_i is the observed value, $\widehat{y}_i$ is the predicted value and N represents the total number of predictions.

7.5 Experiments and Evaluation

We implemented NeuTM as a traffic matrix prediction application on top of POX controller [189]. NeuTM's LSTM model is implemented using Keras library [208] on top of Google's TensorFlow machine learning framework [190]. We evaluate the prediction accuracy of our method using real traffic data from the GÉANT backbone networks [212] made up of 23 peer nodes interconnected using 38 links (as of 2004). 2004-timeslot traffic matrix data is sampled from the GÉANT network by 15-min interval [188] for several months.

To evaluate our method on short term traffic matrix prediction, we consider a set of 309 traffic matrices. As detailed in section 7.4.2, we transform the matrices to vectors of size 529 each and we concatenate the vectors to obtain the traffic-over-time matrix M of size 309×529. We split M into two matrices, training matrix M_{train} and validation matrix M_{test} of sizes 263×529 and 46×529 consecutively. M_{train} is used to train the LSTM model and M_{test} is used to evaluate and validate its accuracy. Finally, We normalize the data by dividing by the maximum value.

Figure 7.5 depicts the MSE obtained over different numbers of hidden layers (depths). The prediction accuracy is better with deeper networks. Figure 7.6 depicts the variation of the training time over different depths. Note that it takes less than 5 minutes to train a 6 layers network on 20 epochs. Finally, figure 7.7 compares the prediction error of the different prediction methods presented in this chapter and shows the superiority of LSTM.

7.6 Related Work

Various methods have been proposed to predict traffic matrix. [184] evaluates and compares traditional linear prediction models (ARMA, ARAR, HW) and neural network based prediction with multi-resolution learning. The results show that NNs outperform traditional linear prediction methods which cannot meet the accuracy requirements. [193] proposes a FARIMA predictor based on an α-stable non-Gaussian self-similar traffic model. [191] compares three prediction methods: Independent Node Prediction (INP), Total Matrix Prediction with Key Element Correction (TMP-KEC) and Principle Component Prediction with Fluctuation Component Correction (PCP-FCC). INP method does not consider the correlations among the nodes, resulting in unsatisfying prediction error. TMP-KEC method reduces the forecasting error of key elements as well as that of the total matrix. PCP-FCC method improves the overall prediction error for most of the OD flows.

7.7 Conclusion

In this work, we have shown that LSTM architectures are well suited for traffic matrix prediction. We have proposed a data pre-processing and RNN feeding technique that achieves high prediction accuracy in a very short training time. The results of our evaluations show that LSTMs outperforms traditional linear methods and feed forward neural networks by many orders of magnitude.

Chapter 8

Predictive Dynamic Routing for OpenFlow Networks

Summary

Current SDN/OpenFlow controllers use a default routing based on Dijkstra's algorithm for shortest paths, and provide APIs to develop custom routing applications. In this chapter, we introduce NeuRoute, a dynamic routing framework for SDN entirely based on machine learning, specifically, Deep Neural Networks. NeuRoute is a controller-agnostic dynamic routing framework that (i) predicts traffic matrix in real time, (ii) uses a neural network to learn traffic characteristics and (iii) generates forwarding rules accordingly to optimize the network throughput. NeuRoute achieves the same results as the most efficient dynamic routing heuristic but in much less execution time.

8.1 Introduction

The modern Internet is experiencing an explosion of the Machine-to-Machine (M2M) communications and Internet-of-Things (IoT) applications, in addition to other bandwidth intensive applications such as voice over IP (VoIP), video conferencing and video streaming services. Thus leading to a high pressure on carrier operators to increase their network capacity in order to support all these applications with an acceptable Quality of Service (QoS). The common practice to ensure a good QoS so far is to over-provision network resources. Operators over-provision a network so that capacity is based on peak traffic load estimates. Although this approach is simple for networks with predictable peak loads, it is not economically justified in the long-term.

In addition, most ISP networks today use Shortest Path First (SPF) routing algorithms, namely the Open Shortest Path First (OSPF) [194]. OSPF routes packets statically by assigning weights to links hence the routing tables are recalculated only when a topology change occurs. OSPF is a best effort routing protocol, meaning that when a packet experiences congestion, the routing subsystem cannot send it through an alternate path, thus failing to provide desired QoS during congestion even when the total traffic load is not particularly high.

Although OSPF has a QoS extension [195] that dynamically changes link weights based on measured traffic, it is still not implemented in the Internet for two major reasons. First, changing the cost of a link in one part of the network may cause a lot of routing updates and in turn negatively affect traffic in a completely different part of the network. This can be disruptive to many (or all) traffic flows. Another problem concerns routing loops that may occur before the routing protocol converges. Therefore, in networks with distributed control plane, changing the link cost is considered just as disruptive as link-failures. On the other hand, without the possibility to differentiate between traffic flows more granularly (not only based on destination IP address), dynamic routing cannot positively contribute to load balancing [196].

The dynamic routing problem, also known as QoS routing or concurrent flow routing, is a case of Multi-commodity flow problem where flows are packets or traffic flows and the goal is to maximize the total network flow while respecting routing constraints such as load balancing the total network traffic or minimizing the traffic delay. Due to their high computational complexity, multi-commodity flow algorithms are rarely implemented in practice.

There are many variants of the dynamic routing problem including the maximum throughput dynamic routing, the maximum throughput minimum cost dynamic routing and the maximum throughput minimum cost multicast dynamic routing. In this work, we focus on the maximum throughput minimum cost unicast dynamic routing where given a traffic demand matrix, the objective is to maximize the total throughput of the network while minimizing the cost of routing the total traffic knowing that each flow can be routed through only one end-to-end path. We present NeuRoute, a Neural Network based hyperheuristic that is capable of computing dynamic paths in real time. NeuRoute learns from a dynamic routing algorithm then imitates it achieving the same results but in only 25% of its execution time. The basic

motivation behind NeuRoute is that dynamic routing using traditional algorithmic solutions is not practical due to their high computational complexity. That is, at every execution round the routing algorithm uses measured link loads as input and performs a graph search to find the near optimal paths.

The main contributions of this chapter are summarized as follows: (i) We introduce for the first time an integral routing system based on machine learning and detail its architecture, (ii) we detail the design of the neural network responsible for matching traffic demands to routing paths and (iii) we evaluate our proposal against an efficient dynamic routing heuristic and show our solution's superiority.

The remainder of this chapter is organized as follows: Section 8.2 formally states the dynamic routing problem and discusses its most prominent heuristic solutions. Section 8.3 details NeuRoute design. In section 8.4, we evaluate NeuRoute on real world network data and topology and we conclude the paper in section 8.6

8.2 The Dynamic Routing Problem

In this section, we formulate the maximum throughput minimum cost dynamic routing problem (MT-MC-DRP) as a linear program, and then prove its NP-completeness. The problem is equivalent to the known Unsplittable Constrained Multicommodity Max-Flow-Min-Cost problem. We want to find routings for multiple unicast flows which maximizes the aggregate flow in a graph, while minimizing the routing-cost. By focusing on unsplittable multicommodity flow we exclude multipath routing where a flow can be split and routed through multiple end-to-end paths.

We consider a software-defined network $G(V, L)$, where V is the set of SDN-enabled switch nodes, and L is the set of links that connect the switches where each link $l_{i,j}$ has a capacity $C(l)$. Each unicast flow f has source and destination nodes denoted s_f and d_f respectively, a requested traffic rate R^f and a minimum necessary traffic rate N^f. Let $r^f_{in}(v)$ and $r^f_{out}(v)$ denote the aggregate flow rate into/out of node v due to flow f, respectively. The traffic rate related to flow f and flowing through link l is denoted by $r^f(l)$. Each ink has a routing cost denoted by $\Theta(l)$ that can represent any linear function of the traffic flowing on it , i.e., delay, jitter, congestion probability or reliability. We define an Admissible Routing as an assignment of flows to the links in G, such that no capacity constraints are violated, and flow-conservation applies at every node. The MT-MC-DRP problem can be stated as follows: Does there exist an admissible routing for the flows, where each flow receives its requested rate r^f while the total routing cost is minimized?

8.2.1 MT-MC-DRP As Two Linear Problems

We formulate MT-MC-DRP as a succession of two linear problems (LPs): A Constrained-Maximum-Flow LP (CMaxF-LP) and a Constrained-Minimum-Cost LP (CMinC-LP).

8.2.1.1 CMaxF-LP

$$maximize(\sum_{f \in F} r^f_{in}(d_f)) \tag{8.2.1}$$

subject to:

$$r^f(l) \geq 0 \qquad \forall f \in F, \forall l \in L^f \tag{8.2.2}$$

$$r^f(l) \leq C(l) \quad \forall f \in F, \forall l \in L^f \tag{8.2.3}$$

$$\sum_{f \in F} r^f(l) \leq C(l) \qquad \forall l \in L \tag{8.2.4}$$

$$r^f_{in}(v) = r^f_{out}(v) \qquad \forall f \in F, \forall v \in V^f - \{s_f, d_f\} \tag{8.2.5}$$

$$r^f_{in}(s_f) = 0 \qquad \forall f \in F \tag{8.2.6}$$

$$r^f_{out}(d_f) = 0 \qquad \forall f \in F \tag{8.2.7}$$

$$r^f_{out}(s_f) \leq R^f \qquad \forall f \in F \tag{8.2.8}$$

$$r^f_{out}(s_f) \geq N^f \qquad \forall f \in F \tag{8.2.9}$$

8.2.1.2 CMinC-LP

$$minimize(\sum_{f \in F} \sum_{l \in L} r^f(l) \times \Theta(l)) \tag{8.2.10}$$

subject to:

$$r^f_{out}(s_f) = \Pi^f + / - \epsilon \qquad \forall f \in F, \forall l \in L^f \tag{8.2.11}$$

$$\sum_{f \in F} r^c(l) \leq C(l) \qquad \forall l \in L \tag{8.2.12}$$

$$r^f_{in}(v) = r^f_{out}(v) \qquad \forall f \in F, \forall v \in V' \tag{8.2.13}$$

$$r^f_{in}(s_f) = 0 \qquad \forall f \in F \tag{8.2.14}$$

$$r^f_{out}(d_f) = 0 \qquad \forall f \in F \tag{8.2.15}$$

Theorem. The Maximum Throughput Minimum Cost Dynamic Routing Problem as presented above is NP-hard.

Proof. refer to [198] [202] □

8.2.2 Heuristic Solution for The MT-MC-DRP

Due to its NP-completeness, an exact solution for the MT-MC-DRP as defined above is not practical to be implemented in the network controller. It is more practical to design an approximate but fast solution. Therefore, a major research effort was put into designing efficient fully polynomial-time approximation schemes (FPTAS) for multicommodity flow problems including max flow min cost multicommodity problem. A fully polynomial-time approximation scheme for a flow maximization problem is an algorithm that, given an accuracy parameter $\epsilon > 0$, computes, in polynomial time in the size of the input and $1/\epsilon$, a solution with an objective value within a factor of $(1 - \epsilon)$ of the optimal one [199]. The multicommodity problem literature has a rich body of work providing FPTASes. In this work, we use the novel method proposed in [199] as a baseline heuristic to solve the MT-MC-DRP. We also refer to the same paper for more literature on other existing heuristics.

8.3 System Design

As shown in figure 8.1, NeuRoute is designed as an integral routing application for the SDN controller. NeuRoute is composed of three key components: a Traffic Matrix Estimator (TME), a Traffic Matrix Predictor (TMP) and a Traffic Routing Unit (TRU). In this chapter, we focus on and detail the TRU but also describe briefly the two other components for the sake of completeness.

8.3.1 Traffic Matrix Estimator

As mentioned earlier,detailed design of the traffic matrix (TM) estimator is out of the scope of this chapter. Here we only motivate the need for a traffic matrix estimator and define its interfaces with the rest of NeuRoute components.

A network TM presents the traffic volume between all pairs of origin-destination (OD) nodes of the network at a certain time t. The nodes in a traffic matrix can be Points- of-Presence (PoPs), switches, routers or links. In OpenFlow SDNs, the controller leverages packet_in messages to build a global view of the network. When a new flow arrives to a switch, it is matched against forwarding rules to determine a forwarding path for it. If the flow does not match any rule, the switch forwards the first packet or only the packet header to the controller. In addition, the controller can query switches for packet counts that track the number of packets and bytes handled by the switch. However, the number of packet_in and the number of controller queries, necessary for a near real-time measurement, increases rapidly with a large number of switches and flows, making this measurement mechanism not practical. Also, there is a chance that by the time the controller receives the message, the values of the counters become out of date and do not reflect the near real-time state of the switch anymore. These and a number of other issues listed in [28] call for an efficient measurement mechanism

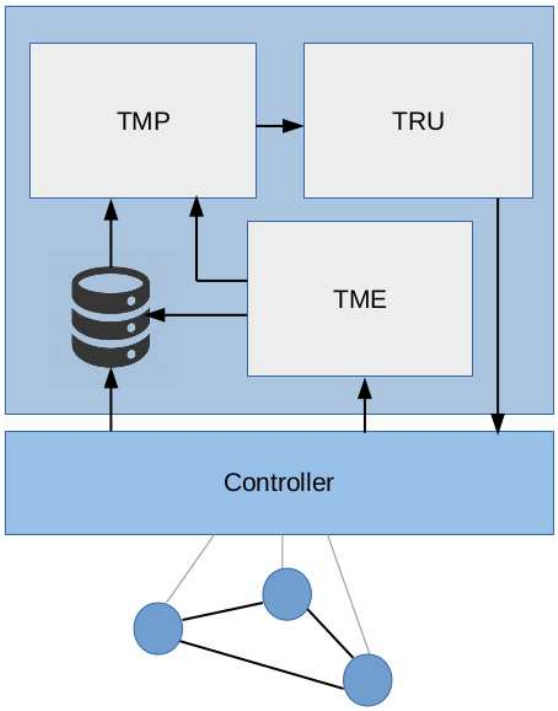


Figure 8.1: NeuRoute architecture

to capture traffic matrix in near real-time. In its current implementation, NeuRoute uses a variant of a recent proposal called openMeasure [25] to estimate traffic matrix.

8.3.2 Traffic Matrix Predictor

Network Traffic Matrix prediction refers to the problem of estimating future network traffic from the past and current network traffic data. Internet traffic is known to be self-similar enabling it to be predictable with high accuracy [171]. NeuRoute's Traffic Matrix Predictor (TMP) uses a Long Short Term Memory Recurrent Neural Network (LSTM-RNN) described in [200]. Figure 8.2 shows the sliding prediction window where at each time instant t, the TMP takes a fixed size set of achieved traffic matrices as input and outputs the traffic matrix of time instant $t + 1$

Prediction using NNs involves two phases: a) the training phase and b) the test (prediction) phase. During the training phase, the NN is supervised to learn from the data by presenting the training data at the input layer and dynamically adjusting the parameters of the NN to achieve the desired output value for the input set. The most commonly used learning algorithm to train NNs is called the backpropagation algorithm. The underlying idea is to propagate the error backward, from the output to the input, where the weights are changed continuously until the output error falls below a preset value. In this way, the NN learns correlated patterns

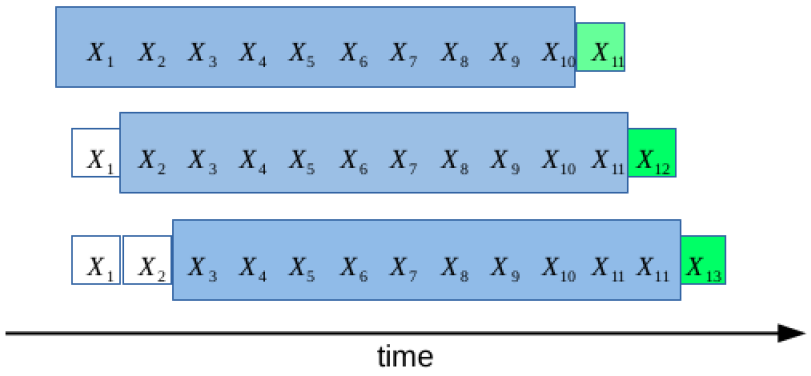


Figure 8.2: Traffic Matrix Prediction Over Time

between input sets and the corresponding target val- ues. The prediction phase represents the testing of the NN. A new unseen input is presented to the NN and the output is calculated, thereby predicting the outcome of new input data.

8.3.3 Traffic Routing Unit

The core component of the NeuRoute system is the Traffic Routing Unit (TRU) which is responsible of selecting optimal routes based on the predicted traffic matrix. TRU is based on the supervised learning approach where an agent is trained to infer a function from labeled training data. It consists of a Deep Feed Forward Neural Network that learns to match traffic demands to routing paths by observing the output of a heuristic, that we call the Baseline Heuristic (BH). In this chapter we present our experimentations with a BH that is built following the algorithm discussed in section 8.2.2.

To bootstrap, only the TME is activated to continuously provide the BH with timely estimated traffic matrices. Copies of these estimated traffic matrices are stored to be used later on by the TMP and the TRU. NeuRoute collects the output of the BH for a period of time that can be configured based on the desired performance. Once enough BH-output data is gathered, NeuRoute's components, TMP and TRU, are fired up. The TMP uses the stored history of estimated traffic matrices to predict the future traffic matrix, continuously as detailed in [200]. On the other hand, the TRU takes the BH output data and the stored history of estimated traffic matrices along with corresponding Network States (NSs) as input to train its routing neural network. Each tuple (NS+traffic matrix, BH output) constitutes one learning sample for the TRU. NS at a time instant t (or NS_t) is the set of all links available capacities and links costs at time instant t (links costs usually do not change frequently). Once the learning phase is done (within a few seconds to a few minutes depending on the volume of data and desired performance), the trained model is fired up to route new traffic flows. The reason why we predict the traffic matrix is that the real-time measurement of traffic matrix is not practical and by the time the controller gets the measured information, the flows to be routed are already on their way on the existing paths, before even the controller computes the

new paths. In the following, we detail the design elements and the design challenges of TRU.

8.3.3.1 Deep Feed Forward Neural Networks

Deep neural networks are currently the most successful machine learning technique for solving a variety of tasks including language translation, image classification and image generation. TRU is similar to an image classifier that has a set of images in input and tries to find a function that matches these images to a set of classes. In the routing case, the traffic matrices are the images and the routing paths represent the output classes. The deep neural network used in TRU is presented in figure 8.3. It takes a traffic matrix and an NS instance as input and matches them to a unique path y_i as output.

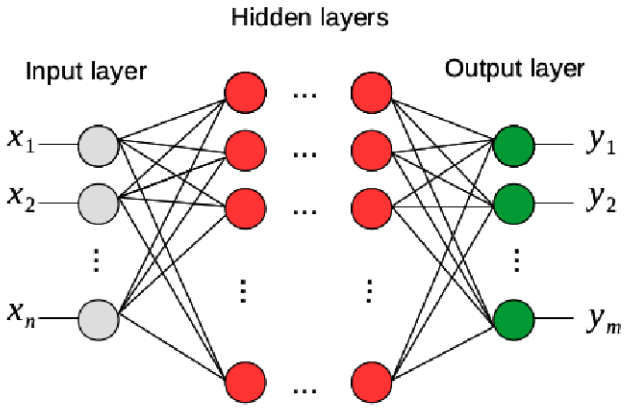


Figure 8.3: Deep Feed Forward Neural Network

In a deep feed forward network, the information flows only forward through the network from the input nodes, through the hidden nodes to the output nodes, with no cycles or loops. Each node has an activation function which acts like a threshold for the node to fire up: A node n produces a value for its output nodes only if the weighted sum of the input values of n is equal or exceeds the threshold. Each edge has a weight and permits transfer of value from node to node.

Learning Algorithm. We use the Backpropagation learning algorithm that was first introduced in the 70s and now is the most widely used algorithm for supervised learning in deep feed-forward networks. The goal is to make the network learn some target function, in our case, matching traffic matrices to routing paths. The basic idea of the algorithm is to look for the minimum of the error function in weight space by repeatedly applying the chain rule to compute the influence of each weight in the network with respect to the error function: The output values of the network are compared with the learning sample (correct answer) to compute the value of the error function. The calculated error is then fed back through the network and used to adjust the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large

number of training cycles, the network will usually converge to some state where the error is small enough. In other words, we say that the network has learned the target function to some extend. We refer to [204] for more details about the algorithm.

Optimization Algorithm. In this work, we use Adam (short for Adaptive Moment Estimation) optimizer, one of the most adopted optimization algorithms among deep learning practitioners for applications in computer vision and natural language processing. Adam optimizer is an improvement of the gradient descent algorithm that can yield quicker convergence in training deep networks [205].

Learning Rate. The learning rate determines how quickly or how slowly we want the network weights to be updated (by the backpropagation algorithm). In other words, how quickly or how slowly we want the network to forget learned features and learn new ones. Picking a learning rate is problem dependent since the optimum learning rate can differ based on a number of parameters including epoch size, number of learning iterations, number of hidden layers and/or neurons and number and format of the inputs. Trial and error is often used in order to determine the ideal learning condition for each problem studied. We describe our empirical approach for choosing the learning rate in the implementation section 8.4.

8.3.3.2 Input Pre-Processing and Normalization

The input (NS+traffic matrix) are merged into one single vector of numbers then normalized by dividing all numbers by the greatest number. The result is a vector of numbers ranging between 0 and 1. This normalization is a good practice that can make training faster and reduce the chance of getting stuck in local optima [201].

8.3.3.3 Routing Over Time

At each time instant t, the TRU's trained model takes predicted traffic matrix of time instant $t+1$ (TM_{t+1}) and corresponding NS as input. The model function is applied and the output is a set of path probabilities where the highest value indicates the best routing path. TRU then sends the chosen path to the controller in order to be installed in switches as flow rules. By the time $t+1$, when the flows arrive, the forwarding rules are already installed which minimizes considerably the network delay.

Matching traffic matrices and network states to routing paths is similar to classifying a stream of frames in a video, witch is not a common and well studied problem since the usual image classification is applied to individual images. Besides tweaking the neural network architecture and parameters to obtain a high classification performance, there are two unique challenges that arise in our problem:

- The runtime performance of the trained model is critical and needs to be optimized to perform continuous routing over time. We achieve high performance by keeping the

predicted traffic matrices in memory before feeding them to the LRU's neural network.

- Unlike images and videos, there is no camera bias in traffic matrices (Camera bias refers to the fact that in many images and videos, the object of interest often occupies the center region), hence it is not possible to work around resolutions to optimize training time as it was done in [203].

8.4 Implementation and Evaluation

We implemented NeuRoute as a routing application on top of POX controller [189]. The TRU's neural network is implemented using Keras library [208] on top of Google's TensorFlow machine learning framework [209]. We have chosen the GÉANT network topology for our testbed as GÉANT's traffic matrices are already available online [210]. We implemented the GÉANT topology (shown in figure 8.4) as an SDN network using Mininet [211] setting link capacities at 10Mbps. We use link delay as the cost function with 2ms delay per link.

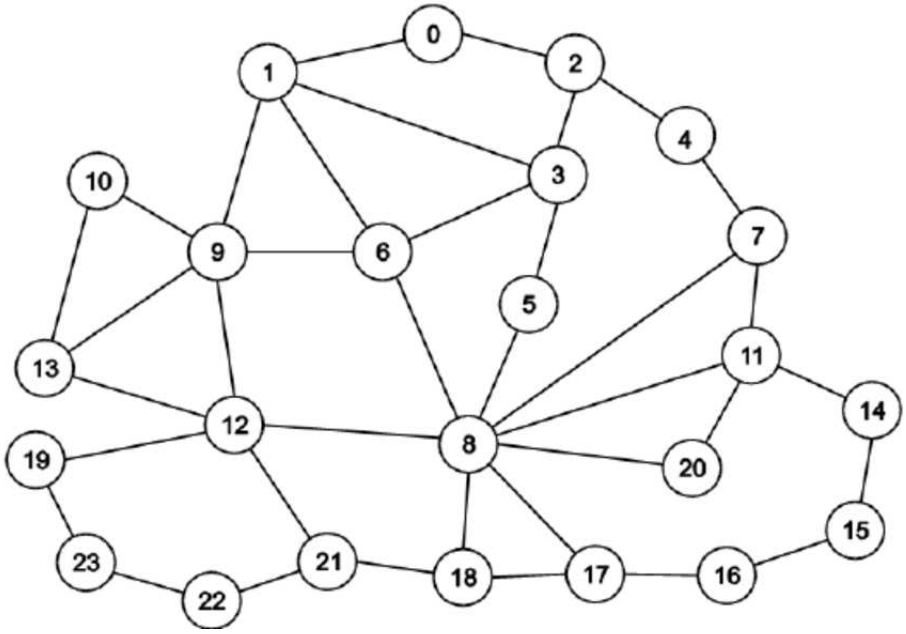
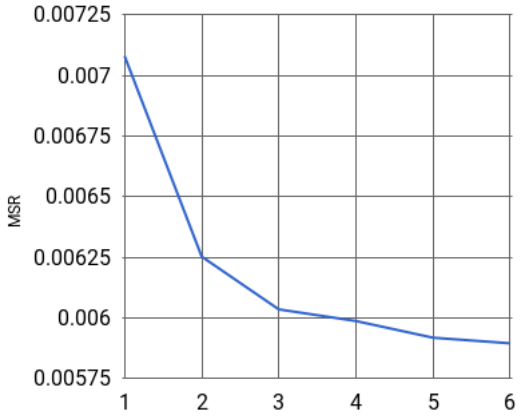


Figure 8.4: GÉANT2 Network Topology [212]

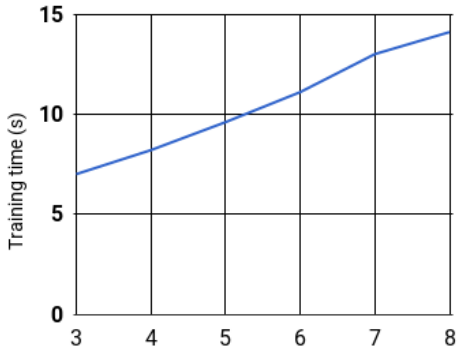
Data generation. In order to generate the learning data, we applied the BH on the testbed described above with GÉANT's traffic matrices as input. We obtained a data set of 10000 samples (traffic matrix+network state, near optimal path) that we split to training data set of 7000 samples and test data set of 3000 samples.

The neural network architecture. Determining the neural network architecture is problem dependent, hence we adopted an empirical approach to determine the number of

hidden layers and the size of each hidden layer. We measured the training time and the learning performance (GÉANT traffic matrices + related network states as input and the results of the BH as output) for different numbers of hidden layers and different hidden layer sizes. This allowed us to pick an optimal number of hidden layers of 6 with 100 nodes per hidden layer. Note that we choose the architecture parameters based on the measured learning performance, and we stop experimenting when the training time becomes too long.

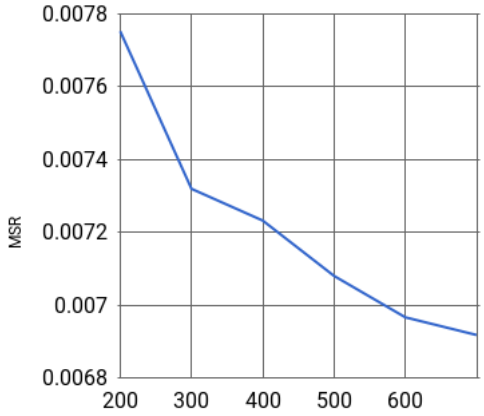


(a) MSR over number of hidden layers

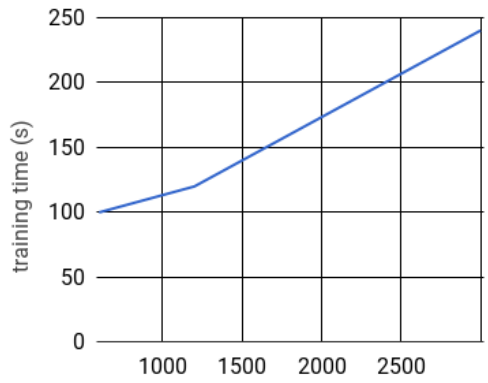


(b) Training time over number of hidden layers

Figure 8.5: Picking the number of hidden layers



(a) Training time over number of hidden nodes



(b) Training time over number of hidden nodes

Figure 8.6: Picking the number of hidden nodes

Figure 8.5a depicts the measured MSR over different numbers of hidden layers. The MSR diminishes at high numbers of hidden layers (deep network) but figure 8.5b shows that the deeper is the network the longer it takes to train it. To select a good compromise, we fix the training time to 2 minutes. This training time corresponds to a depth of 6 hidden layers.

Similarly, figure 8.6a shows that the MSR diminishes at higher network sizes but the training time goes up as figure 8.6b shows. We fix again the training time to 2 minutes and obtain the corresponding hidden nodes number of 600, or 100 nodes per hidden layer. Note that a 2 minutes training time is not too long but is chosen proportionally to the size of the data set. Larger data sets may take hours or days to train.

Data preparation. We prepared the input data as follows: we split the total learning data into batches of size 100 each. Each input sample is a vector of size $506 + 38 = 544$, 506 being the size of a vector representing one traffic matrix of 23 nodes (23*22) and 38 being the number of links in the GÉANT topology, which is equal to the size of one network state vector. The output vector is of size $23 * 22 * 5$ with 23*22 being the number of origin-destination (OD) pairs and we arbitrarily fix the number of possible paths per OD pair to 6.

The learning rate. Like the neural network's architecture, the learning rate is problem dependent. Our approach is to start with a high value and go down to lower values, recording the learning performance and training time for every learning rate value.

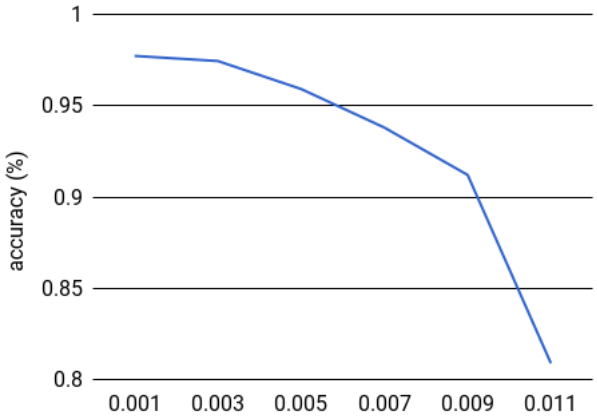


Figure 8.7: Accuracy over different learning rate values

Figure 8.7 depicts the MSR variation over different learning rate values. The training time does not change for different learning rates (5s per epoch).

The overfitting problem. Overfitting is a serious problem that occurs when training a neural network on limited data. It happens when a model learns the detail and noise in the training data to the extent that it negatively impacts its performance on new data. This means that the noise or random fluctuations in the training data is picked up and learned as features by the model. The problem is that these features do not apply to new data and negatively impact the model's ability to generalize. Various methods have been proposed to avoid or reduce overfitting, including stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization and Dropout [213]. In this work, we use the Dropout technique which is proven to be the most effective [213]. Dropout is a technique that addresses both these issues.

It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. The term "dropout" refers to dropping out units (hidden and visible) in a neural network.

Evaluation of TRU. Finally, we applied the trained model on the test data and recorded the accuracy (number of correctly chosen paths from the test set) over number of training epochs in figure 8.8. One epoch is a one complete training pass over the whole training data set where each epoch takes roughly 2s to complete. Figure 8.8 shows that the model picks the near optimal path learned from the BH with an estimated error of less than 0.05% when trained well (3min of training is enough to reach this error rate). Furthermore, the trained model executes and finds the near optimal path in 30ms compared to the BH execution time of 120ms.

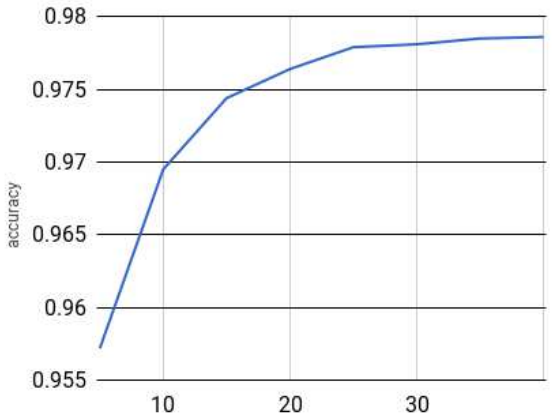


Figure 8.8: Accuracy over number of training epochs

8.5 Related Work

The authors of paper [207] propose a machine learning meta-layer composed of multiple modules. Each module works only for one OD pair. The proposed scheme is however not practical since the number of OD pairs (hence the number of neural networks associated) explodes in large networks. Knowing that each neural network is trained separately and each trained model operates separately, this approach does not capture the relations between ODs requests that arrive at the same time. It is also much more complicated to implement and computationally expensive than our approach.

8.6 Conclusion

In this chapter, we introduced NeuRoute, a machine learning based dynamic routing framework for SDN. NeuRoute learns a routing algorithm and imitates it with higher performance. We implemented NeuRoute as a routing application on top of Pox Controller and performed proof of concept experiments that showed our solution's superiority compared to an efficient dynamic routing heuristic. Experiments on larger data sets are being conducted and will be presented in a future work along with more details about the system.

Part V

Conclusion

Chapter 9

Conclusions and Future Work

Summary

In this chapter, we present the general conclusions of this manuscript and then list a number of perspectives for future work.

9.1 Conclusions

The Telecom and ICT sector is witnessing an unprecedented techno-economic shift due to the emerging Network Softwarization technologies. By separating the hardware on which network functions/services run and the software that realizes and controls the network functions/services, Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are creating an open ecosystem that drastically reduces the cost of building networks and changes the way operators operate their networks. SDN and NFV paradigms add more flexibility and enable more control over networks, thus, related technologies are expected to dominate a large part of the networking market in the next few years (estimated at USD 3.68B in 2017 and forecasted to reach $54B by 2022 at a Compound Annual Growth Rate (CAGR) of 71.4%).

However, one of the major operators' concerns about Network Softwarization is security. In the first part of this thesis, we have addressed some of the most sensitive security issues in SDN. First, we have designed and implemented a pentesting (penetration testing) framework for SDN controllers. We have proposed a set of algorithms to fingerprint a remote SDN

controller without having direct connection to it. Using our framework, network operators can evaluate the security of their SDN deployments (including Opendaylight, Floodlight and Cisco Open SDN Controller) before putting them into production. Second, we have studied the Topology Discovery problem in SDN controllers and discovered major security (as well as performance) issues around the current de-facto OpenFlow Topology Discovery Protocol (OFDP). In order to fix these major issues, we have designed and implemented a new secure and efficient OpenFlow Topology Discovery Protocol (called sOFTDP). sOFTDP requires minimal changes to the OpenFlow switch design and is shown to be more secure than previous workarounds on traditional OFDP. Also, sOFTDP outperforms OFDP by several orders of magnitude which we confirmed by extensive experiments.

In the second part of this thesis, we have proposed a novel traffic engineering scheme for softwarized networks. Inspired by the recent breakthroughs in machine learning techniques, notably, Deep Neural Networks (DNNs), we have built a traffic engineering engine for SDN called NeuRoute, entirely based on DNNs. Current SDN/OpenFlow controllers use a default routing based on Dijkstra's algorithm for shortest paths, and provide APIs to develop custom routing applications. NeuRoute is a controller-agnostic dynamic routing framework that (i) predicts traffic matrix in real time, (ii) uses a neural network to learn traffic characteristics and (iii) generates forwarding rules accordingly to optimize the network throughput. NeuRoute is composed of two main components: NeuTM and NeuRoute-TRU. NeuTM is a traffic matrix (TM) prediction framework that uses Long-Short Term Memory (LSTM) Neural Network architecture to learn long-range traffic dependencies and characteristics then accurately predicts future TMs. NeuRoute-TRU is a path selection engine that computes optimal paths for traffic matrices predicted by NeuTM. NeuRoute-TRU achieves the same results as the most efficient dynamic routing heuristic but in much less execution time.

9.2 Future Work

Several future works can be added to this study to advance both security and traffic engineering in softwarized networks.

First, to further investigate and harden the security of SDN and NFV, the SDN controller fingerprinting framework can be extended by fingerprinting a larger set of controllers and designing more techniques for fingerprinting controllers. Also, formal methods could be investigated to evaluate fingerprinting techniques and how they can be possibly combined to increase success rate. Furthermore, various security countermeasures must be explored and deployed to harden the security of SDN networks against controller fingerprinting and subsequent attacks. On the topology discovery side, our protocol sOFDTP needs to be evaluated at the scale of large software defined data center networks before pushing it into a standardization process. Designing new attacks on sOFTDP would improve its security as well.

Second, our machine learning based traffic engineering system NeuRoute is being tested on large SDNs. We aim to collaborate with industrial partners to test it in their data center

and campus networks. On the research side, we see a great potential in using reinforcement learning for traffic engineering in SDN, especially when the learning data is too large to be labeled, which is mostly the case in networking.

9.3 Publications

Azzouni, Abdelhadi, et al. "Fingerprinting OpenFlow controllers: The first step to attack an SDN control plane."Global Communications Conference (GLOBECOM). IEEE, 2016.

Azzouni, Abdelhadi, et al. "Limitations of openflow topology discovery protocol." Ad Hoc Networking Workshop (Med-Hoc-Net), 2017 16th Annual Mediterranean. IEEE, 2017.

Azzouni, Abdelhadi, Raouf Boutaba, and Guy Pujolle. "NeuRoute: Predictive Dynamic Routing for Software-Defined Networks." 4th International Workshop on Management of SDN and NFV Systems (ManSDN/NFV)-CNSM, IEEE, 2017.

Azzouni, Abdelhadi, et al. "sOFTDP: Secure and Efficient OpenFlow Topology Discovery Protocol." IEEE/IFIP NOMS 2018. to appear.

Azzouni, Abdelhadi, and Guy Pujolle. "NeuTM: A Neural Network-based Framework for Traffic Matrix Prediction in SDN." IEEE/IFIP NOMS 2018. to appear.

Bibliography

[1] BERDE, Pankaj, GEROLA, Matteo, et al. ONOS: towards an open, distributed SDN OS. In : Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014. p. 1-6.

[2] FIELDING, Roy. Representational state transfer. Architectural Styles and the Design of Netowork-based Software Architecture, 2000, p. 76-85.

[3] YAP, Kok-Kiong, HUANG, Te-Yuan, DODSON, Ben, et al. Towards software-friendly networks. In : Proceedings of the first ACM asia-pacific workshop on Workshop on systems. ACM, 2010. p. 49-54.

[4] KREUTZ, Diego, RAMOS, Fernando MV, VERISSIMO, Paulo Esteves, et al. Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 2015, vol. 103, no 1, p. 14-76.

[5] Open Networking Foundation (ONF), 2014. [Online]. Available: https://www.opennetworking.org/

[6] LARA, Adrian, KOLASANI, Anisha, et RAMAMURTHY, Byrav. Network innovation using openflow: A survey. IEEE communications surveys & tutorials, 2014, vol. 16, no 1, p. 493-512.

[7] ALLIANCE, N. G. M. N. Further study on critical C-RAN technologies. Next Generation Mobile Networks, 2015.

[8] HAERICK, W. et GUPTA, M. White Paper: 5G and the Factories of the Future. 5G-PPP, Tech. Rep, 2015.

[9] SDXCentral. Virtual Versus Reality: The Challenges of Enterprise NFV Adoption. https://www.sdxcentral.com/articles/contributed/the-challenges-of-enterprise-nfv-adoption/2017/10/

[10] ETSI GS NFV. Network Functions Virtualisation (NFV) Release 3; Security; Security Management and Monitoring specification. http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/013/03.01.01_60/gs_NFV-SEC013v030101p.pdf

[11] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, X. Xiao, Overview and Principles of Internet Traffic Engineering, RFC 3272, Tech. Rep., May 2002

[12] Ivan Pepelnjak. Traffic engineering the service provider network. searchtelecom.techtarget.com (http://searchtelecom.techtarget.com/feature/Traffic-engineering-the-service-provider-network)

[13] Akyildiz, Ian F., et al. "A roadmap for traffic engineering in SDN-OpenFlow networks." Computer Networks 71 (2014): 1-30.

[14] Shu, Zhaogang, et al. "Traffic engineering in software-defined networking: Measurement and management." IEEE Access 4 (2016): 3246-3256.

[15] MCKEOWN, Nick, ANDERSON, Tom, BALAKRISHNAN, Hari, et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, vol. 38, no 2, p. 69-74.

[16] B. Niven-Jenkins, D. Brungard, M. Betts, N. Sprecher, S. Ueno, Requirements of an MPLS Transport Profile, RFC 5654, Tech. Rep., September 2009.

[17] A.R. Curtis, W. Kim, P. Yalagandula, Mahout: low-overhead datacenter traffic management using end-host-based elephant detection, April 2011, pp. 1629–1637.

[18] Netflow. http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.html

[19] sflow. http://www.sflow.org/sFlowOverview.pdf.

[20] A.C. Myers, Jflow: practical mostly-static information flow control, in: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'99, January 1999, pp. 228–241.

[21] S.R. Chowdhury, M.F. Bari, R. Ahmed, R. Boutaba, Payless: a low cost network monitoring framework for software defined networks, in: Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium, NOMS'14, May 2014.

[22] A. Tootoonchian, M. Ghobadi, Y. Ganjali, Opentm: traffic matrix estimator for openflow networks, in: Proceedings of the 11th International Conference on Passive and Active Measurement, PAM'10, April 2010, pp. 201–210.

[23] M. Malboubi, L. Wang, C.-N. Chuah, and P. Sharma, "Intelligent SDN based traffic (de) aggregation and measurement paradigm (iSTAMP)," in Proc. IEEE Conf. Comput. Commun. (INFOCOM), Apr./May 2014, pp. 934–942.

[24] Gong, Yanlei, et al. "Towards accurate online traffic matrix estimation in software-defined networks." Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. ACM, 2015.

[25] Liu, Chang, AMehdi Malboubi, and Chen-Nee Chuah. "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN." Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on. IEEE, 2016.

[26] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H.V. Madhyastha, Flowsense: monitoring network utilization with zero measurement cost, in: Proceedings of the 14th International Conference on Passive and Active Measurement, PAM'13, March 2013, pp. 31–41.

[27] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with opensketch, in: Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI'13, vol. 13, April 2013, pp. 29–42.

[28] J. Suh, T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for sdn," IBM Research Report, January 2014.

[29] Techcrunch. Google's AlphaGo AI beats the world's best human Go player. https://techcrunch.com/2017/05/23/googles-alphago-ai-beats-the-worlds-best-human-go-player/

[30] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: a reinforcement learning approach," in Advances in Neural Inform. Process. Syst., J. Jack, D. Cowan, G. Tesauro, and J. Alspector, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1994, vol. 6, pp. 671–678.

[31] S. P. M. Choi and D. Y. Yeung, "Predictive q-routing: a memory-based reinforcement learning approach to adaptive traffic control," in Advances in Neural Inform. Process. Syst., D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. Cambridge, MA, USA: The MIT Press, 1996, vol. 8, pp. 945–951.

[32] A. Nowe, K. Steenhaut, M. Fakir, and K. Verbeeck, "Q-learning for adaptive load based routing," in Proc. IEEE Int. Conf. Syst., Man, and Cybern., vol. 4, San Diego, CA, USA, Oct. 1998, pp. 3965–3970.

[33] L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing," in Proc. Int. Joint Conf. Neural Netw., vol. 2, Honolulu, HI, USA, May 2002, pp. 1825–1830.

[34] Soroush Haeri. Applications of Reinforcement Learning to Routing and Virtualization in Computer Networks. 2016. PHD thesis. School of Engineering Science Faculty of Applied Science, B. Eng., Multimedia University, Malaysia.

[35] Yanjun, Li, Li Xiaobo, and Yoshie Osamu. "Traffic engineering framework with machine learning based meta-layer in software-defined networks." Network Infrastructure and Digital Content (IC-NIDC), 2014 4th IEEE International Conference on. IEEE, 2014.

[36] Lin, Shih-Chun, et al. "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach." Services Computing (SCC), 2016 IEEE International Conference on. IEEE, 2016.

[37] Pasca, S. Thomas Valerrian, Siva Sairam Prasad Kodali, and Kotaro Kataoka. "AMPS: Application aware multipath flow routing using machine learning in SDN. " Communications (NCC), 2017 Twenty-third National Conference on. IEEE, 2017.

[38] Valadarsky, Asaf, et al. "A machine learning approach to routing." arXiv preprint arXiv:1708.03074 (2017).

[39] Feamster, Nick, Jennifer Rexford, and Ellen Zegura. "The road to SDN." Queue 11.12 (2013): 20.

[40] Harvey Freeman and Raouf Boutaba. Networking industry transformation through softwarization. IEEE Communications Magazine, August 2016.

[41] B. Han, V. Gopalakrishnan, L. S. Ji, and S. J. Lee, "Network Function Virtualization: Challenges and Opportunities for Innovations," IEEE Communications Magazine, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[42] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, R. Natella, F. Jiang, and P. Wang, "Network Function Virtualization: Challenges and Directions for Reliability Assurance," in ISSREW, Nov. 2014.

[43] Open Networking Foundation (ONF). https://www.opennetworking.org/.

[44] Open Networking Foundation (ONF), "Software-Defined Networking: The New Norm for Networks," Apr. 2012.

[45] ETSI NFV ISG, "Network Functions Virtualization Introductory White Paper: An Introduction, Benefits, Enablers, Challenges & Call for Action," in SDN and OpenFlow World Congress, Oct. 2012.

[46] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: State of the Art, Challenges, and Implementation in Next Generation Mobile Networks (vEPC)," IEEE Network, vol. 28, no. 6, pp. 18–26, Nov./Dec. 2014.

[47] Alcatel-Lucent White Paper, "Network Functions Virtualization: Challenges and Solutions," Jun. 2013.

[48] D. F. Macedo, D. Guedes, L. F. M. Vieira, M. A. M. Vieira, and M. Nogueira, "Programmable Networks-From Software-Defined Radio to Software-Defined Networking," IEEE Communications Surveys & Tutorials, vol. 17, no. 2, pp. 1102–1125, May 2015.

[49] A.-T. Campbell, I. Katzela, K. Miki, and J. Vicente, "Open signaling for ATM, internet and mobile networks (OPENSIG'98)," ACM SIGCOMM Computer Communication Review, vol. 29, no. 1, pp. 97–108, Jan. 1999.

[50] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A Survey of Active Network Research," IEEE Communications Magazine, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[51] S. da Silva, Y. Yemini, and D. Florissi, "The NetScript Active Network System," IEEE Journal on Selected Areas in Communications, vol. 19, no. 3, pp. 538–551, Mar. 2001.

[52] A.-T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, "A Survey of Programmable Networks," ACM SIGCOMM Computer Communication Review, vol. 29, no. 2, pp. 7–23, Apr. 1999.

[53] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," IEEE Communications Surveys & Tutorials, vol. 16, no. 3, pp. 1617–1634, Aug. 2014.

[54] K. L. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz, "Directions in Active Networks," IEEE Communications Magazine, vol. 36, no. 10, pp. 72–78, Oct. 1998.

[55] J. M. Smith and S. M. Nettles, "Active Networking: One View of the Past, Present, and Future," IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 34, no. 1, pp. 4–18, Feb. 2004.

[56] D.-L. Tennenhouse and D. J. Wetherall, "Towards an Active Network Architecture," ACM SIGCOMM Computer Communication Review, vol. 37, no. 5, pp. 81–94, Oct. 2007.

[57] K. Calvert, "Reflections on Network Architecture: an Active sNetworking Perspective," ACM SIGCOMM Computer Communication Review, vol. 36, no. 2, pp. 27–30, Apr. 2006.

[58] J. E. Van der Merwe, S. Rooney, I. Leslie, and S. Crosby, "The Tempest-A Practical Framework for Network Programmability," IEEE Network, vol. 12, no. 3, pp. 20–8, May/Jun. 1998.

[59] E. Haleplidis, J. H. Salim, J. M. Halpern, S. Hares, K. Pentikousis, K. Ogawa, W.-M. Wang, S. Denazis, and O. Koufopavlou, "Network Programmability With ForCES," IEEE Communications Surveys & Tutorials, vol. 17, no. 3, pp. 1423–1440, Aug. 2015.

[60] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," in Internet Engineering Task Force (IETF), Apr. 2004, pp. 1–40.

[61] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI), vol. 2, 2005, pp. 15–28.

[62] J. Vasseur and J. L. Roux, "Path Computation Element (PCE) Com- munication Protocol (PCEP)," Internet Engineering Task Force (IETF), pp. 1–87, Mar. 2009.

[63] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Rexford, J.and Xie, H. Yan, J.-B. Zhan, and H. Zhang, "A Clean Slate 4D Approach to Network Control and Man-

agement," ACM SIGCOMM Computer Communication Review, vol. 35, no. 5, pp. 41–54, Oct. 2005.

[64] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A Protection Architecture for Enterprise Networks," in Proceedings of the 15th conference on USENIX Security Symposium (USENIX-SS'), vol. 15, no. 10, 2006, pp. 1–15.

[65] M. Casado, J. Freedman, M. J.and Pettit, J.-Y. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," ACM SIGCOMM Computer Communication Review, vol. 37, no. 4, pp. 1–12, Oct. 2007.

[66] O.S.Specification, "1.5.1," 2015, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf.

[67] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," IEEE Communica- tions Surveys & Tutorials, vol. 16, no. 4, pp. 2181–2206, Nov. 2014.

[68] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[69] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," in Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation (OSDI), vol. 10, 2010.

[70] Open Network Operating System (ONOS), http://onosproject.org/.

[71] V. Bollapragada, R. White, and C. Murphy, Inside Cisco IOS Software Architecture. Cisco Press, 2008.

[72] Junos OS, http://www.juniper.net/us/en/products-services/nos/junos/.

[73] ExtremeXOS, http://www.extremenetworks.com/product/extremexos- network-operating-system.

[74] Service Router Operating System (SR OS), https://www.alcatel- lucent.com/products/sros.

[75] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[76] "European telecommunications standards institute, the network functions virtualization industry specification group," http://www.etsi.org/technologies-clusters/technologies/nfv.

[77] ETSI NFV ISG, “Network Functions Virtualization Update White Paper: Network Operator Perspectives on Industry Progress,” in SDN and OpenFlow World Congress, Oct. 2013.

[78] ETSI NFV ISG, “Network Functions Virtualization White Paper 3: Network Operator Perspectives on Industry Progress,” in SDN and OpenFlow World Congress, Oct. 2014.

[79] Y. Jarraya, T. Madi, and M. Debbabi, “A Survey and a Layered Taxonomy of Software-Defined Networking,” IEEE Communications Surveys & Tutorials, vol. 16, no. 4, pp. 1955–1980, Nov. 2014.

[80] W.-F. Xia, Y.-G. Wen, C.-H. Foh, D. Niyato, and H.-Y. Xie, “A Survey on Software-Defined Networking,” IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 27–51, Mar. 2015.

[81] ETSI. "Network Functions Virtualization (NFV) Management and Orchestration,” Dec. 2014.

[82] “ETSI Group Specification: Network Functions Virtualization (NFV) Virtualization Requirements,” Oct. 2013.

[83] “ETSI Group Specification: Network Functions Virtualization (NFV) Resiliency Requirements,” Jan. 2015.

[84] J.-L. Izquierdo-Zaragoza, A. Fernandez-Gambin, J.-J. Pedreno- Manresa, and P. Pavon-Marino, “Leveraging Net2Plan planning tool for network orchestration in OpenDaylight,” in SaCoNeT, Jun. 2014.

[85] S. Ristov, M. Gusev, and A. Donevski, “Security Vulnerability Assessment of OpenStack Cloud,” in CICSyN, May 2014.

[86] A. Mayoral, R. Vilalta, R. Munoz, R. Casellas, R. Martinez, and J. Vilchez, “Integrated IT and network orchestration using OpenStack, OpenDaylight and active stateful PCE for intra and inter data center connectivity,” in ECOC, Sep. 2014.

[87] OPNFV. https://www.opnfv.org/.

[88] G. Monteleone and P. Paglierani, “Session Border Controller Virtualization Towards “Service-Defined" Networks Based on NFV and SDN,” in SDN4FNS, Nov. 2013.

[89] G.-Y. Liu and T. Wood, “Cloud-Scale Application Performance Monitoring with SDN and NFV,” in IC2E, Mar. 2015.

[90] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, “Toward a Software-Based Network: Integrating Software Defined Networking and Network Function Virtualization,” IEEE Network, vol. 29, no. 3, pp. 36–41, May/Jun. 2015.

[91] S. M. M. Gilani, T. Hong, and G.-F. Zhao, “SN-FMIA: SDN and NFV enabled Future Mobile Internet Architecture,” in ICACT, Jul. 2015.

[92] N. Omnes, M. Bouillon, G. Fromentoux, and O. Le Grand, "A Programmable and Virtualized Network & IT Infrastructure for the Internet of Things," in ICIN, Feb. 2015.

[93] K. Giotis, Y. Kryftis, and V. Maglaris, "Policy-based Orchestration of NFV Services in Software-Defined Networks," in NetSoft, Apr. 2015.

[94] Google, "Inter-Datacenter WAN with centralized TE using SDN and OpenFlow," 2012, pp. 1–2.

[95] S. Jain et al., "B4: Experience with a Globally-Deployed Software Defined WAN," ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pp. 3–14, Oct. 2013.

[96] "The cloudnfv project," 2013, http://www.cloudnfv.com/.

[97] "The cloudnfv project white paper," 2013, http://www.cloudnfv.com/WhitePaper.pdf.

[98] M. Dillon and T. Winters, "Virtualization of home network gateways," Computer, no. 11, pp. 62–65, 2014.

[99] Z. Bronstein and E. Shraga, "Nfv virtualisation of the home environment," in Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th. IEEE, 2014, pp. 899–904.

[100] M. Ibanez, N. M. Madrid, and R. Seepold, "Virtualization of Residential Gateways," in WISES, Jun. 2007.

[101] V. Aggarwal, V. Gopalakrishnan, R. Jana, K. K. Ramakrishnan, and V. A. Vaishampayan, "Optimizing Cloud Resources for Delivering IPTV Services Through Virtualization," IEEE Transactions on Mul- timedia, vol. 15, no. 4, pp. 789–801, Jun. 2013.

[102] "ETSI Group Specification: Network Functions Virtualization (NFV) Use Cases," Oct. 2013.

[103] I. Giannoulakis, E. Kafetzakis, G. Xylouris, G. Gardikis, and A. Kourtis, "On the Applications of Efficient NFV Management Towards 5G Networking," in 5GU, Nov. 2014.

[104] J. Soares, C. Goncalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a Telco Cloud Environment for Service Functions," IEEE Communications Magazine, vol. 53, no. 2, pp. 98–106, Feb. 2015.

[105] C. Liang, F. R. Yu, and X. Zhang, "Information-Centric Network Function Virtualization over 5G Mobile Wireless Networks," IEEE Network, vol. 29, no. 3, pp. 68–74, May/Jun. 2015.

[106] Huawei. "Observation to NFV, White Paper" Nov. 2014.

[107] "ETSI Group Specification: Network Functions Virtualization (NFV) NFV Security and Trust Guidance," Dec. 2014.

[108] J. Keeney, S. van der Meer, and L. Fallon, "Towards Real-time Management of Virtualized Telecommunication Networks," in CNSM, Nov. 2014.

[109] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford, "A slick control plane for network middleboxes," in Proceedings of ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013, pp. 147–148.

[110] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions," in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013, pp. 19–24.

[111] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in ACM SIGCOMM computer communication review, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[112] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in HotSDN. ACM, 2012, pp. 127–132.

[113] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in Local Computer Networks (LCN), 2010 IEEE 35th Conference on. IEEE, 2010, pp. 408–415.

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

[115] J. Naous et al., "Delegating Network Security with More Information," in Proceedings of the 1st ACM workshop on Research on Enterprise Networking (WREN), Aug. 2009, pp. 19–26.

[116] R. Skowyra, S. Bahargam, and A. Bestavros, "Software-defined ids for securing embedded mobile devices," in High Performance Extreme Computing Conference (HPEC), 2013 IEEE. IEEE, 2013, pp. 1–7.

[117] A. Goodney, S. Narayan, V. Bhandwalkar, and Y. H. Cho, "Pattern based packet filtering using netfpga in deter infrastructure," in 1st Asia NetFPGA Developers Workshop, Daejeon, Korea, 2010.

[118] S.-A. Mehdi, J. Khalid, and S.-A. Khayam, "Revisiting Traffic Anomaly Detection using Software Defined Networking," in Proceed- ings of the 14th international conference on Recent Advances in Intrusion Detection (RAID), 2011, pp. 161–180.

[119] C. C. Liang and F. R. Yu, "Wireless Network Virtualization: A Survey, Some Research Issues and Challenges," IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 358–380, Aug. 2015.

[120] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 236–262, Jan. 2016.

[121] Y.-D. Lin et al., "An Extended SDN Architecture for Network Function Virtualization with a Case Study on Intrusion Prevention," IEEE Network, vol. 29, no. 3, pp. 48–53, May/Jun. 2015.

[122] Y. Li and M. Chen, "Software-Defined Network Function Virtualization: A Survey," IEEE Access, vol. 3, pp. 2542–2553, Dec. 2015.

[123] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," IEEE Communications Surveys & Tutorials, DOI:10.1109/COMST.2015.2453114.

[124] "ETSI Group Specification: Network Functions Virtualization (NFV) NFV Security Problem Statement," Oct. 2014.

[125] "ETSI Group Specification: Network Functions Virtualization (NFV) NFV Security and Trust Guidance," Dec. 2014.

[126] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013, pp. 165–166.

[127] R. Kloti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in Network Protocols (ICNP), 2013 21st IEEE International Conference on. IEEE, 2013, pp. 1–6.

[128] J. Leng, Y. Zhou, J. Zhang, and C. Hu, "An inference attack model for flow table capacity and usage: Exploiting the vulnerability of flow table overflow in software-defined network," arXiv preprint arXiv:1504.03095, 2015.

[129] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, "A Survey of Securing Networks Using Software Defined Networking," IEEE Transactions on Reliability, vol. 64, no. 3, pp. 1086–1097, Sep. 2015.

[130] R. Sherwood et al., "FlowVisor: A Network Virtualization Layer," OPENFLOW-TR-2009-1, pp. 1–14, Oct. 2009.

[131] A. Al-Shabibi et al., "OpenVirteX: Make Your Virtual SDNs Programmable," in HotSDN, 2014, pp. 25–30.

[132] D. Drutskoy, E. Keller, and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," IEEE Internet Computing, vol. 17, no. 2, pp. 20–27, Mar/Apr. 2013.

[133] A. Feghali, R. Kilany, and M. Chamoun, "SDN Security Problems and Solutions Analysis," in ICPE and NTDS, Jul. 2015, pp. 1–5.

[134] Alcatel-Lucent. Providing Security in NFV: Challenges and Opportunities, White Paper. May 2014.

[135] Ayadi, I. and Diaz, G. and Simoni, N. "QoS-based Network Virtualization to Future Networks: An approach based on network constraints ". Fourth International Conference on the Network of the Future (NOF). Oct, 2013.

[136] A. Akhunzada, E. Ahmed, A. Gani, M. Khan, M. Imran, and S. Guizani, “Securing Software Defined Networks: Taxonomy, Requirements, and Open Issues,” IEEE Communications Magazine, vol. 53, no. 4, pp. 36–44, Apr. 2015.

[137] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, “Security in Software Defined Networks: A Survey,” IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2317–2346, Nov. 2015.

[138] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, “A Replication Component for Resilient OpenFlow-based Networking,” in IEEE Network Operations and Management Symposium (NOMS), Apr. 2012, pp. 933–939.

[139] E. P. Krishna, E. Sandhya, and M. G. Karthik, “Managing ddos attacks on virtual machines by segregated policy management,” Global Journal of Computer Science and Technology, vol. 14, no. 6-E, p. 19, 2014.

[140] A. Kamisinski and C. Fung, “FlowMon: Detecting Malicious Switches in Software-Defined Networks,” in Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense (SafeConfig), Oct. 2015, pp. 39–45.

[141] “ETSI Group Specification: Network Functions Virtualization (NFV) Infrastructure Compute Domain,” Dec. 2014.

[142] “ETSI Group Specification: Network Functions Virtualization (NFV) Infrastructure Hypervisor Domain,” Jan. 2015.

[143] “ETSI Group Specification: Network Functions Virtualization (NFV) Infrastructure Network Domain,” Dec. 2014.

[144] Open Networking Foundation. "SDN Security Considerations in the Data Center", Version 1.4.0 (Wire Protocol 0x05). October 14, 2013

[145] NMAP. https://nmap.org/

[146] OWASP Zed Attack Proxy Project. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy

[147] Open Networking Foundation. "Software-Defined Networking". https://www.opennetworking.org/sdn-resources/sdn-definition.

[148] Open Networking Foundation. "OpenFlow". https://www.opennetworking.org/sdn-resources/openflow.

[149] Open Networking Foundation. "OpenFlow Switch Specification", Version 1.5.0 (Wire Protocol 0x06). December 19, 2014.

[150] Bifulco, Roberto, et al. "Fingerprinting software-defined networks." 2015 IEEE 23rd International Conference on Network Protocols (ICNP). IEEE, 2015.

[151] Azzouni, Abdelhadi, et al. "Fingerprinting OpenFlow controllers: The first step to attack an SDN control plane." Global Communications Conference (GLOBECOM), 2016 IEEE. IEEE, 2016.

[152] Project Floodlight. https://floodlight.atlassian.net/wiki/display/floodlight- controller/Supported+Topologies

[153] Linux Foundation. "OpenDaylight". https://www.opendaylight.org/.

[154] What is Beacon?. https://openflow.stanford.edu/display/Beacon/Home/

[155] Floodlight. http://Floodlight.openflowhub.org/

[156] Azzouni, Abdelhadi, et al. "Limitations of OpenFlow Topology Discovery Protocol." arXiv preprint arXiv:1705.00706 (2017).

[157] Hong, Sungmin, et al. "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures." NDSS. 2015.

[158] Congdon, P. (2002). Link layer discovery protocol and MIB. V1. 0 May 20. 2002, http://www. IEEE802.

[159] Alharbi, Talal, Marius Portmann, and Farzaneh Pakzad. "The (In) Security of Topology Discovery in Software Defined Networks. " Local Computer Networks (LCN), 2015 IEEE 40th Conference on. IEEE, 2015.

[160] Pakzad, Farzaneh, et al. "Efficient topology discovery in software defined networks." Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on. IEEE, 2014.

[161] Erickson, David. "The beacon openflow controller." In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pp. 13-18. ACM, 2013.

[162] Ryu. http://osrg.github.com/ryu/

[163] Cisco. "Cisco Open SDN Controller". http://www.cisco.com/c/en/us/products/cloud-systems-management/open-sdn-controller/index.html

[164] Azzouni, A et al. "Fingerprinting OpenFlow controllers: The first step to attack an SDN control plane". GLOBECOM. 2016.

[165] IXIA and NEC. "White paper: SDN Controller Testing, Part 1". https://www.necam.com/docs/?id=2709888a-ecfd-4157-8849-1d18144a6dda

[166] IETF, Bidirectional Forwarding Detection (BFD), https://tools.ietf.org/html/rfc5880

[167] Tarnaras, George, Evangelos Haleplidis, and Spyros Denazis. "SDN and ForCES based optimal network topology discovery. " Network Softwarization (NetSoft), 2015 1st IEEE Conference on. IEEE, 2015.

[168] Antonenko, Vitaly, and Ruslan Smelyanskiy. "Global network modelling based on mininet approach." Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013.

[169] Pfaff, Ben, et al. "The design and implementation of open vswitch." 12th USENIX symposium on networked systems design and implementation (NSDI 15). 2015.

[170] IXIA and NEC. ”White paper: SDN Controller Testing, Part 1”. https://www.necam.com/docs/?id=2709888a-ecfd-4157-8849-1d18144a6dda

[171] W. Leland, M. Taqqu, W. Willinger and D. Wilson, “On the self-similar nature of Ethernet traffic,” In Proc. SIGCOMM ’93, pp.183–193, 1993.

[172] Liwicki, Marcus, et al. "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks." Proc. 9th Int. Conf. on Document Analysis and Recognition. Vol. 1. 2007.

[173] P. Cortez, M. Rio, M. Rocha, P. Sousa, Internet Traffic Forecasting using Neural Networks, International Joint Conference on Neural Networks, pp. 2635–2642. Vancouver, Canada, 2006.

[174] Felix A. Gers, Nicol N. Schraudolph, and Jurgen Schmidhuber, “Learning precise timing with LSTM recurrent networks,” Journal of Machine Learning Research , vol. 3, pp. 115–143, Mar. 2003

[175] Felix A. Gers and Jurgen Schmidhuber, “LSTM recurrent networks learn simple context free and context sensitive languages,” IEEE Transactions on Neural Networks , vol. 12, no. 6, pp. 1333–1340, 2001

[176] H. Feng, Y. Shu, Study on Network Traffic Prediction Techniques, International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1041–1044. Wuhan, China, 2005.

[177] Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6.02 (1998): 107-116.

[178] J. Dai, J. Li, VBR MPEG Video Traffic Dynamic Prediction Based on the Modeling and Forecast of Time Series, Fifth International Joint Conference on INC, IMS and IDC, pp. 1752–1757. Seoul, Korea, 2009.

[179] V. B. Dharmadhikari, J. D. Gavade, An NN Approach for MPEG Video Traffic Prediction, 2nd International Conference on Software Technology and Engineering, pp. V1-57–V1-61. San Juan, USA, 2010.

[180] A. Abdennour, Evaluation of neural network architectures for MPEG-4 video traffic prediction, IEEE Transactions on Broadcasting, Volume 52, No. 2, pp. 184–192. ISSN 0018-9316, 2006.

[181] Sak, Hasim, Andrew W. Senior, and Françoise Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." Interspeech. 2014.

[182] Sak et al. Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling. https://arxiv.org/pdf/1402.1128.pdf

[183] Alex Graves. Supervised Sequence Labelling with Recurrent Neural Networks. http://www.cs.toronto.edu/ graves/phd.pdf

[184] Barabas, Melinda, et al. "Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition." Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on. IEEE, 2011.

[185] H. Feng, Y. Shu, Study on Network Traffic Prediction Techniques, International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1041–1044. Wuhan, China, 2005.

[186] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[187] https://www.geant.org/Projects/GEANT_Project_GN4

[188] Uhlig, Steve, et al. "Providing public intradomain traffic matrices to the research community." ACM SIGCOMM Computer Communication Review 36.1 (2006): 83-86.

[189] https://openflow.stanford.edu/display/ONL/POX+Wiki

[190] Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

[191] Liu, Wei, et al. "Prediction and correction of traffic matrix in an IP backbone network." Performance Computing and Communications Conference (IPCCC), 2014 IEEE International. IEEE, 2014.

[192] P. J. Brockwell, R. A. Davis, Introduction to Time Series and Forecast- ing, Second Edition. Springer-Verlag,ISBN 0-387-95351-5, 2002.

[193] Wen, Yong, and Guangxi Zhu. "Prediction for non-gaussian self-similar traffic with neural network." Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on. Vol. 1. IEEE, 2006.

[194] Moy, John. "OSPF version 2." (1997).

[195] QoS Routing Mechanisms and OSPF Extensions, IETF, RFC 2676, Aug. 1999.

[196] Tomovic, Slavica, et al. "A new approach to dynamic routing in SDN networks." Electrotechnical Conference (MELECON), 2016 18th Mediterranean. IEEE, 2016.

[197] Szymanski, Ted H. "Max-flow min-cost routing in a future-Internet with improved QoS guarantees." IEEE Transactions on Communications 61.4 (2013): 1485-1497.

[198] Hall, Alex, Steffen Hippler, and Martin Skutella. "Multicommodity flows over time: Efficient algorithms and complexity." Theoretical Computer Science 379.3 (2007): 387-404.

[199] Madry, Aleksander. "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms." Proceedings of the forty-second ACM symposium on Theory of computing. ACM, 2010.

[200] Azzouni, Abdelhadi, and Guy Pujolle. "A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction." arXiv preprint arXiv:1705.05690 (2017).

[201] Sola, J., and J. Sevilla. "Importance of input data normalization for the application of neural networks to complex industrial problems." IEEE Transactions on Nuclear Science 44.3 (1997): 1464-1468.

[202] Szymanski, Ted H. "Max-flow min-cost routing in a future-Internet with improved QoS guarantees." IEEE Transactions on Communications 61.4 (2013): 1485-1497.

[203] Karpathy, Andrej, et al. "Large-scale video classification with convolutional neural networks." Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2014.

[204] Demuth, Howard B., et al. Neural network design. Martin Hagan, 2014.

[205] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[206] Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection." Ijcai. Vol. 14. No. 2. 1995.

[207] Yanjun, Li, Li Xiaobo, and Yoshie Osamu. "Traffic engineering framework with machine learning based meta-layer in software-defined networks." Network Infrastructure and Digital Content (IC-NIDC), 2014 4th IEEE International Conference on. IEEE, 2014.

[208] Keras Documentation. https://keras.io/

[209] Google TensorFlow. https://www.tensorflow.org/

[210] https://goo.gl/JD6t78

[211] mininet. http://mininet.org/

[212] Barreto, Fernando, Emílio CG Wille, and Luiz Nacamura Jr. "Fast emergency paths schema to overcome transient link failures in ospf routing." arXiv preprint arXiv:1204.2465 (2012).

[213] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research 15.1 (2014): 1929-1958.