UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LEONARDO RICHTER BAYS

Virtual Network Embedding in Software-Defined Networks

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Luciano Paschoal Gaspary

Porto Alegre
April 2018

CIP — CATALOGING-IN-PUBLICATION

Bays, Leonardo Richter

Virtual Network Embedding in
Software-Defined Networks / Leonardo Richter Bays. – Porto Alegre: PPGC da UFRGS, 2018.

111 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Advisor: Luciano Paschoal Gaspary.

1. Network Virtualization. 2. Virtual Network Embedding. 3. Software-Defined Networking. 4. Privacy. I. Gaspary, Luciano Paschoal. II. Título.

“But someday you will notice
on those shoulders of yours
there are wings to guide you
to the far-off future.”
— Neko Oikawa

AGRADECIMENTOS / *ACKNOWLEDGMENTS*

Due to the need to thank people from all sorts of different places, I will be mixing English and Portuguese here. English sections, such as this one, are written in italics.

Agradeço primeiramente à minha família, por toda a ajuda e todo o apoio que recebi ao longo da minha vida. Agradeço, também, aos amigos que, de uma forma ou de outra, me deram forças para continuar seguindo meu caminho.

Agradeço aos colegas e amigos de graduação, mestrado e doutorado, em especial aos colegas de laboratório ao longo de todos esses anos. Nossa convivência tornou essa jornada muito mais agradável, e a troca de experiências foi fundamental para o desenvolvimento desse trabalho.

Agradeço aos professores da UFRGS e do Unilasalle Canoas com quem tive o privilégio de trabalhar – em especial aos professores Luciano Gaspary, Marinho Barcellos, Luciana Buriol, Lisandro Granville, Mozart Siqueira, Gaspare Bruno e Marcos Barreto. A orientação e o apoio que recebi foram imprescindíveis para o meu crescimento pessoal, minha formação como pesquisador e, além disso, para garantir a qualidade desse trabalho.

Agradeço, ainda, à equipe de técnicos administrativos do Instituto de Informática da UFRGS, que desempenham papeis fundamentais para garantir a excelência do mesmo em diversos aspectos.

I would also like to thank Prof. Raouf Boutaba, who provided me with an amazing and unique opportunity for personal and professional growth, as well as my colleagues and friends from UWaterloo, who not only welcomed me but also provided invaluable insights on my research.

Last, but not least, I would like to thank my distant friends – some of whom I've had the privilege to meet in person within the last few years. You have become a big part of my life, and I'm not sure where I would be right now without you. Thank you for your friendship, your support, and the many laughs we've had over the years, and I hope we stay in touch for many more to come – both remotely and in person whenever possible.

ABSTRACT

Research on network virtualization has been active for a number of years, during which a number of virtual network embedding (VNE) approaches have been proposed. These approaches, however, neglect important operational requirements imposed by the underlying virtualization platforms. In the case of SDN/OpenFlow-based virtualization, a crucial example of an operational requirement is the availability of enough memory space for storing flow rules in OpenFlow devices. Due to these circumstances, we advocate that VNE must be performed with some degree of knowledge of the underlying physical networks, otherwise the deployment may suffer from unpredictable or even unsatisfactory performance. Considering SDN/OpenFlow-based physical networks as an important virtualization scenario, we propose a framework based on VNE and OpenFlow coordination for proper deployment of virtual networks (VNs). The proposed approach unfolds in the following main contributions:

- a virtual infrastructure abstraction that allows a service provider to represent the details of his/her VN requirements in a comprehensive manner;
- a privacy-aware compiler that is able to preprocess this detailed VN request in order to obfuscate sensitive information and derive computable operational requirements;
- a model for embedding requested VNs that aims at maximizing their feasibility at the physical level.

Results obtained through an evaluation of our framework demonstrate that taking such operational requirements into account, as well as accurately assessing them, is of paramount importance to ensure the "health" of VNs hosted on top of the virtualization platform.

Keywords: Network Virtualization. Virtual Network Embedding. Software-Defined Networking. Privacy.

Alocação de Redes Virtuais em Redes Definidas por Software

RESUMO

Pesquisas acadêmicas em virtualização de redes vêm sendo realizadas durante diversos anos, nos quais diferentes abordagens de alocação de redes virtuais foram propostas. Tais abordagens, no entanto, negligenciam requisitos operacionais importantes impostos por plataformas de virtualização. No caso de virtualização baseada em SDN/OpenFlow, um exemplo fundamental de tais requisitos operacionais é a disponibilidade de espaço de memória para armazenar regras em dispositivos OpenFlow. Diante dessas circunstâncias, argumentamos que a alocação de redes virtuais deve ser realizada com certo grau de conhecimento sobre infraestruturas físicas; caso contrário, após instanciadas, tais redes podem sofrer instabilidade ou desempenho insatisfatório. Considerando redes físicas baseadas em SDN/OpenFlow como um cenário importante de virtualização, propõe-se um arcabouço baseado na coordenação entre a alocação de redes virtuais e redes OpenFlow para realizar a instanciação de redes virtuais de forma adequada. A abordagem proposta desdobra-se nas seguintes contribuições principais:

- uma abstração de infraestruturas virtuais que permite que um requisitante represente os detalhes de seus requerimentos de rede de maneira aprofundada;
- um compilador ciente de privacidade que é capaz de pré-processar requisições com tal grau de detalhamento, ofuscando informações sensíveis e derivando requisitos operacionais computáveis;
- um modelo para a alocação de redes virtuais que visa a maximizar a viabilidade no nível físico.

Resultados obtidos por meio de uma avaliação da nossa abordagem evidenciam que considerar tais requisitos operacionais, bem como computá-los de forma precisa, é imprescindível para garantir a "saúde" das redes virtuais hospedadas na plataforma de virtualização considerada.

Palavras-chave: Virtualização de Redes, Alocação de Redes Virtuais, Redes Definidas por Software, Privacidade.

LIST OF ABBREVIATIONS AND ACRONYMS

CAPEX	Capital Expenditure
OPEX	Operating Expense
SDN	Software-Defined Network
VN	Virtual Network
VNE	Virtual Network Embedding
CPU	Central Processing Unit
SLA	Service-Level Agreement
InP	Infrastructure Provider
QoS	Quality of Service
ONF	Open Networking Foundation
TCAM	Ternary Content-Addressable Memory
RAM	Random-Access Memory
MAC	Media Access Control
UDP	User Datagram Protocol
IP	Internet Protocol
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
LLDP	Link Layer Discovery Protocol
OS3E	Internet2 Open Science, Scholarship and Services Exchange network
TIG	Tenant Infrastructure Graph
PAC	Privacy-Aware Compiler
DSCP	Differentiated Services Code Point
ILP	Integer Linear Programming
MIP	Mixed Integer Programming

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Network virtualization has emerged in recent years as a promising technique for dealing with the complexity and dynamic requirements of today's networks. It enables the instantiation of virtual networks with arbitrary topologies, policies, and capacities as well as the dynamic scaling of these networks in order to cope with high variability of demand. This, in turn, allows network providers to make better use of their physical resources, potentially reducing their CAPEX/OPEX expenditures. The provisioning of network virtualization as a service is expected to be implemented in over 50% of networks between 2021 and 2023, primarily driven by data center networks and large enterprise business networks. (COMMISSION, 2017)

The need for higher adaptability and scalability in network environments has also given rise to the concept of network programmability. Software-Defined Networking (SDN) enables such programmability by decoupling the data and control planes of the network and logically centralizing the control plane. This centralized control enables the creation of algorithmic solutions for dynamic network management and provides a favorable environment for the instantiation of virtual networks.

Research on network virtualization has been active for a number of years. During this period, several approaches for embedding VNs on top of physical infrastructures[1] have been proposed. However, with few exceptions, these approaches tend to limit their scope to generic VN requirements, such as CPU, memory, and bandwidth guarantees, as well as location constraints. Some approaches consider additional aspects such as virtual router image transfer and instantiation overheads, network survivability, or communication security. In contrast, relevant operational requirements related to the instantiation of VNs on different virtualization platforms are typically neglected (either fully or partially – in most cases, the former). This simplification enables the streamlining of the optimization models and heuristics used in these approaches. Moreover, it renders them generic enough to be applied to a number of different scenarios. However, by not taking into account operational requirements of the underlying virtualization platforms, the mappings produced by these VNE approaches may *(a)* not be feasible in practice, *(b)* be unable to properly fulfill SLA (Service Level Agreement) requirements, or *(c)* fail to use infrastructure resources in an efficient manner. In this work, we focus on maximizing the feasibility of VNE mappings on a multi-tenant, SDN/OpenFlow-based network en-

[1] Throughout this work, the expressions *physical infrastructure*, *physical network*, and *physical substrate* are used interchangeably.

vironment so as not to put at risk satisfactory performance and/or network predictability of embedded VNs. Although we focus mainly on OpenFlow-based SDN networks, we consider that many of the proposed concepts and mechanisms could be extended to other implementations of SDN.

1.1 Problem Statement

Software-Defined Networking (SDN) offers a promising platform for network virtualization. In addition to slicing physical resources among customers (SHERWOOD et al., 2009; BOZAKOV; PAPADIMITRIOU, 2012; CORIN et al., 2012; SALVADORI et al., 2011; DRUTSKOY; KELLER; REXFORD, 2013; AL-SHABIBI et al., 2014), SDN-based environments provide abstractions that allow different virtualization functionality, including the instantiation of arbitrary virtual topologies (BOZAKOV; PAPADIMITRIOU, 2012; CORIN et al., 2012; SALVADORI et al., 2011; DRUTSKOY; KELLER; REXFORD, 2013; AL-SHABIBI et al., 2014) and the use of overlapping address spaces (SALVADORI et al., 2011; DRUTSKOY; KELLER; REXFORD, 2013; AL-SHABIBI et al., 2014).

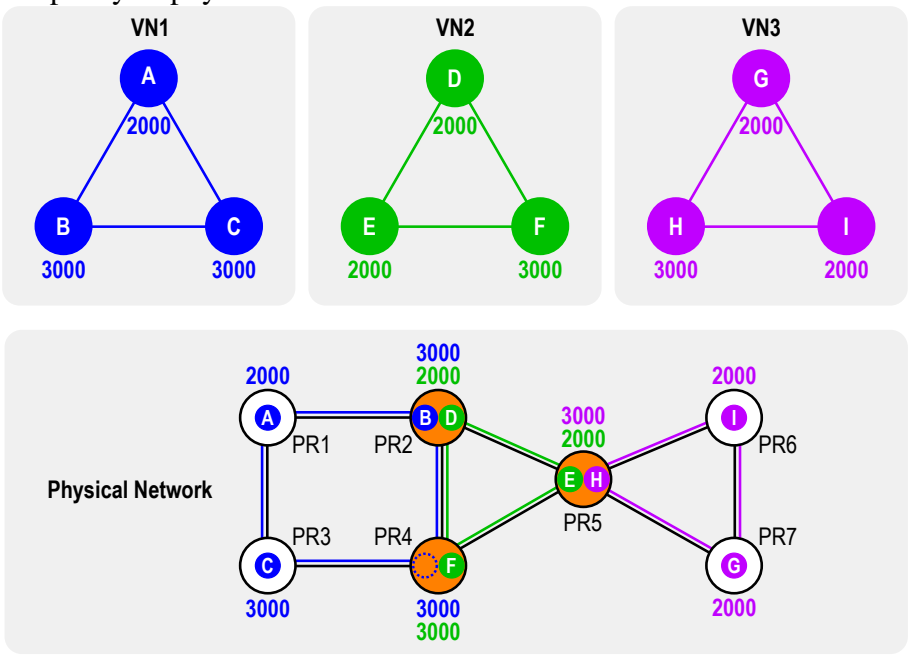
In the case of SDN-based virtualization, there are a number of operational requirements that may render VNE-provided mappings inadequate in real environments. In the following paragraphs, we provide an overview of what we consider operational constraints in such environments.

Flow table overflow. A crucial example of an operational requirement that may render VNE-provided mappings inadequate in real SDN environments is the unavailability of enough memory space for storing flow rules in OpenFlow devices. If this critical issue is not handled by the VNE algorithm, OpenFlow devices may not be able to accommodate all flow rules required by the virtual routers assigned to them. As a consequence, these devices would need to frequently contact the controller in order to handle incoming packets. High rates of controller intervention, in turn, could hinder network performance predictability and potentially render the multi-tenant environment unstable.

Figure 1.1 depicts an example of mappings that would be considered valid by a standard VNE approach but could ultimately lead to performance issues in practice. In this example, three VN requests (VN1–3) are embedded on top of a physical network. In this example, physical routers support up to 4,000 flow rules each, while routers in each VN request require either 2,000 or 3,000 flow rules to be installed on the physical routers

hosting them. Moreover, a number of flow rules must also be installed on "auxiliary routers" – *i.e.*, routers that are not part of the VN request directly, but are necessary in order to create physical paths to host virtual links between such routers. As one can observe, the computed mappings exceed the capacity of some of the physical routers – namely, PR2 (which is hosting virtual routers B and D), PR4 (hosting virtual router F and an auxiliary router in the path between virtual routers B and C), and PR5 (hosting virtual routers E and H).

Figure 1.1: Example of mappings generated by a standard VNE approach that exceed the flow table capacity of physical devices.



The example illustrated above not only underscores that VNE must be performed with some knowledge of the underlying physical networks but also sheds light to the importance of going further in terms of expressing what a VN needs from the physical network. Applications running on top of VNs have distinct traffic patterns and are often subject to different policies or network functions (*e.g.*, load balancing, access control, or deep packet inspection). We advocate that the specification of the expected behavior of a VN, translated into an estimate of flow table usage and communicated to the infrastructure provider (InP), would potentially lead to both a more accurate orchestration of VN embeddings and, ultimately, an overall better quality of service.

Meter table overflow. Meter tables provide QoS-related operations such as rate limiting, which is crucial in a multi-tenant environment. If the meter table of a physical device runs out of space, some flows may be temporarily forwarded while there are no

specific rules to determine at which rate they should be forwarded. The consequences of this range from overloading physical links to breaking SLA requirements regarding QoS. Much like flow tables, the available space in meter tables of physical devices must be taken into account during the process of virtual network embedding, ensuring enough space is reserved for necessary table entries.

Virtual controller allocation. Each OpenFlow network requires at least one controller, which – as previously explained – centralizes all control plane operations. The controller manages all forwarding devices in the network, *e.g.*, instructing devices to add or remove entries from their flow and meter tables. As such, this entity must be accounted for in the mapping process. Moreover, reserving sufficient resources to the controller software is crucial for ensuring the network will accurately follow its intended behavior as well as to prevent performance degradation.

Channel allocation for in-band switch–controller communication. In order to ensure all virtual routers of a VN are reachable by its virtual controller, it is necessary to establish dedicated communication channels between these entities. Failing to properly consider this communication may lead to saturation of physical links as controllers may periodically need to perform bandwidth-heavy operations. Moreover, one cannot assume these communication channels are allocated in an out of band manner, as this would not be feasible in real physical networks.

Physical hypervisor positioning. In order to manage the virtual routers of a given VN, the virtual controller must send and receive data through a physical hypervisor. In other words, the communication channels between a virtual controller and the routers it manages must always pass through a hypervisor. By not considering the position of physical hypervisors in the topology, virtual controllers may be placed near the virtual routers they need to manage but far from a hypervisor. This, in turn, would make switch–controller communication channels exceedingly long, consuming a significantly higher amount of bandwidth than necessary. Therefore, taking into account the position of physical hypervisors in the infrastructure is of paramount importance in order to efficiently establish these paths.

Taking into account all aforementioned issues at the same time, however, introduces a large set of constraints to the VNE strategy. This, by itself, constitutes another problem, as the large number of constraints significantly reduces the solution space. In other words, the number of feasible mappings the VNE model can produce is severely limited. This, in turn, would likely lead to a significant increase in rejection rates of VN

requests, further exacerbated by other known issues in the field of VNE such as resource fragmentation (LUIZELLI et al., 2016). As such, one must be aware of this additional, important challenge when devising a solution that encompasses the aforementioned operational constraints.

1.2 Hypothesis

The hypothesis we formulate as the fundamental research issue to guide this Ph.D. research work is the following: *In order to provide network predictability, efficient resource usage, and quality of service, it is necessary to coordinate virtual network embedding and operational constraints found in SDN-based networks.* Traditional VNE proposals disregard such operational constraints subjecting SDN environments to unnecessary resource depletion, potentially threatening the proper operation of VNs hosted on top of it. The objective of this research is to confirm this hypothesis and answer the research questions presented below.

Research Question 1. When performing virtual network embedding while disregarding operational constraints, how significant is the negative impact on predictability and quality of service (*e.g.*, performance degradation due to controller intervention)?

Research Question 2. Once coordination between virtual network embedding and software-defined networks is achieved, what are the quantifiable gains that can be attained?

Research Question 3. Is the cost to be paid for VNE/SDN coordination acceptable in terms of solution space reduction, computing power, and timeliness?

1.3 Goals and Contributions

Our approach consists of a VNE framework that is aware of operational requirements related to the instantiation of VNs on top of an SDN/OpenFlow environment. The central idea of the proposed approach is the specification, by VN requesters, of VN requests enriched with information about how (to be) provisioned networks will be used (*e.g.*, important application flows, network functions/policies to which packets will be subjected to, etc.). These VN specifications are then used to derive operational requirements, still at the customer's end. The resulting specifications – reflecting requesters

willingness (or not) to disclose information about the VNs – are sent to the InP, which will ultimately correctly embed the requested VNs favoring incoming requests with well defined operational requirements. We consider a number of pieces of information that, if known in advance by the InP, can lead to improved allocation of network resources and, in turn, to improved network utilization. The main contributions of the thesis are therefore as follows.

- A deep understanding of the influence of SDN operational constraints on the process of virtual network embedding and the "health" of hosted VNs and the multi-tenant environment as a whole.
- An abstraction model for expressing requirements related to internal VN policies and traffic patterns.
- A strategy for accurately deriving the operational requirements associated with each particular VN based on the aforementioned abstraction model.
- A VNE method that leverages this computed information in order to properly and efficiently allocate resources in an SDN-based virtualization environment.
- A strategy for mitigating VN rejection based on customer-controlled flexibilization of VN requirements.

1.4 Organization

The remainder of this thesis is organized as follows.

- Chapter 2 presents background information that we consider a prerequisite for fully comprehending our proposal. This background review covers the topics of Sotware-Defined Networking and its main implementation, OpenFlow, as well as network virtualization.
- Chapter 3 discusses state-of-the-art VNE proposals, including general purpose VNE approaches and SDN-oriented ones.
- Chapter 4 describes our first step towards tackling the issue of virtual network embedding on top of software-defined networks, focusing on one of the most crucial operational constraints of SDN substrates – flow table overflow. Moreover, it presents an evaluation centered around issues pertaining this particular operational constraint.

- Chapter 5 presents our fully realized architecture taking into account all research problems previously explained in this chapter. Additionally, it discusses the results of a second round of experiments, focusing on the ability of our solution to handle a rich set of operational constraints simultaneously.
- Chapter 6, in turn, concludes this thesis with our final considerations and directions for future work.

2 BACKGROUND

In this chapter, we provide a detailed background of the area of Software-Defined Networking and the OpenFlow protocol. Afterwards, we review the main concepts related to network virtualization. The structure of this chapter is inspired by that of a journal paper published early during the Ph.D. course (BAYS et al., 2015).

2.1 Software-Defined Networking and OpenFlow

Although the SDN paradigm was conceived relatively recently, the idea of programmable networks was envisioned much longer ago, in the mid-1990s (CALVERT, 2006). The growing prevalence of the Internet as well as the development of an increasingly diverse array of applications led to the necessity of developing and testing new protocols in realistic network scenarios. Driven by this need, researchers started to explore ways to open up network control and make networks more easily programmable.

Active networking was the first major novel approach to network programmability. The core concept behind it was the presence of a network programming interface. This interface allowed developers to access resources on individual network nodes and program specific functions to be applied to certain traffic passing through them. Through this programmability, active networking aimed at keeping the network core simple and lowering the barrier to innovation.

Ultimately, active networking failed to achieve widespread adoption. Feamster *et al.* (FEAMSTER; REXFORD; ZEGURA, 2014) argue that this failure may have been due to "the lack of an immediately compelling problem or a clear path to deployment". In spite of this, active networking paved the way for subsequent efforts on achieving network programmability as well as related concepts such as network virtualization.

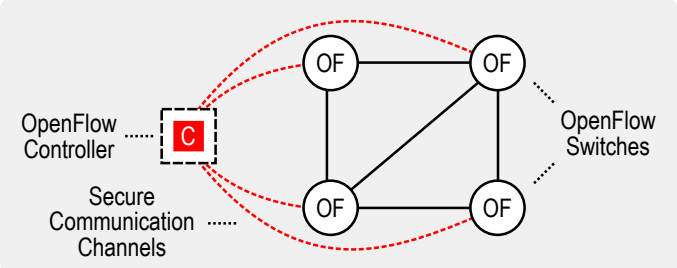
Fundamentals of SDN and OpenFlow. Software-Defined Networking follows the foundation laid out by active networking, albeit focusing on a narrower, more streamlined set of problems to solve. SDN employs the concept of plane separation in order to address the issues of routing and configuration management. The control plane – responsible for making routing decisions based on network policies – is decoupled from the data plane – which simply forwards traffic according to decisions made by the control plane. Through this decoupling, SDN enables network intelligence and state to be logically centralized, while abstracting low level network infrastructure information.

The OpenFlow protocol is the most prominent and widely adopted implementation of the SDN paradigm. It specifies an open, well defined protocol for communication between network devices and the logically centralized control plane, ensuring interoperability with network devices manufactured by different vendors. Moreover, it formalizes requisites that network devices must be able to fulfill, such as the set of operations they must be able to perform on incoming traffic.

OpenFlow was conceived (and initially released) as an academic initiative from the Stanford University (MCKEOWN et al., 2008) in 2008. Over time, interest in this project grew beyond academia and reached big players in the areas of networking and telecommunications. Thus, in 2011, the Open Networking Foundation[1] (ONF) – an organization aimed at improving, standardizing, and promoting SDN and OpenFlow – was formed. Development of the OpenFlow protocol has continued steadily since the creation of the ONF, with at least four major versions released since then. Currently, the ONF encompasses 140 member companies, including 28 startups.

Figure 2.1 presents an overview of the entities that comprise the OpenFlow architecture. This figure depicts a network of four OpenFlow-enabled devices (*i.e.*, switches) managed by a single OpenFlow controller. The OpenFlow switches represent the data plane of this software-defined network, while the controller acts as the centralized control plane. The controller sends and receives information to/from each network device through secure communication channels. OpenFlow networks may optionally employ multiple controllers simultaneously (*e.g.*, in order to improve performance and/or redundancy). In this case, while the control plane remains logically centralized, it is physically distributed among a number of controllers.

Figure 2.1: Overview of the OpenFlow architecture.



OpenFlow controllers. The controller provides a global view of the network (including information regarding its topology, statistics, and events) as well as interfaces that abstract low level network information. These features allow network administrators

[1]Open Networking Foundation: https://www.opennetworking.org/

to create high level algorithmic solutions for network management, enabling dynamic, automated network control applications. To be more precise, these applications interface directly with the controller, leveraging the high level information provided by it in order to algorithmically make network management decisions. The controller itself is, in turn, responsible for programming network devices in order to ensure they behave as intended by the application.

Switch–controller communication follows a strict, well defined protocol. Through this protocol, the controller is able to send messages requesting the addition, removal, or modification of flow rules on specific OpenFlow switches. Moreover, it may periodically request network statistics, which may be crucial for decision-making.

Channel allocation for switch–controller communication may be done in two different manners – "in-band" or "out of band". In-band communication consists in using preexisting links in the infrastructure for carrying data between switch–controller pairs. Out of band communication assumes the presence of dedicated channels between these pairs (as illustrated in Figure 2.1). In the stages of modeling and prototyping OpenFlow-based solutions, it is common to assume switch–controller communication is performed in an out of band manner and without bandwidth constraints. Such assumptions can be observed in mathematical models as well as OpenFlow emulators such as Mininet (LANTZ; HELLER; MCKEOWN, 2010). As will be explained later, these assumptions are not in line with real network virtualization environments.

Due to the presence of a standardized switch–controller communication protocol, multiple implementations of OpenFlow controllers exist. These typically differ in terms of the programming language they allow controller applications to be built in – such as Python (*e.g.,* POX[2] and Ryu [3]) or Java (*e.g.*, Beacon[4] and Floodlight[5]) – and/or their feature sets – such as enabling the federation of multiple controllers (*e.g.*, Flowvisor (SHERWOOD et al., 2009)) or providing capabilities for network virtualization (*e.g.*, RouteFlow (NASCIMENTO et al., 2011) and OpenVirteX (AL-SHABIBI et al., 2014)).

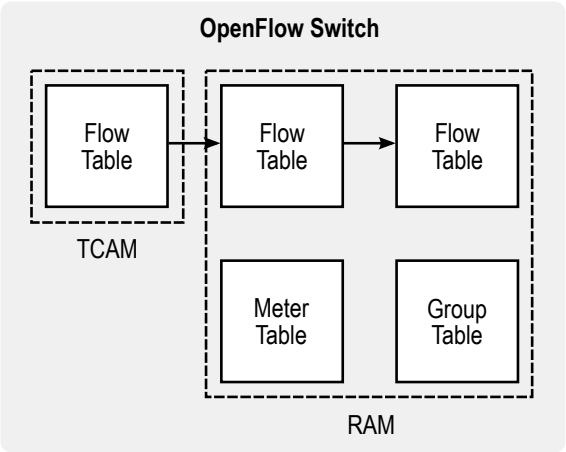
OpenFlow switches. In contrast with traditional network paradigms and as just mentioned, SDN-enabled network devices are limited to simple data processing and forwarding operations. The operations to be applied to an incoming packet depend on how a particular device is programmed by the controller. OpenFlow employs the concept of

[2]POX OpenFlow Controller: https://openflow.stanford.edu/display/ONL/POX+Wiki
[3]Ryu SDN Framework: https://osrg.github.io/ryu/
[4]Beacon OpenFlow Controller: https://openflow.stanford.edu/display/Beacon/Home
[5]Floodlight OpenFlow Controller: http://www.projectfloodlight.org/floodlight/

"traffic flows" in order to classify and treat network packets, allowing varying degrees of granularity. A flow is composed of a number of matching fields and one or more actions. Matching fields are used to distinguish among different network flows and typically represent packet header fields, such as source, destination, and port. Actions, in turn, define the operations to be performed on packets that match the described flow. In addition to forwarding packets through specific network interfaces on a switch, these actions also allow the manipulation of packet header fields (*e.g.,* changing the source or destination IP or MAC address). Tables 2.1 and 2.2 list all matching fields and actions available in the latest publicly available version of the OpenFlow protocol (1.5.1)[6].

Figure 2.2 depicts the main elements within each OpenFlow device. Switches store information regarding flow matching fields and actions into flow tables. Whenever a packet is received, the device attempts to find a flow table entry that matches its characteristics. If a match is found, the device applies the operations defined in the "actions" field of the matching entry. If there are no matches, the switch may contact the controller in order to determine which action(s) should be taken. Actions specified in flow tables may direct packets for further processing in a different flow table or to group actions (which are stored in the group table). OpenFlow devices also keep track of both individual and aggregate statistics regarding traffic that passes through them. The meter table is used to measure packet rates and implement QoS-related operations such as rate limiting.

Figure 2.2: Main components of an OpenFlow switch.



Flow tables are often distributed between two different types of memory – namely, TCAM (Ternary Content-Addressable Memory) and RAM (Random-Access Memory). TCAMs are highly suited for flow table operations, as they allow parallel lookups with

[6]As of the writing of this thesis, a more recent version, OpenFlow 1.6, is available – however, its specifications are only available to members of the ONF and not to the general public.

Table 2.1: OpenFlow matching fields.

Field	Description	Introduced in
IN_PORT	Switch input port.	v0.8
IN_PHY_PORT	Switch physical input port.	v1.2
METADATA	Metadata passed between tables.	v1.1
ETH_DST	Ethernet destination address.	v0.8
ETH_SRC	Ethernet source address.	v0.8
ETH_TYPE	Ethernet frame type.	v0.8
VLAN_VID	VLAN id.	v0.8
VLAN_PCP	VLAN priority.	v0.9
IP_DSCP	IP DSCP (6 bits in ToS field).	v1.0
IP_ECN	IP ECN (2 bits in ToS field).	v1.2
IP_PROTO	IP protocol.	v0.8
IPV4_SRC	IPv4 source address.	v0.8
IPV4_DST	IPv4 destination address.	v0.8
TCP_SRC	TCP source port.	v0.8 (v1.2)[7]
TCP_DST	TCP destination port.	v0.8 (v1.2)[7]
UDP_SRC	UDP source port.	v0.8 (v1.2)[7]
UDP_DST	UDP destination port.	v0.8 (v1.2)[7]
SCTP_SRC	SCTP source port.	v1.1 (v1.2)[7]
SCTP_DST	SCTP destination port.	v1.1 (v1.2)[7]
ICMPV4_TYPE	ICMP type.	v1.2
ICMPV4_CODE	ICMP code.	v1.2
ARP_OP	ARP opcode.	v1.2
ARP_SPA	ARP source IPv4 address.	v1.2
ARP_TPA	ARP target IPv4 address.	v1.2
ARP_SHA	ARP source hardware address.	v1.2
ARP_THA	ARP target hardware address.	v1.2
IPV6_SRC	IPv6 source address.	v1.2
IPV6_DST	IPv6 destination address.	v1.2
IPV6_FLABEL	IPv6 Flow Label.	v1.2
ICMPV6_TYPE	ICMPv6 type.	v1.2
ICMPV6_CODE	ICMPv6 code.	v1.2
IPV6_ND_TARGET	Target address for ND.	v1.2
IPV6_ND_SLL	Source link-layer for ND.	v1.2
IPV6_ND_TLL	Target link-layer for ND.	v1.2
MPLS_LABEL	MPLS label.	v1.1
MPLS_TC	MPLS TC.	v1.1
MPLS_BOS	MPLS BoS bit.	v1.3
PBB_ISID	PBB I-SID.	v1.3
TUNNEL_ID	Logical Port Metadata.	v1.3
IPV6_EXTHDR	IPv6 Extension Header pseudo-field.	v1.3
PBB_UCA	PBB UCA header field.	v1.4
TCP_FLAGS	TCP flags.	v1.5
ACTSET_OUTPUT	Output port from action set metadata.	v1.5
PACKET_TYPE	Packet type value.	v1.5

very high performance. Traditional RAM memory, in turn, offers limited performance as lookups must be performed in a sequential manner. In spite of their performance advantage, TCAMs are 400x more expensive (LIAO, 2012) and consume 100x more power (SPITZNAGEL; TAYLOR; TURNER, 2003) per megabit than RAM. Therefore, TCAM space tends to be extremely limited in OpenFlow devices, adding to the necessity of properly managing flow tables and keeping track of table occupation.

Due to the magnitude and importance of the issue of properly managing flow tables, research in this area is currently highly active. Neves *et al.* (NEVES et al., 2016) provide a detailed look at these strategies, which the authors classify as "spatial" and "temporal". Spatial strategies employ techniques such as aggregating multiple flow rules

[7]Originally, these matching fields were grouped together as TP_SRC and TP_DST, which matched

Table 2.2: OpenFlow actions.

Action	Description	Introduced in
OUTPUT	Output to switch port.	v0.8
SET_VLAN_VID	Set the 802.1q VLAN id.	v0.8
SET_VLAN_PCP	Set the 802.1q priority.	v0.8
STRIP_VLAN	Strip the 802.1q header.	v0.8
SET_DL_SRC	Ethernet source address.	v0.8
SET_DL_DST	Ethernet destination address.	v0.8
SET_NW_SRC	IP source address.	v0.8
SET_NW_DST	IP destination address.	v0.8
SET_TP_SRC	TCP/UDP source port.	v0.8
SET_TP_DST	TCP/UDP destination port.	v0.8
COPY_TTL_OUT	Copy TTL "outwards" – from next-to-outermost to outermost	v1.1
COPY_TTL_IN	Copy TTL "inwards" – from outermost to next-to-outermost	v1.1
SET_MPLS_TTL	MPLS TTL	v1.1
DEC_MPLS_TTL	Decrement MPLS TTL	v1.1
PUSH_VLAN	Push a new VLAN tag	v1.1
POP_VLAN	Pop the outer VLAN tag	v1.1
PUSH_MPLS	Push a new MPLS tag	v1.1
POP_MPLS	Pop the outer MPLS tag	v1.1
SET_QUEUE	Set queue id when outputting to a port	v1.1
GROUP	Apply group.	v1.1
SET_NW_TTL	IP TTL.	v1.1
DEC_NW_TTL	Decrement IP TTL.	v1.1
SET_FIELD	Set a header field using OXM TLV format.	v1.2
PUSH_PBB	Push a new PBB service tag (I-TAG)	v1.3
POP_PBB	Pop the outer PBB service tag (I-TAG)	v1.3
COPY_FIELD	Copy value between header and register.	v1.5
METER	Apply meter (rate limiter)	v1.5

into a smaller subset that covers a wider range of flows or distributing rules that would be stored in a single network device among multiple ones. Temporal strategies, in contrast, attempt to minimize the number of rules installed in each device at any one time by dynamically adding and removing rules as flows become active or inactive. While successful in reducing flow table usage to an extent, both classes of strategies are based on tradeoffs. Spatial strategies may force the use of flow rules with low granularity or hinder network predictability, while temporal strategies may have a substantial negative impact on network performance due to a significant increase in controller intervention.

A more detailed view of entries that may be added to a flow table is shown in Table 2.3. As previously explained, the matching fields of a flow rule are used to discriminate network flows. In this example, rule 0 matches packets with a source MAC address that starts with “4C:80:93”, while rule 1 matches packets with a destination IP address that starts with “192.168”. Rule 2 matches all UDP packets (represented by the IP protocol number 0x11). The actions associated with rules 0 and 1 forward packets through specific physical network interfaces. As the “actions” field of rule 2 is empty, packets matching this rule will not be forwarded (*i.e.*, all UDP traffic will be dropped). The priority field of the flow table is used to establish matching precedence. If a packet matches more than

source and destination ports, respectively, regardless of protocol. In version 1.2, specific matching fields for TCP, UDP, and SCTP source and destination ports were created, deprecating the original fields.

one rule at the same time (*e.g.*, rules 0 and 2), only the rule with the highest priority will be used to treat it (in this case, rule 2).

Each rule may also have an idle timeout and/or a hard timeout. As rule 0 has an idle timeout of 120, the switch will automatically remove this rule if no traffic belonging to this flow is received during 120 consecutive seconds. Rule 1, in turn, has a hard timeout of 600. Therefore, it will be automatically removed 600 seconds after its creation, regardless of the activity or inactivity of this flow. Setting both of these fields to 0 allows the creation of permanent rules, which will only be removed if the switch is instructed to do so by the controller (as is the case of rule 2). Managing these timeouts is a crucial part of temporal strategies for flow rule management. Last, counter fields allow the device to keep track of how much traffic (in terms of packets and bytes) pertaining to each flow has been received and treated by each rule.

Table 2.3: Example of flow table entries that may be installed on an OpenFlow device.

Entry	Matching Fields	Actions	Priority	Timeouts	Counter
0	ETH_SRC=4C:80:93:*	OUTPUT:1	16384	idle_timeout=120	4925
1	IPV4_DST=192.168.*.*	OUTPUT:2	16384	hard_timeout=600	6030
2	IP_PROTO=0x11	∅	32768	∅	1377

After presenting an overview of Software-Defined Networking, we now proceed to a review of the area of network virtualization. This review includes a discussion on the employment of SDN as a platform for instantiating virtual networks.

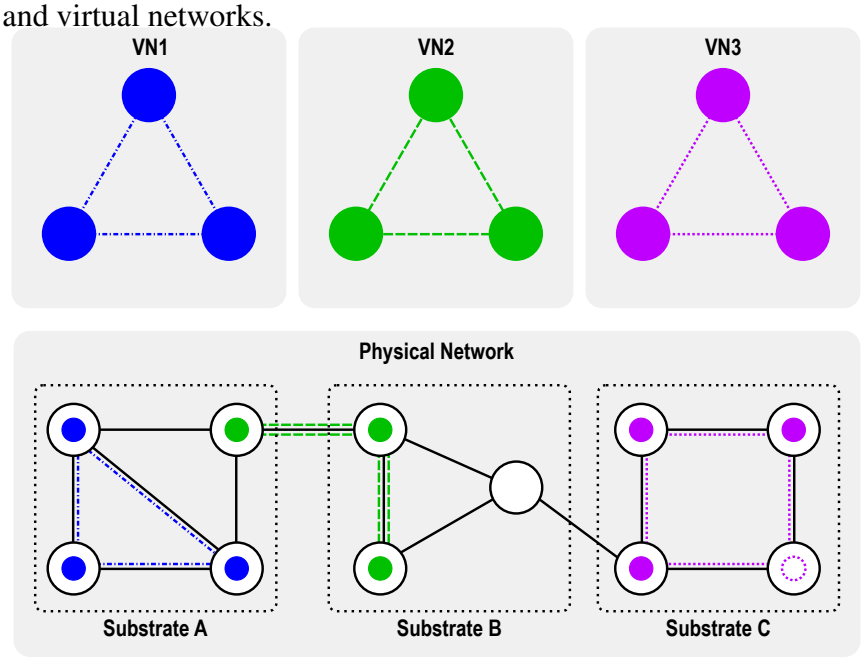
2.2 Network Virtualization

Network virtualization consists of sharing resources from physical network devices (routers, switches, etc.) among different virtual networks. It allows the coexistence of multiple, possibly heterogeneous networks, on top of a single physical infrastructure. The basic elements of a network virtualization environment are shown in Figure 2.3. At the physical network level, a number of autonomous systems are represented by interconnected network substrates (*i.e.,* substrates A, B, and C). Physical network devices are represented by nodes supporting virtualization technologies. Virtual network topologies (VN1–3), in turn, are mapped to a subset of nodes from one or more substrates. These topologies are composed of virtual routers, which use a portion of the resources available in physical ones, and virtual links, which are mapped to physical paths composed of one or more physical links and their respective intermediate routers (also referred to as

auxiliary routers).

From the point of view of a virtual network, virtual routers and links are seen as dedicated physical devices. However, in practice, they share physical resources with routers and links from other virtual networks. For this reason, the virtualization technology used to create this environment must provide an adequate level of resource isolation in order to enable the use of network virtualization in real, large scale environments.

Figure 2.3: Network virtualization model, denoting a scenario with multiple physical substrates and virtual networks.



Over the years, different methods for instantiating virtual networks have been used. Typical approaches include VLANs (Virtual Local Area Networks) and VPNs (Virtual Private Networks). Recently, Virtual Machine Monitors and Software-Defined Networks have also been employed to create virtual routers and links over physical devices and communication channels. These approaches are briefly revisited next.

Protocol-based approaches. These approaches consist of implementing a network protocol that enables the distinction of virtual networks through techniques such as tagging or tunneling. The only requirement of this kind of approach is that physical devices (or a subset of them) support the selected protocol.

One example of protocol-based network virtualization are VLANs. VLANs consist of logical partitions of a single underlying network. Devices in a VLAN communicate with each other as if they were on the same Local Area Network, regardless of physical location or connectivity. All frames sent through a network are tagged with their corresponding VLAN ID, processed by VLAN-enabled routers and forwarded as necessary

Table 2.4: Virtualization techniques.

Technique	Description	Examples
Full Virtualization	The Virtual Machine Monitor emulates a complete machine, based on the underlying hardware architecture. The guest Operating System runs without any modification.	VMware Workstation, VirtualBox, QEMU
Paravirtualization	The Virtual Machine monitor emulates a machine which is similar to the underlying hardware, with the addition of a hypervisor. The hypervisor allows the guest Operating System to run complex tasks directly on non-virtualized hardware. The guest OS must be modified in order to take advantage of this feature.	VMware ESXi, Xen, KVM, Hyper-V
Container-based Virtualization	Instead of running a full Virtual Machine, this technique provides Operating System-level containers, based on separate userspaces. In each container, the hardware, as well as the Operating System and its kernel, are identical to the underlying ones.	OpenVZ, Linux VServer, VMware ThinApp, Docker

(LAN/MAN STANDARDS COMMITTEE, 2006). As this technique is typically based only on packet tagging, it does not provide any guarantees regarding data or resource isolation.

Another commonly used approach is the creation of Virtual Private Networks. VPNs are typically used to provide a secure communication channel between geographically distributed nodes. VPNs can be provided in the physical, data link, or network layers according to the tunneling protocols being employed (ROSEN et al., 2006). As these protocols commonly make use of cryptography to establish tunnels, most VPNs natively provide data isolation (but not resource isolation).

Machine virtualization-based approaches. These approaches consist of creating virtual networks by means of groups of interconnected virtual machines. Virtual Machine Monitors are used to instantiate virtual routers, and virtual links are established between them, regardless of physical network topology. Table 2.4 summarizes different machine virtualization-based techniques that can be used to create virtual networks, as well as a brief explanation and an example of each.

This alternative is remarkably flexible and relatively cheap, as it allows the use of customized software and does not require the use of specific hardware[8]. Additionally,

[8]Machine virtualization is available for personal computers, in commonly used operating systems (*e.g.*,

machine virtualization-based approaches are typically capable of ensuring both data and resource isolation. However, they are more demanding in terms of resource usage in comparison to previously described protocol-based approaches.

Software-Defined Networks. As previously mentioned, SDN-based network environments may be used as platforms for the instantiation of virtual networks. Network virtualization may be achieved at the hardware level – directly slicing the physical resources of each device – or at the flow level – in which a network hypervisor manages physical flow tables on behalf of virtual controllers. Next, we explain the specifics of each level of virtualization, what may be achieved through them, and the tradeoffs between them.

In hardware-level virtualization, physical flow tables are sliced among a number of tenants, who are able to directly install, remove, and modify rules. As there is no layer between controllers and switches, this type of virtualization is very lightweight. However, it does not offer much flexibility or isolation. Each tenant has access to all traffic being routed through the devices it controls. Moreover, conflicts may occur if tenants attempt to, for example, use overlapping address spaces. FlowVisor (SHERWOOD et al., 2009) is an example of a platform that employs this type of virtualization.

In flow-level virtualization, virtual network controllers manage flow tables indirectly by communicating with a network hypervisor. When a hypervisor receives a message from a controller, it must first translate the flow rule and only then install it on the physical device. While this translation adds some overhead to the process of adding, removing, or modifying rules, it enables a number of features that cannot be achieved in hardware-level virtualization (such as topology and address space virtualization, which will be explained next). Moreover, it allows control over the number of flow rules each tenant is allowed to install and provides isolation among different tenants.

Topology virtualization enables an SDN network virtualization platform to create arbitrary topologies on top of physical ones. Such topologies may include, for example, virtual links comprised of paths that traverse multiple physical links – which would not be possible with hardware-level virtualization as the tenant would have access to the physical routers within this path. This may be achieved through a number of different techniques, such as VLAN encapsulation or on-the-fly manipulation of level 2 packet header fields or Link Layer Discovery Protocol (LLDP) messages. Topology virtualization is offered by a number of SDN-based virtualization technologies, such as Au-

---

Windows, Linux, and macOS).

toSlice (BOZAKOV; PAPADIMITRIOU, 2012), VeRTIGO (CORIN et al., 2012), FlowN (DRUTSKOY; KELLER; REXFORD, 2013), ADVisor (SALVADORI et al., 2011), and OpenVirteX (AL-SHABIBI et al., 2014).

Address space virtualization consists in allowing multiple virtual networks to use overlapping address spaces. This may be achieved through VLAN encapsulation or by dynamically translating overlapping virtual addresses to unique ones at the physical level. With regards to the latter, this may be done by assigning a unique prefix to each virtual network, and modifying rules and packets from each virtual network to use addresses within its own prefix. Three of the platforms mentioned above – namely, FlowN, ADVisor, and OpenVirteX – provide address space virtualization in addition to topology virtualization. Table 2.5 summarizes information regarding which of the previously discussed platforms support each of these features.

Table 2.5: SDN-based network virtualization platforms and the features they support.

Platform	Topology Virtualization	Address Space Virtualization
FlowVisor	Not supported	Not supported
AutoSlice	Supported	Not supported
VeRTIGO	Supported	Not supported
FlowN	Supported	Supported
ADVisor	Supported	Supported
OpenVirteX	Supported	Supported

Taking into account the network virtualization model considered in this thesis, both topology and address space virtualization would be necessary to support the instantiation of virtual networks mapped by our VNE approach. Therefore, platforms such as FlowN, ADVisor, and OpenVirteX are more closely aligned with our work. We now proceed to an analysis of the state of the art regarding VNE approaches in Chapter 3, culminating in a discussion of the shortcomings of current approaches and how we intend to address them.

3 STATE OF THE ART

In this chapter, we first revisit the most prominent approaches to the virtual network embedding problem, highlighting the particularities of each one. We explore both "traditional" (*i.e.*, platform-agnostic) VNE approaches and SDN-oriented ones. Afterwards, we provide a brief discussion on the shortcomings of these approaches when considering the full deployment of VNs on an SDN substrate and how we intend to tackle these issues.

3.1 Traditional VNE Approaches

Virtual network embedding has been a highly active area of research for a number of years. The earliest efforts to tackle this problem were published circa 2008 and, to this day, this extensive body of work continues to grow. Over the years, researchers have focused on different objectives, such as improved efficiency, higher coverage of constraints, or particular demands such as security or energy efficiency.

One of the earliest major efforts towards tackling the VNE problem was devised by Yu *et al.* (YU et al., 2008). The heuristic-based VNE approach presented by the authors embeds virtual routers and links in separate phases, and prioritizes VNs with largest revenue. This approach takes into account CPU and location constraints for routers, and bandwidth constraints for links.

Chowdhury *et al.* (CHOWDHURY; RAHMAN; BOUTABA, 2012) propose two optimization models, one being a relaxed version of the other. Routers and links are embedded in distinct phases; however, the authors improve upon previous work by enhancing coordination between these two phases. This is achieved by preselecting router mappings taking into account their location constraints in order to facilitate link mapping. Similarly to the work of Yu *et al.*, CPU, location, and bandwidth requirements are considered by the proposed optimization models. Moreover, link delay is used to determine how far a virtual router may be embedded from its preferred location.

Cheng *et al.* (CHENG et al., 2011) introduce the concept of "node ranking", in which virtual and physical nodes are ranked according to their own capacity and the capacities of their neighbors. Two embedding algorithms are proposed, one mapping routers and links in distinct stages while the other performs both simultaneously. The algorithms take into account CPU and bandwidth constraints but do not include location

constraints.

Alkmim *et al.* (ALKMIM; BATISTA; FONSECA, 2013) present two VNE approaches based on optimization models. One employs a traditional Integer Linear Programming (ILP) model, while the other employs a relaxation technique in order to reduce running times. The authors focus on constraints related to overheads incurred when transferring and instantiating virtual router software images. As such, in addition to CPU, location, bandwidth, and link delay, the size of virtual router images (and the memory needed to support them), the locations in which they are stored, and the time needed to transfer and instantiate them are also taken into account.

Bays *et al.* (BAYS et al., 2014) propose both an optimization model and a heuristic algorithm for virtual network embedding focusing on privacy. Both approaches take into account throughput capacity and location requirements of routers as well as link bandwidth. Additionally, a number of security related constraints are considered, namely which physical routers are capable of supporting the necessary security protocols, overheads associated with cryptographic operations, and which VNs may not share physical resources.

Guan *et al.* (GUAN; CHOI; SONG, 2015) propose a VNE approach aimed at tackling the issue of energy efficiency in data center networks. To this end, they formulate an optimization model as well as a heuristic algorithm. The authors take into account the energy costs of operation and migration of routers and links, in addition to CPU and bandwidth requirements. Moreover, the proposed algorithm accounts for varying resource demands throughout the lifecycle of each VN.

Zhang *et al.* (ZHANG et al., 2015) propose both an optimization model and a heuristic approach which, similarly to the work of Guan *et al.*, take into account energy efficiency. In addition to CPU and bandwidth requirements, these approaches also consider location constraints. The optimization model is a multi-objective one, simultaneously aiming at minimizing energy consumption and maximizing profit. The heuristic algorithm, in turn, is based on artificial immune systems.

Esposito *et al.* (ESPOSITO; PAOLA; MATTA, 2016) formulate a VNE algorithm based on an auction system. Physical routers "bid" on virtual ones from incoming VN requests, prioritizing those with highest capacity in order to maximize revenue. After a consensus is reached regarding which virtual router is the "winner" of each auctioned virtual node, a separate link embedding phase is carried out. No additional constraints other than CPU and bandwidth are considered.

Last, Jarray *et al.* (JARRAY; KARMOUCH, 2015) propose an optimization model which is also based on an auction system; however, this auction system follows a different method of operation. In this approach, each VN requester offers a bid, which represents how much it is willing to pay for the embedding of its VN. Additionally, in contrast to the work of Esposito *et al.*, router and link embedding is performed simultaneously. In order to tackle scalability issues, the authors employ relaxation and rounding techniques to their model. Only CPU and bandwidth requirements are taken into account.

Table 3.1: Summary of traditional VNE approaches.

Authors	Objective	Specific Constraints	Optimization Method	Evaluated Topologies	Deployment Coordination
Yu *et al.*	Maximize revenue	None	Heuristic	100 routers, random (GT-ITM)	No
Chowdhury *et al.*	Minimize cost, balance load	None	Optimal with relaxations	50 routers, random (GT-ITM)	No
Cheng *et al.*	Balance load (through NodeRank)	None	Heuristic	100 routers, random (GT-ITM)	No
Alkmim *et al.*	Minimize cost	Link delay; location, size, and time to instantiate virtual router images	Optimal with relaxations	10 to 50 routers, BA-2 model (BRITE)	No
Bays *et al.*	Minimize cost	Security support and related overheads, non-overlapping VNs	Optimal and heuristic	100 and 500 routers, BA-2 model (BRITE)	No
Guan *et al.*	Minimize energy consumption	Energy costs of operation and migration of routers and links	Optimal and heuristic	Random number of routers, datacenter-like (NetworkX)	No
Zhang *et al.*	Maximize revenue, minimize energy consumption	Energy costs of operation of routers and links	Optimal and heuristic	50 routers, random (GT-ITM)	No
Esposito *et al.*	Maximize revenue	None	Heuristic	50 routers, BA-2 model (BRITE)	No
Jarray *et al.*	Maximize revenue	None	Optimal	20 and 50 routers, generated based on geographical data	No

Information presented thus far is summarized in Table 3.1. As one can observe, researchers have employed a number of different objective functions to guide their approaches – although all are related in one way or another to embedding costs. Five of the approaches presented in this subsection only take into account basic VNE-related constraints (CPU, bandwidth, and location), while the remaining four consider additional

constraints with varying degrees of specificity. Due to scalability concerns, most authors propose heuristic methods or relaxations to their optimal approaches. Most proposals are evaluated on physical networks with size ranging from 20 to 100, generated using tools such as GT-ITM and BRITE. As the only exception, Jarray *et al.* employ topologies based on geographical data from the United States and Europe. Last, as all VNE approaches presented thus far are platform-agnostic, none of them take into account SDN deployment coordination.

3.2 SDN-Oriented VNE Approaches

Recently, there have been efforts towards adapting the VNE problem to cover different aspects related to SDN/OpenFlow networks. We now proceed to a review of the most prominent efforts in this scope.

Demirci *et al.* (DEMIRCI; AMMAR, 2014) focus on the issue of controller placement in addition to virtual router and link mapping. The authors devise two different embedding strategies. The first one aims at balancing the load on physical elements, while the other aims at minimizing communication delay between virtual routers and controllers. The authors consider bandwidth capacity constraints, in addition to controller location requirements. Embedding is performed in an offline manner, assuming all requests are known in advance.

Blenk *et al.* (BLENK et al., 2016) devise strategies for dealing with multiple SDN network hypervisors. The authors propose an optimization model with four different variations which aim at minimizing control plane latency in four different manners (*i.e.*, maximum latency, average latency, average maximum latency, and maximum average latency). Capacity constraints are not considered.

Huang *et al.* (HUANG et al., 2017) focus on the issue of flow table capacity. In their case, this constraint must be explicitly specified by the customer when requesting a VN. The authors devise both an optimization model and a heuristic algorithm that, in addition to flow table occupation, considers bandwidth capacities of physical links. The optimization model directly aims at minimizing embedding costs, while the heuristic algorithm favors the selection of physical nodes with greater amounts of residual resources.

Tegueu *et al.* (TEGUEU et al., 2017) introduce an embedding algorithm focused on minimizing resource fragmentation and migration costs. This is achieved by maximizing resource distribution in addition to minimizing embedding costs. Similarly to the

approach of Huang *et al.*, the authors take into account flow table occupation, with the addition of group table occupation. The proposed strategy is composed of a heuristic algorithm that employs an optimization model as one of the steps of its search for feasible mappings.

Last, Zhong *et al.* (ZHONG et al., 2016) propose an embedding approach that takes into account both controller placement and flow table capacities. The authors represent their solution as an optimization model and, additionally, propose a heuristic and a meta-heuristic algorithm. The proposed approaches aim at minimizing embedding costs, including link bandwidth and flow table occupation. The meta-heuristic algorithm devised by the authors is based on particle swarm optimization.

Table 3.2: Summary of SDN-oriented VNE approaches.

Authors	Objective	Specific Constraints	Optimization Method	Evaluated Topologies	Deployment Coordination
Demirci *et al.*	Balance load or minimize controller delay	Controller positioning	Heuristic	10 topologies from the Internet Topology Zoo (sizes not reported)	Partial
Blenk *et al.*	Minimize control plane latency	Multiple hypervisor placement	Optimal	34 routers (OS3E network), other topologies from the Internet Topology Zoo	Partial
Huang *et al.*	Minimize cost	Flow table occupation	Optimal and heuristic	24 routers, random (GT-ITM)	Partial
Tegueu *et al.*	Minimize cost, maximize distribution	Flow and group table occupation	Optimal and heuristic	41 routers (GEANT network – Europe)	Partial
Zhong *et al.*	Minimize cost	Controller placement, flow table occupation	Optimal and heuristic	65 routers (TA2 network – Austria)	Partial

Table 3.2 summarizes information regarding the SDN-oriented VNE approaches just discussed. Trends similar to those present in traditional VNE approaches can also be observed here. More specifically, most approaches follow the objective of minimizing embedding costs and, additionally, a number of authors propose heuristic (or meta-heuristic) algorithms in order to improve scalability. In contrast to previously analyzed traditional approaches, most papers studied in this sub-area employ topologies based on real networks – *e.g.*, the Internet2 Open Science, Scholarship and Services Exchange network (OS3E) and topologies based on the Internet Topology Zoo[1]. Most importantly, however, the studied approaches only take into account a very limited subset of SDN-specific constraints (namely, controller/hypervisor placement and/or flow/group table occupation),

[1]The Internet Topology Zoo: http://topology-zoo.org/

sometimes to the detriment of basic VNE constraints such as router and link capacities.

3.3 Discussion

The traditional VNE algorithms presented in this chapter can be seen as a first step towards the actual deployment of VNs on physical networks. Mappings computed by a VNE algorithm can be integrated with a platform such as the ones discussed in Section 2.2 (and summarized in Table 2.5). Traditional VNE algorithms, however, are platform-agnostic – *i.e.*, they do not consider any particularities of the virtualization environment other than fundamental aspects such as basic capacity constraints. Therefore, they are unable to ensure the viability of the mappings they generate when deployed on a real substrate network.

The SDN-oriented approaches discussed in Section 3.2 are more closely aligned to our proposal when compared to the traditional VNE approaches presented earlier in Section 3.1. These efforts take into account certain relevant SDN-related operational constraints, such as virtual controller placement and flow table limitations. However, the operational requirements explored by these authors is still restricted to a very limited subset of crucial constraints observed on real SDN/OpenFlow networks. For this reason, mappings generated through these strategies may still be ultimately impossible to instantiate in practice.

Aware of the shortcomings of VNE strategies proposed thus far, our objective is to provide a full framework for the instantiation of virtual networks on top of SDN substrates. Through this approach, we aim at encompassing a rich set of aspects deemed essential to mapping and deploying virtual networks on this type of environment – flow and meter table constraints, virtual controller allocation, channel allocation for in-band switch–controller communication, and physical hypervisor positioning. By taking these operational requirements into account, we are able to ensure with a high level of confidence that all generated VNE mappings are valid while simultaneously preventing performance degradation. It is worth noting that, although we propose an entirely new VNE model as part of our framework, other VNE algorithms could be used in its place in a "plug-and-play" manner. Alternatively, this model could be partially modified in order to incorporate different strategies (*e.g.*, different objective functions) if one is demonstrated to be more effective than the current one.

4 VIRTUAL NETWORK EMBEDDING WITH FLOW-RELATED OPERATIONAL CONSTRAINTS

Next, we present our first step towards coordinating VNE and SDN infrastructures. First, we briefly explain the characteristics of SDN environments considered in this work and provide an overview of our initial approach. Right after, we detail each of its main components. Last, we present an evaluation centered around issues pertaining flow-related operational constraints.

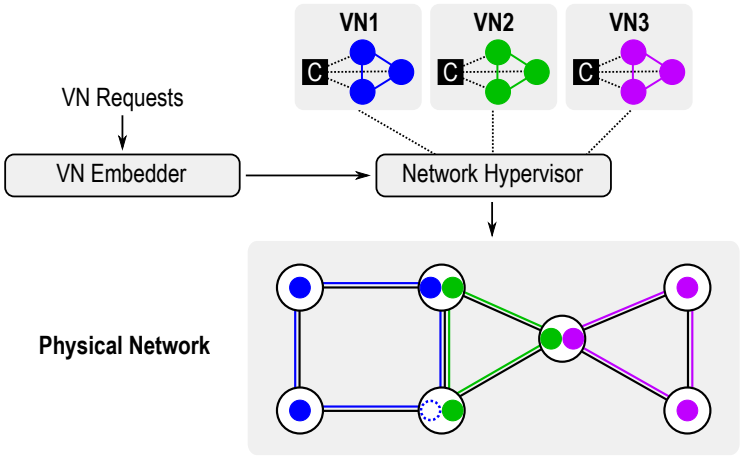
4.1 Multi-tenant Infrastructure Model and Network Virtualization Paradigm Considered

Our proposed approach targets an SDN/OpenFlow-based network environment in which a number of customers (service providers) request and, if possible, are granted virtual infrastructures. More specifically, we focus on correctly provisioning the VNs that interconnect the elements of these infrastructures.

An example of a multi-tenant network virtualization environment is depicted in Figure 4.1. VN requests are received by the infrastructure provider and processed by a VN embedder. Accepted VN requests are ultimately instantiated on top of the physical infrastructure through a network hypervisor, following the mappings produced by the VN embedder. The controller associated with each VN (represented as black boxes with the letter C in the figure), in turn, is hosted within a virtual machine, and communication channels are established between it and the network hypervisor. The hypervisor intermediates flow rule instantiation and monitoring actions sent by VN controllers in order to enforce properties such as isolation at the physical level.

In this work, we consider an SDN virtualization platform capable of providing the aforementioned features, such as OpenVirteX (introduced in Section 2.2). However, it is worth mentioning that our VNE/SDN coordination approach may be adapted to interface with other virtualization platforms, even ones that do not follow the flow-level virtualization model. Moreover, we focus on VN embedding (*i.e.*, not including host embedding), assuming end hosts are located on the premises of the customer.

Figure 4.1: Multi-tenant OpenFlow/SDN-based network virtualization model considered in our approach.



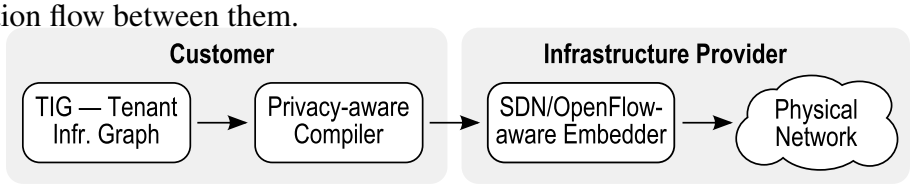
4.2 Overview of our Proposed VNE and SDN Coordination Approach

As previously explained, VN mappings generated by standard VNE algorithms may violate operational requirements when an infrastructure provider attempts to instantiate the requested VNs on a real physical substrate. In order for the VNE algorithm to take such requirements into account, they would have to be part of the VN request provided by the customer. However, we believe it is unreasonable to expect customers to be aware of operational requirements that affect the environment on a physical level. Moreover, customers may be averse to disclosing too much information regarding the internal behavior of their network. Therefore, the main goals of our approach are to: *(i)* allow the customer to represent the needs of his/her VN in a detailed manner; *(ii)* preprocess this detailed representation, removing sensitive information and deriving data regarding the operational requirements associated with this particular request; and *(iii)* embed the requested VNs ensuring both feasibility and adequate performance by making use of this "distilled" information.

Figure 5.2 depicts the components of our proposed approach. The customer first creates a Tenant Infrastructure Graph (TIG), which represents not only virtual routers and links (and their capacity requirements) but also elements (such as hosts and end users) connected to the network, the traffic patterns among them, and the network functions that will be applied to each traffic flow. As some of this information may be considered sensitive by the customer, the TIG is preprocessed by a Privacy-aware Compiler (PAC) running on the customer's premises. This Privacy-aware Compiler allows customers to only reveal as much information about their networks as they want, while still generating

an enhanced VN request that aids the VNE process in order to provide feasible, high-quality VN deployments. The preprocessed request is then sent to the InP, which makes use of our SDN/OpenFlow-aware Embedder in order to properly embed and deploy (although the latter is out of the scope of our work) the customer's network. The main elements of our proposal – Tenant Infrastructure Graphs, the Privacy-aware Compiler, and the SDN/OpenFlow-aware Embedder – will be further explained in the following sections.

Figure 4.2: Overview of our proposed approach, depicting its main elements and the information flow between them.



4.3 Specification of Infrastructure Resources

We now proceed to a detailed explanation of the process that is carried out on the customer's end in order to request a VN.

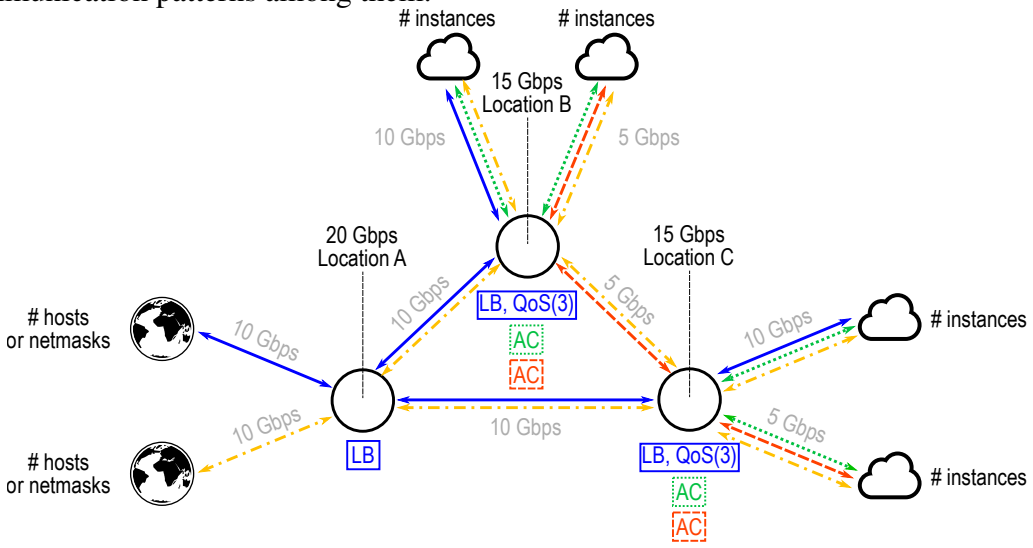
4.3.1 Tenant Infrastructure Graph (TIG) – A Detailed Abstraction of a Virtual Network and its Communication Patterns

The TIG enables customers to represent the needs of their requested VNs with a high level of detail. In addition to the information contained in a standard VN request (*e.g.*, network topology and capacity and location requirements), a TIG also represents: *(i)* elements such as end hosts (represented as network prefixes or individual addresses) connected to the network; *(ii)* the communication patterns among such elements; and (iii) network functions/policies each communication pattern must be subjected to.

Figure 5.3 depicts an example of a TIG. Each cloud represents a group of application servers executing a common task (*e.g.*, within the same tier). Globes represent groups of external users (*e.g.*, network administrators or end users accessing applications running on the customer's premises through the to be deployed VN). Each of these elements has some information associated to it – namely, the number of instances of each group of application instances and the number of network prefixes of each group of exter-

nal users. Last, circles in the graph represent virtual routers, and colored edges represent communication patterns (*i.e.*, traffic flows) among network elements.

Figure 4.3: Tenant Infrastructure Graph representing elements connected to a VN and the communication patterns among them.



Each group of edges represented with the same color and style in Figure 5.3 denotes a distinct traffic flow. As an example, the solid edges represented in blue interconnect end users to externally accessible applications running on the customer's premises (*e.g.*, the front-end of a two-tier web application), while the dotted green edges interconnect the front-end to the application database back-end. Dashed red edges interconnect databases running in different locations for synchronization/replication purposes, while the dashed and dotted yellow pattern provides an administrator access to all applications.

In addition to forwarding packets, routers may need to perform other functions specific to each traffic flow. Some of these functions – Load Balancing (LB), Quality of Service (QoS), and Access Control (AC) – are represented in Figure 5.3. In order to discriminate between different network flows and apply the appropriate functions to each, routers use a number of packet header fields (or combinations of fields[1]). In order to apply load balancing, for example, both the source and destination of a packet should be taken into account. For the purpose of QoS, in turn, the Differentiated Services Code Point (DSCP) header field may be used. The TIG represents these (combinations of) fields as sets of Traffic Discriminators (TDs). Moreover, each TD contains a number of entries – *i.e.*, the number of different values a given (combination of) header field(s) may be set to. In the aforementioned QoS example, the DSCP field may be set to a value between 0 and

[1]The source or destination of a packet, for example, may be composed of a combination of the IP address, MAC address, network protocol, and port fields.

63. Therefore, the number of entries in a traffic discriminator that uses this field may be anywhere between 2 (if only two different QoS classes are used) and 64 (if all possible classes are used).

Through the information represented in this graph, it is possible to accurately derive the number of flow rules each router in a VN will need in order to ensure its correct and optimal operation. Moreover, as shown in Figure 5.3, standard VNE constraints (router throughput, link bandwidth, and location requirements) are also represented in a TIG.

We emphasize that the TIG has been enhanced with additional elements since this first iteration of our work was completed. The full definition of the TIG, as well as a formal specification of it, are presented in Chapter 5.

4.3.2 Privacy-aware Compiler

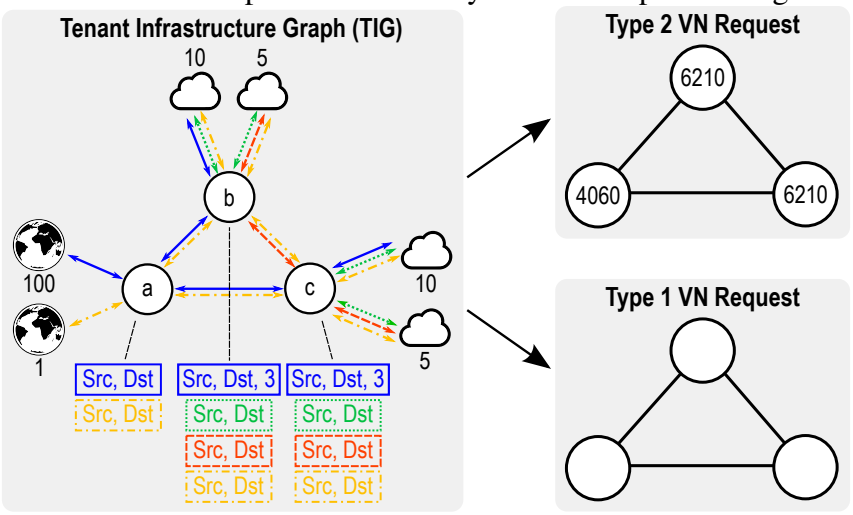
While a TIG enables the representation of operational constraints associated with the instantiation of each VN on the physical infrastructure of an InP, it also exposes information that the customer may consider sensitive. As an example, the TIG exposes the location of entities within the network and interactions between different applications with a high level of detail, which could be used to infer the business model of the VN requester or potentially as an attack vector. Therefore, TIGs are expected to be preprocessed by a Privacy-aware Compiler on the customer's end before being sent to an InP.

As previously mentioned, the TIG represents distinct communication patterns within elements of the requested VN. A number of flow rules will ultimately need to be installed on each router in order to ensure the correct operation of each traffic flow. The number of necessary flow rules depends on a number of pieces of information, namely: *(i)* the number of network applications and/or network prefixes of external users associated with each flow; *(ii)* the number of traffic discriminators associated with each router for handling each network flow; and *(iii)* the number of entries of each traffic discriminator.

The left side of Figure 4.4 shows an example TIG populated with numerical values. The communication pattern represented in solid blue interconnects a number of users (comprising 100 network masks) to a number of applications through the customer's VN hosted on the physical infrastructure of the InP (10 app instances connected to virtual router b and 10 connected to router c). The dotted green patterns interconnect each group of 10 application instances to a group of 5 database servers. The dashed red pattern,

in turn, interconnects both groups of database (server) instances. Last, the dashed and dotted yellow pattern interconnects all network services to a specific external network mask (used by a network administrator to manage all network services).

Figure 4.4: Possible outputs of the Privacy-aware Compiler for a given TIG.



If no network functions need to be applied to a particular communication pattern, its flow table requirements are calculated by adding up the total number of source-destination pairs (including all applications and network masks) that are part of this traffic. The pattern represented in dashed and dotted yellow on the TIG shown in Figure 5.4 interconnects a single external user (or network mask) to all (30) applications running within the network, adding up to a total of 60 source-destination pairs (considering both directions of each possible communication flow). Therefore, this flow requires a total of 60 rules on each router it traverses (a, b, and c). This is also the case for the flow represented in solid blue when traversing router a. This flow interconnects 100 external network prefixes to 20 network applications, which – accounting for all possible combinations in both directions – adds up to a total of 4,000 source-destination pairs (and, therefore, 4,000 rules to be installed in a). If additional traffic discriminators are used, they enter the calculation as multiplying factors – the number of flow rules is multiplied by the number of entries in each discriminator. As an example, if the DSCP field is used as a discriminator with 3 possible QoS values (*i.e.*, 3 different traffic classes), the number of flow rules is multiplied by 3. In the example shown in Figure 5.4, packets that belong to the solid blue communication pattern traversing routers b and c are subjected to this traffic discriminator. Therefore, the total number of source-destination pairs (2,000) connected (directly or indirectly) to each of these routers is multiplied by the number of entries in the respective traffic discriminator (3), adding up to a total of 6,000 flow rules to be installed on routers

b and c.

After being processed, the TIG is compiled into a VN request which will be shared with the InP. This request may contain more or less information according to what the customer is willing to reveal. The right side of Figure 5.4 shows the two different types of requests we consider. A "type 1" request is equivalent to a standard VN request. A "type 2" request, in contrast, includes the accurate number of flow rules required by each router. While not represented in this figure, standard VN requirements – namely, the throughput capacity of routers, bandwidth capacity of links, and location constraints – are also considered for both types. We envision that a larger gradient of VN request types could be considered. As an example, an intermediate level between our "type 1" and "type 2" requests could contain estimates for flow table requirements rather than exact values. We intend to further explore this aspect in future work.

Similarly to the TIG, the PAC has also been enhanced to cope with new elements after the completion of this phase. A description of these enhancements can be found in Chapter 5.

4.4 SDN/OpenFlow-aware Embedder

The SDN/OpenFlow-aware Embedder is run by the InP, receiving VN requests that have been preprocessed by the Privacy-aware Compiler and embedding them on a physical substrate. It has been modeled as an Integer Linear Program (ILP), and its formulation is presented next. Before presenting our model, we introduce the syntax for our formulation. Capital letters represent sets or variables, and superscripts denote whether a given set or variable refers to physical (P) or virtual (V) entities, or to routers (R) or links (L). Moreover, subscript letters represent indices associated to variables or paths. For ease of reference, the full list of symbols used in this model is summarized in Table 4.1.

Topologies. The topology of each VN request, as well as that of the physical network, are represented as a directed graph $N = (R, L)$. Bidirectional links are represented by pairs of edges in opposite directions. Each virtual router is mapped to a single physical router, while virtual links may be mapped to either a physical link or a substrate path.

Physical and Virtual Capacities. The capacity of physical routers is measured in terms of throughput. The capacity of a physical router i is expressed as T_i^P. Likewise, $T_{r,i}^V$ denotes the throughput required by virtual router i from VN r. Likewise, the bandwidth capacity of a physical link (i, j) is represented as $B_{i,j}^P$, and the bandwidth requirement of

Table 4.1: Symbols used in this model.

Decision Variables	
$A^R_{i,r,j} \in \{0,1\}$	Router allocation.
$A^L_{i,j,r,k,l} \in \{0,1\}$	Link allocation.
Input Sets	
R^P	Physical routers.
R^V	Virtual routers.
L^P	Physical links.
L^V	Virtual links.
T^P	Throughput of physical routers.
T^V	Throughput of virtual routers.
B^P	Bandwidth of physical links.
B^V	Bandwidth of virtual links.
S^P	Location of physical routers.
S^V	Location requirement of virtual routers.
F^P	Flow table capacity of physical routers reserved for type 2 requests.
F^V	Flow table requirement of virtual routers in type 2 requests.
$\mathcal{F}^P$	Flow table capacity of physical routers reserved for type 1 requests.
$\mathcal{F}^V$	Flow table requirement of virtual routers in type 1 requests.
E^R	Previously mapped routers.
E^L	Previously mapped links.

a virtual link (k, l) from VN r is represented as $B^V_{r,k,l}$.

Locations. All physical routers are associated with a location identifier – an integer number stored in set S^P. This enables customers to demand some of their virtual routers to be instantiated in specific geographic locations. If a virtual router has a location requirement, it is stored in set S^V.

Flow Table Usage. As previously explained, the flow table requirements of VN requests are calculated by the Privacy-aware Compiler based on a given TIG and added to the generated request. The flow table capacity of a physical router is divided in two – one part (the majority of the available flow table space) will be used for requests with specific flow table requirements (type 2), while the remaining capacity will be used for type 1 requests. The flow table capacity of a physical router i reserved for type 2 requests is represented as F^P_i, while the remaining capacity reserved for type 1 requests is represented as $\mathcal{F}^P_i$. As for virtual routers, those that belong to type 2 VN requests have their flow table requirement represented as $F^V_{r,j}$, while the estimated flow table requirement for type 1 requests is represented as $\mathcal{F}^V_{r,j}$.

Previous Mappings. As VN requests are handled in an online manner, the mappings of previously embedded VNs must be taken into account and preserved while processing new incoming requests. Mappings of previously embedded routers and links are

stored in sets $E^R_{i,r,j}$ and $E^L_{i,j,r,k,l}$, respectively.

Variables. The variables of the ILP model indicate the optimal placement of routers and links on the substrate.

- $A^R_{i,r,j} \in \{0,1\}$ – Router allocation, indicates whether virtual router j from VN r is embedded on physical router i.
- $A^L_{i,j,r,k,l} \in \{0,1\}$ – Link allocation, indicates whether virtual link (k,l) from VN r is embedded on physical link (i,j).

Next, we present the objective function of our SDN/OpenFlow-aware Embedder and its constraint sets (C1–C9). The objective function aims at minimizing overall flow table occupation – *i.e.*, the aggregated number of flow table entries needed to instantiate incoming VN requests. The calculation of flow table usage will be presented in further detail after all constraint sets are listed and explained.

Objective

$$Minimize \sum_{(i,j)\in L^P, r\in N^V, (k,l)\in L^V} \frac{(\min_{(F^V_{r,k},F^V_{r,l})} + \min_{(\mathcal{F}^V_{r,k},\mathcal{F}^V_{r,l})}) A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})}{2} + \sum_{i\in R^P, r\in N^V, k\in R^V} (F^V_{r,k} + \mathcal{F}^V_{r,k}) A^R_{i,r,k}$$

Subject to

$$\sum_{j\in R^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(T^V_{r,k},T^V_{r,l})} A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})}{2} + \sum_{r\in N^V, k\in R^V} T^V_{r,k} A^R_{i,r,k} \leq T^P_i \quad \forall i \in R^P \tag{C1}$$

$$\sum_{r\in N^V, (k,l)\in L^V} B^V_{r,k,l} A^L_{i,j,r,k,l} \leq B^P_{i,j} \quad \forall (i,j) \in L^P \tag{C2}$$

$$\sum_{j \in R^V} A^R_{i,r,j} \leq 1 \qquad \forall i \in R^P, r \in N^V \quad \text{(C3)}$$

$$\sum_{j \in R^P, r \in N^V, (k,l) \in L^V} \frac{\min_{(F^V_{r,k}, F^V_{r,l})} A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})}{2} + \sum_{r \in N^V, k \in R^V} F^V_{r,k} A^R_{i,r,k} \leq F^P_i \qquad \forall i \in R^P \quad \text{(C4)}$$

$$\sum_{j \in R^P, r \in N^V, (k,l) \in L^V} \frac{\min_{(\mathcal{F}^V_{r,k}, \mathcal{F}^V_{r,l})} A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})}{2} + \sum_{r \in N^V, k \in R^V} \mathcal{F}^V_{r,k} A^R_{i,r,k} \leq \mathcal{F}^P_i \qquad \forall i \in R^P \quad \text{(C5)}$$

$$\sum_{i \in R^P} A^R_{i,r,j} = 1 \qquad \forall r \in N^V, j \in R^V \quad \text{(C6)}$$

$$\sum_{j \in R^P} A^L_{i,j,r,k,l} - \sum_{j \in R^P} A^L_{j,i,r,k,l} = A^R_{i,r,k} - A^R_{i,r,l} \qquad \forall r \in N^V, (k,l) \in L^V, i \in R^P \quad \text{(C7)}$$

$$jA^R_{i,r,k} = lA^R_{i,r,k} \qquad \forall (i,j) \in S^P, r \in N^V, (k,l) \in S^V \quad \text{(C8)}$$

$$A^R_{i,r,j} = E^R_{i,r,j} \qquad \forall (i,r,j) \in E^R \quad \text{(C9)}$$

$$A^L_{i,j,r,k,l} = E^L_{i,j,r,k,l} \qquad \forall (i,j,r,k,l) \in E^L \quad \text{(C10)}$$

Constraint sets C1 and C2 ensure, respectively, that the throughput capacity of physical routers and that the bandwidth capacity of physical links is not exceeded. C3 prevents multiple virtual routers from a single VN request from being mapped to the same physical router. C4 and C5 ensure that the flow table capacity of physical routers is not exceeded. Constraint set C4 deals with type 2 requests, while C5 handles type 1 requests. C6 guarantees that all routers in an incoming VN request are mapped to physical routers. C7 ensures that each virtual link is mapped to a physical path whose end-points match the physical routers hosting the end-points of this link. C8 ensures all virtual routers with location requirements are mapped to physical routers in the correct location. Last,

constraint sets C9 and C10 preserve the mappings of previously embedded VNs.

For the sake of clarity, our objective function, as well as constraint sets C1, C4, and C5, are shown in non-linear form. However, in practice, they are linearized by replacing the multiplication $A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})$ with an auxiliary variable $Z_{i,j,r,k,l} \in \{0, 1\}$ and adding constraint sets C11, C12, and C13 – shown below – to the model. Moreover, function $\min(a, b)$ returns the lowest number between a and b and can be defined as $\frac{1}{2}(a + b - |a - b|))$.

$$Z_{i,j,r,k,l} <= A^L_{i,j,r,k,l} \qquad \forall (i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C11)}$$

$$Z_{i,j,r,k,l} <= (1 - A^R_{i,r,k}) \qquad \forall (i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C12)}$$

$$Z_{i,j,r,k,l} >= (1 - A^R_{i,r,k}) + A^L_{i,j,r,k,l} - 1 \qquad \forall (i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C13)}$$

In order to properly account for flow table usage, the objective function must not only consider explicit flow table requirements of virtual routers, but also the flow rules that must be installed on auxiliary routers through which virtual links traverse. The number of rules that must be installed on the auxiliary routers used by a virtual link (k, l) corresponds to the lowest number between the flow table requirements of virtual routers k and l. In the objective function, the first summation refers to the flow table constraints of auxiliary routers, while the second one refers to that of physical routers hosting virtual routers of each network. Constraint sets C1, C4, and C5 employ the same strategy in order to compute the throughput and flow table usage of auxiliary routers.

4.5 Evaluation

We now proceed to a performance evaluation of our proposed VNE and SDN coordination approach. Experiments were performed on a machine with an Intel Core i5 4278U CPU, 8 GB of RAM and Operating System Mac OS X 10.10.5. The previously introduced ILP model was implemented and run using the IBM ILOG CPLEX Interactive Optimizer 12.4.

4.5.1 Workloads

In order to evaluate our proposal, we developed a simulator that creates virtual topologies according to a series of parameters. Each virtual topology is then converted to a VN request in the format required by our ILP model. The simulator is run for 500 rounds, generating a new request on each one[2]. If accepted, VNs remain embedded for 25 rounds before being removed.

Fixed parameters. In all experiments, physical and virtual topologies are generated with BRITE using the Barabási-Albert model (ALBERT; BARABÁSI, 2000). Generated physical networks contain 100 routers. Physical routers have a throughput capacity of 150 Gbps, while physical links have a bandwidth capacity of 30 Gbps. The flow table of each device is capable of storing up to 16,000 rules. Physical routers are uniformly distributed among 16 geographic locations.

Each generated VN request contains 5 routers. Virtual router throughput and link bandwidth requirements are, respectively, 50 Gbps and 10 Gbps. Each VN has two edge routers with randomly generated location requirements. 50% of the generated VNs are type 1 requests, while the remaining 50% are type 2 requests. Flow table requirements of type 2 requests are set to 3,000 rules per router, while those of type 1 requests (which are not known) are treated in different ways according to the experiment being performed.

Variable parameters. We performed a number of different experiments with variations regarding flow table space reserved for type 2 requests as well as how flow table requirements are considered. The first three experiments – henceforth referred to as "Flow-70/30", "Flow-80/20", and "Flow-90/10" – reserve 70%, 80%, and 90% of each physical router flow table space for type 2 requests. The remaining flow table space is reserved to accommodate type 1 requests. As we do not know their requirements, a minimal set of 1,500 rules is reserved for each router from these networks. The idea here is to deliberately allocate little table space to these virtual routers[3].

In the fourth experiment, all flow table space is used to embed type 2 requests, while accepted type 1 requests are supported on a "best effort" manner. More specifically, type 1 requests are embedded as long as their other capacity requirements (throughput and bandwidth) can be fulfilled, with no flow table space guarantees. This experiment is referred to as "Flow-100/0". In the last experiment, no flow table requirements are

[2] On average, each request was optimally mapped in less than 5 seconds.

[3] The amount of flow table space reserved for each router in type 1 requests may be fine-tuned as desired by the InP.

considered by the embedding model. This experiment behaves similarly to an environment running a traditional VNE algorithm and is carried out to assess the impact of our proposed approach. This experiment is referred to as "NoFlow".

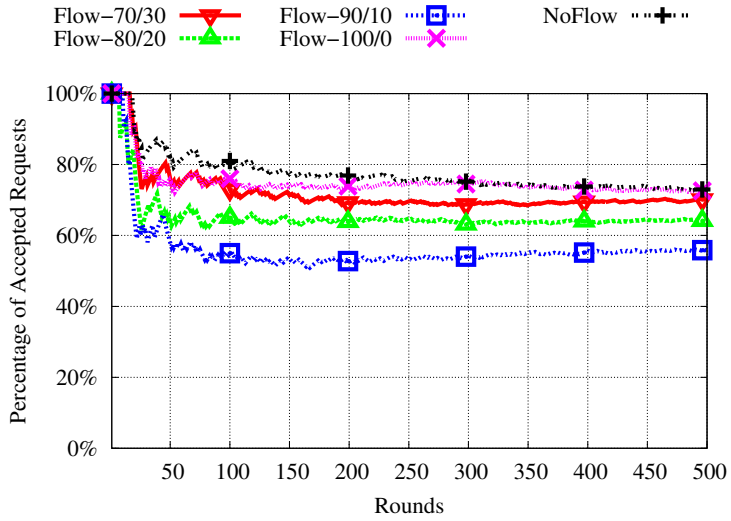
4.5.2 Results

We first analyze the overall acceptance rate in all experiments, shown in Figure 5.5. Each point in this graph represents the average acceptance rate from the beginning of the experiment up until the round depicted in the horizontal axis. The acceptance rates achieved throughout experiments Flow-70/30, Flow-80/20, and Flow-90/10 were, respectively, 70.2%, 64.2%, and 55.8%. As the residual flow table space for type 1 requests decreases, more VNs of this type (which require less resources and represent half of all generated requests) are rejected, leading to lower acceptance rates. Although the overall acceptance rate is lower, this, in turn, favors the acceptance of a higher amount of type 2 requests. This is a desirable outcome for the InP in order to prevent under- or overestimation of resources (as type 2 requests contain precise flow table occupation requirements), potentially leading to improved resource usage (in the case of overestimation) and/or lower rates of controller intervention (if flow table requirements are underestimated). The acceptance rates of individual types of requests and their effect on the rate of controller interventions will be further analyzed in the remainder of this section.

The acceptance rates observed in experiments Flow-100/0 and NoFlow were higher than those of other experiments – 72.6% and 73%, respectively. This is due to the former not reserving any flow table space for type 1 requests and the latter disregarding flow table requirements entirely. While seemingly a positive result at first glance, this is likely to result in severe underestimation of resources needed to adequately support the embedded VNs, potentially leading to high rates of controller intervention. We emphasize that both Flow-100/0 and NoFlow are used as baseline scenarios, *i.e.,* the best results one would achieve in terms of accepted requests at the cost of compromising network predictability and, in extreme cases, its technical feasibility.

Next, Figure 4.6 depicts the acceptance rate of each type of request in all evaluation scenarios. In experiments Flow-70/30, Flow-80/20, and Flow-90/10, the acceptance rates of type 1 requests were, respectively, 73.7%, 57.7%, and 30.83%. Acceptance rates of type 2 requests observed for the same experiments were of 67.18%, 69.77%, and 82.04%, respectively. These increasing acceptance rates of type 2 requests are a desirable

Figure 4.5: Overall acceptance rate in all experiments.

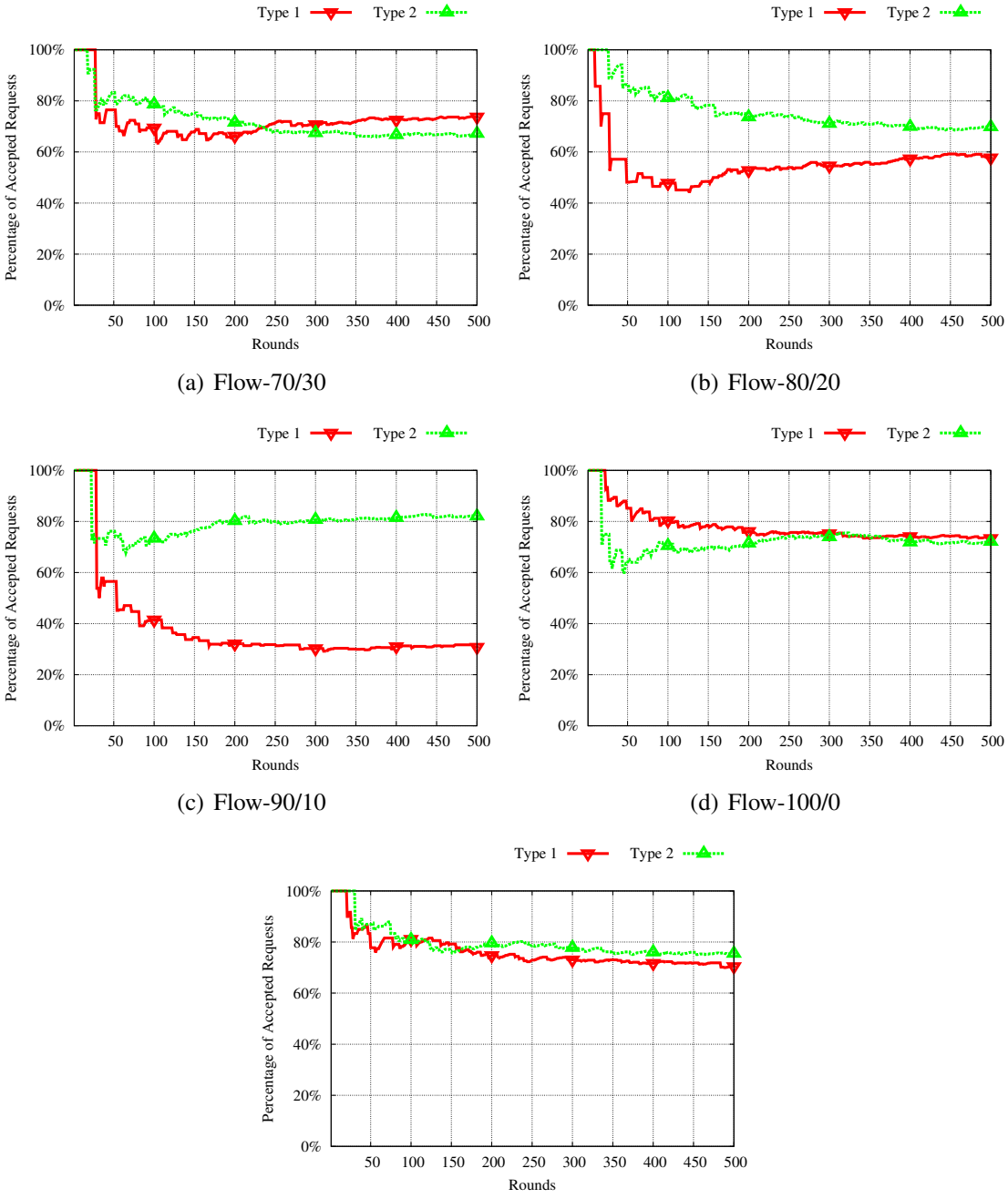


outcome for InPs, as they would likely desire to prioritize the embedding of this type of request. This happens as a result of the fine-tuning of the amount of flow entries reserved for each type of request, leading to an acceptance rate of over 80% for type 2 requests in the most extreme scenario (Flow-90/10). InPs may fine-tune these reservations as desired in order to allow more or less of each type of request to be embedded and potentially minimize issues caused by type 1 requests (as a result of under- or overestimation of operational requirements). Moreover, all scenarios exhibit variations in acceptance rates within the first 250 rounds. In scenario Flow-70/30, the acceptance rate of type 1 requests is initially higher than that of type 2 requests. In scenarios Flow-80/20 and Flow-90/10, in turn, acceptance rates for both types of requests either increase or decrease during the first 250 rounds. This can be attributed to two factors. First, the substrate is completely empty at the beginning of the experiment, requiring some time for acceptance rates to stabilize. Second, when the average acceptance rate is calculated considering a lower number of rounds (as is the case in the early stages of the experiment), each acceptance or rejection has a higher impact on the calculated average. In all cases, acceptance rates become stable towards the last 250 rounds.

In the remaining experiments (Flow-100/0 and NoFlow), acceptance rates of type 1 requests were, respectively, 73.4% and 70.4%. Acceptance rates of type 2 requests, in turn, were of 72.1% and 75.49%, respectively. As previously explained, the former does not reserve any flow table space for type 1 requests while the latter completely disregards flow table requirements. Therefore, the main causes of rejection are likely related to topological factors or throughput/bandwidth resource scarcity, leading to similar acceptance

rates for both types of requests on both experiments.

Figure 4.6: Acceptance rate of requests per TIG type in all experiments.



(a) Flow-70/30

(b) Flow-80/20

(c) Flow-90/10

(d) Flow-100/0

(e) NoFlow

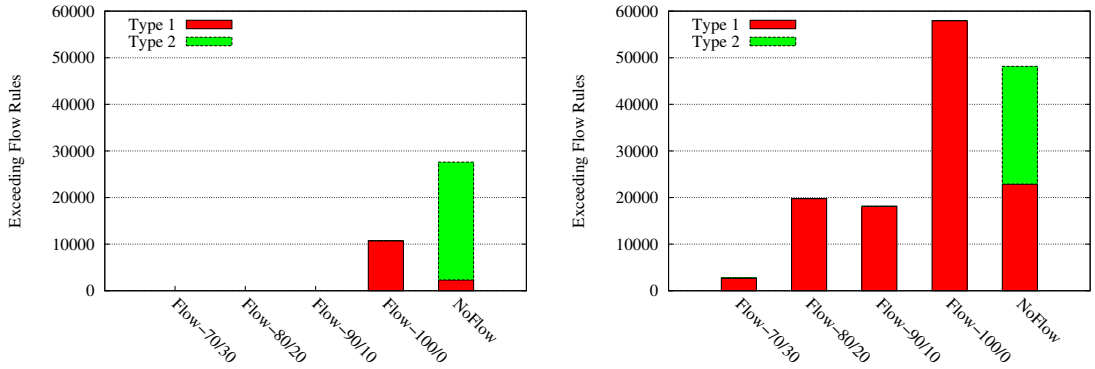
Last, we analyze the potential impacts of accurately or inaccurately estimating operational requirements. To this end, we consider that VN requests in each scenario have been embedded and deployed, and we calculate the number of flow rules each router would need to have installed in order to support the embedded VNs. We first assume the 1,500 rules reserved for each router of type 1 requests were sufficient. Afterwards, we assume this lower bound was not adequate and that type 1 requests would actually

require 3,000 flow rules per router. These scenarios represent extreme cases – assuming either that a minimal set of flow rules was sufficient or that actual requirements exceed this lower bound by a factor of 2. Our goal is to determine in which cases the number of necessary flow rules would exceed the capacity of physical routers. As previously mentioned, if physical devices are not able to accommodate all necessary flow rules, they would need to frequently contact the controller in order to handle incoming packets. This, in turn, would likely degrade the performance of physical devices, hindering the quality of service experienced by customers.

Figure 4.7 depicts the average number of flow rules that would exceed the capacity of physical routers in each experiment, considering the scenarios just described. Assuming the flow table usage of type 1 requests fits within the minimal provided flow table space, no exceeding rules were observed in scenarios Flow-70/30, Flow-80/20, and Flow-90/10. The ILP model used in these experiments takes into account flow table constraints for both types of requests, ensuring that the capacity of physical devices will not be exceeded as long as the reserved flow table space is sufficient. In experiment Flow-100/0, in turn, the average number of exceeding flow rules was 10,732, all belonging to type 1 requests. This is due to this experiment disregarding flow table requirements for this type of request, embedding them in a "best effort" manner. In experiment NoFlow, which mirrors the behavior of a standard VNE algorithm without flow-related constraints, the average number of exceeding flow rules was significantly higher, with both types of VN requests contributing to flow table saturation. More specifically, an average of 27,593 exceeding rules were observed (2,331 incurred by type 1 requests and 25,262 incurred by type 2 requests). The substantial numbers of exceeding flow rules observed in scenarios Flow-100/0 and NoFlow highlight the importance of considering this operational constraint in the embedding process.

Last, assuming the flow table usage of type 1 requests exceeds the minimal provided flow table space (a likely scenario in practice), all experiments exhibit flow table saturation. In experiments Flow-70/30, Flow-80/20, and Flow-90/10, the average numbers of exceeding flow rules were, respectively, 2,731, 19,783, and 18,130. Although these experiments took into account flow-related constraints, assuming flow table usage exceeds the established lower bounds led to significant saturation – particularly in scenarios Flow-80/20 and Flow-90/10, in which less resources were reserved for this type of request. The highest numbers of exceeding flow rules were observed in experiments Flow-100/0 and NoFlow – which, as previously explained, either do not reserve flow table

Figure 4.7: Number of flow rules exceeding the capacity of physical routers.



(a) Assuming flow table space allocated for type 1 requests was sufficient.

(b) Assuming flow table space allocated for type 1 requests was insufficient.

space for type 1 requests or disregard flow constraints entirely. The average amount of exceeding flow rules observed in experiment Flow-100/0 was of 57,970 – all incurred by type 1 requests. In experiment NoFlow, an average of 22,886 exceeding flow rules were incurred by type 1 requests and 25,262 by type 2 requests, adding up to a total of 48,148. These results evidence that, in addition to the aforementioned importance of considering operational constraints, accurately determining flow table requirements plays a crucial role in ensuring the feasibility of supporting the embedded VNs. Regarding the former, it is important to note that, with the exception of scenario NoFlow, all rules of type 2 requests were properly installed in physical routers. Therefore, these VNs will be able to operate with minimal controller interventions, minimizing potential negative impacts on quality of service (as controller intervention may increase latency by up to twice the round-trip time between the switch and the controller (CURTIS et al., 2011)).

In summary, through the evaluation conducted thus far, we demonstrated that taking operational constraints into account is of paramount importance to maintain the "health" of the environment. Neglecting such constraints may render the environment unable to cope with a substantial number of flow rules and, therefore, susceptible to performance degradation or instability. With this knowledge, we now proceed to the implementation of the full architecture we envisioned, detailed in Chapter 5.

5 VIRTUAL NETWORK EMBEDDING WITH A RICH SET OF OPERATIONAL CONSTRAINTS

In this section, we present our fully realized architecture, encompassing all operational constraints included in the scope of this thesis. We first reiterate and explain in further detail each operational constraint and how they affect virtual network embedding. Next, we explain our approach to mitigate the issue of solution space reduction – a byproduct of considering a large number of constraints. We then present the formalization of our VN Embedder as an MIP (Mixed Integer Program) model, as well as its overarching framework. Last, we discuss the results of a second round of experiments, focusing on the ability of our solution to handle a rich set of operational constraints simultaneously.

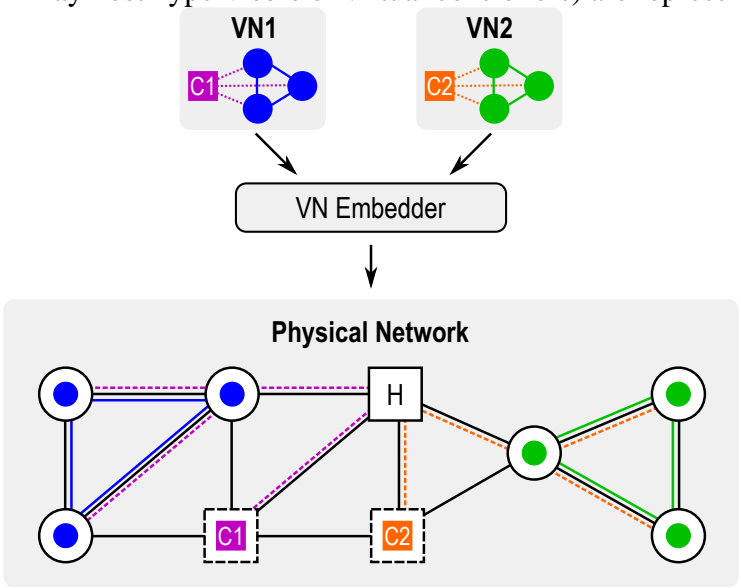
5.1 SDN/OpenFlow-related Operational Constraints

In this work, we consider an SDN/OpenFlow-based network infrastructure capable of hosting virtual networks on top of it. Each VN is controlled by an individual entity independent from the Infrastructure Provider and is created or removed on-demand. Each incoming VN request is processed by a VN embedder, which attempts to map the elements within the VN request (*i.e.*, virtual routers, links, and controller) to their physical counterparts. If sufficient resources are available, the VN is ultimately instantiated on top of the physical infrastructure.

An example of an SDN-based multi-tenant network virtualization environment is depicted in Figure 5.1. Two VN requests, VN1 and VN2, each composed of 3 routers (represented as circles), one controller (C1 and C2, respectively), and a number of links between the routers and controllers in each VN (represented as solid and dashed lines), are received by the InP. Once processed by the VN embedder, both VNs are instantiated on top of the physical network. Each virtual router is mapped to a physical router, while each virtual controller is hosted on a Virtual Machine (VM) running on a physical server. Moreover, a virtual link between any two virtual nodes is established on a path between the physical nodes hosting each one.

Due to the intricacies of mapping VNs on top of an SDN environment, taking into account only traditional, platform-agnostic constraints is not sufficient. These platform-agnostic constraints include router throughput capacity and location requirements, as well

Figure 5.1: Multi-tenant OpenFlow/SDN-based network virtualization model considered in this thesis. Routers are represented as circles, while virtual controllers and physical machines (which may host hypervisors or virtual controllers) are represented as squares.



as link bandwidth and delay. In addition to these, we consider a richer set of SDN-related constraints in order to ensure the proper mapping and operation of VNs. Each of these additional SDN-related constraints is described next.

Flow table capacity. The OpenFlow protocol employs the concept of flow rules in order to define the behavior of physical forwarding devices. Each flow rule contains one or more matching fields (*e.g.*, source and destination IP) as well as a number of actions (*e.g.*, drop the packet or forward it through a specific port). Whenever a device receives an incoming packet, it attempts to find a match between the packet and the matching fields in its flow rules. If a match is found, the actions defined in the appropriate flow rule are applied to the packet. If not, the controller must be contacted in order to install the proper flow rule on the forwarding device. Flow rules commonly need to combine a number of different matching fields in order to properly route packets as well as to enforce network policies, potentially leading to very large sets of flow rules to be stored in each device. As storage space is limited, it is crucial to ensure the flow table capacity of each device is not exceeded.

SDN controllers. Each embedded virtual network must have its own virtual controller, which will communicate with a physical hypervisor in order to manage virtual devices. The communication between OpenFlow devices and controllers should be kept as fast as possible in order to ensure virtual networks do not suffer significant degradation when controller intervention is necessary. Therefore, virtual controller placement must be

performed as close to optimality as possible.

Switch–controller communication channels. In order to ensure the controller of a VN is able to reach the forwarding devices hosting the virtual routers of this VN, communication channels must be established between those entities. These communication channels are not considered in traditional VNE models. Moreover, SDN/OpenFlow network emulators such as Mininet assume this communication is performed in an "out of band" manner (*i.e.*, through separate communication channels that interconnect each device directly to the controller). This assumption is not realistic, and would either be extremely costly or impossible depending on the topology of the physical network. Therefore, the embedding model should explicitly establish these channels, including well defined bandwidth allocations. In the example shown in Figure 5.1, the communication channels between controllers C1 and C2 and the virtual devices managed by them are represented as dashed lines.

SDN hypervisors. With regards to allocating switch–controller communication channels on the physical side, another operational constraint must be considered. In order to manage the flow table of a virtual router (*e.g.*, to add or remove flow rules), a virtual controller must first send the appropriate command to a physical hypervisor. The physical hypervisor, in turn, checks the validity of the received command (and may apply changes, if necessary) before forwarding it to the physical device hosting the appropriate virtual router. Therefore, the distance between the suggested location of the virtual controller and the physical hypervisor – as well as the distance between the hypervisor and each virtual router – must be considered by the embedding algorithm. Whether there is a single or multiple hypervisors, we assume this to be a hard constraint on the side of the Infrastructure Provider. Therefore, in order to properly place virtual controllers and establish the appropriate communication channels, the embedding method must be aware of the location of hypervisors on the physical substrate. This issue is depicted in Figure 5.1, as virtual controllers C1 and C2 are placed near a hypervisor H which is, in turn, in near proximity of the virtual devices they manage. Moreover, communication channels between these entities are established on a path that passes through hypervisor H.

Meter table capacity. Meter tables were added to OpenFlow in version 1.3 and enable a number of QoS-related operations. One such operation is rate limiting, which is crucial in a multi-tenant environment in order to restrict the bandwidth of each virtual link to the allocated limit. The maximum number of entries that can be added to the meter table of each device must be accounted for, ensuring enough space is reserved for

necessary entries.

5.2 Mitigating Solution Space Reduction

While the necessity of considering underlying operational constraints in VNE is clear, doing so may also lead to unintended side effects. Each constraint added to an optimization model limits the solution space to a varying degree. As we consider a significantly wider set of constraints in comparison to a standard VNE model, this reduction can be drastic. In the context of VNE, this could lead to a number of issues, such as less efficient resource usage, fragmentation, and a significant increase in the rejection of VN requests. Balancing the need to ensure VNE mappings are valid in real environments and the issue of solution space reduction is a challenge in and of itself.

To mitigate this issue, we propose a different approach towards handling VN requirements. We allow the flexibilization of specific traditional VNE constraints – namely, router throughput, link bandwidth, and delay. This is achieved by allowing customers to express their requirements as a range, which stretches from a *minimal* requisite to a *desired* level of fulfillment. By allowing customers to be in control of this range, we ensure their minimal requirements will be met – even though the amount of resources allocated to their VN may not achieve their desirable level.

On the InP's side, the VNE model leverages the concept of soft constraints in mathematical optimization to materialize the proposed flexibilization. It is important to note, however, that VN requisites are only flexibilized when it would be otherwise impossible to embed a VN, which would normally lead to the outright rejection of the request. In other words, the VNE model always attempts to fulfill VN requirements as much as possible. Moreover, the minimal level of each requirement is always guaranteed, and SDN-related operational requirements are still satisfied to their full extent.

The main objective of this flexibilization is to turn an unconditional rejection into a one-step negotiation between customer and InP. We envision this will lead to a more favorable outcome for both parties. The InP will be able to host a higher number of VNs, potentially leading to an increase in revenue. Customers, in turn, will be granted a VN that meets their minimal requirements without having to wait for more resources to become available. Moreover, if additional resources are made available in the future, the customer may be able to augment their VN through further negotiation with the InP (although this supplementary, post-embedding negotiation step – which would include issues such as

resource expansion and, potentially, migration – is out of the scope of this thesis).

5.3 SDN/OpenFlow-aware Embedder

The SDN/OpenFlow-aware Embedder handles VN requests received by the InP, embedding them on a physical substrate. It is modeled as a Mixed Integer Program (MIP). We first introduce the inputs to our model, followed by its variables, objective function, and constraints. For ease of reference, the full list of symbols used in this model is summarized in Table 5.1.

Physical and virtual router capacities. R^P represents the set of physical routers, while R^V, the set of virtual routers. Each physical router is associated with inputs that represent its residual capacity, while virtual routers are associated with capacity requirements. The residual throughput capacity of a physical router i is represented as T^P_i. The required throughput of a virtual router is modeled as a flexible requirement. The desired throughput capacity of virtual router j from VN r is represented as $T^V_{r,j}$, while the minimum throughput for the same router is represented as $\mathcal{T}^V_{r,j}$. Similarly, the residual flow table and meter table capacities of physical routers are represented as F^P_i and M^P_i, respectively. The flow and meter table requirements of virtual routers, modeled as hard constraints, are represented as $F^V_{r,j}$ and $M^V_{r,j}$, respectively. Each physical router is also associated with a location, stored in set S^P. Virtual routers may optionally have location requirements, which are stored in set S^V.

Physical and virtual link capacities. Physical and virtual links are represented as edges between pairs of physical or virtual nodes. Physical links between routers are stored in set E^P, while virtual ones are stored in E^V. Likewise, physical and virtual links between routers and other elements (physical machines, hypervisors, or virtual controllers) are stored in $\mathcal{E}^P$ and $\mathcal{E}^V$, respectively. Each link is associated with bandwidth capacity and delay – both modeled as flexible constraints. The residual bandwidth capacity and delay of a physical link (i, j) are represented as $B^P_{i,j}$ and $D^P_{i,j}$, respectively. The desired bandwidth capacity of a virtual link (k, l) from VN r is represented as $B^V_{r,k,l}$, while the minimum acceptable bandwidth is represented as $\mathcal{B}^V_{r,k,l}$. Analogously, the desired delay of a virtual link is represented as $D^V_{r,k,l}$, while the maximum tolerated delay is represented as $\mathcal{D}^V_{r,k,l}$.

Controllers and hypervisors. Virtual controllers are stored in set C^V, while physical machines that host them are stored in C^P. Each controller j from VN r is associated

Table 5.1: Symbols used in this model.

Decision Variables	
$A^R_{i,r,j} \in \{0,1\}$	Router allocation.
$A^C_{i,r,j} \in \{0,1\}$	Controller allocation.
$A^L_{i,j,r,k,l} \in \{0,1\}$	Link allocation.
Auxiliary Variables	
$X^T_{r,j} \in \mathbb{R}$	Fulfillment of throughput capacity.
$X^B_{r,k,l} \in \mathbb{R}$	Fulfillment of bandwidth capacity.
$X^D_{r,k,l} \in \mathbb{R}$	Fulfillment of delay constraint.
Input Sets	
R^P	Physical routers.
R^V	Virtual routers.
T^P	Throughput of physical routers.
T^V	Desired throughput of virtual routers.
$\mathcal{T}^V$	Minimum throughput of virtual routers.
F^P	Flow table capacity of physical routers.
F^V	Flow table requirement of virtual routers.
M^P	Meter table capacity of physical routers.
M^V	Meter table requirement of virtual routers.
S^P	Location of physical routers.
S^V	Location requirement of virtual routers.
L^P	All physical links.
L^V	All virtual links.
E^P	Links between physical routers.
E^V	Links between virtual routers.
$\mathcal{E}^P$	Links between physical routers and other elements.
$\mathcal{E}^V$	Links between virtual routers and other elements.
B^P	Bandwidth of physical links.
B^V	Desired bandwidth of virtual links.
$\mathcal{B}^V$	Minimum bandwidth of virtual links.
D^P	Delay of physical links.
D^V	Desired delay of virtual links.
$\mathcal{D}^V$	Maximum delay of virtual links.
C^P	Physical machines.
C^V	Virtual controllers.
H	Physical hypervisors.
F^H	Flow table capacity of physical hypervisors.
$\mathcal{C}^P$	CPU capacity of physical machines.
$\mathcal{C}^V$	CPU capacity of virtual controllers.
$\mathcal{M}^P$	Memory capacity of physical machines.
$\mathcal{M}^V$	Memory capacity of virtual controllers.
K	Auxiliary input, set to 1 if a physical device is a hypervisor.

with CPU and memory requirements, represented as $\mathcal{C}^V_{r,j}$ and $\mathcal{M}^V_{r,j}$, respectively. Likewise, the residual CPU and memory capacity of each physical machine is represented as $\mathcal{C}^P_i$ and $\mathcal{M}^P_i$. Physical hypervisors, in turn, are stored in set H. For the sake of simplicity, the capacity of a physical hypervisor is expressed as the number of flow rules it is able to manage at any one time. It is stored in set F^H_i.

Decision variables. The decision variables indicate the placement of all virtual elements on the substrate.

- $A^R_{i,r,j} \in \{0,1\}$ – Router allocation, indicates whether virtual router j from VN r is embedded on physical router i.
- $A^C_{i,r,j} \in \{0,1\}$ – Controller allocation, indicates whether controller j from VN r is embedded on physical machine i.
- $A^L_{i,j,r,k,l} \in \{0,1\}$ – Link allocation, indicates whether virtual link (k,l) from VN r is embedded on physical link (i,j).

Auxiliary variables. The auxiliary variables define the extent to which a flexible capacity constraint is fulfilled. As an example, if the throughput capacity of router j from VN r was only 80% fulfilled, $X^T_{r,j}$ is set to 0.8. In the case of link delay, as it varies between a desired value and a *maximum* (rather than a *minimum*), $X^D_{r,k,l}$ will be greater than 1.0 if this requirement is partially unfilfilled.

- $X^T_{r,j} \in \mathbb{R}$ – Defines the throughput capacity fulfillment of router j in VN r.
- $X^B_{r,k,l} \in \mathbb{R}$ – Defines the bandwidth capacity fulfillment of link (k,l) in VN r.
- $X^D_{r,k,l} \in \mathbb{R}$ – Defines the delay fulfillment of link (k,l) in VN r.

Objective function. Our objective is to minimize the cost of embedding each VN while also maximizing the fulfillment of flexible VN requirements. The latter is achieved by applying penalties to the calculation of each partially fulfilled requirement. These penalties are proportional to the degree of unfulfillment of each requirement (*e.g.*, a penalty of 20% if the requirement is 20% unfilfilled). Additionally, a weight (ω) is applied to each factor in the objective function – delay (α), throughput (β), bandwidth (γ), and flow table capacities (δ) – , allowing the prioritization of specific factors.

Objective:

$$Minimize\, \omega^D\alpha + \omega^T\beta + \omega^B\gamma + \omega^F\delta$$

Where:

$$\alpha = \sum_{(i,j)\in L^P, r\in N^V, (k,l)\in L^V} A^L_{i,j,r,k,l} D^V_{r,k,l} X^D_{r,k,l}$$

$$\beta = \sum_{(i,j)\in L^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(T^V_{r,k}, T^V_{r,l})} A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})}{2} + \sum_{i\in R^P, r\in N^V, k\in R^V} T^V_{r,k} A^R_{i,r,k}(2 - X^T_{r,k})$$

$$\gamma = \sum_{(i,j)\in L^P, r\in N^V, (k,l)\in L^V} A^L_{i,j,r,k,l} B^V_{r,k,l}(2 - X^B_{r,k,l})$$

$$\delta = \sum_{(i,j)\in L^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(F^V_{r,k}, F^V_{r,l})} A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})}{2} + \sum_{i\in R^P, r\in N^V, k\in R^V} F^V_{r,k} A^R_{i,r,k}$$

Factors α, β, and γ refer to the flexible constraints in our model. In the case of delay (α), the penalty for partial unfulfillment of this requirement is calculated by multiplying the desired delay D^V by X^D (which, for example, will be equal to 1.2 if the delay of a given link is 20% higher than the desirable level). For throughput (β) and bandwidth (γ), this penalty is calculated by $(2 - X^T)$ and $(2 - X^B)$, respectively. As such, if one of these requirements is 20% unfulfilled (leading to an X^T or X^B of 0.8), this subtraction sets the penalty to 1.2. Last, δ is the sum of flow table usage of physical devices. The calculation for both β and γ include throughput and flow table occupation of auxiliary routers through which virtual links traverse (equal to the minimum value between the requirements of the end points of such links). Inputs L^P and L^V represent sets including all physical and virtual links, respectively (*i.e.*, $L^P = E^P \cup \mathcal{E}^P$ and $L^V = E^V \cup \mathcal{E}^V$).

Constraints. Next, we formalize and explain the constraint sets of our model. For ease of comprehension, constraints are organized into groups according to the function they serve.

$$\sum_{j \in R^P, r \in N^V, (k,l) \in L^V} \frac{\min_{(X^T_{r,k}T^V_{r,k}, X^T_{r,k}T^V_{r,l})} A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})}{2} + \sum_{r \in N^V, k \in R^V} X^T_{r,k} T^V_{r,k} A^R_{i,r,k} \leq T^P_i \quad \forall i \in R^P \tag{C1}$$

$$\sum_{j \in R^P, r \in N^V, (k,l) \in L^V} \frac{\min_{(F^V_{r,k}, F^V_{r,l})} A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})}{2} + \sum_{r \in N^V, k \in R^V} F^V_{r,k} A^R_{i,r,k} \leq F^P_i \quad \forall i \in R^P \tag{C2}$$

$$\sum_{r \in N^V, k \in R^V} M^V_{r,k} A^R_{i,r,k} \leq M^P_i \quad \forall i \in R^P \tag{C3}$$

Constraint C1 ensures the throughput capacity of each physical router will not be exceeded, while constraint C2 performs the same function regarding flow tables of physical routers. Both C1 and C2 take into account resource consumption in "auxiliary arouters" – *i.e.*, physical routers that are not explicitly hosting virtual routers of a certain VN but through which their links traverse. C3, in turn, handles meter table occupation in a similar manner.

$$\sum_{r \in N^V, k \in C^V} \mathcal{C}^V_{r,k} A^C_{i,r,k} \leq \mathcal{C}^P_i \quad \forall i \in C^P \tag{C4}$$

$$\sum_{r \in N^V, k \in C^V} \mathcal{M}^V_{r,k} A^C_{i,r,k} \leq \mathcal{M}^P_i \quad \forall i \in C^P \tag{C5}$$

$$\sum_{r \in N^V, (k,l) \in L^V} A^L_{i,j,r,k,l} F^V_{r,l} \leq F^H_i \quad \forall i \in H, j \in R^P \tag{C6}$$

Constraints C4 and C5 ensure the CPU and memory capacity of VMs hosting

virtual controllers is not exceeded. C6, in turn, deals with physical hypervisor occupation by ensuring the sum of the number of flow rules in virtual routers being managed by a physical hypervisor does not exceed its capacity.

$$\sum_{r \in N^V, (k,l) \in L^V} X^B_{r,k,l} B^V_{r,k,l} A^L_{i,j,r,k,l} \leq B^P_{i,j} \qquad \forall (i,j) \in L^P \quad \text{(C7)}$$

$$\sum_{(i,j) \in L^P} D^P_{i,j} A^L_{i,j,r,k,l} \leq X^D_{r,k,l} D^V_{r,k,l} \qquad \forall r \in N^V, (k,l) \in L^V \quad \text{(C8)}$$

Constraint C7 ensures the bandwidth requirement of all virtual links traversing a physical link does not exceed its capacity. Similarly, Constraint C8 ensures the delay requirement of a virtual link is lower than the total delay of the physical path it traverses.

$$\sum_{i \in C^P} A^C_{i,r,j} = 1 \qquad \forall r \in N^V, j \in C^V \quad \text{(C9)}$$

$$\sum_{i \in R^P} A^R_{i,r,j} = 1 \qquad \forall r \in N^V, j \in R^V \quad \text{(C10)}$$

$$\sum_{j \in R^V} A^R_{i,r,j} \leq 1 \qquad \forall i \in R^P, r \in N^V \quad \text{(C11)}$$

$$j A^R_{i,r,k} = l A^R_{i,r,k} \qquad \forall (i,j) \in S^P, r \in N^V, (k,l) \in S^V \quad \text{(C12)}$$

Constraints C9 and C10 ensure all virtual controllers and routers are mapped to physical controllers and routers, respectively. Additionally, C11 prevents multiple virtual routers from the same VN from being mapped to the same physical router. The purpose of this constraint is to ensure costs associated with virtual links between routers are properly accounted for. Constraint C12, in turn, guarantees all virtual routers with a location requirement are mapped to physical routers within the requested locations.

$$\sum_{j \in R^P} A^L_{i,j,r,k,l} - \sum_{j \in R^P} A^L_{j,i,r,k,l} = A^R_{i,r,k} - A^R_{i,r,l}$$
$$\forall r \in N^V, (k,l) \in L^V, i \in R^P \quad \text{(C13)}$$

$$\sum_{(i,j) \in \mathcal{E}^P} (K_i + K_j) A^L_{i,j,r,k,l} > 0 \qquad \forall r \in N^V, (k,l) \in \mathcal{E}^V \quad \text{(C14)}$$

Constraint C13 ensures any virtual link (a, b) is correctly mapped to a path between the physical routers hosting a and b. Moreover, Constraint C14 forces links between virtual controllers and routers to pass through a physical hypervisor. In this case, K is an auxiliary input which is set to 1 if a physical node is a hypervisor (and 0 otherwise).

$$\frac{\mathcal{T}^V_{r,k}}{T^V_{r,k}} \leq X^T_{r,k} \leq 1.0 \qquad \forall r \in N^V, k \in R^V \quad \text{(C15)}$$

$$\frac{\mathcal{B}^V_{r,k,l}}{B^V_{r,k,l}} \leq X^B_{r,k,l} \leq 1.0 \qquad \forall r \in N^V, (k,l) \in L^V \quad \text{(C16)}$$

$$1.0 \leq X^D_{r,k,l} \leq \frac{\mathcal{D}^V_{r,k,l}}{D^V_{r,k,l}} \qquad \forall r \in N^V, (k,l) \in L^V \quad \text{(C17)}$$

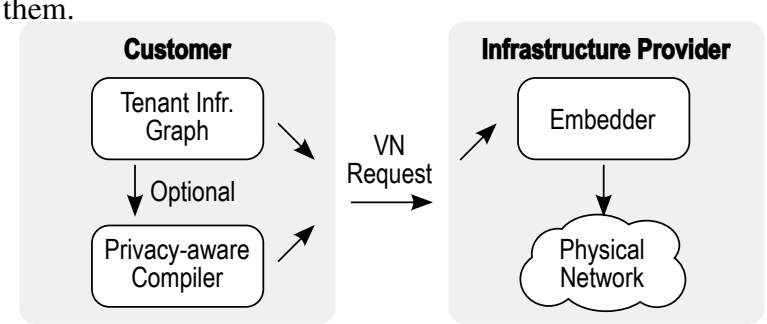
Constraints C15, C16, and C17 define the range of auxiliary variables X^T, X^B, and X^D, respectively. These variables determine the degree of flexibilization of throughput, bandwidth, and delay constraints for each virtual router or link. In the case of throughput and bandwidth, the lower limit of this range is defined as the ratio between the minimum acceptable capacity and the desired capacity (which is a value between 0 and 1.0) , while the upper limit is set to 1.0 (representing the desirable capacity). In the case of delay, the lower limit is set to 1.0 while the upper limit is a ratio between the maximum tolerated delay and the desirable delay (which is a value greater than 1.0).

5.4 Overarching Framework

The main elements of the overarching framework of our architecture were initially introduced in Chapter 4. However, as previously explained, these elements have been reformulated in order to properly handle the full scope of our proposed solution. Next, we explain our overarching framework in further detail and, afterwards, each of its elements – highlighting the changes that have been made since the previous phase of development.

Figure 5.2 depicts the main elements of our complete framework, as well as the information flow between them. To request a VN, the customer first creates a Tenant Infrastructure Graph (TIG). This graph is similar to the standard representation of a VN request – however, it is augmented with a wide number of additional elements that allow for a more precise calculation of SDN-related operational requirements. The TIG may be sent directly to the InP or preprocessed by the Privacy-aware Compiler (PAC). The purpose of the PAC is to leverage the augmented information present in the TIG to precalculate operational constraints at the customer's end and remove this augmented information (as the customer may deem it sensitive) before submitting the VN request to the InP. On the InP's end, the VN request is received and handled by the SDN/OpenFlow-aware Embedder. The request is then embedded on top of the physical network, as long as there are enough resources to fulfill the minimal requirements set by the customer.

Figure 5.2: Overview of our architecture, depicting its main elements and the information flow between them.

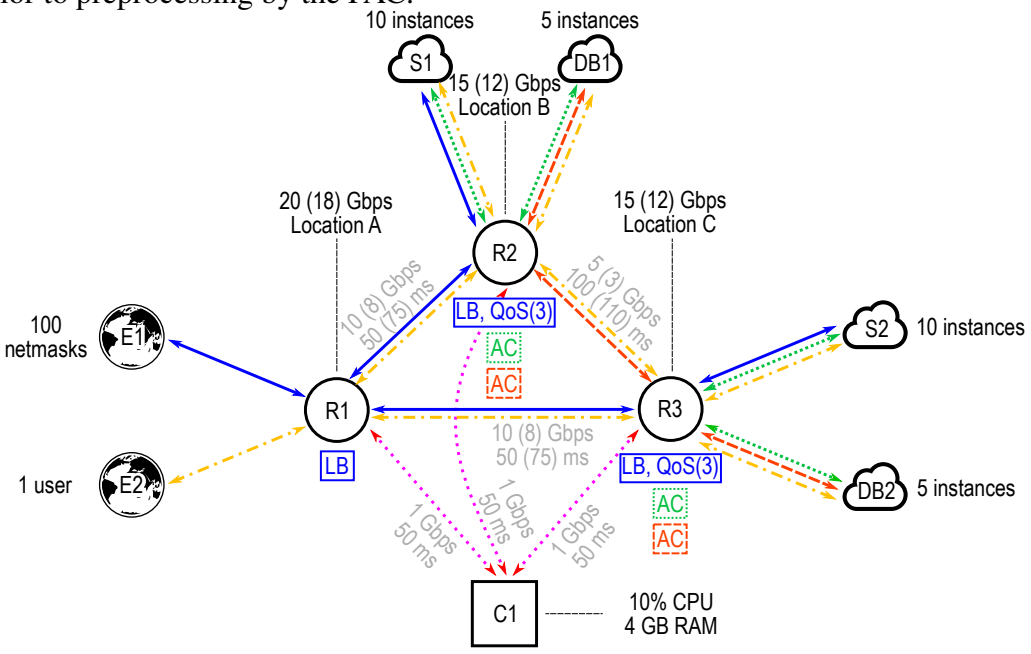


Tenant Infrastructure Graph. The TIG is an enhanced version of what is traditionally considered a VN request in VNE algorithms. A "traditional" VN request is a graph composed of routers and links (and potentially VMs) and attributes associated with each of these entities (*e.g.*, link bandwidth and delay). The TIG expands upon this model, encompassing a number of elements that help to compute SDN-related operational requirements with a higher level of precision. It includes VN requirements traditionally considered in state of the art VNE approaches – *i.e.*, router throughput and

location constraints, as well as link bandwidth and delay. Moreover, it takes into account SDN/OpenFlow controllers as part of the topology, as well as requirements associated with these entities as well – *i.e.*, CPU and memory requirements. Most importantly, it enables customers to represent external end hosts, traffic flows among these hosts, and network functions that routers need to perform on each traffic flow.

An example of a Tenant Infrastructure Graph is presented in Figure 5.3. The example in this figure depicts a network that handles two groups of user-facing servers (S1 and S2) and internal database servers (DB1 and DB2). The solid blue lines represent the traffic flow of external users (E1, adding up to 100 netmasks) accessing a user-facing server. Router R1 performs load balancing (LB) in order to redirect traffic from specific clients to a server in either S1 or S2, while routers R2 and R3 additionally differentiate traffic through different QoS levels. Similarly, the other flows depicted in this example represent other traffic patterns within the VN, each with their own network functions that need to be handled by specific routers. The dotted green lines represent communication between groups of user-facing servers and their respective database servers. The dashed red lines represent communication between different groups of database servers for the purpose of synchronization. Last, the dashed and dotted yellow line grants a single entity (a system administrator) access to all servers.

Figure 5.3: Example of a Tenant Infrastructure Graph, depicting all elements included in it prior to preprocessing by the PAC.



As previously explained, we allow the flexibilization of certain requirements – represented as the numbers in parenthesis in Figure 5.3. As an example, the link between

R1 and R2 has a desired bandwidth capacity of 10 Gbps and a minimal capacity of 8 Gbps, as well as a desired delay of 50 ms and a maximum tolerated delay of 75 ms.

In contrast to the description presented in Chapter 4, a number of new elements have been added to the TIG. First, we now consider an explicit representation of SDN/OpenFlow controllers, the communication channels between controllers and routers within each VN, and CPU and memory requirements associated with them. This enables our model to properly take into account controller-related requirements and leverage this knowledge to efficiently embed these elements on the physical network. Second, the TIG now includes link delay as an additional constraint. Last, it now allows customers to represent router throughput, link bandwidth, and delay as flexible constraints.

We now present an example of the formal representation of a TIG. For clarity, this representation is divided into Algorithms 1 and 2 – although, in practice, all elements within a TIG are represented as a single file. This example is based on the same TIG represented in Figure 5.3. Algorithm 1 encompasses elements related to network topology and services – *e.g.*, routers, links, and controllers. Each entity is defined as a (unique) name (*e.g.*, R1, R2, and R3) followed by a declaration of its type (router, compute[1], controller, link, or flow) and, finally, by a list of attributes associated with it. As an example, the attributes associated with a router are *throughput*, *throughput_min*, and *location* (where *throughput* represents the desired capacity and *throughput_min* represents the minimum acceptable capacity).

Algorithm 2 demonstrates the representation of network flows. Compared to other elements, flows are represented in a slightly more complex manner in order to fully capture the potential intricacies they may exhibit. However, they still follow the same standard syntax. The path of a flow is composed of any number of links (which must have been previously defined). Groups of links within the path of a flow can be represented in a nested manner, in order to represent branching paths (as is the case of the blue flow). Moreover, each flow is associated with a number of functions – and each function, in turn, associated with a specific virtual router. This representation is necessary in order to properly determine traffic patterns between external entities and how network functions will be applied to each flow when being processed by specific routers.

Privacy-Aware Compiler. Before being handled by the Embedder, the TIG is preprocessed in order to compute SDN/OpenFlow-related requirements. This can be performed by the Privacy-aware Compiler (PAC) at the customer's end or directly at the InP's

[1]For the sake of simplicity, any entities external to the VN (*e.g.*, end users/netmasks and servers) are defined as compute instances.

Algorithm 1 Formal representation of a TIG – network and service representation

```
/* routers */
   R1={type=router,throughput=20,throughput_min=18,
      location=1};
   R2={type=router,throughput=15,throughput_min=12,
      location=2};
   R3={type=router,throughput=15,throughput_min=12,
      location=3};
/* compute instances */
   E1={type=compute,instances=100};
   E2={type=compute,instances=1};
   S1={type=compute,instances=10};
   DB1={type=compute,instances=5};
   S2={type=compute,instances=10};
   DB2={type=compute,instances=5};
/* controllers */
   C1={type=controller,cpu=10,memory=4};
/* internal links between routers */
   R1_R2={type=link,path={R1,R2},bandwidth=10,
      bandwidth_min=8,delay=50,delay_max=75};
   R1_R3={type=link,path={R1,R3},bandwidth=10,
      bandwidth_min=8,delay=50,delay_max=75};
   R2_R3={type=link,path={R2,R3},bandwidth=5,
      bandwidth_min=3,delay=100,delay_max=110};
/* internal links between controller and routers */
   C1_R1={type=link,path={C1,R1},bandwidth=1,delay=50};
   C1_R2={type=link,path={C1,R2},bandwidth=1,delay=50};
   C1_R3={type=link,path={C1,R3},bandwidth=1,delay=50};
/* external links */
   E1_R1={type=link,path={E1,R1}};
   E2_R1={type=link,path={E2,R1}};
   S1_R2={type=link,path={S1,R2}};
   DB1_R2={type=link,path={DB1,R2}};
   S2_R3={type=link,path={S2,R3}};
   DB2_R3={type=link,path={DB2,R3}};
```

end. The PAC leverages augmented high-level network information in the TIG in order to calculate these requirements in a precise manner. At the same time, it removes this augmented information from its output, generating a “sanitized” VN request.

As the TIG is considerably dense in terms of the amount of information it carries, the PAC must be equipped to process all the different pieces of information within a TIG. This includes newly added elements such as controllers and their respective communication channels, capacity requirements such as controller CPU and memory as well as link delay, and the representation of capacity constraints as flexible ranges. Additionally, as the Embedder now takes into account meter table occupation, the VN request generated by the PAC also includes the number of meter table entries required by each router in order to properly enforce bandwidth constraints.

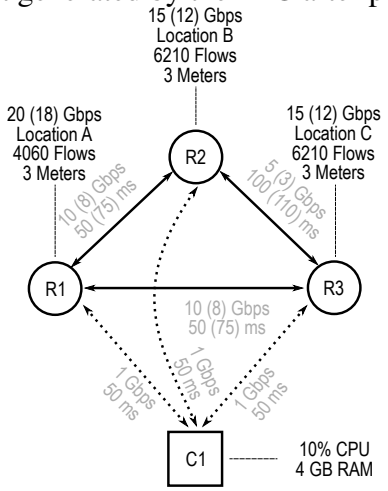
Figure 5.4 depicts an example of how this process is carried out on the same TIG

Algorithm 2 Formal representation of a TIG – flow representation

```
1: /* flows */
2:    yellow={type=flow,path={E2_R1,R1_R2,S1_R2,
3:       DB1_R2,R1_R3,S2_R3,DB2_R3,R2_R3}};
4:    blue={type=flow,path={{E1_R1},{{R1_R2,S1_R2},
5:       {R1_R3,S2_R3}}},functions={
6:       R1_blue_LB={entity=R1,purpose=load_balancing},
7:       R2_blue_LB={entity=R2,purpose=load_balancing},
8:       R2_blue_QoS={entity=R2,purpose=qos,levels=3},
9:       R3_blue_LB={entity=R3,purpose=load_balancing},
10:      R3_blue_QoS={entity=R3,purpose=qos,levels=3}
11:      }};
12:   green={type=flow,path={S1_R2,DB1_R2},functions={
13:      R2_green_AC={entity=R2,purpose=access_control},
14:      R3_green_AC={entity=R3,purpose=access_control}
15:      }};
16:   red={type=flow,path={DB1_R2,DB2_R3,R2_R3},
17:      functions={
18:      R2_red_AC={entity=R2,purpose=access_control},
19:      R3_red_AC={entity=R3,purpose=access_control}
20:      }};
21:   magenta={type=flow,path={C1_R1,C1_R2,C1_R3}};
```

shown previously in Figure 5.3. The input TIG includes information regarding external network instances, the traffic flows among them, and functions performed by each router on each flow. This information is converted into simple, numerical capacity requirements while the source information is removed from the generated VN request. As a result, the output of the PAC is similar to a traditional VN request, with all the information within a TIG being "distilled" and converted into a format that is simple for the Embedder to parse.

Figure 5.4: VN request generated by the PAC after processing a given TIG.



With regards to the computation of flow table requirements, depending on the functions to be performed on a given flow, the initial number of flow rules required by

a router that carries this flow may be either based on the number of end points interconnected by this flow or the total number of source-destination pairs communicating through it. Moreover, this initial number may be further multiplied depending on the functions – or a combination of functions – performed on it.

As an example of the calculation performed to determine flow table requirements performed by the PAC, consider router R2 and its requirements shown in Figure 5.4. As previously shown (in Figure 5.3 and Algorithms 1 and 2), four flows – yellow, red, green, and blue – traverse this router. The flow represented in yellow interconnects an external entity to all (30) compute instances within the network, adding up to a total of **60** source-destination pairs. As no additional functions are performed on this flow, this is the total number of flow rules it requires. Similarly, the red flow interconnects 5 compute instances within DB1 and 5 compute instances within DB2, adding up to **50** source-destination pairs, while the green flow connects 10 instances in S1 to 5 instances in DB1, comprising a total of **100** source-destination pairs. While both the red and green flows require access control, this does not directly translate into an increase in flow table requirements, as the router only needs to keep track of packet sources and destinations in order to perform this function. Last, the blue flow connects 100 external netmasks to 10 compute instances within S1. This leads to an initial number of 2,000 source-destination pairs. Additionally, this flow requires both load balancing and QoS with 3 distinct levels, the latter of which leads to a 3x multiplication in the initial computed number, adding up to **6,000** flow rules. The total amount of flow entries required by this router is, thus, the combination of the number of entries required by each flow that traverses it, adding up to 6,210 entries (60 + 50 + 100 + 6,000).

In addition to flow table requirements, the PAC also computes meter table requirements. Meter table entries are employed by the InP at the substrate level to ensure virtual routers do not exceed the maximum bandwidth capacity assigned to them. In this case, as bandwidth capacity is handled on a per-link basis, the necessary number of meter table entries is equal to the number of links traversing a router. Analogously, the PAC may be extended to handle other cases in which certain VN features may introduce operational requirements at the physical level that the customer is not – and should not be – aware of.

5.5 Evaluation

We now discuss the experiments used to evaluate our proposed framework and the results obtained through this evaluation. Experiments were performed on a machine with an Intel Xeon E5-2420 CPU, 32 GB of RAM and Operating System Ubuntu GNU/Linux 16.04. The SDN/OpenFlow-aware Embedder was implemented and run using the IBM ILOG CPLEX Interactive Optimizer 12.4.

5.5.1 Workloads

For this evaluation, we algorithmically generate a series of VN requests according to a series of parameters. The algorithm is run for a total of 250 rounds, generating a new VN request per round. If a request is accepted, the VN remains embedded for 25 rounds and is subsequently removed.

Fixed parameters. In all experiments, physical and virtual topologies are generated through BRITE[2] using the Barabási-Albert model (ALBERT; BARABÁSI, 2000). Physical networks used in this evaluation contain 100 routers, 4 hypervisors, and 8 physical machines for hosting virtual controllers. Physical routers have a throughput capacity of 150 Gbps and are uniformly distributed among 16 geographic locations. The flow table of each device is capable of storing up to 16,000 rules, while the meter table has a capacity of 4,000 rules. The total CPU and memory capacity of each physical machine are set to 100% and 64 GB, respectively. Physical hypervisors are capable or handling up to 32,000 flow rules each. Physical links have a bandwidth capacity of 30 Gbps while the delay of each link is uniformly distributed between 20 and 40 ms.

Each VN request contains 1 controller and 5 routers. Within each VN, two "edge" routers contain randomly generated location requirements. The capacity requirements of virtual routers, links, and controllers differ depending on the experiment, and will be explained next.

Variable parameters. We performed experiments with two variations regarding VN requirements – one set of experiments was carried out with lower VN requirements, while another set had higher requirements. In "lower cost" (LC) experiments, router throughput, flow table entries, link bandwidth, and controller CPU and memory requirements were set to 25% of the total capacity of physical devices, while link delay require-

[2]BRITE: Boston university Representative Internet Topology gEnerator – https://www.cs.bu.edu/brite/

ments were set to 100 ms. "Higher cost" (HC), in turn, refers to experiments in which the aforementioned capacity requirements were set to 50% of physical device capacities, while link delay constraints were set to 50 ms. Additionally, five variations of each of these experiments were carried out by adjusting the amount of flexibilization allowed in router throughput, link bandwidth, and delay requirements of each VN. Different experiments were performed in which the tolerance for sub-optimal requirements was set to 0%[3], 5%, 10%, 15%, and 20% of the optimal required capacity. Tables 5.2 and 5.3 summarize all the aforementioned variable parameters used in our experiments.

Table 5.2: Variable VN requirements in "lower cost" experiments.

Parameter	Desired	Minimal (per experiment)				
		0%	5%	10%	15%	20%
Router Throughput	37.5 Gbps	37.5	35.62	33.75	31.87	30
Link Bandwidth	7.5 Gbps	7.5	7.12	6.75	6.37	6
Link Delay	100 ms	100	105	110	115	120
Flow Table Entries	4,000	N/A				
Controller CPU	25%	N/A				
Controller Memory	16 GB	N/A				

Table 5.3: Variable VN requirements in "higher cost" experiments.

Parameter	Desired	Minimal (per experiment)				
		0%	5%	10%	15%	20%
Router Throughput	75 Gbps	75	71.25	67.5	63.75	60
Link Bandwidth	15 Gbps	15	14.25	13.5	12.75	12
Link Delay	50 ms	50	52.5	55	57.5	60
Flow Table Entries	8,000	N/A				
Controller CPU	50%	N/A				
Controller Memory	32 GB	N/A				

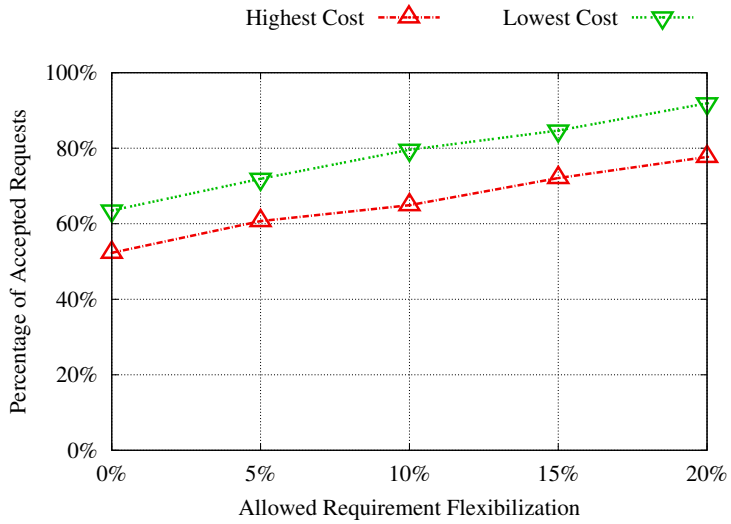
5.5.2 Results

First, we analyze the acceptance rate observed in each experiment. This data is represented in Figure 5.5. In higher cost experiments with no constraint flexibilization, the achieved acceptance rate was of 52.3%. In lower cost experiments, in turn, the acceptance rate was 63.5%. These two experiments in particular can be considered "baseline"

[3] *I.e.*, both the optimal and minimum/maximum tolerated capacity were set to the same value, not allowing any flexibilization.

cases when compared to their variations – in which different degrees of constraint flexibilization are considered. Increasing amounts of flexibilization led to an approximately linear gain in both cases, with acceptance rates reaching 77.7% and 91.9% for lower and higher cost variants, respectively. Compared to baseline experiments with no flexibilization, these rates represent growths of 25.4% and 28.4% in VN acceptance, respectively – exceeding the degree of flexibilization of 20% for a subset of VN constraints. These results highlight the potential advantages of employing flexible constraints in order to maximize the number of VNs hosted on a given physical infrastructure while maintaining a "hard" limit on minimum requirements set by customers – which should not be violated. Moreover, as expected, the difference in terms of capacity constraints between higher and lower cost scenarios is reflected in this graph. The acceptance rate observed in higher cost scenarios remains between 11.2% and 14.2% lower than that of their lower cost counterparts – considering the same amount of flexibilization.

Figure 5.5: Acceptance rate of VN requests in each evaluation scenario.



Next, the graphs in Figure 5.6 depict the level of flexibilization achieved in each experiment. While throughput, bandwidth, and delay requirements may allow flexibilization up to a certain limit, this metric expresses the extent to which this potential flexibilization was actually used by the Embedder. In these graphs, the solid black line represents the theoretical maximum that could be achieved given the level of flexibilization allowed in each scenario. In experiments where minimal requirements are limited to 5% below or above the desired ones, the achieved flexibilization of all requirements was similar, remaining between 4.2 and 4.5%. At 20%, the achieved flexibilization of throughput and bandwidth requirements remained between 16.9 and 17.5%, while that of delay require-

ments was of 13.7 in higher capacity experiments and 14.4% in lower capacity experiments. As evidenced by these results, the achieved levels of flexibilization remain close to the theoretical maximum in each experiment. However, the values observed for delay requirements exhibit a slightly lower growth when compared to throughput and bandwidth. This is likely due to topological features of the physical network making it impossible to establish longer paths with a delay that remains within the tolerated limit (as the delay of a path is defined as the sum of all links that form this path). Moreover, a correlation can be seen between the achieved level of flexibilization and the acceptance rate of each experiment. As more VN requests are accepted over time and physical network resources become increasingly scarce, higher degrees of flexibilization are effectively used by the Embedder in order to accommodate incoming requests – resulting in higher acceptance rates.

Figure 5.6: Level of constraint flexibilization effectively achieved in each experiment.

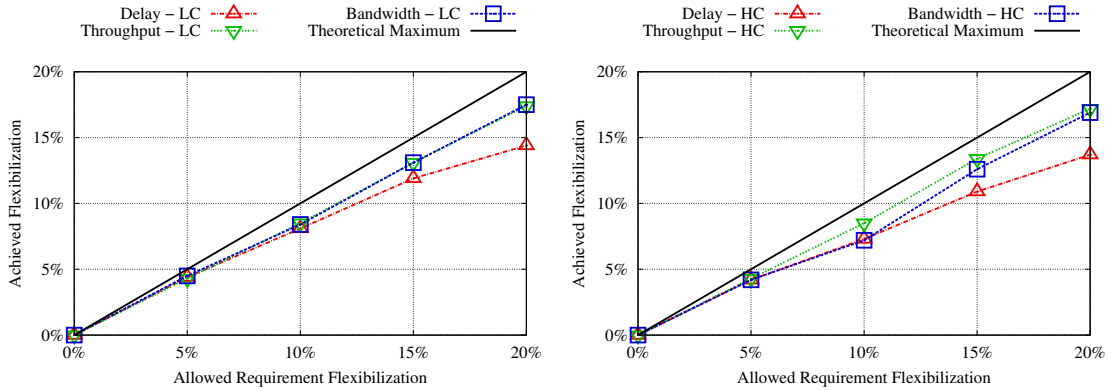


Table 5.4: Time to reach the optimal solution in each scenario (in seconds).

Allowed Requirement Flexibilization	Solution Time	
	LC	HC
0%	8.8	9.1
5%	37.3	34.2
10%	35.2	34.3
15%	36.7	36.7
20%	35.9	35.2

Last, we analyze the time needed to reach the optimal solution in each experiment. This data is summarized in Table 5.4. In experiments where no constraint flexibilization was allowed (represented as 0%), the average solution time was of 8.8 seconds in experiments with lower capacity requirements. In experiments with higher requirements, in turn, the average observed time was of 9.1 seconds. In remaining experiments, in which

constraint flexibilization is allowed in varying degrees, solution times remained between 34.2 and 37.3 seconds. As evidenced by these observed values, solution times suffer an increase when the Embedder has to take into account flexible resource requirements. However, in spite of this increase, solution times remained shorter than 40 seconds in all experiments – well within the acceptable limits for this type of environment. Moreover, greater amounts of flexibilization did not result in further increases to solution times. The observed values remain highly similar in all experiments that allow resource flexibilization, regardless of the extent to which this feature is allowed.

In summary, these experiments demonstrate the feasibility of simultaneously taking into account a large number of operational constraints during the VN embedding process. Potential negative impacts on acceptance rates are mitigated by the flexibilization of resource demands – which was utilized nearly to its full extent in order to maximize request acceptance. Moreover, the impact caused by the employment of this flexibilization on the time needed to compute VN mappings was limited to the order of seconds. The impact of the evaluation results obtained throughout the development of this work on our hypothesis and research questions is further discussed in Chapter 6.

6 FINAL CONSIDERATIONS

Network virtualization and Software-Defined Networking have been playing increasingly prominent roles in today's networks. The provisioning of network virtualization as a service is expected to experience continuous growth, reaching a majority of production networks within the next three to five years. However, although a substantial body of work exists in the area of virtual network embedding, existing approaches do not take into account relevant operational constraints related to the instantiation of VNs on different embedding platforms. At the same time, demanding information regarding such operational constraints from customers may be unrealistic as they may be either not aware of how their VN affects the InP's substrate at the physical level or unwilling to share detailed information about the inner working of their VNs.

Based on this reasoning, we proposed an abstraction model for expressing requirements related to internal VN policies and traffic patterns. This model – the Tenant Infrastructure Graph – is built in a way that is familiar to customers in a network virtualization environment. Moreover, it may be preprocessed on the customer's end by a Privacy-aware Compiler in order to derive information that is valuable to the InP and, at the same time, remove sensitive data that the customer may not be willing to disclose. The output of this compiler is then forwarded to the InP, which can employ the SDN/OpenFlow-aware Embedder to ensure embedded VNs do not break any crucial operational constraints of its network virtualization platform.

We also proposed a strategy for mitigating the solution space reduction that results from taking into account a large set of operational constraints simultaneously. As doing so severely reduces the number of feasible mappings the VNE model can produce, this could likely lead to a significant increase in rejection rates of VN requests. Our framework enables customers to represent their needs – in terms of capacity constraints – as a range which extends from their minimum acceptable requirements to the maximum desired capacity. Our strategy leverages this added flexibility to minimize the impact of increased restrictions imposed by SDN/OpenFlow-related operational constraints.

6.1 Conclusions

Through a comprehensive evaluation, we first demonstrated that taking flow-related operational constraints into account is of paramount importance to maintain a desired

level of quality of service. Neglecting such constraints may render the environment unable to cope with a substantial number of flow rules that are crucial to ensure proper VN behavior. As physical devices become unable to store all necessary flow rules internally, they need to frequently contact the controller in order to route incoming packets, which, in turn, may lead to significant performance degradation for VNs hosted on such devices.

In our experiments, assuming the flow table space reserved for requests with unknown requirements was sufficient, the proposed approach was able to eliminate controller intervention due to flow table saturation. Additionally, assuming the actual requirements of such requests exceeded the allocated space by a factor of 2, the number of exceeding flow rules was still reduced by 40.8% on average (compared to a traditional VNE approach). Moreover, the reduction of acceptance rates due to the added constraints was limited to, on average, 9.6%. Our proposed approach enables InPs to accurately assess operational constraints and correctly embed incoming requests without violating these constraints. Further, by adjusting the ratio of flow table space dedicated to different types of incoming requests, the InP may choose to which degree requests that include all the necessary information are favored in detriment of requests that do not (and that, therefore, rely on estimation of necessary resources in order to be embedded).

Subsequently, we demonstrated the feasibility of taking into account a rich set of SDN/OpenFlow-related operational constraints at the same time during the VN embedding process. In a second set of experiments, we demonstrated that the proposed VNE approach is able to optimally embed incoming requests in a time frame limited to a few seconds. While enabling requirement flexibilization has an impact on the time needed to compute the optimal mapping of each request, the total time to embed incoming VNs was still limited to the order of seconds. Namely, solution times were lower than 10 seconds in scenarios with no constraint flexibilization and lower than 40 seconds in scenarios with different degrees of tolerance. Additionally, the obtained results evidence that the level of flexibilization achieved in each experiment remains near the theoretical maximum for that scenario and promotes significant increases in acceptance rates. By allowing up to 20% flexibilization of some constraints, we were able to observe growths of up to 28.4% in terms of VN acceptance.

As a result of the work carried out throughout the course of this Ph.D. research, we were able to confirm our previously formulated hypothesis – "*In order to provide network predictability, efficient resource usage, and quality of service, it is necessary to coordinate virtual network embedding and operational constraints found in SDN-based networks.*".

Moreover, we were able to successfully answer the research questions we initially posed. More specifically, we quantified the negative impact of neglecting operational constraints in terms of predictability and quality of service (Research Question 1) as well as the gains achieved through VNE/SDN coordination (Research Question 2). Further, we demonstrated that the cost to be paid for this coordination – in terms of solution space reduction, computing power, and timeliness – are acceptable using the proposed strategy (Research Question 3). Next, we list the main outcomes of this thesis in terms of peer-reviewed publications.

Main publications:

- **Virtual network security: threats, countermeasures, and challenges**
  Leonardo Richter Bays, Rodrigo Ruas Oliveira, Marinho Barcellos, Luciano Paschoal Gaspary, Edmundo Roberto Mauro Madeira.
  Journal of Internet Services and Applications, 2015.
  (Included as Appendix A of this document)
- **Virtual Network Embedding in Software-Defined Networks**
  Leonardo Richter Bays, Luciano Paschoal Gaspary, Reaz Ahmed, Raouf Boutaba.
  IEEE/IFIP Network Operations and Management Symposium, 2016.
  (Included as Appendix B of this document)

Related publications:

- **A toolset for efficient privacy-oriented virtual network embedding and its instantiation on SDN/OpenFlow-based substrates**
  Leonardo Richter Bays, Rodrigo Ruas Oliveira, Luciana Buriol, Marinho Barcellos, Luciano Paschoal Gaspary.
  Computer Communications, 2016.
- **How physical network topologies affect virtual network embedding quality: A characterization study based on ISP and datacenter networks**
  Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Buriol, Marinho Barcellos, Luciano Paschoal Gaspary.
  Journal of Network and Computer Applications, 2016.

6.2 Future Work

Based on the experience acquired throughout the Ph.D. research presented in this thesis, we envision two main possibilities for future work, which are discussed next.

Routing orchestration. In a multi-tenant virtualization environment such as the one we consider in this work, customers may make use of routing policies that conflict with those of other VNs. This can cause unnecessary strain on the substrate, hindering the performance of VNs hosted on top of it or preventing the InP from hosting a higher number of networks. Moreover, as each VN is controlled by a different entity, they cannot directly coordinate with each other in order to choose strategies that may favor them mutually in the long term. For this reason, a strategy for analyzing the behavior of VNs, suggesting improved strategies, and offering benefits to VNs that adopt these strategies could potentially benefit the environment as a whole.

Iterative requirement negotiation. Currently, we consider customer demands as a range that stretches from minimum acceptable to maximum desirable resources. The VN Embedder takes this range into account in order to flexibilize demands during the embedding process. A further refinement could be made to this process, enabling iterative, back-and-forth negotiation of demands between customers and InPs prior to the actual embedding of the requested VN. Through this strategy, if the Embedder is unable to fully accommodate a certain request on the substrate, it first analyzes what adjustments (if any) would allow it to be embedded. A modified request – with lower requirements – is then forwarded to the customer, who may accept it or not. If the customer accepts the proposed changes, the modified VN is then embedded on the substrate. Otherwise, both entities may attempt to iteratively reach an agreement, with the customer proposing his/her own modifications. If an agreement cannot be reached within a reasonable amount of time, the VN request may be eventually discarded.

Dynamic reoptimization of allocated resources. Presently, the VN Embedder takes into account the amount of resources available in the infrastructure at the moment in which a request is received in order to determine the level of flexibilization that will be necessary in order to embed the requested VN. However, the amount of available resources may increase at a later time, *e.g.*, if a previously embedded VN is removed from the substrate. Therefore, this process could potentially be refined by reconsidering the amount of resources allocated to remaining VNs whenever further resources become available. This process of reoptimization could be leveraged to fulfill VN requests to a

level that increasingly approaches the maximum desirable capacity of each request. On the other hand, this process would conversely result in a lower amount of resources being made available for future incoming requests. Therefore, a more comprehensive study on both the positive and negative impacts of such a strategy would need to be carried out in order to determine whether its potential gains outweigh any inherent drawbacks.

REFERENCES

AL-SHABIBI, A. et al. Openvirtex: make your virtual sdns programmable. In: ACM WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, 2014... **Proceedings.** New York, NY, USA: ACM, 2014. p. 25–30. Disponível em: <http://doi.acm.org/10.1145/2620728.2620741>.

ALBERT, R.; BARABÁSI, A.-L. Topology of evolving networks: Local events and universality. **Physical Review Letters**, American Physical Society, v. 85, p. 5234–5237, Dec 2000.

ALKMIM, G. P.; BATISTA, D. M.; FONSECA, N. L. S. Mapping virtual networks onto substrate networks. **Journal of Internet Services and Applications**, v. 3, n. 4, p. 1–15, 2013. ISSN 1869-0238.

BAYS, L. R. et al. Virtual network security: threats, countermeasures, and challenges. **Journal of Internet Services and Applications**, Springer London, v. 6, n. 1, p. 1, 2015.

BAYS, L. R. et al. A heuristic-based algorithm for privacy-oriented virtual network embedding. In: IEEE NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 2014... **Proceedings.** Krakow, Poland: IEEE, 2014. p. 1–8. Disponível em: <http://doi.org/10.1109/NOMS.2014.6838360>.

BLENK, A. et al. Control plane latency with sdn network hypervisors: The cost of virtualization. **IEEE Transactions on Network and Service Management**, IEEE, v. 13, n. 3, p. 366–380, 2016.

BOZAKOV, Z.; PAPADIMITRIOU, P. Autoslice: automated and scalable slicing for software-defined networks. In: ACM CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES, 2012...**Proceedings.** Nice, France: IEEE, 2012. p. 3–4. Disponível em: <https://dl.acm.org/citation.cfm?id=2413251>.

CALVERT, K. Reflections on network architecture: an active networking perspective. **ACM SIGCOMM Computer Communication Review**, ACM, v. 36, n. 2, p. 27–30, 2006.

CHENG, X. et al. Virtual network embedding through topology-aware node ranking. **SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 41, n. 2, p. 38–47, abr. 2011. ISSN 0146-4833.

CHOWDHURY, M.; RAHMAN, M. R.; BOUTABA, R. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. **IEEE/ACM Transactions on Networking**, IEEE Press, Piscataway, NJ, USA, v. 20, n. 1, p. 206–219, fev. 2012. ISSN 1063-6692.

COMMISSION, E. **Implications of the emerging technologies Software-Defined Networking and Network Function Virtualisation on the future Telecommunications Landscape**. 2017. Disponível em: <http://sdn.wik-consult.com/index.php?id=762>.

CORIN, R. D. et al. Vertigo: network virtualization and beyond. In: EUROPEAN WORKSHOP ON SOFTWARE DEFINED NETWORKING, 2012...**Proceedings.** Darmstadt, Germany: IEEE, 2012. p. 24–29. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6385043/>.

CURTIS, A. R. et al. Devoflow: Scaling flow management for high-performance networks. **SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, v. 41, p. 254–265, 2011. ISSN 0146-4833.

DEMIRCI, M.; AMMAR, M. Design and analysis of techniques for mapping virtual networks to software-defined network substrates. **Computer Communications**, v. 45, p. 1 – 10, 2014. ISSN 0140-3664.

DRUTSKOY, D.; KELLER, E.; REXFORD, J. Scalable network virtualization in software-defined networks. **IEEE Internet Computing**, IEEE, v. 17, n. 2, p. 20–27, 2013.

ESPOSITO, F.; PAOLA, D. D.; MATTA, I. On distributed virtual network embedding with guarantees. **IEEE/ACM Transactions on Networking**, IEEE, v. 24, n. 1, p. 569–582, 2016.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: an intellectual history of programmable networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 44, n. 2, p. 87–98, 2014.

GUAN, X.; CHOI, B.-Y.; SONG, S. Energy efficient virtual network embedding for green data centers using data center topology and future migration. **Computer Communications**, Elsevier, v. 69, p. 50–59, 2015.

HUANG, H. et al. Embedding virtual software-defined networks over distributed hypervisors for vdc formulation. In: INTERNATIONAL CONFERENCE ON COMMUNICATIONS, 2017...**Proceedings.** Paris, France: IEEE, 2017. p. 1–6. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7996721/>.

JARRAY, A.; KARMOUCH, A. Decomposition approaches for virtual network embedding with one-shot node and link mapping. **IEEE/ACM Transactions on Networking (TON)**, IEEE Press, v. 23, n. 3, p. 1012–1025, 2015.

LAN/MAN STANDARDS COMMITTEE. **IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks**. 2006. IEEE Std 802.1Q-2005 (incorpora IEEE Std 802.1Q1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, e IEEE Std 802.1s-2002).

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: ACM SIGCOMM WORKSHOP ON HOT TOPICS IN NETWORKS, 2010...**Proceedings.** Monterey, California: ACM, 2010. p. 19:1–19:6. Disponível em: <http://doi.acm.org/10.1145/1868447.1868466>.

LIAO, J. **SDN System Performance**. 2012. Disponível em: <http://www.pica8.com/pica8-deep-dive/sdn-system-performance/>.

LUIZELLI, M. C. et al. How physical network topologies affect virtual network embedding quality: A characterization study based on isp and datacenter networks. **Journal of Network and Computer Applications**, Elsevier, v. 70, p. 1–16, 2016.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. **SIGCOMM Computer Communication Review**, ACM, New York, USA, v. 38, p. 69–74, March 2008. ISSN 0146-4833.

NASCIMENTO, M. R. et al. Virtual routers as a service: the routeflow approach leveraging software-defined networks. In: INTERNATIONAL CONFERENCE ON FUTURE INTERNET TECHNOLOGIES, 2011...**Proceedings.** Seoul, Korea: ACM, 2011. p. 34–37. Disponível em: <http://doi.acm.org/10.1145/2002396.2002405>.

NEVES, M. et al. Contando os segundos: avaliação de estratégias de domínio temporal para a gerência de regras em redes sdn. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS - SBRC, Salvador, 2016... **Anais.** [S.l.: s.n.], 2016. p. 542–555.

ROSEN, E. et al. **RFC 4364: BGP/MPLS IP Virtual Private Networks (VPNs)**. 2006. 2006. Disponível em: <<http://www.ietf.org/rfc/rfc4364.txt>>. Acesso em: 30 apr. 2013.

SALVADORI, E. et al. Generalizing virtual network topologies in openflow-based networks. In: IEEE GLOBAL TELECOMMUNICATIONS CONFERENCE, 2011...**Proceedings.** Kathmandu, Nepal: IEEE, 2011. p. 1–6. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6134525/>.

SHERWOOD, R. et al. Flowvisor: A network virtualization layer. **OpenFlow Switch Consortium, Technical Report**, 2009.

SPITZNAGEL, E.; TAYLOR, D.; TURNER, J. Packet classification using extended tcams. In: IEEE INTERNATIONAL CONFERENCE ON NETWORK PROTOCOLS, 2003...**Proceedings.** Atlanta, Georgia: IEEE, 2003. p. 120–131. Disponível em: <https://ieeexplore.ieee.org/abstract/document/1249762/>.

TEGUEU, A. S. et al. A reactive resource defragmentation method for virtual links mapping in software-defined networks. In: IEEE INTERNATIONAL SYMPOSIUM ON LOCAL AND METROPOLITAN AREA NETWORKS, 2017...**Proceedings.** Osaka, Japan: IEEE, 2017. p. 1–6. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7972143/>.

YU, M. et al. Rethinking virtual network embedding: substrate support for path splitting and migration. **SIGCOMM Computer Communication Review**, ACM, New York, USA, v. 38, n. 2, p. 17–29, mar. 2008. ISSN 0146-4833.

ZHANG, Z. et al. Adaptive multi-objective artificial immune system based virtual network embedding. **Journal of Network and Computer Applications**, Elsevier, v. 53, p. 140–155, 2015.

ZHONG, X. et al. Flowvisor-based cost-aware vn embedding in openflow networks. **International Journal of Network Management**, Wiley Online Library, v. 26, n. 5, p. 373–395, 2016.

APPENDIX A – PUBLISHED PAPER – JISA, 2015

- Title: Virtual network security: threats, countermeasures, and challenges
- Journal: Journal of Internet Services and Applications
- Date: January, 2015
- DOI: <https://doi.org/10.1186/s13174-014-0015-z>

Bays *et al. Journal of Internet Services and Applications* (2015) 6:1
DOI 10.1186/s13174-014-0015-z

Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH Open Access

Virtual network security: threats, countermeasures, and challenges

Leonardo Richter Bays[1], Rodrigo Ruas Oliveira[1], Marinho Pilla Barcellos[1], Luciano Paschoal Gaspary[1*] and Edmundo Roberto Mauro Madeira[2]

Abstract

Network virtualization has become increasingly prominent in recent years. It enables the creation of network infrastructures that are specifically tailored to the needs of distinct network applications and supports the instantiation of favorable environments for the development and evaluation of new architectures and protocols. Despite the wide applicability of network virtualization, the shared use of routing devices and communication channels leads to a series of security-related concerns. It is necessary to provide protection to virtual network infrastructures in order to enable their use in real, large scale environments. In this paper, we present an overview of the state of the art concerning virtual network security. We discuss the main challenges related to this kind of environment, some of the major threats, as well as solutions proposed in the literature that aim to deal with different security aspects.

Keywords: Network virtualization; Security; Threats; Countermeasures

1 Introduction

Virtualization is a well established concept, with applications spanning several areas of computing. This technique enables the creation of multiple virtual platforms over a single physical infrastructure, allowing heterogeneous architectures to run on the same hardware. Additionally, it may be used to optimize the usage of physical resources, as an administrator is able to dynamically instantiate and remove virtual nodes in order to satisfy varying levels of demand.

In recent years, there has been a growing demand for adaptive network services with increasingly distinct requirements. Driven by such demands, and stimulated by the successful employment of virtualization for hosting custom-built servers, researchers have started to explore the use of this technique in network infrastructures. Network virtualization allows the creation of multiple independent virtual network instances on top of a single physical substrate [1]. This is made possible by instantiating one or more virtual routers on physical devices and establishing virtual links between these routers, forming topologies that are not limited by the structure of the physical network.

In addition to the ability to create different topological structures, virtual networks are also not bound by other characteristics of the physical network, such as its protocol stack. Thus, it is possible to instantiate virtual network infrastructures that are specifically tailored to the needs of different network applications [2]. These features also enable the creation of virtual testbeds that are similar to real infrastructures, a valuable asset for evaluating newly developed architectures and protocols without interfering with production traffic. [3] For these reasons, network virtualization has attracted the interest of a number of researchers worldwide, especially in the context of Future Internet research. Network virtualization has been embraced by the Industry as well. Major Industry players – such as Cisco and Juniper – nowadays offer devices that support virtualization, and this new functionality allowed infrastructure providers to offer new services.

In contrast to the benefits brought by network virtualization, the shared use of routing devices and communication channels introduces a series of security-related concerns. Without adequate protection, users from a virtual network might be able to access or even interfere with traffic that belongs to other virtual networks, violating

*Correspondence: paschoal@inf.ufrgs.br
[1] Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
Full list of author information is available at the end of the article

security properties such as confidentiality and integrity [4,5]. Additionally, the infrastructure could be a target for denial of service attacks, causing availability issues for virtual networks instantiated on top of it [6,7]. Therefore, it is of great importance that network virtualization architectures offer protection against these and other types of threats that might compromise security.

Recently, attention has been drawn to security concerns in network infrastructures due to the discovery of pervasive electronic surveillance around the globe. Although all kinds of networks are potentially affected, the shared use of physical resources in virtual network environments exacerbates these concerns. As such, these recent circumstances highlight the need for a comprehensive analysis of current developments in the area of virtual network security.

In this paper, we characterize the current state of the art regarding security in network virtualization. We identify the main threats to network virtualization environments, as well as efforts aiming to secure such environments. For this study, an extensive literature search has been conducted. Major publications from the literature have been studied and grouped according to well known classifications in the area of network security, as well as subcategories proposed by the authors of this paper. This organization allows the analysis and discussion of multiple aspects of virtual network security.

The remainder of this paper is organized as follows. Section 2 presents a brief background on the area of network virtualization as well as a review of related literature. Section 3 introduces the taxonomy used to classify the selected publications. Section 4 exposes the security vulnerabilities and threats found in the literature, while Section 5 presents the security countermeasures provided by solutions found in previous proposals. In Section 6, we discuss the results of this study, and in Section 7 we summarize the main current research challenges in the area of virtual network security. Last, in Section 8 we present our conclusions.

2 Background and literature review

In this section, we first provide a brief background on the area of network virtualization, highlighting its most relevant concepts. Next, we present a review of literature closely related to virtual network security.

2.1 Background

Network virtualization consists of sharing resources from physical network devices (routers, switches, etc.) among different virtual networks. It allows the coexistence of multiple, possibly heterogeneous networks, on top of a single physical infrastructure. The basic elements of a network virtualization environment are shown in Figure 1. At the physical network level, a number of autonomous systems are represented by interconnected network substrates (e.g., substrates A, B, and C). Physical network devices are represented by nodes supporting virtualization technologies. Virtual network topologies (e.g., virtual networks 1 and 2), in turn, are mapped to a subset of nodes from one or more substrates. These topologies are composed of virtual routers, which use a portion of the resources available in physical ones, and virtual links, which are mapped to physical paths composed of one or more physical links and their respective intermediate routers.

From the point of view of a virtual network, virtual routers and links are seen as dedicated physical devices. However, in practice, they share physical resources with routers and links from other virtual networks. For this reason, the virtualization technology used to create this environment must provide an adequate level of isolation in order to enable the use of network virtualization in real, large scale environments.

Over the years, different methods for instantiating virtual networks have been used. Typical approaches include VLANs (Virtual Local Area Networks) and VPNs (Virtual Private Networks). Recently, Virtual Machine Monitors and programmable networks have been employed to create virtual routers and links over physical devices and communication channels. These approaches are briefly revisited next.

2.1.1 Protocol-based approaches

Protocol-based approaches consist of implementing a network protocol that enables the distinction of virtual networks through techniques such as tagging or tunneling. The only requirement of this kind of approach is that physical devices (or a subset of them) support the selected protocol.

One example of protocol-based network virtualization are VLANs. VLANs consist of logical partitions of a single underlying network. Devices in a VLAN communicate with each other as if they were on the same Local Area Network, regardless of physical location or connectivity. All frames sent through a network are tagged with their corresponding VLAN ID, processed by VLAN-enabled routers and forwarded as necessary [8]. Since isolation is typically based only on packet tagging, this approach is susceptible to eavesdropping attacks.

Another commonly used approach is the creation of Virtual Private Networks. VPNs are typically used to provide a secure communication channel between geographically distributed nodes. Cryptographic tunneling protocols enable data confidentiality and user authentication, providing a higher level of security in comparison with VLANs. VPNs can be provided in the physical, data link, or network layers according to the protocols being employed [9].

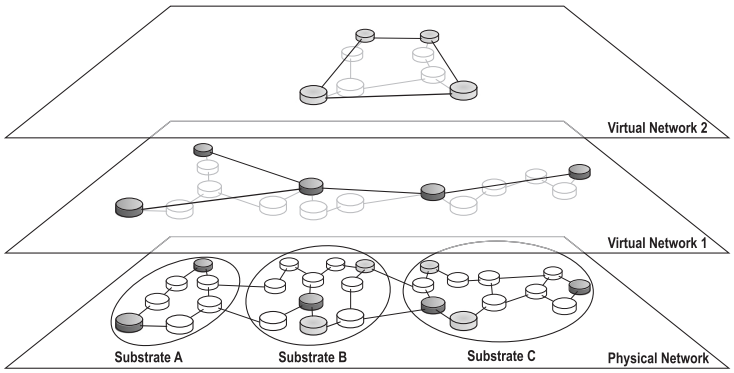


Figure 1 Network virtualization model, denoting a scenario with multiple physical substrates (Substrate A, B, and C) and virtual networks (Virtual Network 1 and 2).

2.1.2 Machine virtualization-based approaches

Machine virtualization-based approaches consist of creating virtual networks by means of groups of interconnected virtual machines. Virtual Machine Monitors are used to instantiate virtual routers, and virtual links are established between them, regardless of physical network topology. Table 1 shows different machine virtualization-based techniques that can be used to create virtual networks, as well as a brief explanation and an example of each.

This alternative is remarkably flexible and relatively cheap, as it allows the use of customized software and does not require the use of specific hardware[1]. However, it is more demanding in terms of resource usage in comparison to previously described protocol-based approaches. Additionally, it may introduce security concerns associated with server virtualization, some of which are mentioned in Sections 4 and 5. A general study on the security issues that arise from the use of machine virtualization was performed by van Cleeff *et al.* [10].

2.1.3 Programmable networks

Programmable routers have been used to enable the creation of virtual networks. Although this is not a new concept, research in this area has been recently stimulated by the inception of Software-Defined Networking (SDN). This paradigm consists of decoupling the data plane and the control plane in network devices. More specifically, devices such as routers and links retain only the data plane, and a separated control plane manages such devices based on an overview of the entire network.

OpenFlow [11], one of the most promising techniques for implementing this paradigm, defines a protocol that allows a centralized controller to act as the control plane, managing the behavior of network devices in a dynamic manner. The controller communicates with network devices through a secure connection, creating and managing flow rules. Flow rules instruct network devices on how to properly process and route network traffics with distinct characteristics. Through the establishment of specific flow rules, it is possible to logically partition physical networks and achieve data plane isolation. This isolation enables the creation of virtual networks on top of an SDN environment. OpenFlow gave rise to the Open Networking Foundation, an organization ran by major companies within the area of computer networks that aims to disseminate this type of technology.

Table 1 Virtualization techniques

Technique	Description	Examples
Full virtualization	The Virtual Machine Monitor emulates a complete machine, based on the underlying hardware architecture. The guest Operating System runs without any modification.	VMware Workstation, VirtualBox
Paravirtualization	The Virtual Machine monitor emulates a machine which is similar to the underlying hardware, with the addition of a hypervisor. The hypervisor allows the guest Operating System to run complex tasks directly on non-virtualized hardware. The guest OS must be modified in order to take advantage of this feature.	VMware ESX, Xen
Container-based virtualization	Instead of running a full Virtual Machine, this technique provides Operating System-level containers, based on separate userspaces. In each container, the hardware, as well as the Operating System and its kernel, are identical to the underlying ones.	OpenVZ, Linux VServer

2.2 Literature review

To the best of our knowledge, there have been no previous attempts at characterizing the state of the art regarding security in network virtualization. However, there have been a number of similar studies in other, closely related fields of research. We now proceed to a review of some of the main such studies.

Chowdhury *et al.* [1] provide a general survey in the area of network virtualization. The authors analyze the main projects in this area (both past projects and, at the time of publication, current ones) and discuss a number of key directions for future research. The authors touch upon the issues of security and privacy both while reviewing projects and discussing open challenges; however, as this is not the main focus of this survey, there is no in-depth analysis of security issues found in the literature.

Bari *et al.* [12] present a survey that focuses on data center network virtualization. Similarly to the aforementioned study, the authors survey a number of key projects and discuss potential directions for future work. When analyzing such projects, the authors provide insights on the fault-tolerance capabilities of each one, in addition to a brief discussion on security issues as one of the potential opportunities for future research.

In addition to the general studies on network virtualization presented so far, a number of surveys on cloud computing security have also been carried out. Cloud computing environments tend to make use of both machine and network virtualization, making this a highly relevant related topic for our study. However, while there is some overlap between cloud computing security and virtual network security, we emphasize that cloud computing represents a very specific use case of network virtualization and, therefore, poses a significantly distinct set of security challenges. Zhou *et al.* [13] provide an investigation on security and privacy issues of cloud computing system providers. Additionally, the authors highlight a number of government acts that originally intended to uphold privacy rights but fail to do so in light of advances in technology. Hashizume *et al.* [14], in turn, focus on security vulnerabilities, threats, and countermeasures found in the literature and the relationships among them.

Last, Scott-Hayward *et al.* [15] conducted a study on SDN security. As explained in Section 2.1.3, this is one of the technologies on top of which network virtualization environments can be instantiated. The authors first analyze security issues associated with the SDN paradigm and, afterwards, investigate approaches aiming at enhancing SDN security. Last, the authors discuss security challenges associated with the SDN model.

3 Taxonomy

The first step towards a comprehensive analysis of the literature was the selection of a number of publications from quality conferences and journals. To this end, we performed extensive searches in the ACM and IEEE digital libraries using a number of keywords related to network virtualization and security. We then ranked the literature found through this process according to the average ratio of citations per publication of the conferences or journals in which these papers were published. All publications from top tier conferences or journals with a consistent number of citations per publication were considered relevant and, therefore, selected. The remaining papers were analyzed and generally discarded.

Following the aforementioned process, a taxonomy was created in order to aid the organization and discussion of the selected publications. For this purpose, two well known classifications in the area of network security were chosen. Papers are organized according to the *security threats* they aim to mitigate, and afterwards, according to the *security countermeasures* they provide. As different authors have different definitions for each of these concepts, these classifications are briefly explained in the following subsections. The direct connection between them and the area of virtual network security is explained in sections 4 and 5, respectively.

In addition to these broad classifications, subcategories were created in order better organize this body of work. Figure 2 presents the full hierarchical organization that will be used in sections 4 and 5. Dark gray boxes represent broad categories used in the literature [16,17], while white boxes denote subdivisions proposed and created by the authors of this paper.

3.1 Security vulnerabilities and threats

There are a number of potential malicious actions, or threats, that may violate security constraints of computational systems. Shirey [16] describes and divides the consequences of these threats into four categories, namely *disclosure*, *deception*, *disruption*, and *usurpation*.

Unauthorized *disclosure* is defined as gaining unauthorized access to protected information. Sensitive data may be erroneously exposed to unauthorized entities, or acquired by an attacker that circumvents the system's security provisions.

Deception is characterized by intentionally attempting to mislead other entities. For example, a malicious entity may send false or incorrect information to others, leading them to believe that this information is correct. Fake identities may be used in order to incriminate others or gain illegitimate access.

Disruption means causing failure or degradation of systems, negatively affecting the services they provide.

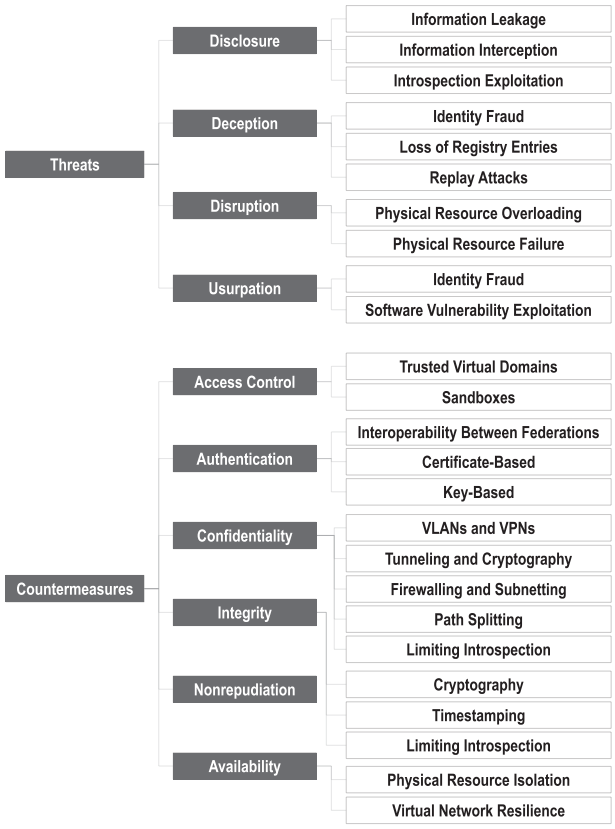


Figure 2 Taxonomy used to classify publications in the area of virtual network security.

This may be done by directly incapacitating a system component or the channel through which information is delivered, or by inducing the system to deliver corrupted information.

Last, through *usurpation*, an attacker may gain unauthorized control over a system. This unauthorized control may allow the attacker to illegitimately access protected data or services, or tamper with the system itself in order to cause incorrect or malicious behavior.

These threat categories, as well as the previously mentioned subcategories we have created, also cover vulnerabilities and attacks. For ease of comprehension, vulnerabilities and threats are discussed collectively in Section 4. Table 2 presents the relationships between vulnerabilities and threats in network virtualization environments. This table is organized according to the previously described taxonomy and lists all vulnerabilities found in the literature and the threats associated with each one. Additionally, the terms threat and attack are used interchangeably throughout the paper, as a threat may be understood as a potential attack (while an attack is the proper action that takes advantage of a vulnerability to violate a security policy).

3.2 Security countermeasures

Due to the existence of the previously described threats, computational systems must provide a series of countermeasures in order to maintain a desirable level of security. Stallings [17] categorizes these essential countermeasures into six subdivisions (referred to by Stallings as "security services"), namely *access control*, *authentication*, *data confidentiality*, *data integrity*, *nonrepudiation*, and *availability*.

Access control allows a system to administer which entities will be able to access its functions, and what permissions each of these entities will have. In order to grant

Table 2 Relationships between vulnerabilities and threats in network virtualization environments

Threat categories		Vulnerabilities	Threats
Disclosure	Information Leakage	Lack of ARP table protection	ARP table poisoning
		Placement of firewall rules inside virtual nodes	Subversion of firewall rules
	Information Interception	Lack of ARP table protection	ARP table poisoning
		Transmission of data in predictable patterns	Traffic Analysis attacks
		Uncontrolled handling of multiple, sequential virtual network requests from a single entity	Inference and disclosure of sensitive topological information
		Unprotected exchange of routing information among virtual routers	Disclosure of sensitive routing information
	Introspection Exploitation	Uncontrolled Introspection	Data theft
Deception	Identity Fraud	Improper handling of identities: - within individual networks;	Injection of malicious messages with forged sources
		- among federated networks;	Privilege escalation
		- during migration procedures.	Abuse of node removal and re-addition in order to obtain new (clean) identities
	Loss of registry entries	Uncontrolled rollback operations	Loss of registry entries
	Replay attacks	Lack of unique message identifiers	Replay attacks
Disruption	Physical Resource Overloading	Uncontrolled resource allocation	Performance degradation
			Abusive resource consumption
		Uncontrolled handling of virtual network requests	Exhaustion of resources in specific parts of the infrastructure
		Lack of proactive or reactive recovery strategies	Denial of Service attacks
	Physical Resource Failure	Lack of proactive or reactive recovery strategies	Failure of virtual routers/networks
		Uncontrolled resource reallocation after failures	Overloading of remaining virtual routers after failures
Usurpation	Identity Fraud	Improper handling of identities and associated privileges	Privilege escalation
	Software Vulnerability Exploitation	Privilege escalation in Virtual Machine Monitors	Unauthorized control of physical routers

individual access rights and permissions, entities must be properly authenticated in the system.

The purpose of *authentication* is to ensure that entities communicating with each other are, in fact, the entities they claim to be. The receiver of a message must be able to correctly identify its sender, and an entity must not be able to impersonate another.

Providing adequate *data confidentiality* means ensuring that third parties do not have access to confidential information being transmitted between two entities. Additionally, the system should inhibit attackers from deriving information by analyzing traffic flow characteristics.

Data integrity has the purpose of assuring that data stored by entities or transmitted through a network are not corrupted, adulterated or destroyed. Attacks such as duplication, modification, reordering, and replay of messages must be prevented. Furthermore, mechanisms for recovering from data corruption may also be provided.

In communications between peers, *nonrepudiation* provides a way to settle disputes when an entity denies having performed a certain action. The goal of this service is to prevent entities from falsely denying participation in any (possibly malicious) network-related activity.

The last security countermeasure is *availability*. System resources must be available upon request by an authorized entity, and the system must also conform to its performance specifications. In order to maintain availability, countermeasures against attacks such as *denial of service* must be provided.

4 Security vulnerabilities and threats

In this section, we present a comprehensive list of vulnerabilities and threats found in network virtualization environments. The interested reader should refer to Table 2 for a systematic review of such vulnerabilities and threats.

While some of the threats listed in this section are a result of accidental actions, we emphasize that all

threats – intentional or accidental – have an effect on security. As an example of an accidental attack, it is common for virtual routers to attempt to use all available resources (as virtualization tends to be transparent and virtual routers are typically not aware that they are not running on dedicated physical hardware). If the network virtualization environment does not adequately limit the resource usage of each virtual router, even this unintentional abuse may cause disruption on other networks hosted on the same substrate or cause the degradation or failure of critical services provided by the virtualization environment.

4.1 Disclosure

In an environment where physical resources are shared between a number of virtual networks, there is a series of behaviors that may result in undesired disclosure of information. Threats related to disclosure of private or sensitive information are explained next.

4.1.1 Information leakage

Cavalcanti *et al.* [18] mention the possibility of messages being leaked from one virtual network to another. In this type of attack, an entity may disclose private or sensitive information to members of other virtual networks, who should not have access to such information. The authors state that this may be achieved through ARP table poisoning. For example, a malicious user may spoof the IP address of a node that is able to send messages to the virtual network with which it intends to communicate. Wolinsky *et al.* [19] describe a similar attack, in which virtual nodes send messages to outside the boundaries of a network virtualization environment. This would make it possible for messages to reach physical nodes that not only do not belong to any virtual network, but are hosted outside of the virtualized network infrastructure. According to the authors, if data isolation is achieved by means of firewall rules, malicious users may be able to subvert such rules by escalating privileges and gaining root access on a virtual node.

4.1.2 Information interception

Attackers in a virtual network environment may capture messages being exchanged between two entities in order to access their content. This type of attack, often referred to as "eavesdropping" or "sniffing", may lead to theft of confidential information [4,5,20]. Wu *et al.* [20], specifically, mention ARP table poisoning as a means of achieving this. In contrast to the ARP poisoning attack described by Cavalcanti *et al.* [18] (explained in Section 4.1.1), in this case the attack would be used in order to mislead physical routers into forwarding packets meant to one entity to another one, allowing a malicious entity to sniff such packets. This is a common threat in any networking environment, but the use of shared physical resources by multiple virtual networks further exacerbates this problem. According to these and other authors, such as Cui *et al.* [21], networking solutions provided by virtual machine monitors may not properly isolate data belonging to different virtual networks. This means that members of one virtual network may be able to access data being transferred by other virtual networks sharing the same substrate.

Even if data inside network packets is protected (e.g. through the use of cryptography), entities may be able to derive sensitive information by analyzing them. In traffic analysis attacks, described by Huang *et al.* [22], entities acquire such information by analyzing characteristics of traffic flows between communicating entities in virtual networks. These characteristics include which entities communicate with which other entities, frequency of communication, and packet sizes, among others. For example, an entity that is involved in frequent, short communications with a high number of other entities may be a central point of control in the network. Knowing this, a malicious user could launch an attack directed at that entity, aiming to cause a considerable amount of disruption with limited effort. As previously mentioned, this attack is effective even if traffic is encrypted, making any type of virtual networking environment a potential target.

In addition to the previously detailed forms of information interception, which may also affect traditional network environments, other forms are specific to network virtualization. One such form is the use of multiple virtual network requests to disclose the topology of the physical infrastructure, explored by Pignolet *et al.* [23]. This constitutes a security threat, as infrastructure providers typically do not wish to disclose this information. The authors demonstrate that by sequentially requesting a number of virtual networks with varying topological characteristics and analyzing the response given by the infrastructure provider (*i.e.*, whether the request can be embedded or not), they are able to gradually obtain information about the physical topology. Moreover, the authors determine the number of requests needed to fully disclose the physical topology on networks with different topological structures (tree, cactus, and arbitrary graphs). Conversely, Fukushima *et al.* [24] state that the entity controlling a physical network may obtain confidential routing information from virtual networks hosted on top of it. As current routing algorithms require routing information to be sent and received through virtual routers, sensitive information may be disclosed to the underlying network.

4.1.3 Introspection exploitation

Introspection is a feature present in virtual machine monitors that allows system administrators to verify the current state of virtual machines in real time. It enables external

observers to inspect data stored in different parts of the virtual machine (including processor registers, disk, and memory) without interfering with it. While this feature has valuable, legitimate uses (*e.g.*, enabling administrators to verify that a virtual machine is operating correctly), it may be misused or exploited by attackers in order to access (and potentially disclose) sensitive data inside virtual machines [10]. This problem is aggravated by the fact that virtual nodes may be moved or copied between multiple virtual machine monitors, as sensitive data may be compromised through the exploitation of this feature on any virtual machine monitor permanently or temporarily hosting such virtual nodes.

4.2 Deception

We have identified three subcategories of threats that may lead to deception in virtual network environments. These subdivisions – namely identity fraud, loss of registry entries and replay attacks – are explained next.

4.2.1 Identity fraud

In addition to dealing with unauthorized disclosure, Cabuk *et al.* [5] and Wu *et al.* [20] also describe threats related to deception in virtual network environments. Specifically, virtual entities may inject malicious messages into a virtual network, and deceive others into believing that such messages came from another entity.

Certain characteristics of virtualized network environments increase the difficulty of handling identity fraud. The aggregation of different virtual networks into one compound network, known as federation, is indicated by Chowdhury *et al.* [25] as one of such characteristics. Federation raises issues such as the presence of separate roles and possible incompatibility between security provisions or policies from aggregated networks. Another complicating factor mentioned by the authors is the dynamic addition and removal of entities. An attacker may force a malicious node to be removed and re-added in order to obtain a new identity.

Other characteristics that complicate the handling of identity fraud involve operations such as migration and duplication of virtual nodes, as mentioned by van Cleeff *et al.* [10]. The study presented by the authors refers to virtualization environments in general. Therefore, in the context of this study, a virtual node may refer to either a virtual router or a virtual workstation. If a virtual node is migrated from one physical point to another, the identity of the machine that contains this virtual node may change. Moreover, virtual nodes may be copied to one or more physical points in order to provide redundancy, which may lead to multiple entities sharing a single identity. Both of these issues may cause inconsistencies in the process of properly identifying the origin of network messages, which may be exploited in identity fraud attacks.

4.2.2 Loss of registry entries

Van Cleeff *et al.* [10] also mention issues related to logging of operations in virtualization environments. If information regarding which entity was responsible for each operation in the network is stored in logs inside virtual machines, entries may be lost during rollback procedures. Likewise, logs of malicious activities performed by attackers may also be lost.

4.2.3 Replay attacks

Fernandes and Duarte [26] mention replay attacks as another form of deception in virtual networks. In this type of attack, a malicious entity captures legitimate packets being transfered through the network and retransmits them, leading other entities to believe that a message was sent multiple times. The authors explain that virtual routers may launch attacks in which they repeat old control messages with the intention of corrupting the data plane of the attacked domain.

4.3 Disruption

In a network virtualization environment, proper management of resources is crucial to avoid disruption. The main sources of disruption in such environments are related to the abuse of physical resources (either intentional or unintentional) and the failure of physical devices.

4.3.1 Physical resource overloading

Physical resource overloading may lead to failure of virtual nodes, or cause the network performance to degrade below its minimum requirements. This degradation may cause congestion and packet loss in virtual networks, as stated by Zhang *et al.* [27]. In addition to causing disruption in already established networks, overloading may also hinder the deployment of new ones.

Resource requirements themselves can be a point of conflict in virtual network environments. As explained by Marquezan *et al.* [28], multiple virtual networks may require an excessive amount of resources in the same area of the substrate network. While such prohibitive demands may be unintentional, they may also be due to a coordinated attack. This may not only happen during deployment operations, but also during the lifetime of virtual networks.

It is also possible for one virtual network to disrupt another by using more than its fair share of resources. This concern is explored by a number of authors in their respective publications [26,29-31]. Isolation and fair distribution of physical resources among virtual networks are essential to maintain the network virtualization environment operating properly. This includes assuring that the minimum requirements of each network will be fulfilled, as well as prohibiting networks from consuming more resources than they are allowed to.

Overloading may also be caused by attacks aimed at the physical network infrastructure. Attacks may originate from within a virtual network hosted in the same environment, or from outside sources. The most common threats are Denial of Service (DoS) attacks, as presented by Yu *et al.* [6] and Oliveira *et al.* [7]. A single physical router or link compromised by a DoS attack may cause disruption on several virtual networks currently using its resources.

4.3.2 Physical resource failure

As previously stated, the failure of physical devices is one of the sources of disruption in virtual infrastructures [32-34]. Possible causes range from the failure of single devices (a physical router, for example, may become inoperative if one of its components malfunctions) to natural disasters that damage several routers or links in one or more locations [35]. Additionally, further complications may arise as the remainder of the network may be overloaded during attempts to relocate lost virtual resources. In addition to being valuable from the point of view of fault tolerance, countermeasures for mitigating the effect of failures may also be applied in the event of attacks such as DoS, as in both cases there is a need for redirecting network resources away from compromised routers or links.

4.4 Usurpation

In virtual network environments, usurpation attacks may allow an attacker to gain access to privileged information on virtual routers, or to sensitive data stored in them. Such attacks may be a consequence of identity fraud or exploited vulnerabilities, which are explained next.

4.4.1 Identity fraud

As previously mentioned in Section 4.2, identity fraud attacks can be used to impersonate other entities within a virtual network. By impersonating entities with high levels of privilege in the network, attackers may be able to perform usurpation attacks. As an example, the injection of messages with fake sources mentioned by Cabuk *et al.* [5] is used for this purpose. By sending a message that appears to have been originated from a privileged entity, attackers may perform actions restricted to such entities, including elevating their own privilege level.

4.4.2 Software vulnerability exploitation

Roschke *et al.* [36] mention that virtual machine monitors are susceptible to the exploit of vulnerabilities in their implementation. According to the authors, by gaining control over a virtual machine monitor, attackers can break out of the virtual machine, obtaining access to the hardware layer. In an environment that uses full virtualization or paravirtualization to instantiate virtual routers, exploiting such vulnerabilities may enable an attacker to have full control over physical routers. By gaining access to physical devices, attackers could easily compromise any virtual networks provided by the infrastructure. As examples of such threats in practice, the Common Vulnerabilities and Exposures system lists a number of vulnerabilities in different versions of VMware products that allow guest Operating System users to potentially execute arbitrary code on the host Operating System [37-40].

5 Security countermeasures

In this section, we explore solutions published in the literature that aim to provide security and protect the environment from the aforementioned security threats.

5.1 Access control

Access control makes use of authentication and authorization mechanisms in order to verify the identity of network entities and enforce distinct privilege levels for each. This countermeasure is approached in two different manners in the literature, namely Trusted Virtual Domains and sandboxes. While these approaches are closely related to the notion of controlled execution domains, note that access control is performed in order to ensure that entities are granted the appropriate privilege levels.

5.1.1 Trusted virtual domains

Cabuk *et al.* [5] devised a framework to provide secure networking between groups of virtual machines. Their security goals include providing isolation, confidentiality, integrity, and information flow control in these networks. The framework provides the aforementioned security countermeasures through the use of Trusted Virtual Domains (TVDs). Each TVD represents an isolated domain, composed of "virtualization elements" and communication channels between such elements. In Cabuk's proposal, the virtualization elements are virtual workstations. However, the concept of TVDs may be applied to any device supporting virtualization.

Figure 3 depicts a virtual network infrastructure with three TVDs (A, B, and C). Gray routers represent gateways between these domains. While the gateway between TVDs B and C is simultaneously within both domains, the gateways between A and B are isolated – making use of an auxiliary TVD (AB) in order to communicate.

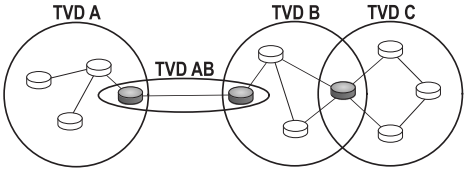


Figure 3 Example of a virtual infrastructure with three Trusted Virtual Domains, as described by Cabuk *et al.* [5].

Access control is performed when virtual machines join a TVD, ensuring that only machines that satisfy a given set of conditions are able to join. This admission control may be applied continuously in case prerequisites to join a TVD are changed. Additionally, TVDs leverage access policies to prevent unauthorized access.

5.1.2 Sandboxes

Wolinsky *et al.* [19] use virtual machine sandboxes in order to provide security in large scale collaborative environments. Although this work focuses on networked virtual machines hosting virtual workstations, this concept can be extended to virtual networks. Sandboxes are used to limit virtual machine access to physical resources, preventing malicious virtual machines from accessing data within other virtual machines. Moreover, each virtual machine supports IPSec, enabling the creation of secure communication channels, and X.509, providing virtual machine authentication. The authentication process is detailed in Section 5.2.

5.2 Authentication

Authentication aims to ensure that entities in a network environment are who they claim to be. In virtual network environments, providing proper authentication is complicated by factors such as the federation of virtual networks or mobility of virtual routers and links. Approaches that aim to deal with such difficulties are explained next.

5.2.1 Interoperability between federated virtual networks

Although isolation is one of the main security requirements in virtual networking, there are cases in which distinct virtual networks must be able to cooperate. The federation of virtual networks can, for example, enable end-to-end connectivity – through virtual devices of distinct virtual networks – or allow access to distinct services. However, it may not be possible to provide interoperability due to the heterogeneous nature of virtual networks (which may implement different, incompatible protocols). Chowdhury *et al.* [25] partially tackle this issue with a framework that manages identities in this kind of environment. The main objective of the work is to provide a global identification system. To this end, the authors employ a decentralized approach in which controllers and adapters are placed in each virtual network. Controllers provide functionalities such as address allocation and name resolution, while adapters act as gateways between virtual networks, performing address and protocol translations. The proposed global identification system does not restrict the internal identification mechanisms used locally by virtual networks, allowing each virtual network to keep its own internal naming scheme. Additionally, global identifiers used by this framework are unique, immutable, and not associated with physical location, in order to not hinder the security or mobility of virtual devices.

5.2.2 Certificate-based

As previously mentioned, the framework presented by Cabuk *et al.* [5] makes use of Trusted Virtual Domains (TVDs) to provide access control and network isolation. The authentication necessary to support access control is provided by means of digital certificates. These certificates ensure the identity of entities joining the network. Additionally, the system makes use of Virtual Private Networks (VPNs) to authenticate entities in network communications.

Analogously, Wolinsky *et al.* [19] use IPSec with X.509-based authentication for the purpose of access control in their system. In order to access the system, joining machines must request a certificate to the Certification Authority (CA). The CA responds by sending back a signed certificate to the node. The IP address of the requesting node is embedded into the certificate in order to prevent other nodes from reusing it.

5.2.3 Key-based

Fernandes and Duarte [26,31] present an architecture that aims to provide efficient routing, proper resource isolation and a secure communication channel between routers and the Virtual Machine Monitor (VMM) in a physical router. In order to ensure efficiency, virtual routers copy routing-related information to the VMM – in this case, the hypervisor. This process is performed by a plane separation module, which separates the data plane (which contains routing rules) and the control plane (responsible for creating routing rules). As a result, packets matching rules in the hypervisor routing table do not need to be redirected to virtual routers, resulting in a significant performance speedup. However, the process of copying routing information needs to be authenticated such that a malicious router is not able to compromise the data plane of another router.

In order to prevent identity fraud, the system requires mutual authentication between virtual routers and the VMM. Figure 4 depicts a simplified representation of the proposed architecture. The authors consider a Xen (paravirtualization)-based environment, in which virtual routers reside in unprivileged domains (DomUs) while the hypervisor resides within the privileged domain (Dom0). Each virtual router, upon instantiation, connects to the hypervisor following the client–server paradigm and performs an initial exchange of session keys using asymmetrical cryptography. The use of unique keys allows the hypervisor to verify the identity of distinct virtual routers in different unprivileged domains (in this example, $DomU_1$, $DomU_2$, and $DomU_3$) and to isolate traffic between them. After this initial key exchange, the secure

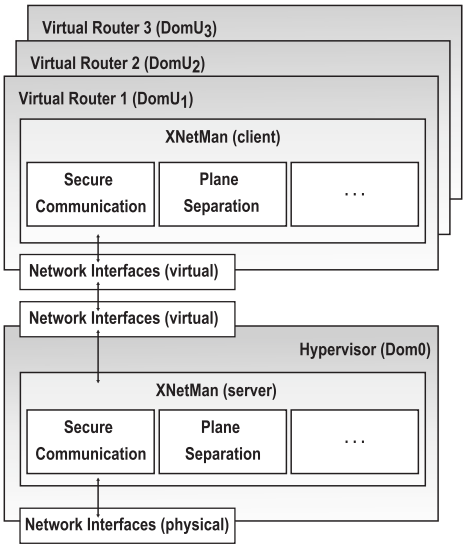


Figure 4 Simplified version of the architecture presented by Fernandes *et al.* [26,31], showing the secure communication modules.

communication module is used by other system modules in order to securely exchange messages with the hypervisor.

5.3 Data confidentiality

As network virtualization promotes the sharing of network devices and links among multiple entities, data confidentiality is a major security-related concern. Next, we explore approaches that leverage different protocols and techniques in order to provide secure communication within virtual networks.

5.3.1 VLANs and VPNs

The security goals approached by Cabuk *et al.* [5] include integrity, data isolation, confidentiality, and information flow control. Other than integrity, the remaining three goals, are directly related, and are tackled by a data confidentiality mechanism. The framework uses TVDs to control data access. However, virtual machines that belong to different TVDs may be hosted in the same physical machine. Therefore, it is necessary to ensure proper isolation, preventing a TVD from accessing data that belongs to another TVD.

The proposed solution for this challenge employs a combination of VLANs and VPNs. VLANs are used to identify packets belonging to different networks, allowing VLAN-enabled devices to route packets to the appropriate network interfaces, thus providing adequate isolation. Untrusted physical channels, however, may require a higher level of security. Therefore, if necessary, VPNs are used to provide data confidentiality by means of end-to-end cryptography.

5.3.2 Tunneling and cryptography

Wolinsky *et al.* [19] make use of tunneling in order to isolate network traffic between virtual machines (in this case, virtual workstations). Two tunneling approaches are employed. In the first approach, the host system runs a tunneling software that captures packets incoming from physical interfaces and forwards them to virtual machines. In the second approach, the tunneling software runs inside virtual machines, and traffic is restricted within virtual networks through the use of firewall rules. According to the authors, while the second approach is easier to deploy, malicious users may be able to subvert this firewall, compromising the system. Although the focus of Wolinsky *et al.* is isolation between virtual workstations, we believe that the techniques used to achieve such isolation could be extended to virtual routers in network virtualization environments.

Fernandes and Duarte [26,31] deal with data confidentiality in communications between a virtual router and the Virtual Machine Monitor (VMM) hosting it. After the authentication process, described in Section 5.2, virtual routers use symmetrical cryptography in order to securely communicate with the VMM.

Huang *et al.* [22] present a framework that provides secure routing. In the environment presented by the authors, routing information that is propagated through a virtual network is confidential and needs to be kept secret from unauthorized network entities. Routing information is categorized in groups, and group keys are assigned to virtual routers. Therefore, routing information can be encrypted, ensuring that only routers with the correct key are able to decrypt this information. Thus, routing information relative to a given group is protected against unauthorized access from other groups, other virtual networks or the physical network itself.

Similarly to the previously described approach, Fukushima *et al.* [24] aim to protect sensitive routing information in virtual networks from being disclosed to entities controlling the physical network. To achieve this goal, the authors make use of a strategy based on Secure Multi-party Computation (SMC). SMC allows multiple entities to perform joint computations on sensitive data they hold without disclosing such data. Each entity has access to the result of the global computation, but not to any data held by other entities. This is achieved through the use of one-way functions, which are easy to evaluate but hard to invert. In the context of virtual network routing, SMC allows a virtual router to compute optimal

routes without needing to share the information that it holds. As SMC requires full-mesh connectivity between computing nodes, the authors decompose the virtual network into locally connected subsets of routers, called *cliques*. The SMC-based distributed routing algorithm is run locally in each *clique*, and the results of local computations are then shared between *cliques*.

As the employment of cryptographic techniques requires physical devices that are capable of supporting protocols that enable them and generates processing and bandwidth overheads, Bays *et al.* [4] devise an optimization model and a heuristic algorithm for online, privacy-oriented virtual network embedding. Clients may require end-to-end or point-to-point cryptography for their networks, as well as requiring that none of their resources overlap with other specific virtual networks. Both the optimal and heuristic approaches take into account whether physical routers are capable of supporting cryptographic algorithms in order to ensure the desired level of confidentiality and guarantee the non-overlapping of resources (if requested). Additionally, both methods feature precise modeling of overhead costs of security mechanisms in order to not underestimate the capacity requirements of virtual network requests. This proposal is in line with research performed in the area of virtual network embedding, such as the work of Alkmim *et al.* [41].

5.3.3 Firewalling and subnetting

As previously mentioned in Section 5.3.2, Wolinsky *et al.* [19] make use of firewall rules (in addition to tunneling techniques) in order to prevent communications between different virtual networks. In addition to using firewalls for this purpose, Wu *et al.* [20] also employ subnetting (*i.e.*, each virtual network is bound to a unique subnet) in order to provide an additional layer of security against unauthorized information disclosure.

5.3.4 Path splitting

In addition to encryption of routing information, Huang *et al.* [22] use variable paths in virtual networks to propagate data flows. Figure 5 illustrates the employment of path splitting in order to hinder an information interception attack. Communication between a virtual router hosted on Physical Router (PR) 1 and another one hosted on PR 7 is split among two different paths – one passing through PR 3 and 6, and the other, through PR 2 and 4 (represented by dashed lines). Even if traffic between these two virtual routers is not encrypted, the threat is partially mitigated as the attacker only has access to part of the information being exchanged (packets passing through the link between PR 3 and 6). Moreover, when used in combination with encryption (as in the work of Huang *et al.*), this approach helps mitigate traffic analysis attacks.

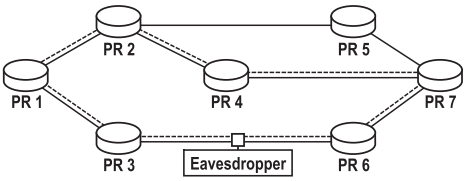


Figure 5 Example of path splitting used to mitigate an information interception attack.

It is worth noting that while in this example the attacker is only eavesdropping on one physical path, in reality, multiple devices may be compromised. In this case, splitting traffic among an increasing number of paths would lead to progressively higher levels of security (or, conversely, to increasingly higher costs for an attacker to capture the full traffic).

5.3.5 Limiting introspection

Finally, van Cleeff *et al.* [10] present recommendations for safer use of virtualization. One of these recommendations is to limit, or even disable, the introspection feature, which allows virtual machine monitors to access data inside virtual machines. While useful, this functionality may be exploited by attackers, as previously explained on Subsection 4.1.3.

5.4 Data integrity

Similarly to confidentiality, data integrity is a major concern as a result of shared network devices and communication channels. Next, we describe approaches that aim to establish a desired level of integrity in virtual network environments.

5.4.1 Cryptography

In addition to authentication (*i.e.*, source integrity) and confidentiality, the framework developed by Cabuk *et al.* [5] makes use of VPNs to provide data integrity to virtual networks. The use of cryptographic tunneling protocols prevents malicious entities from manipulating messages going through the network. As previously discussed, the authors use IPSec as the tunneling protocol.

5.4.2 Timestamping

As previously discussed, replay attacks are one of the threats to data integrity that may be present in network virtualization environments. The addition of unique identifiers inside encrypted messages makes it possible to detect duplicated messages, and therefore, replay attacks. For this purpose, the architecture proposed by Fernandes and Duarte [26,31] inserts timestamps inside encrypted messages in order to ensure that messages are non-reproducible.

5.4.3 Limiting introspection

Besides mitigating information theft, disabling or limiting introspection also prevents data tampering. According to van Cleeff *et al.* [10], this functionality allows the VMM to modify applications running inside it, which may cause inconsistencies. Another recommendation consists of specifically designing applications that facilitate batch processing and checkpointing. According to the authors, this minimizes security issues associated with rollback and restore operations that may otherwise threaten integrity.

5.5 Nonrepudiation

Nonrepudiation provides evidences regarding which (potentially malicious) actions have been performed by which entities. This security countermeasure is highly valuable in the context of network virtualization environments, in which a number of physical devices are shared by different users. Nevertheless, we are not aware of any publication that targets this countermeasure specifically.

5.6 Availability

Last, we present proposals that aim to maintain the availability of network virtualization environments. The key concerns in this area of security are providing proper resource isolation and mitigating attacks that target physical or virtual devices. Approaches aiming to deal with such concerns are explained in the following subsections.

5.6.1 Physical resource isolation

One of the main concerns regarding availability is the abuse of physical resources by virtual networks. Virtual networks may attempt to use as much resources as possible in order to maximize their performance. If the environment is not adequately protected, this behavior may lead to the exhaustion of physical resources, compromising the availability of other virtual networks hosted on the same substrate. Therefore, physical resources must be shared in a fair manner, and actions performed by a virtual network must not negatively impact others.

According to Wu *et al.* [29], the sharing of physical resources by packet processors is usually only performed at a granularity of entire processor cores. The authors claim that finer-grained processor sharing is required in order to provide scalability for network virtualization environments. Thus, the authors propose a system that allows multiple threads to share processor cores concurrently while maintaining isolation and fair resource sharing. However, typical multithreading approaches consider a cooperative environment, which is not the case in network virtualization. The authors devise a fair multithreading mechanism that allows the assignment of different priorities to each thread. Additionally, this mechanism takes into account the history of how much processing has been performed by each thread. Inactivity times are also considered in order to guarantee that threads will not stay idle for too long. The evaluation performed by the authors shows that the proposed mechanism is able to properly distribute processing resources according to the defined priorities. Furthermore, while it requires more processing power, it is able to provide better resource utilization in comparison to coarse-grained approaches.

Kokku *et al.* [30] propose a network virtualization scheme that provides resource isolation while aiming to maximize substrate utilization. It allows virtual networks to have either resource-based reservations (*i.e.*, reservations calculated as a percentage of available resources in the substrate) or bandwidth-based reservations (*i.e.*, reservations based on the aggregate throughput of the virtual network). Virtual networks are divided in two groups according to the type of reservation required, and treated independently by a scheduler. This scheduler treats flows that belong to different virtual networks with distinct priorities, based on the reservations and average resource usage rate of each network. The authors present an evaluation performed on an implemented prototype, showing that the proposed scheme was capable of ensuring that each virtual network met its reservations.

Fernandes and Duarte [26] present a network monitor that employs plane separation in order to provide resource isolation in network virtualization environments. The system is able to allocate resources based on fixed reservations, as well as to redistribute idle resources between virtual networks that have a higher demand. Additionally, an administrator is able to control the amount of resources to be used by each virtual network, as well as set priorities for using idle resources. The system continuously monitors the consumption of physical resources by each virtual router. If any virtual router exceeds its allowed use of bandwidth, processing power, or memory, it is adequately punished by having packets dropped, or a percentage of its stored routes erased. Harsher punishments are instituted if there are no idle resources available. Conversely, given punishments are gradually reduced if the router stops using more than its allocated resources. This system is capable of adequately preventing physical resources from being overloaded, and packet drops employed by the punishment mechanism do not cause a major impact on network traffic.

In another publication [31], the same authors extend the previously described network monitor. This new system introduces the idea of short term and long term requirements, based on the time frame in which they must be met. Short term requirements may be allocated in an exclusive or non-exclusive manner, while long term requirements are always non-exclusive. In this context, exclusive requirements are always allocated (even if part of the allocated resources is idle), while non-exclusive

requirements are only allocated when necessary. The system prioritizes virtual networks that have used the lowest portion of their requirements, and an adaptive control scheme is used in order to improve the probability that long term requirements, if needed, will be met. The presented evaluation shows the improvement of this system over the original [26] in terms of guaranteeing that the demands of each virtual network will be met, as well as reducing resource load on the physical substrate.

5.6.2 Virtual network resilience

Even with proper physical resource isolation, maintaining availability remains a challenge in virtualized networks. The virtualization layer must be resilient, maintaining its performance and mitigating attacks in order to sustain its availability. Some of the publications described next approach the issue of virtual network resilience from the point of view of fault tolerance. Nonetheless, we emphasize that the solutions described in these publications may also be used as a response to attacks that cause the failure or degradation of physical devices or links.

The solution presented by Yeow *et al.* [32] aims to provide network infrastructures that are resilient to physical router failures. This objective is achieved through the use of backups (*i.e.,* redundant routers and links). However, redundant resources remain idle, reducing the utilization of the physical substrate. To minimize this problem, the authors propose a scheme that dynamically creates and manages shared backup resources. This mechanism minimizes the number of necessary backup instances needed to achieve a certain level of reliability. While backup resources are shared, each physical router is restricted to hosting a maximum number of backup instances in order not to sacrifice reliability. The connectivity between each virtual router and its neighbors is preserved in all of its backups, both in terms of number of links and bandwidth reservations.

The illustration on the left side of Figure 6 shows a simple representation of how backup nodes (represented as circles) may be shared among different virtual networks. For example, the two backup nodes at the right side of this figure are shared between Virtual Network 1 and Virtual Network 3, regardless of whether they belong to one or the other. The right side of Figure 6, in turn, depicts in greater detail how backups are allocated to virtual routers. A virtual router C_1 has virtual routers B_1 and B_2 as its backups. Since C_1 has a virtual link connecting it to another router, N_1, a virtual link with the same bandwidth reservation (depicted as 1 in the figure) is also established between each backup node and N_1 in order to preserve the connectivity of the original router.

Meixner *et al.* [35] devise a probabilistic model for providing virtual networks that are resilient to physical disasters. Disasters are characterized by the occurrence of multiple failures in the physical network, as well as the possibility of correlated cascading failures during attempts to recover network resources. The virtual link mapping strategy guarantees that the failure of a single physical link will not disconnect any virtual network, and aims at minimizing virtual network disconnection in the event of a disaster (*i.e.*, simultaneous failure of multiple links). Additionally, excess processing capacity in the physical network is used to create a backup router for each virtual network, which reduces disconnection in the event of disasters and provides additional processing capacity for the recovery phase. When attempting to recover virtual network resources, the model analyzes all possible virtual router replacements in an effort to replace affected virtual routers in a way that ensures the virtual network will not be disconnected by any post-disaster failures.

The system presented by Zhang *et al.* [27] uses redundant virtual networks in order to provide reliable live streaming services. It is able to detect path failures and traffic congestion, dynamically redirecting data flows.

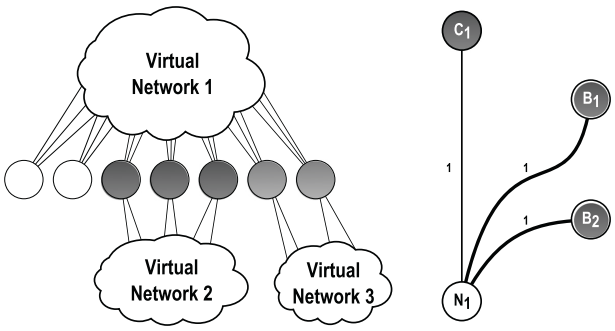


Figure 6 Examples of sharing and mapping of backup instances, used by Yeow *et al.* [32] to provide resilient virtual networks.

Initially, the data flow is distributed equally through available virtual networks. Figure 7 depicts the distribution of a data flow through virtual networks, using multiple paths between a server and a client. Gradually, the number of packets routed through each virtual network is adapted according to its relative bandwidth capacity. Additionally, an active probing mechanism is used to detect failures in the physical network or routing problems (changes in routing tables, for example, may have a significant impact in live streaming applications). If an issue is detected, the system is able to redirect data flows away from problematic networks and redistribute it among the remaining ones. Experiments performed by the authors demonstrate advantages in using multiple networks instead of a single one, with increasing gains when using up to four virtual networks. Additionally, the authors claim that the bandwidth cost of the probing mechanism is negligible.

Chen *et al.* [33] propose a virtual network embedding strategy that aims at ensuring survivability. Load balancing is employed in the embedding process in order to balance the bandwidth consumption of substrate links. Moreover, backup links are reserved for each accepted virtual network, but not activated until a failure occurs. Backup links are allocated in physical paths that do not overlap with the path hosting the original link, guaranteeing that a single physical link failure will not simultaneously affect the original virtual link and one or more of its backups. These backup resources may be shared by multiple virtual networks or reconfigured over time in order to improve efficiency.

Zhang *et al.* [34] devise a strategy for computing the availability of Virtual Data Centers (VDCs), as well as an algorithm for reliable VDC embedding. In order to determine VDC availability, the authors consider the availability of individual, heterogeneous components, as well as dependencies among them. The embedding mechanism aims at meeting minimum availability criteria while optimizing resource usage. Virtual devices are divided into replication groups (groups in which any virtual device may serve as a backup if another fails). In order to minimize resource consumption, VDCs are embedded on physical devices with the lowest level of availability that still meets the desired level. In a similar way, a minimum number of backups is assigned to each replication group in order to meet availability requirements.

Unlike the previously described approaches in the area of virtual network resilience, Oliveira *et al.* [7] present a strategy based on "opportunistic resilience", which does not employ backup resources. The bandwidth demand of each virtual link is split over multiple physical paths. As a consequence, physical link failures are less likely to cause a virtual link disconnection (an affected virtual link will remain operational, albeit with less capacity). Additionally, when link failures occur, a reactive strategy is used in order to reallocate the lost capacity over unaffected paths, attempting to fully restore the bandwidth of degraded virtual links.

Distributed Denial of Service (DDoS) attacks are a common threat to the availability of network services. The system proposed by Yu and Zhou [6] aims to detect such attacks on community networks (federated virtual networks that belong to cooperating entities). The devised solution leverages communication between virtual routers that belong to different entities in this collaborative environment to detect possible attacks at an early stage. Virtual routers located on the edges of the community network monitor traffic passing through them and calculate the entropy of its flows. Traffic surges in any of these flows will cause the entropy to drop, indicating a possible attack. If this occurs, other routers are notified

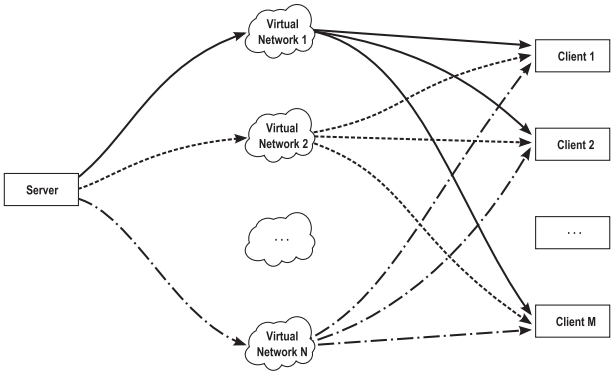


Figure 7 A live streaming data flow is distributed among different virtual networks, a mechanism used by Zhang *et al.* [27].

and instructed to calculate the entropy rate of this suspected flow. Calculated values are compared, and if they are similar, a DDoS attack is confirmed.

6 Discussion

A number of insights can be obtained from the extensive investigation of the state of the art reported in this paper. First, it is possible to observe that the publications in the area are not equally distributed between the main security categories. Tables 3 and 4 show, respectively, the security threats and security countermeasures approached in these publications. In both tables, publications have been grouped together according to the security elements they approach, whenever possible. It is noticeable that disruption and availability – a security threat and a countermeasure that are directly correlated – are approached in the majority of these publications. This is likely due to the high prevalence of attacks aiming at causing disruption. These attacks are relatively simple but can be highly devastating, especially in an environment that makes heavy use of shared resources (as a single physical failure may disrupt several virtual networks). Disclosure and confidentiality follow closely behind, being present in a similar number of publications as disruption/availability. Once again, this is linked to physical resource sharing. Similarly to disruption attacks, such sharing means that a single well-placed sniffer may be able to acquire sensitive information from multiple virtual networks at once. Moreover, there are also privacy concerns between infrastructure providers and virtual network requesters (as the former may have access to data that the latter considers confidential).

Second, only a small number of publications approach more than one threat or countermeasure simultaneously. No single publication has dealt with threats in more than two of the four categories, or presented solutions that provide more than four security countermeasures, out of a total of six. Additionally, one security countermeasure in particular – nonrepudiation – was not approached by any of the publications. The combination of authentication and integrity, which exists in some publications, can be considered as the basis for the provision

Table 3 Security threats mentioned in publications in the area of virtual network security

Publication	Threats			
	DI	DE	DR	US
[4]	×			
[19]	×			
[21]	×			
[22]	×			
[23]	×			
[24]	×			
[10]	×	×		
[20]	×	×		
[5]	×	×		×
[25]		×		
[26]		×	×	
[36]				×
[6]			×	
[7]			×	
[27]			×	
[28]			×	
[29]			×	
[30]			×	
[31]			×	
[32]			×	
[33]			×	
[34]			×	
[35]			×	

From left to right: Disclosure, Deception, Disruption, Usurpation.

Table 4 Security countermeasures provided by publications in the area of virtual network security

Publication	Countermeasures					
	AC	AU	CO	IN	NR	AV
[4]			×			
[20]			×			
[22]			×			
[24]			×			
[19]	×	×	×			
[5]	×	×	×	×		
[26]		×	×	×		×
[31]		×	×	×		×
[10]			×	×		
[25]		×				
[6]						×
[7]						×
[27]						×
[29]						×
[30]						×
[32]						×
[33]						×
[34]						×
[35]						×

From left to right: Access Control, Authentication, Confidentiality, Integrity, Nonrepudiation, Availability.

of nonrepudiation, but this specific countermeasure is not targeted. Nonrepudiation is a highly valuable (albeit challenging) security countermeasure for network virtualization environments, and will be further discussed in Section 7.

Third, we were able to conclude that many of the threats that affect network virtualization environments also affect traditional networks. However, we emphasize that these threats affect traditional and virtual network environments in different ways. In most cases, the effects of these threats are greatly exacerbated by certain characteristics of virtual network environments. Information interception, physical resource overloading, physical resource failure, and software vulnerability exploitation are aggravated by the fact that a number of virtual routers may share a physical router. Therefore, as previously explained, an attack of any of these types targeting a single physical router may affect several virtual networks. Further, it is more difficult to recognize (and, therefore, to perform countermeasures against) identity fraud and replay attacks due to the dynamicity of network virtualization environments, as virtual routers may be freely moved among physical routers and assume different identities. Loss of registry entries and information leakage, as described in the studied literature, are limited to virtual network environments. Moreover, threats related to introspection are also inherent to these types of environments, as this is a (potentially exploitable) feature of virtual machine monitors.

Last, we can observe the employment of different virtualization techniques in some publications. For example, Cabuk *et al.* [5] implemented a prototype of their framework based on a paravirtualization platform, while Huang *et al.* [22] consider an underlying network based on programmable routers. Further, Fernandes and Duarte [26,31] build a hybrid solution that combines paravirtualization with plane separation, a core idea of programmable networks. Although the majority of publications do not target specific network virtualization techniques, we emphasize that different types of platforms have their own sets of benefits, as well as security concerns, which need to be taken into account.

7 Challenges

Despite the existence of a sizable body of work in virtual network security, some challenges remain open. In this section, we summarize some of the main research challenges in this area. We emphasize, however, that these challenges should not be considered exhaustive, but rather as a starting point for further discussions in the area.

One clear opportunity for research in virtual network security is the provisioning of nonrepudiation – which, to the best of our knowledge, has not yet been approached. Nonrepudiation requires providing proof of actions performed by entities on a network, which can be used for holding entities accountable for malicious activity. We deem nonrepudiation an essential security countermeasure for virtual networking environments in order to accurately backtrace attacks – not only to ensure that punitive actions will be taken against the attackers but also to properly contain the attacks themselves. In the event of a DDoS attack, for example, this countermeasure could enable administrators to pinpoint the origins of the attack with a high level of precision – which otherwise tends to be a very difficult task. Moreover, nonrepudiation may even prevent attacks, in the sense that malicious users who are aware that such a mechanism is in use may refrain from carrying out attacks in order to avoid exposing themselves. Provisioning nonrepudiation can be challenging for a number of reasons, such as the complexity of securely storing and handling digital certificates – used for proving that an action was, indeed, performed by a given entity – and the negative impact this has on network performance. Moreover, it is necessary to maintain a desired level of privacy for virtual network requesters as well as end users. Nevertheless, we envision that the importance of this countermeasure will grow steadily as network virtualization becomes increasingly prominent in production environments.

In addition to privacy issues related to nonrepudiation, there are also concerns regarding the privacy of general data stored in virtual routers or sent through virtual networks. Although such data may be protected from being intercepted by other entities, infrastructure providers have physical access to all data stored in virtual networks they are hosting. Although this issue has been approached by some authors, their proposed strategies are often based on strong assumptions, such as the ability to choose which physical entity (out of a number of entities controlling the physical substrate) will host each of its routers – a feature that may not commonly be available in practice.

Another opportunity stems from the multiple levels of heterogeneity present in network infrastructures. As previously mentioned, in addition to the use of heterogeneous hardware devices, it is common for network substrates to be composed of a number of physical networks that belong to different entities. As such, there is a need for uniform methods for requesting, negotiating, and enforcing security requirements across devices that may have incompatible interfaces and entities with potentially conflicting policies.

Last, software platforms used to instantiate virtual networks may not always offer adequate protection against security threats. Moreover, although virtualization technologies are gradually evolving and becoming more mature, both hardware and software are susceptible to vulnerabilities that may be exploited by attackers.

Consequently, research efforts that build on top of network virtualization need to consider these security issues and, most importantly, overhead costs of additional security mechanisms that may be necessary, in order to ensure that they will be suitable for real world environments.

We emphasize, once again, that this is not an exhaustive list of challenges in the area. The essence of network virtualization is based on layers upon layers with increasing levels of abstraction (*e.g.*, the physical substrate, the virtualization layer, virtual networks, and services running on top of them). Consequently, we envision that a number of other challenges may be present in all of these layers – much like the ones listed in this section.

8 Conclusions

Network virtualization enables the subdivision of a single network infrastructure into multiple virtual architectures. The benefits of this technique apply to a wide range of applications, including the creation of virtual testbeds, community networks, and cloud computing infrastructures. Furthermore, network virtualization has been proposed by researchers as the basis for the creation of a new architecture for the Internet, allowing pluralist network environments that support a number of different network protocols simultaneously.

In spite of the benefits provided by network virtualization, there is a series of security issues that need to be considered. Our study revealed a number of security threats, covering the four categories defined by Shirey [16]. The very act of sharing a physical infrastructure among multiple parties is shown to be the source of several of these threats.

This study shows that there have been several efforts to provide security in virtual networks. However, these efforts were not organized in a comprehensible manner. This study provides a systematic overview of the available research results in the field, categorizing work that represents the state of the art and highlighting different approaches for providing security. Additionally, it also evidences imbalances between different sub-areas of security research in network virtualization, which can be used as guidance for future work in this area. Usurpation and access control, for example, are significantly underrepresented in relation to other security countermeasures, and nonrepudiation is not targeted by any publication. Additionally, while a significant body of work exists in the sub-area of availability, only one publication deals with detection and prevention of attacks. Such gaps may represent valuable opportunities for future work.

To summarize, the categorization of security threats and countermeasures presented in this paper simplifies the analysis of which security aspects have not yet been approached and which types of threats need to be mitigated. Furthermore, it makes it easier to identify a number of existing solutions that aim to provide security in virtual networks.

Endnotes

[1]Machine virtualization is available for personal computers, in commonly used operating systems (e.g., Windows, Linux, and Mac OS X).

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All authors read and approved the final manuscript.

Acknowledgements
The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work has been partially supported by FP7/CNPq (Project SecFuNet, FP7-ICT-2011-EU-Brazil), RNP-CTIC (Project ReVir), as well as PRONEM/FAPERGS/CNPq (Project NPRV).

Author details
[1]Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil. [2]Institute of Computing, University of Campinas, Campinas, Brazil.

Received: 29 August 2014 Accepted: 8 December 2014
Published online: 27 January 2015

References

1. Chowdhury NMMK, Boutaba R (2010) A survey of network virtualization. Comput Netw 54(5):862–876
2. Fernandes N, Moreira MD, Moraes I, Ferraz L, Couto R, Carvalho HT, Campista M, Costa LK, Duarte OB (2011) Virtual networks: isolation, performance, and trends. Ann Telecommun 66(5–6):339–355
3. Anderson T, Peterson L, Shenker S, Turner J (2005) Overcoming the internet impasse through virtualization. Computer 38(4):34–41
4. Bays LR, Oliveira RR, Buriol LS, Barcellos MP, Gaspary LP (2014) A heuristic-based algorithm for privacy-oriented virtual network embedding. In: IEEE/IFIP Network Operations and Management Symposium (NOMS). IEEE, Krakow, Poland
5. Cabuk S, Dalton CI, Ramasamy H, Schunter M (2007) Towards automated provisioning of secure virtualized networks. In: ACM Conference on Computer and Communications Security. New York, USA
6. Yu S, Zhou W (2008) Entropy-based collaborative detection of ddos attacks on community networks. In: IEEE International Conference on Pervasive Computing and Communications. IEEE Computer Society, Washington, DC, USA
7. Oliveira RR, Marcon DS, Bays LR, Neves MC, Buriol LS, Gaspary LP, Barcellos MP (2013) No more backups: Toward efficient embedding of survivable virtual networks. In: IEEE International Conference on Communications. IEEE, Budapest, Hungary
8. LAN/MAN Standards Committee (2006) IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks. IEEE Std 802.1Q-2005 (incorporates IEEE Std 802.1Q1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, and IEEE Std 802.1s-2002). http://www.ieee802.org/1/pages/802.1Q-2005.html
9. Rosen E, Cisco Systems I, Rekhter Y, Juniper Networks I (2006) RFC 4364: BGP/MPLS IP Virtual Private Networks (VPNs). http://www.ietf.org/rfc/rfc4364.txt
10. van Cleeff A, Pieters W, Wieringa RJ (2009) Security implications of virtualization: A literature study. In: International Conference on Computational Science and Engineering. IEEE Computer Society, Washington, DC, USA
11. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J (2008) Openflow: enabling innovation in campus networks. SIGCOMM Comput Commun Rev 38:69–74
12. Bari MF, Boutaba R, Esteves R, Granville LZ, Podlesny M, Rabbani MG, Zhang Q, Zhani MF (2013) Data center network virtualization: A survey. Communications Surveys Tutorials, IEEE 15:909–928

13. Zhou M, Zhang R, Xie W, Qian W, Zhou A (2010) Security and privacy in cloud computing: A survey. In: Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference On. IEEE, Beijing, China
14. Hashizume K, Rosado DG, Fernández-Medina E, Fernandez EB (2013) An analysis of security issues for cloud computing. J Internet Serv Appl 4:1–13
15. Scott-Hayward S, O'Callaghan G, Sezer S (2013) Sdn security: A survey. In: Future Networks and Services (SDN4FNS), 2013 IEEE SDN For. IEEE, Trento, Italy
16. Shirey R (2000) RFC 2828: Internet Security Glossary. http://www.ietf.org/rfc/rfc2828.txt
17. Stallings W (2006) Cryptography and Network Security: Principles and Practice. Pearson/Prentice Hall, Upper Saddle River, New Jersey, USA
18. Cavalcanti E, Assis L, Gaudencio M, Cirne W, Brasileiro F (2006) Sandboxing for a free-to-join grid with support for secure site-wide storage area. In: International Workshop on Virtualization Technology in Distributed Computing. IEEE Computer Society, Washington, USA
19. Wolinsky DI, Agrawal A, Boykin PO, Davis JR, Ganguly A, Paramygin V, Sheng YP, Figueiredo RJ (2006) On the design of virtual machine sandboxes for distributed computing in wide-area overlays of virtual workstations. In: International Workshop on Virtualization Technology in Distributed Computing. IEEE Computer Society, Washington, DC, USA
20. Wu H, Ding Y, Winer C, Yao L (2010) Network security for virtual machine in cloud computing. In: Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference On. IEEE, Seoul, South Korea
21. Cui Q, Shi W, Wang Y (2009) Design and implementation of a network supporting environment for virtual experimental platforms. In: WRI International Conference on Communications and Mobile Computing. IEEE Computer Society, Washington, DC, USA
22. Huang D, Ata S, Medhi D (2010) Establishing secure virtual trust routing and provisioning domains for future internet. In: IEEE Conference on Global Telecommunications, Miami, USA
23. Pignolet Y-A, Schmid S, Tredan G (2013) Adversarial vnet embeddings: A threat for isps?. In: IEEE INFOCOM. IEEE, Turin, Italy
24. Fukushima M, Sugiyama K, Hasegawa T, Hasegawa T, Nakao A (2013) Minimum disclosure routing for network virtualization and its experimental evaluation. IEEE/ACM Trans Netw PP(99):1839–1851
25. Chowdhury NMMK, Zaheer F-E, Boutaba R (2009) imark: an identity management framework for network virtualization environment. In: IFIP/IEEE International Symposium on Integrated Network Management. IEEE Press, Piscataway, USA
26. Fernandes NC, Duarte OCMB (2011) Xnetmon: A network monitor for securing virtual networks. In: IEEE International Conference on Communications. IEEE, Kyoto, Japan
27. Zhang Y, Gao L, Wang C (2009) Multinet: multiple virtual networks for a reliable live streaming service. In: IEEE Conference on Global Telecommunications. IEEE Press, Piscataway, USA
28. Marquezan CC, Granville LZ, Nunzi G, Brunner M (2010) Distributed autonomic resource management for network virtualization. In: IEEE/IFIP Network Operations and Management Symposium, Osaka, Japan
29. Wu Q, Shanbhag S, Wolf T (2010) Fair multithreading on packet processors for scalable network virtualization. In: ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ACM, New York, USA
30. Kokku R, Mahindra R, Zhang H, Rangarajan S (2010) Nvs: a virtualization substrate for wimax networks. In: International Conference on Mobile Computing and Networking. ACM, New York, USA
31. Fernandes NC, Duarte OCMB (2011) Provendo isolamento e qualidade de serviço em redes virtuais. In: Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, Campo Grande, Brazil. (in Portuguese)
32. Yeow W-L, Westphal C, Kozat UC (2011) Designing and embedding reliable virtual infrastructures. SIGCOMM Comput Commun Rev 41(2):57–64
33. Chen Q, Wan Y, Qiu X, Li W, Xiao A (2014) A survivable virtual network embedding scheme based on load balancing and reconfiguration. In: IEEE Network Operations and Management Symposium. IEEE, Krakow, Poland
34. Zhang Q, Zhani MF, Jabri M, Boutaba R (2014) Venice: Reliable virtual data center embedding in clouds. In: IEEE INFOCOM. IEEE, Toronto, Canada
35. Meixner CC, Dikbiyik F, Tornatore M, Chuah C, Mukherjee B (2013) Disaster-resilient virtual-network mapping and adaptation in optical networks. In: International Conference on Optical Network Design and Modeling
36. Roschke S, Cheng F, Meinel C (2009) Intrusion detection in the cloud. In: IEEE International Conference on Dependable, Autonomic and Secure Computing. IEEE Computer Society, Washington, DC, USA
37. Common Vulnerabilities and Exposures (2012) CVE-2012-1516. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1516
38. Common Vulnerabilities and Exposures (2012) CVE-2012-1517. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1517
39. Common Vulnerabilities and Exposures (2012) CVE-2012-2449. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2449
40. Common Vulnerabilities and Exposures (2012) CVE-2012-2450. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2450
41. Alkmim GP, Batista DM, Fonseca NLS (2013) Mapping virtual networks onto substrate networks. J Internet Serv Appl 3(4):1–15

APPENDIX B – PUBLISHED PAPER – NOMS, 2016

- Title: Virtual Network Embedding in Software-Defined Networks
- Conference: IEEE/IFIP Network Operations and Management Symposium
- Date: April, 2016
- DOI: <https://doi.org/10.1109/NOMS.2016.7502791>

Virtual Network Embedding in Software-Defined Networks

Leonardo Richter Bays, Luciano Paschoal Gaspary
Institute of Informatics – Federal University of Rio Grande do Sul (UFRGS)
{lrbays,paschoal}@inf.ufrgs.br
Reaz Ahmed, Raouf Boutaba
David R. Cheriton School of Computer Science – University of Waterloo
{r5ahmed,rboutaba}@uwaterloo.ca

***Abstract*—Research on network virtualization has been active for a number of years, during which a number of virtual network embedding (VNE) approaches have been proposed. These approaches, however, neglect important operational requirements imposed by the underlying virtualization platforms. In the case of SDN/OpenFlow-based virtualization, a crucial example of an operational requirement is the availability of enough memory space for storing flow rules in OpenFlow devices. In this paper, we advocate that VNE must be performed with some knowledge of the underlying physical networks, otherwise the deployment may suffer from unpredictable or even unsatisfactory performance. Considering SDN/OpenFlow-based physical networks as an important virtualization scenario, we propose an approach based on VNE and OpenFlow coordination for proper deployment of virtual networks (VNs). The proposed approach unfolds in the following main contributions: *(i)* a virtual infrastructure abstraction that allows a service provider to represent the details of his/her VN requirements in a comprehensive manner; *(ii)* a privacy-aware compiler that is able to preprocess this detailed VN request in order to obfuscate sensitive information and derive computable operational requirements; and *(iii)* a model for embedding requested VNs ensuring their feasibility at the physical level. The results obtained through our evaluation demonstrate that taking such operational requirements into account, as well as accurately assessing them, is of paramount importance to ensure the correct behavior of VNs hosted on top of the virtualization platform.**

I. Introduction

Research on network virtualization has been active for a number of years. During this period, several approaches for embedding VNs on top of physical infrastructures have been proposed [1]–[6]. Most approaches consider a similar set of VN requirements, such as CPU, memory, and bandwidth guarantees, as well as location constraints. Some approaches take into account additional aspects such as virtual router image transfer and instantiation overheads, network survivability, or communication security. In contrast, relevant operational requirements related to the instantiation of VNs on different virtualization platforms are neglected. This simplification enables the streamlining of the optimization models and heuristics used in these approaches. Moreover, it renders them generic enough to be applied to a number of different scenarios. However, by not taking into account operational requirements of the underlying virtualization platforms, the mappings produced by these VNE approaches may *(a)* not be feasible in practice, *(b)* be unable to properly fulfill SLA requirements, or *(c)* fail to use infrastructure resources in an efficient manner. In this work, we focus on ensuring the feasibility of VNE mappings on a multi-tenant, SDN/OpenFlow-based network environment so as not to put at risk satisfactory performance and/or network predictability of embedded VNs.

Software-Defined Networking (SDN) offers a promising platform for network virtualization. In addition to slicing physical resources among customers [7]–[12], SDN-based environments provide abstractions that allow different virtualization functionality, including the instantiation of arbitrary virtual topologies [8]–[12] and the use of overlapping address spaces [10]–[12]. In the case of SDN/OpenFlow-based virtualization, a crucial example of an operational requirement that may render VNE-provided mappings inadequate in real environments is the unavailability of enough memory space for storing flow rules in OpenFlow devices. If this critical issue is not taken into account by the VNE algorithm, OpenFlow devices may not be able to accommodate all flow rules required by the virtual routers assigned to them. As a consequence, these devices would need to frequently contact the controller in order to handle incoming packets. High rates of controller intervention, in turn, could hinder network performance predictability and potentially render the multi-tenant environment unstable.

Figure 1 depicts an example of mappings that would be considered valid by a standard VNE approach but could ultimately lead to performance issues in practice. In this example, three VN requests (VN1–3) are embedded on top of a physical network. In this example, physical routers support up to 4,000 flow rules each, while routers in each VN request require either 2,000 or 3,000 flow rules to be installed on the physical routers hosting them. Moreover, a number of flow rules must also be installed on "auxiliary routers" – *i.e.*, routers that are not part of the VN request directly, but are necessary in order to create physical paths to host virtual links between such routers. As one can observe, the computed mappings exceed the capacity of some of the physical routers – namely, PR2 (which is hosting virtual routers B and D), PR4 (hosting virtual router F and an auxiliary router in the path between virtual routers B and C), and PR5 (hosting virtual routers E and H).

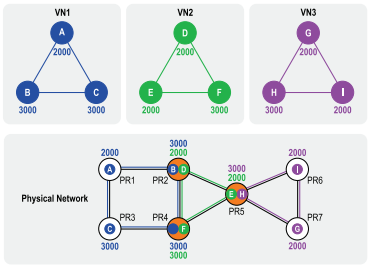


Fig. 1. Example of mappings generated by a standard VNE approach that exceed the flow table capacity of physical devices.

The example illustrated above not only underscores that VNE must be performed with some knowledge of the underlying physical networks but also sheds light to the importance

978-1-5090-0223-8/16/$31.00 ©2016 IEEE

of going further in terms of expressing what a VN needs from the physical network. Applications running on top of VNs have distinct traffic patterns and are often subject to different policies or network functions (*e.g.*, load balancing, access control, or deep packet inspection). The specification of the expected behavior of a VN, translated into an estimate of flow table usage and communicated to the infrastructure provider, would potentially lead to both a more accurate orchestration of VN embeddings and, ultimately, an overall better quality of service.

In this paper, we propose a VNE approach that is aware of operational requirements related to the instantiation of VNs on top of an SDN/OpenFlow environment. The central idea of the proposed approach is the specification, by VN requesters, of VN requests enriched with information about how (to be) provisioned networks will be used (*e.g.*, important application flows, network functions/policies to which packets will be subjected to, etc.). These VN specifications are then used to derive operational requirements, still at the customer's end. The resulting specifications – reflecting requesters willingness (or not) to disclose information about the VNs – are sent to the InP, which will ultimately correctly embed the requested VNs favoring incoming requests with well defined operational requirements. We consider a number of pieces of information that, if known in advance by the InP, can lead to improved allocation of network resources and, in turn, to improved network utilization. The main contributions of this paper are threefold: *(i)* an abstraction model for expressing requirements related to internal VN policies and traffic patterns; *(ii)* a strategy for accurately deriving the number of flow rules needed to instantiate VNs based on the aforementioned requirements; and *(iii)* a VNE method that leverages this information in order to correctly and efficiently allocate resources in an SDN/OpenFlow-based virtualization environment.

The remainder of this paper is organized as follows. In Section II we discuss existing VNE approaches. In Section III we introduce our proposed solution and describe its main elements. In Section IV we describe the evaluation we carried out and present and discuss the obtained results. Last, in Section V we present final remarks and perspectives for future work.

II. Related Work

In this section, we discuss previous work in the area of virtual network embedding, focusing on the constraints considered in each approach.

Yu *et al.* [1] present a heuristic-based VNE approach. The algorithm embeds virtual routers and links in separate phases, and prioritizes VNs with largest revenue. This approach takes into account CPU and location constraints for routers, and bandwidth constraints for links.

Chowdhury *et al.* [2] propose two optimization models, one being a relaxed version of the other. Routers and links are embedded in distinct phases; however, improved coordination between these phases is achieved by preselecting router mappings taking into account their location constraints in order to facilitate link mapping. Similarly to the work of Yu *et al.*, CPU, location, and bandwidth requirements are considered by the proposed optimization models. Moreover, link delay is used to determine how far a virtual router may be embedded from its preferred location.

Cheng *et al.* [3] introduce the concept of "node ranking", in which virtual and physical nodes are ranked according to their own capacity and the capacities of their neighbors. Two embedding algorithms are proposed, one mapping routers and links in distinct stages while the other performs both simultaneously. The algorithms take into account CPU and bandwidth constraints but do not include location constraints.

Alkmim *et al.* [4] present two VNE approaches based on optimization models. One employs a traditional Integer Linear Programming (ILP) model, while the other employs a relaxation technique in order to reduce running times. The authors focus on constraints related to overheads incurred when transferring and instantiating virtual router software images. As such, in addition to CPU, location, bandwidth, and link delay, the size of virtual router images (and the memory needed to support them), the locations in which they are stored, and the time needed to transfer and instantiate them are also taken into account.

Bays *et al.* [5] propose both an optimization model and a heuristic algorithm for virtual network embedding focusing on privacy. Both approaches take into account throughput capacity and location requirements of routers as well as link bandwidth. Additionally, a number of security related constraints are considered, namely which physical routers are capable of supporting the necessary security protocols, overheads associated with cryptographic operations, and which VNs may not share physical resources.

Last, Demirci *et al.* [6] focus on embedding VNs on top of an SDN substrate. More specifically, the issue of controller placement is tackled in addition to virtual router and link mapping. The authors devise two different embedding strategies. The first one aims at balancing the load on physical elements, while the other aims at minimizing communication delay between virtual routers and controllers. The authors consider bandwidth capacity constraints, in addition to controller location requirements. Embedding is performed in an offline manner, assuming all requests are known in advance.

As highlighted in this section, previous work in the area of VNE does not take into account operational requirements related to the instantiation of VNs on top of SDN/OpenFlow substrates. Although the work of Demirci *et al.* [6] is more closely aligned to the approach proposed in this paper, the authors do not take into account hard capacity constraints of SDN routers, only attempting to minimize overall resource usage. As previously explained, this may lead these approaches to generate mappings that are ultimately impossible to instantiate in practice. Moreover, we are not aware of previous attempts to enable customers to represent the needs of their VNs in a level of detail they are comfortable with while simultaneously allowing infrastructure providers to leverage a "distilled" version of this information to ensure no operational constraints are broken.

III. Proposed Solution

Next, we present our proposed solution for coordinating VNE and SDN infrastructures. First, we briefly explain the characteristics of SDN environments considered in this paper and provide an overview of our proposal. Right after, we detail each of its main components.

A. Multi-tenant Infrastructure Model and Network Virtualization Paradigm Considered

The approach proposed in this paper targets an SDN/OpenFlow-based network environment in which a number of customers (service providers) request and, if possible, are granted virtual infrastructures. More specifically, we focus on correctly provisioning the VNs that interconnect the elements of these infrastructures.

An example of a multi-tenant network virtualization environment is depicted in Figure 2. VN requests are received by the infrastructure provider and processed by a VN embedder. Accepted VN requests are ultimately instantiated on top of the physical infrastructure through a network hypervisor, following the mappings produced by the VN embedder. The controller associated with each VN (represented as black boxes

with the letter C in the figure), in turn, is hosted within a virtual machine, and communication channels are established between it and the network hypervisor. The hypervisor intermediates flow rule instantiation and monitoring actions sent by VN controllers in order to enforce properties such as isolation at the physical level.

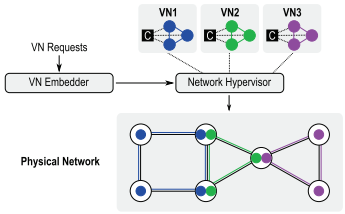


Fig. 2. Multi-tenant OpenFlow/SDN-based network virtualization model considered in this paper.

In recent years, a number of architectures for enabling network virtualization on top of OpenFlow-based SDNs have been proposed. These architectures have evolved from simpler hardware-based flow table slicing to more complex flow level virtualization. While the former achieves better performance, the latter allows a significantly higher degree of flexibility. As an example of such flexibility, recent flow level virtualization approaches [10]–[12] enable tenants to request arbitrary topologies that are not restricted to a subset of the physical network (topology virtualization) as well as to use overlapping address spaces (address space virtualization).

In this work, we consider an SDN virtualization platform capable of providing the aforementioned features, such as OpenVirteX [12]. However, it is worth mentioning that our VNE/SDN coordination approach may be adapted to interface with other virtualization platforms, even ones that do not follow the flow-level virtualization model. Moreover, we focus on VN embedding (*i.e.*, not including host embedding), assuming end hosts are located on the premises of the customer.

B. Overview of our Proposed VNE and SDN Coordination Approach

As previously explained, VN mappings generated by standard VNE algorithms may violate operational requirements when an infrastructure provider attempts to instantiate the requested VNs on a real physical substrate. In order for the VNE algorithm to take such requirements into account, they would have to be part of the VN request provided by the customer. However, we believe it is unreasonable to expect customers to be aware of operational requirements that affect the environment on a physical level. Moreover, customers may be averse to disclosing too much information regarding the internal behavior of their network. Therefore, the main goals of our approach are to: *(i)* allow the customer to represent the needs of his/her VN in a detailed manner; *(ii)* preprocess this detailed representation, removing sensitive information and deriving data regarding the operational requirements associated with this particular request; and *(iii)* embed the requested VNs ensuring both feasibility and adequate performance by making use of this "distilled" information.

Figure 3 depicts the components of our proposed approach. The customer first creates a Tenant Infrastructure Graph (TIG), which represents not only virtual routers and links (and their capacity requirements) but also elements (such as hosts and end users) connected to the network, the traffic patterns among them, and the network functions that will be applied to each traffic flow. As some of this information may be considered sensitive by the customer, the TIG is preprocessed by a Privacy-aware Compiler running on the customer's premises. This Privacy-aware Compiler allows customers to only reveal as much information about their networks as they want, while still generating an enhanced VN request that aids the VNE process in order to ensure feasible, high-quality VN deployments. The preprocessed request is then sent to the InP, which makes use of our SDN/OpenFlow-aware Embedder in order to properly embed and deploy (although the latter is out of the scope of this paper) the customer's network. The main elements of our proposal – Tenant Infrastructure Graphs, the Privacy-aware Compiler, and the SDN/OpenFlow-aware Embedder – will be further explained in the following subsections.

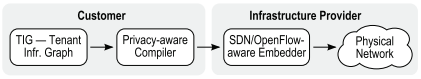


Fig. 3. Overview of our proposed approach, depicting its main elements and the information flow between them.

C. Specification of Infrastructure Resources

We now proceed to a detailed explanation of the process that is carried out on the customer's end in order to request a VN.

1) Tenant Infrastructure Graph (TIG) – A Detailed Abstraction of a Virtual Network and its Communication Patterns: The TIG enables customers to represent the needs of their requested VNs with a high level of detail. In addition to the information contained in a standard VN request (*e.g.*, network topology and capacity and location requirements), a TIG also represents: *(i)* elements such as end hosts (represented as network prefixes or individual addresses) connected to the network; *(ii)* the communication patterns among such elements; and (iii) network functions/policies each communication pattern must be subjected to.

Figure 4 depicts an example of a TIG. Each cloud represents a group of application servers executing a common task (*e.g.*, within the same tier). Globes represent groups of external users (*e.g.*, network administrators or end users accessing applications running on the customer's premises through the to be deployed VN). Each of these elements has some information associated to it – namely, the number of instances of each group of application instances and the number of network prefixes of each group of external users. Last, circles in the graph represent virtual routers, and colored edges represent communication patterns (*i.e.*, traffic flows) among network elements.

Each group of edges represented with the same color and style in Figure 4 denotes a distinct traffic flow. As an example, the solid edges represented in blue interconnect end users to externally accessible applications running on the customer's premises (*e.g.*, the front-end of a two-tier web application), while the dotted green edges interconnect the front-end to the application database back-end. Dashed red edges interconnect databases running in different locations for synchronization/replication purposes, while the dashed and dotted yellow pattern provides an administrator access to all applications.

In addition to forwarding packets, routers may need to perform other functions specific to each traffic flow. Some of these functions – Load Balancing (LB), Quality of Service (QoS), and Access Control (AC) – are represented in Figure 4. In order to discriminate between different network flows

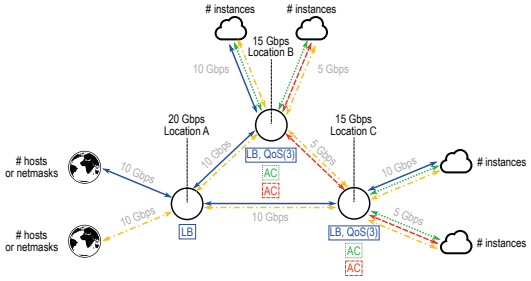


Fig. 4. Tenant Infrastructure Graph representing elements connected to a VN and the communication patterns among them.

and apply the appropriate functions to each, routers use a number of packet header fields (or combinations of fields[1]). In order to apply load balancing, for example, both the source and destination of a packet should be taken into account. For the purpose of QoS, in turn, the Differentiated Services Code Point (DSCP) header field may be used. The TIG represents these (combinations of) fields as sets of Traffic Discriminators (TDs). Moreover, each TD contains a number of entries – *i.e.*, the number of different values a given (combination of) header field(s) may be set to. In the aforementioned QoS example, the DSCP field may be set to a value between 0 and 63. Therefore, the number of entries in a traffic discriminator that uses this field may be anywhere between 2 (if only two different QoS classes are used) and 64 (if all possible classes are used).

Through the information represented in this graph, it is possible to accurately derive the number of flow rules each router in a VN will need in order to ensure its correct and optimal operation. Moreover, as shown in Figure 4, standard VNE constraints (router throughput, link bandwidth, and location requirements) are also represented in a TIG.

2) Privacy-aware Compiler: While a TIG enables the representation of operational constraints associated with the instantiation of each VN on the physical infrastructure of an InP, it also exposes information that the customer may consider sensitive. Therefore, TIGs are expected to be preprocessed by a Privacy-aware Compiler on the customer's end before being sent to an InP.

As previously mentioned, the TIG represents distinct communication patterns within elements of the requested VN. A number of flow rules will ultimately need to be installed on each router in order to ensure the correct operation of each traffic flow. The number of necessary flow rules depends on a number of pieces of information, namely: *(i)* the number of network applications and/or network prefixes of external users associated with each flow; *(ii)* the number of traffic discriminators associated with each router for handling each network flow; and *(iii)* the number of entries of each traffic discriminator.

The left side of Figure 5 shows an example TIG populated with numerical values. The communication pattern represented in solid blue interconnects a number of users (comprising 100 network masks) to a number of applications through the customer's VN hosted on the physical infrastructure of the InP (10 app instances connected to virtual router b and 10 connected to router c). The dotted green patterns interconnect each group of 10 application instances to a group of 5 database servers. The dashed red pattern, in turn, interconnects both groups of database (server) instances. Last, the dashed and dotted yellow pattern interconnects all network services to a specific external network mask (used by a network administrator to manage all network services).

[1]The source or destination of a packet, for example, may be composed of a combination of the IP address, MAC address, network protocol, and port fields.

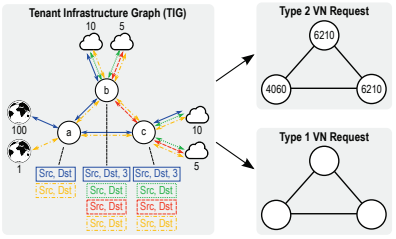


Fig. 5. Possible outputs of the Privacy-aware Compiler for a given TIG.

If no network functions need to be applied to a particular communication pattern, its flow table requirements are calculated by adding up the total number of source-destination pairs (including all applications and network masks) that are part of this traffic. The pattern represented in dashed and dotted yellow on the TIG shown in Figure 5 interconnects a single external user (or network mask) to all (30) applications running within the network, adding up to a total of 60 source-destination pairs (considering both directions of each possible communication flow). Therefore, this flow requires a total of 60 rules on each router it traverses (a, b, and c). This is also the case for the flow represented in solid blue when traversing router a. This flow interconnects 100 external network prefixes to 20 network applications, which – accounting for all possible combinations in both directions – adds up to a total of 4,000 source-destination pairs (and, therefore, 4,000 rules to be installed in a). If additional traffic discriminators are used, they enter the calculation as multiplying factors – the number of flow rules is multiplied by the number of entries in each discriminator. As an example, if the DSCP field is used as a discriminator with 3 possible QoS values (*i.e.*, 3 different traffic classes), the number of flow rules is multiplied by 3. In the example shown in Figure 5, packets that belong to the solid blue communication pattern traversing routers b and c are subjected to this traffic discriminator. Therefore, the total number of source-destination pairs (2,000) connected (directly or indirectly) to each of these routers is multiplied by the number of entries in the respective traffic discriminator (3), adding up to a total of 6,000 flow rules to be installed on routers b and c.

After being processed, the TIG is compiled into a VN request which will be shared with the InP. This request may contain more or less information according to what the customer is willing to reveal. The right side of Figure 5 shows the two different types of requests we consider. A "type 1" request is equivalent to a standard VN request. A "type 2" request, in contrast, includes the accurate number of flow rules required by each router. While not represented in this figure, standard VN requirements – namely, the throughput capacity of routers, bandwidth capacity of links, and location constraints – are also considered for both types. We envision that a larger gradient of VN request types could be considered. As an example, an intermediate level between our "type 1" and "type 2" requests could contain estimates for flow table requirements rather than exact values. We intend to further explore this aspect in future work.

D. SDN/OpenFlow-aware Embedder

The SDN/OpenFlow-aware Embedder is run by the InP, receiving VN requests that have been preprocessed by the Privacy-aware Compiler and embedding them on a physical substrate. It has been modeled as an Integer Linear Program (ILP), and its formulation is presented next. Before presenting our model, we introduce the syntax for our formulation. Capital letters represent sets or variables, and superscripts denote whether a given set or variable refers to physical (P) or virtual (V) entities, or to routers (R) or links (L). Moreover, subscript letters represent indices associated to variables or paths.

Topologies: The topology of each VN request, as well as that of the physical network, are represented as a directed graph $N = (R, L)$. Bidirectional links are represented by pairs of edges in opposite directions. Each virtual router is mapped to a single physical router, while virtual links may be mapped to either a physical link or a substrate path.

Physical and Virtual Capacities: The capacity of physical routers is measured in terms of throughput. The capacity of a physical router i is expressed as T_i^P. Likewise, $T_{r,i}^V$ denotes the throughput required by virtual router i from VN r. Likewise, the bandwidth capacity of a physical link (i, j) is represented as $B_{i,j}^P$, and the bandwidth requirement of a virtual link (k, l) from VN r is represented as $B_{r,k,l}^V$.

Locations: All physical routers are associated with a location identifier – an integer number stored in set S^P. This enables customers to demand some of their virtual routers to be instantiated in specific geographic locations. If a virtual router has a location requirement, it is stored in set S^V.

Flow Table Usage: As previously explained, the flow table requirements of VN requests are calculated by the Privacy-aware Compiler based on a given TIG and added to the generated request. The flow table capacity of a physical router is divided in two – one part (the majority of the available flow table space) will be used for requests with specific flow table requirements (type 2), while the remaining capacity will be used for type 1 requests. The flow table capacity of a physical router i reserved for type 2 requests is represented as F_i^P, while the remaining capacity reserved for type 1 requests is represented as $\mathcal{F}_i^P$. As for virtual routers, those that belong to type 2 VN requests have their flow table requirement represented as $F_{r,j}^V$, while the estimated flow table requirement for type 1 requests is represented as $\mathcal{F}_{r,j}^V$.

Previous Mappings: As VN requests are handled in an online manner, the mappings of previously embedded VNs must be taken into account and preserved while processing new incoming requests. Mappings of previously embedded routers and links are stored in sets $E_{i,r,j}^R$ and $E_{i,j,r,k,l}^L$, respectively.

The variables of the ILP model indicate the optimal placement of routers and links on the substrate.

- $A_{i,r,j}^R \in \{0, 1\}$ – Router allocation, indicates whether virtual router j from VN r is embedded on physical router i.
- $A_{i,j,r,k,l}^L \in \{0, 1\}$ – Link allocation, indicates whether virtual link (k, l) from VN r is embedded on physical link (i, j)

Next, we present the objective function of our SDN/OpenFlow-aware Embedder and its constraint sets (C1–C9). The objective function aims at minimizing overall flow table occupation – *i.e.*, the aggregated number of flow table entries needed to instantiate incoming VN requests. The calculation of flow table usage will be presented in further detail after all constraint sets are listed and explained.

Objective:

$$Minimize \sum_{(i,j)\in L^P, r\in N^V, (k,l)\in L^V} \frac{(\min_{(F_{r,k}^V, F_{r,l}^V)} + \min_{(\mathcal{F}_{r,k}^V, \mathcal{F}_{r,l}^V)}) A_{i,j,r,k,l}^L (1 - A_{i,r,k}^R)}{2} + \sum_{i\in R^P, r\in N^V, k\in R^V} (F_{r,k}^V + \mathcal{F}_{r,k}^V) A_{i,r,k}^R$$

Subject to:

$$\sum_{j\in R^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(T_{r,k}^V, T_{r,l}^V)} A_{i,j,r,k,l}^L (1 - A_{i,r,k}^R)}{2} + \sum_{r\in N^V, k\in R^V} T_{r,k}^V A_{i,r,k}^R \le T_i^P \quad \forall i \in R^P \quad \text{(C1)}$$

$$\sum_{r\in N^V, (k,l)\in L^V} B_{r,k,l}^V A_{i,j,r,k,l}^L \le B_{i,j}^P \quad \forall (i,j) \in L^P \quad \text{(C2)}$$

$$\sum_{j\in R^V} A_{i,r,j}^R \le 1 \quad \forall i \in R^P, r \in N^V \quad \text{(C3)}$$

$$\sum_{j\in R^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(F_{r,k}^V, F_{r,l}^V)} A_{i,j,r,k,l}^L (1 - A_{i,r,k}^R)}{2} + \sum_{r\in N^V, k\in R^V} F_{r,k}^V A_{i,r,k}^R \le F_i^P \quad \forall i \in R^P \quad \text{(C4)}$$

$$\sum_{j\in R^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(\mathcal{F}_{r,k}^V, \mathcal{F}_{r,l}^V)} A_{i,j,r,k,l}^L (1 - A_{i,r,k}^R)}{2} + \sum_{r\in N^V, k\in R^V} \mathcal{F}_{r,k}^V A_{i,r,k}^R \le \mathcal{F}_i^P \quad \forall i \in R^P \quad \text{(C5)}$$

$$\sum_{i\in R^P} A_{i,r,j}^R = 1 \quad \forall r \in N^V, j \in R^V \quad \text{(C6)}$$

$$\sum_{j\in R^P} A_{i,j,r,k,l}^L - \sum_{j\in R^P} A_{j,i,r,k,l}^L = A_{i,r,k}^R - A_{i,r,l}^R \quad \forall r \in N^V, (k,l) \in L^V, i \in R^P \quad \text{(C7)}$$

$$jA_{i,r,k}^R = lA_{i,r,k}^R \quad \forall (i,j) \in S^P, r \in N^V, (k,l) \in S^V \quad \text{(C8)}$$

$$A_{i,r,j}^R = E_{i,r,j}^R \quad \forall (i,r,j) \in E^R \quad \text{(C9)}$$

$$A_{i,j,r,k,l}^L = E_{i,j,r,k,l}^L \quad \forall (i,j,r,k,l) \in E^L \quad \text{(C10)}$$

Constraint sets C1 and C2 ensure, respectively, that the throughput capacity of physical routers and that the bandwidth capacity of physical links is not exceeded. C3 prevents multiple virtual routers from a single VN request from being mapped to the same physical router. C4 and C5 ensure that the flow table capacity of physical routers is not exceeded. Constraint set C4 deals with type 2 requests, while C5 handles type 1 requests. C6 guarantees that all routers in an incoming VN request are mapped to physical routers. C7 ensures that each virtual link is mapped to a physical path whose end-points match the physical routers hosting the end-points of this link. C8 ensures all virtual routers with location requirements are mapped to physical routers in the correct location. Last, constraint sets C9 and C10 preserve the mappings of previously embedded VNs.

For the sake of clarity, our objective function, as well as constraint sets C1, C4, and C5, are shown in non-linear form. However, in practice, they are linearized by replacing the

multiplication $A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})$ with an auxiliary variable $Z_{i,j,r,k,l} \in \{0, 1\}$ and adding constraint sets C11, C12, and C13 – shown below – to the model. Moreover, function $\min(a, b)$ returns the lowest number between a and b and can be defined as $\frac{1}{2}(a + b - |a - b|)$.

$$Z_{i,j,r,k,l} <= A^L_{i,j,r,k,l} \quad \forall (i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C11)}$$

$$Z_{i,j,r,k,l} <= (1 - A^R_{i,r,k}) \quad \forall (i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C12)}$$

$$Z_{i,j,r,k,l} >= (1 - A^R_{i,r,k}) + A^L_{i,j,r,k,l} - 1 \quad \forall (i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C13)}$$

In order to properly account for flow table usage, the objective function must not only consider explicit flow table requirements of virtual routers, but also the flow rules that must be installed on auxiliary routers through which virtual links traverse. The number of rules that must be installed on the auxiliary routers used by a virtual link (k, l) corresponds to the lowest number between the flow table requirements of virtual routers k and l. In the objective function, the first summation refers to the flow table constraints of auxiliary routers, while the second one refers to that of physical routers hosting virtual routers of each network. Constraint sets C1, C4, and C5 employ the same strategy in order to compute the throughput and flow table usage of auxiliary routers.

IV. Evaluation

We now proceed to a performance evaluation of our proposed VNE and SDN coordination approach. Experiments were performed on a machine with an Intel Core i5 4278U CPU, 8 GB of RAM and Operating System Mac OS X 10.10.5. The previously introduced ILP model was implemented and run using the IBM ILOG CPLEX Interactive Optimizer 12.4.

A. Workloads

In order to evaluate our proposal, we developed a simulator that creates virtual topologies according to a series of parameters. Each virtual topology is then converted to a VN request in the format required by our ILP model. The simulator is run for 500 rounds, generating a new request on each one[2]. If accepted, VNs remain embedded for 25 rounds before being removed.

Fixed parameters: In all experiments, physical and virtual topologies are generated with BRITE using the Barabási-Albert model [13]. Generated physical networks contain 100 routers. Physical routers have a throughput capacity of 150 Gbps, while physical links have a bandwidth capacity of 30 Gbps. The flow table of each device is capable of storing up to 16,000 rules. Physical routers are uniformly distributed among 16 geographic locations.

Each generated VN request contains 5 routers. Virtual router throughput and link bandwidth requirements are, respectively, 50 Gbps and 10 Gbps. Each VN has two edge routers with randomly generated location requirements. 50% of the generated VNs are type 1 requests, while the remaining 50% are type 2 requests. Flow table requirements of type 2 requests are set to 3,000 rules per router, while those of type 1 requests (which are not known) are treated in different ways according to the experiment being performed.

Variable parameters: We performed a number of different experiments with variations regarding flow table space reserved for type 2 requests as well as how flow table requirements are considered. The first three experiments – henceforth referred to as "Flow-70/30", "Flow-80/20", and "Flow90/10" – reserve 70%, 80%, and 90% of each physical router flow table space for type 2 requests. The remaining flow table space is reserved to accommodate type 1 requests. As we do not know their requirements, a minimal set of 1,500 rules is reserved for each router from these networks. The idea here is to deliberately allocate little table space to these virtual routers[3].

In the fourth experiment, all flow table space is used to embed type 2 requests, while accepted type 1 requests are supported on a "best effort" manner. More specifically, type 1 requests are embedded as long as their other capacity requirements (throughput and bandwidth) can be fulfilled, with no flow table space guarantees. This experiment is referred to as "Flow-100/0'. In the last experiment, no flow table requirements are considered by the embedding model. This experiment behaves similarly to an environment running a traditional VNE algorithm and is carried out to assess the impact of our proposed approach. This experiment is referred to as "NoFlow".

B. Results

We first analyze the overall acceptance rate in all experiments, shown in Figure 6. The acceptance rates achieved throughout experiments Flow-70/30, Flow-80/20, and Flow-90/10 were, respectively, 70.2%, 64.2%, and 55.8%. As the residual flow table space for type 1 requests decreases, more VNs of this type (which require less resources and represent half of all generated requests) are rejected, leading to lower acceptance rates. Although the overall acceptance rate is lower, this, in turn, favors the acceptance of a higher amount of type 2 requests. This is a desirable outcome for the InP in order to prevent under- or overestimation of resources (as type 2 requests contain precise flow table occupation requirements), potentially leading to improved resource usage (in the case of overestimation) and/or lower rates of controller intervention (if flow table requirements are underestimated). The acceptance rates of individual types of requests and their effect on the rate of controller interventions will be further analyzed in the remainder of this section.

The acceptance rates observed in experiments Flow-100/0 and NoFlow were higher than those of other experiments – 72.6% and 73%, respectively. This is due to the former not reserving any flow table space for type 1 requests and the latter disregarding flow table requirements entirely. While seemingly a positive result at first glance, this is likely to result in severe underestimation of resources needed to adequately support the embedded VNs, potentially leading to high rates of controller intervention. We emphasize that both Flow-100/0 and NoFlow are used as baseline scenarios, *i.e.,* the best results one would achieve in terms of accepted requests at the cost of compromising network predictability and, in extreme cases, its technical feasibility.

Next, Figure 7 depicts the acceptance rate of each type of request in all evaluation scenarios. In experiments Flow-70/30, Flow-80/20, and Flow-90/10, the acceptance rates of type 1 requests were, respectively, 73.7%, 57.7%, and 30.83%. Acceptance rates of type 2 requests observed for the same experiments were of 67.18%, 69.77%, and 82.04%, respectively. These increasing acceptance rates of type 2 requests are a desirable outcome for InPs, as they would likely desire to prioritize the embedding of this type of request. This happens as a result of the fine-tuning of the amount of flow entries reserved for each type of request, leading to an acceptance rate of over 80% for type 2 requests in the most extreme scenario (Flow-90/10). InPs may fine-tune these reservations as desired

[2]On average, each request was optimally mapped in less than 5 seconds.

[3]The amount of flow table space reserved for each router in type 1 requests may be fine-tuned as desired by the InP.

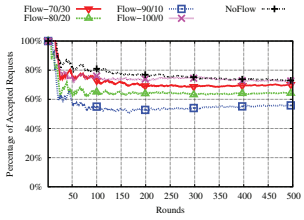


Fig. 6. Overall acceptance rate in all experiments.

in order to allow more or less of each type of request to be embedded and potentially minimize issues caused by type 1 requests (as a result of under- or overestimation of operational requirements). Moreover, all scenarios exhibit variations in acceptance rates within the first 250 rounds. In scenario Flow-70/30, the acceptance rate of type 1 requests is initially higher than that of type 2 requests. In scenarios Flow-80/20 and Flow-90/10, in turn, acceptance rates for both types of requests either increase or decrease during the first 250 rounds. This can be attributed to the fact that the substrate is completely empty at the beginning of the experiment, requiring some time for acceptance rates to stabilize. In all cases, acceptance rates become stable towards the last 250 rounds.

In the remaining experiments (Flow-100/0 and NoFlow), acceptance rates of type 1 requests were, respectively, 73.4% and 70.4%. Acceptance rates of type 2 requests, in turn, were of 72.1% and 75.49%, respectively. As previously explained, the former does not reserve any flow table space for type 1 requests while the latter completely disregards flow table requirements. Therefore, the main causes of rejection are likely related to topological factors or throughput/bandwidth resource scarcity, leading to similar acceptance rates for both types of requests on both experiments.

Last, we analyze the potential impacts of accurately or inaccurately estimating operational requirements. To this end, we consider that VN requests in each scenario have been embedded and deployed, and we calculate the number of flow rules each router would need to have installed in order to support the embedded VNs. We first assume the 1,500 rules reserved for each router of type 1 requests were sufficient. Afterwards, we assume this lower bound was not adequate and that type 1 requests would actually require 3,000 flow rules per router. These scenarios represent extreme cases – assuming either that a minimal set of flow rules was sufficient or that actual requirements exceed this lower bound by a factor of 2. Our goal is to determine in which cases the number of necessary flow rules would exceed the capacity of physical routers. As previously mentioned, if physical devices are not able to accommodate all necessary flow rules, they would need to frequently contact the controller in order to handle incoming packets. This, in turn, would likely degrade the performance of physical devices, hindering the quality of service experienced by customers.

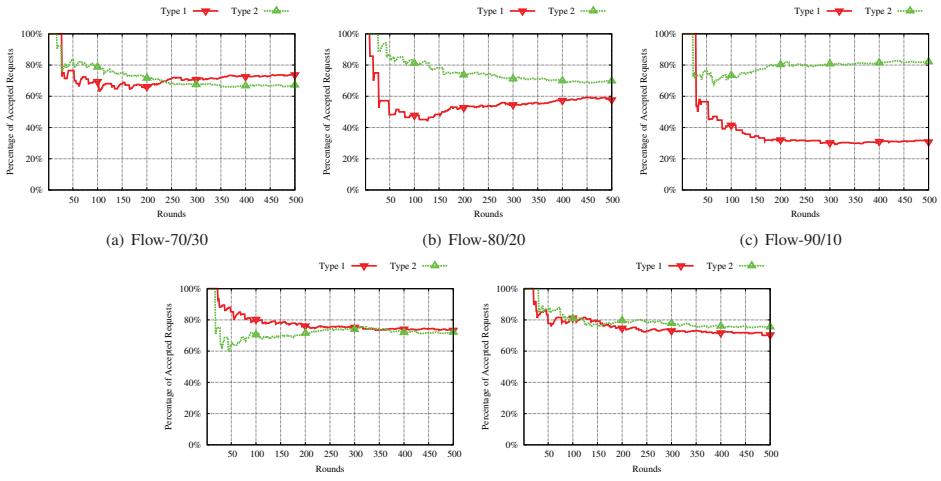
Figure 8 depicts the average number of flow rules that would exceed the capacity of physical routers in each experiment, considering the scenarios just described. Assuming the flow table usage of type 1 requests fits within the minimal provided flow table space, no exceeding rules were observed in scenarios Flow-70/30, Flow-80/20, and Flow-90/10. The ILP model used in these experiments takes into account flow table constraints for both types of requests, ensuring that the capacity of physical devices will not be exceeded as long as the reserved flow table space is sufficient. In experiment Flow-100/0, in turn, the average number of exceeding flow rules was 10,732, all belonging to type 1 requests. This is due to this experiment disregarding flow table requirements for this type of request, embedding them in a "best effort" manner. In experiment NoFlow, which mirrors the behavior of a standard VNE algorithm without flow-related constraints, the average number of exceeding flow rules was significantly higher, with both types of VN requests contributing to flow table saturation. More specifically, an average of 27,593 exceeding rules were observed (2,331 incurred by type 1 requests and 25,262 incurred by type 2 requests). The substantial numbers of exceeding flow rules observed in scenarios Flow-100/0 and NoFlow highlight the importance of considering this operational constraint in the embedding process.

Last, assuming the flow table usage of type 1 requests exceeds the minimal provided flow table space (a likely scenario in practice), all experiments exhibit flow table saturation. In experiments Flow-70/30, Flow-80/20, and Flow-90/10, the average numbers of exceeding flow rules were, respectively, 2,731, 19,783, and 18,130. Although these experiments took into account flow-related constraints, assuming flow table usage exceeds the established lower bounds led to significant saturation – particularly in scenarios Flow-80/20 and Flow-90/10, in which less resources were reserved for this type of request. The highest numbers of exceeding flow rules were observed in experiments Flow-100/0 and NoFlow – which, as previously explained, either do not reserve flow table space for type 1 requests or disregard flow constraints entirely. The average amount of exceeding flow rules observed in experiment Flow-100/0 was of 57,970 – all incurred by type 1 requests. In experiment NoFlow, an average of 22,886 exceeding flow rules were incurred by type 1 requests and 25,262 by type 2 requests, adding up to a total of 48,148. These results evidence that, in addition to the aforementioned importance of considering operational constraints, accurately determining flow table requirements plays a crucial role in ensuring the feasibility of supporting the embedded VNs. Regarding the former, it is important to note that, with the exception of scenario NoFlow, all rules of type 2 requests were properly installed in physical routers. Therefore, these VNs will be able to operate with minimal controller interventions, minimizing potential negative impacts on quality of service (as controller intervention may increase latency by up to twice the round-trip time between the switch and the controller [14]).

V. Conclusions

Although a substantial body of work exists in the area of virtual network embedding, existing approaches do not take into account relevant operational constraints related to the instantiation of VNs on different embedding platforms. In the case of Software-Defined Networking, which offers a promising platform for network virtualization, memory space for storing flow rules is often limited, becoming a crucial operational constraint that may render VNE-provided mappings unsuitable for real environments due to unpredictable or even unsatisfactory VN performance. At the same time, demanding information regarding such operational constraints from customers may be unrealistic as they may be either not aware of how their VN affects the InP's substrate at the physical level or unwilling to share detailed information about the inner working of their VNs.

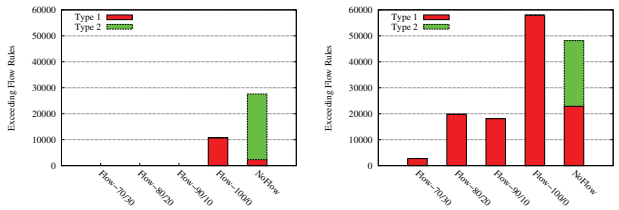
Based on this reasoning, we proposed an abstraction model for expressing requirements related to internal VN policies and traffic patterns. This model – the Tenant Infrastructure Graph – is built in a way that is familiar to customers in a network virtualization environment. Moreover, it is preprocessed on the customer's end by a Privacy-aware Compiler in order to derive information that is valuable to the InP and, at the



(a) Flow-70/30 (b) Flow-80/20 (c) Flow-90/10

(d) Flow-100/0 (e) NoFlow

Fig. 7. Acceptance rate of requests per TIG type in all experiments.



(a) Assuming flow table space allocated for type 1 requests was sufficient.

(b) Assuming flow table space allocated for type 1 requests was insufficient.

Fig. 8. Number of flow rules exceeding the capacity of physical routers.

same time, remove sensitive data that the customer may not be willing to disclose. The output of this compiler is then forwarded to the InP, which can employ the SDN/OpenFlow-aware Embedder to ensure embedded VNs do not break any operational constraints of its network virtualization platform.

Through a comprehensive evaluation, we demonstrated that taking the aforementioned operational constraints into account is of paramount importance to maintain a desired level of quality of service. Neglecting such constraints may render the environment unable to cope with a substantial number of flow rules that are crucial to ensure correct VN behavior. As physical devices become unable to store all necessary flow rules internally, they need to frequently contact the controller in order to route incoming packets, which, in turn, may lead to significant performance degradation for VNs hosted on such devices. In our experiments, assuming the flow table space reserved for requests with unknown requirements was sufficient, the proposed approach was able to eliminate controller intervention due to flow table saturation. Additionally, assuming the actual requirements of such requests exceeded the allocated space by a factor of 2, the number of exceeding flow rules was still reduced by 40.8% on average (compared to a traditional VNE approach). Moreover, the reduction of acceptance rates due to the added constraints was limited to, on average, 9.6%. Our proposed approach enables InPs to accurately assess operational constraints and correctly embed incoming requests without violating these constraints. Moreover, by adjusting the ratio of flow table space dedicated to different types of incoming requests, the InP may choose to which degree requests that include all the necessary information are favored in detriment of requests that do not (and that, therefore, rely on estimation of necessary resources in order to be embedded). Perspectives for future work include: *(i)* taking into account other SDN/OpenFlow-related operational constraints; *(ii)* considering a larger gradient of VN request types with varying amounts of information; and *(iii)* including a negotiation phase between the customer and the InP, providing alternatives for cases in which available resources are not sufficient to embed requested VNs.

ACKNOWLEDGMENTS

This work has been supported by the Brazilian National Council for Scientific and Technological Development (CNPq).

REFERENCES

[1] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, Mar. 2008.

[2] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[3] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 38–47, Apr. 2011.

[4] G. P. Alkmim, D. M. Batista, and N. L. S. Fonseca, "Mapping virtual networks onto substrate networks," *Journal of Internet Services and Applications*, vol. 3, no. 4, pp. 1–15, 2013.

[5] L. R. Bays, R. R. Oliveira, L. Buriol, M. P. Barcellos, and L. Gaspary, "A heuristic-based algorithm for privacy-oriented virtual network embedding," in *IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–8.

[6] M. Demirci and M. Ammar, "Design and analysis of techniques for mapping virtual networks to software-defined network substrates," *Computer Communications*, vol. 45, pp. 1 – 10, 2014.

[7] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Technical Report*, 2009.

[8] Z. Bozakov and P. Papadimitriou, "Autoslice: automated and scalable slicing for software-defined networks," in *ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2012, pp. 3–4.

[9] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, "Vertigo: network virtualization and beyond," in *European Workshop on Software Defined Networking*. IEEE, 2012, pp. 24–29.

[10] E. Salvadori, R. D. Corin, A. Broglio, and M. Gerola, "Generalizing virtual network topologies in openflow-based networks," in *IEEE Global Telecommunications Conference*. IEEE, 2011, pp. 1–6.

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

[12] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: make your virtual sdns programmable," in *Workshop on Hot topics in Software Defined Networking*. ACM, 2014, pp. 25–30.

[13] R. Albert and A.-L. Barabási, "Topology of evolving networks: Local events and universality," *Physical Review Letters*, vol. 85, pp. 5234–5237, Dec 2000.

[14] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM Computer Communication Review*, vol. 41, pp. 254–265, 2011.