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

DANIEL STEFANI MARCON

Achieving Predictable, Guaranteed and Work-Conserving Performance in Datacenter Networks

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

Advisor: Prof. Dr. Marinho Pilla Barcellos

Porto Alegre
February 2017

CIP – CATALOGING-IN-PUBLICATION

Marcon, Daniel Stefani

Achieving Predictable, Guaranteed and Work-Conserving Performance in Datacenter Networks / Daniel Stefani Marcon. – Porto Alegre: PPGC da UFRGS, 2017.

205 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2017. Advisor: Marinho Pilla Barcellos.

1. Datacenter networks. 2. Performance interference. 3. Bandwidth guarantees. 4. Work-conserving sharing. I. Barcellos, Marinho Pilla. II. Título.

"Science is a way of life. Science is a perspective. Science is the process that takes us from confusion to understanding in a manner that's precise, predictive and reliable – a transformation, for those lucky enough to experience it, that is empowering and emotional."

— Brian Greene

ACKNOWLEDGMENTS

First of all, I would like to thank God for always being with me and for all the help given throughout my life.

I would also like to thank my godmother Darcila Stefani and my godfather Fausto Stefani for always supporting me and, especially, for all the help they provided me during the PhD.

Thanks to my advisor, Prof. Marinho Barcellos, for all the guidance on how to conduct a high-level research, on how to correctly write scientific texts and for teaching me how to develop critical thinking. I am also grateful to Prof. Luciano Gaspary for all the contribution he gave to my thesis.

Finally, my sincere gratitude to all my friends and members of the computer networks group of INF/UFRGS who helped me during the PhD, directly or indirectly.

AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus, por estar sempre comigo e por toda a ajuda dada ao longo da minha vida.

Agradeço também à minha dinda-mãe Darcila Stefani e ao meu dindo Fausto Stefani por terem me apoiado durante toda a minha vida e, em especial, pela ajuda em todos os momentos do curso de dutorado.

Agradeço ao meu orientador, Prof. Marinho Barcellos, por toda a orientação durante o doutorado: pelos ensinamentos sobre como realizar uma pesquisa de alto nível, como escrever corretamente textos científicos e como desenvolver o pensamento crítico. Também agradeço ao Prof. Luciano Gaspary pelos conselhos, correções e sugestões.

Por fim, agradeço a todos os meus amigos e membros do grupo de redes do INF/UFRGS que me ajudaram durante o doutorado, direta ou indiretamente.

ABSTRACT

Performance interference has been a well-known problem in datacenter networks (DCNs) and one that remains a constant topic of discussion in the literature. Several measurement studies concluded that throughput achieved by virtual machines (VMs) in current datacenters can vary by a factor of five or more, leading to poor and unpredictable overall application performance. Recent efforts have proposed techniques that present some shortcomings, such as underutilization of resources, significant management overhead or negligence of non-network resources. In this thesis, we introduce three proposals that address performance interference in DCNs: IoNCloud, Predictor and Packer. IoNCloud leverages the key observation that temporal bandwidth demands of cloud applications do not peak at exactly the same time. Therefore, it seeks to provide predictable and guaranteed performance while minimizing network underutilization by (a) grouping applications in virtual networks (VNs) according to their temporal network usage and need of isolation; and (b) allocating these VNs on the cloud substrate. Despite achieving its objective, IoNCloud does not provide work-conserving sharing among VNs, which limits utilization of idle resources. Predictor, an evolution over IoNCloud, dynamically programs the network in Software-Defined Networking (SDN)-based DCNs and uses two novel algorithms to provide network guarantees with work-conserving sharing. Furthermore, Predictor is designed with scalability in mind, taking into consideration the number of entries required in flow tables and flow setup time in DCNs with high turnover and millions of active flows. IoNCloud and Predictor neglect resources other than the network at allocation time. This leads to fragmentation of non-network resources and, consequently, results in less applications being allocated in the infrastructure. Packer, in contrast, aims at providing predictable and guaranteed network performance while minimizing overall multi-resource fragmentation. Extending the observation presented for IoNCloud, the key insight for Packer is that applications have complementary demands across time for multiple resources. To enable multi-resource allocation, we devise (i) a new abstraction for specifying temporal application requirements (called Time-Interleaved Multi-Resource Abstraction – TI-MRA); and (ii) a new allocation strategy. We evaluated IoNCloud, Predictor and Packer, showing their benefits and overheads. In particular, all of them provide predictable and guaranteed network performance; Predictor reduces flow table size in switches and flow setup time; and Packer minimizes multi-resource fragmentation.

Keywords: Datacenter networks. Performance interference. Bandwidth guarantees. Work-conserving sharing.

Atingindo Desempenho Previsível, Garantido e com Conservação de Trabalho em Redes de Datacenter

RESUMO

A interferência de desempenho é um desafio bem conhecido em redes de datacenter (DCNs), permanecendo um tema constante de discussão na literatura. Diversos estudos concluíram que a largura de banda disponível para o envio e recebimento de dados entre máquinas virtuais (VMs) pode variar por um fator superior a cinco, resultando em desempenho baixo e imprevisível para as aplicações. Trabalhos na literatura têm proposto técnicas que resultam em subutilização de recursos, introduzem sobrecarga de gerenciamento ou consideram somente recursos de rede. Nesta tese, são apresentadas três propostas para lidar com a interferência de desempenho em DCNs: IoNCloud, Predictor e Packer. O IoNCloud está baseado na observação que diferentes aplicações não possuem pico de damanda de banda ao mesmo tempo. Portanto, ele busca prover desempenho previsível e garantido enquanto minimiza a subutilização dos recursos de rede. Isso é alcançado por meio (a) do agrupamento de aplicações (de acordo com os seus requisitos temporais de banda) em redes virtuais (VNs); e (b) da alocação dessas VNs no substrato físico. Apesar de alcançar os seus objetivos, ele não provê conservação de trabalho entre VNs, o que limita a utilização de recursos ociosos. Nesse contexto, o Predictor, uma evolução do IoNCloud, programa dinamicamente a rede em DCNs baseadas em redes definidas por software (SDN) e utiliza dois novos algoritmos para prover garantias de desempenho de rede com conservação de trabalho. Além disso, ele foi projetado para ser escalável, considerando o número de regras em tabelas de fluxo e o tempo de instalação das regras para um novo fluxo em DCNs com milhões de fluxos ativos. Apesar dos benefícios, o IoNCloud e o Predictor consideram apenas os recursos de rede no processo de alocação de aplicações na infraestrutura física. Isso leva à fragmentação de outros tipos de recursos e, consequentemente, resulta em um menor número de aplicações sendo alocadas. O Packer, em contraste, busca prover desempenho de rede previsível e garantido e minimizar a fragmentação de diferentes tipos de recursos. Estendendo a observação feita ao IoNCloud, a observação-chave é que as aplicações têm demandas complementares ao longo do tempo para múltiplos recursos. Desse modo, o Packer utiliza (i) uma nova abstração para especificar os requisitos temporais das aplicações, denominada TI-MRA (Time-Interleaved Multi-Resource Abstraction); e (ii) uma nova estratégia de alocação de recursos. As avaliações realizadas mostram os benefícios e as sobrecargas do IoNCloud, do Predictor e do Packer. Em particular, os três esquemas proveem desempenho de rede previsível e garantido; o Predictor reduz o número de regras OpenFlow em switches e o tempo de instalação dessas regras para novos fluxos; e o Packer minimiza a fragmentação de múltiplos tipos de recursos.

Palavras-chave: Redes de datacenter. Interferência de desempenho. Garantias de largura de banda. Compartilhamento com conservação de trabalho.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS AND ACRONYMS

AIB	*Application Information Base*
API	*Application Programming Interface*
COTS	*Commodity Off-The-Shelf*
D-CLAS	*Discretized Coflow-Aware Least-Attained Service*
DCN	*Datacenter Network*
DRF	*Dominant Resource Fairness*
DRM	*Datacenter Resource Manager*
EC2	*Elastic Compute Cloud*
ECMP	*Equal-Cost MultiPath*
FCT	*Flow Completion Time*
FIFO-LM	*FIFO with Limited Multiplexing*
IaaS	*Infrastructure as a Service*
IoNCloud	*Isolation of Networks in the Cloud*
ID	*Identifier*
IP	*Internet Protocol*
IT	*Information Technology*
LAN	*Local Area Network*
MAC	*Media Access Control*
MADD	*Minimum-Allocation-for-Desired-Duration*
MCP	*Minimal-impact Congestion control Protocol*
MPLS	*Multiprotocol Label Switching*
NIB	*Network Information Base*
NUM	*Network Utility Maximization*
ONF	*Open Networking Foundation*
OS	*Operating System*
OXM	*OpenFlow Extensible Match*
PIAC	*Proactive Inter-Application Communication*

PS-L	*Proportional Sharing at Link-level*
PS-N	*Proportional Sharing at Network-level*
PS-P	*Proportional Sharing on Proximate Links*
PSSR	*Port-Switching Based Source Routing*
QoS	*Quality of Service*
RIAC	*Reactive Inter-Application Communication*
RQ	*Research Question*
RTT	*Round Trip Time*
SDN	*Software-Defined Networking*
SEBF	*Smallest-Effective-Bottleneck-First*
SLA	*Service Level Agreement*
SJF	*Shortest Job First*
STP	*Spanning Tree Protocol*
TAG	*Tenant Application Graph*
TCAM	*Ternary Content-Addressable Memory*
TCP	*Transmission Control Protocol*
TI-MRA	*Time-Interleaved Multi-Resource Abstraction*
TIVC	*Time-Interleaved Virtual Cluster*
ToR	*Top-of-Rack*
TVG	*Time-Varying Graph*
UDP	*User Datagram Protocol*
VBP	*Vector Bin Packing*
VC	*Virtual Cluster*
VDC	*Virtual Datacenter*
VLAN	*Virtual Local Area Network*
VLB	*Valiant Load Balancing*
VM	*Virtual Machine*
VN	*Virtual Network*
VOC	*Virtual Oversubscribed Cluster*

CONTENTS

1 INTRODUCTION

Cloud computing has significantly changed the landscape of Information Technology (IT) by offering on-demand provisioning of resources for tenants. In this model, providers increase resource utilization, reduce operational costs and, thus, achieve economies of scale by implementing cloud datacenters as highly multiplexed shared environments, with different applications coexisting on the same set of physical resources (ARMBRUST et al., 2009; ARMBRUST et al., 2010; CRONKITE-RATCLIFF et al., 2016). Tenants, in turn, can scale in and out their applications as they need, with a simple pay-as-you-go pricing model (i.e., users pay according to the time and amount of computing resources consumed). This allows tenants to execute several kinds of services in the cloud platform, both inward computation with bandwidth-intensive requirements and user-facing applications with strict response times (XIE et al., 2012; JANG et al., 2015).

1.1 Motivation

Providers, however, lack practical, efficient and reliable mechanisms to offer bandwidth guarantees for applications (LEE et al., 2014; DOGAR et al., 2014; CHOWDHURY; ZHONG; STOICA, 2014; NAGARAJ et al., 2016). The intra-cloud network is typically oversubscribed and shared in a best-effort manner, relying on TCP to achieve high utilization and scalability. TCP, nonetheless, does not provide robust isolation among flows[1] in the network (GUO et al., 2014; LI; DONG; GODFREY, 2015; CRONKITE-RATCLIFF et al., 2016; HE et al., 2016); in fact, long-lived flows with a large number of packets are privileged over small ones (ABTS; FELDERMAN, 2012), a problem called *performance interference* (SHIEH et al., 2011; GROSVENOR et al., 2015; BALLANI et al., 2013).

Recent studies (GROSVENOR et al., 2015; JUDD, 2015; SCHAD; DITTRICH; QUIANÉ-RUIZ, 2010; WANG; NG, 2010; BALLANI et al., 2011; JANG et al., 2015; SHEA et al., 2014) concluded that, due to performance interference, the network throughput achieved by virtual machines (VMs) can vary by a factor of five or more, resulting in poor and unpredictable network performance (BALLANI et al., 2013). More specifically, when available bandwidth for an application goes below a certain threshold, the total application execution time is elongated (i.e., overall performance is reduced) (XIE et al., 2012). This behavior happens because of three main reasons: (i) applications have mixed communication and computation (CHEN et al., 2014); (ii) applications tend to generate traffic in bursts (JEYAKUMAR et al., 2013); and (iii) the computation often depends on the data received from the network (if communication speed is reduced due to the lack of available bandwidth, the subsequent computation is delayed) (GUO et al., 2013b).

The lack of network guarantees directly impacts both tenants and providers. Tenants do

[1] Flows are characterized by sequences of packets from source to destination hosts.

not receive the allocation of network resources for their requests (which may hinder both throughput-intensive and latency-sensitive applications) and, in some occasions, may only deploy specific enterprise applications in the cloud (POPA et al., 2013). Moreover, costs are unpredictable due to high network variability (applications may take longer to execute (XIE et al., 2012)). Providers, in turn, have the throughput of their datacenters reduced (because of performance interference) (JUDD, 2015; HE et al., 2016), which may hurt revenue (BALLANI et al., 2011).

Related work has proposed techniques to cope with performance interference. Approaches can be divided according to the kind of guarantees they offer, either deterministic (e.g., Silo (JANG et al., 2015) and Hadrian (BALLANI et al., 2013)) or non-deterministic (e.g., PIAS (BAI et al., 2015), QJump (GROSVENOR et al., 2015) and NumFabric (NAGARAJ et al., 2016)). In spite of improving network performance, they present important shortcomings, including (i) under-utilization of resources (Silo and QJump); (ii) significant management overhead to dynamically perform rate calculation and enforce such rate for each flow (Hadrian and NumFabric), as the network can have millions of flows per second (BENSON; AKELLA; MALTZ, 2010); and (iii) starvation of large flows (PIAS). Moreover, none of these proposals optimize the allocation of resources other than the network, which may result in fragmentation of computing resources (e.g., CPU, memory and disk I/O). Therefore, we aim at proposing schemes without these shortcomings, even if it means introducing some complexity (such as requiring that tenants perform a fine-grained specification of their applications).

1.2 Hypothesis and Research Questions

To improve the state-of-the-art in the context of network performance in datacenters, particularly in terms of providing minimum bandwidth guarantees for tenants and their applications, this thesis presents the following hypothesis:

Hypothesis: a datacenter allocation strategy that considers multiple types of resources (CPU, memory, disk I/O and, particularly, the entire – traditional or SDN-based – network) can scalably provide predictable and guaranteed network performance for cloud applications, without hurting multi-resource utilization and provider revenue.

In order to guide the investigations conducted in this thesis, the following research questions (RQ) associated with the hypothesis are defined and presented:

RQ1: How can a distributed resource such as the network be efficiently managed, given the large scale and high dynamicity of cloud platforms?

RQ2: How can bandwidth be reserved for applications without hurting network utilization?

RQ3: How can SDN-based DCNs scalably provide predictable and guaranteed performance?

RQ4: How to enable the detailed specification of temporal requirements of multiple resources for applications, in order to help achieving guaranteed network performance without hurt-

ing utilization of other types of resources?

RQ5: While providing predictable and guaranteed network performance, can multi-resource utilization in an environment as dynamic as the cloud (i.e., with high rate of application arrival and departure) be maximized (i.e., minimizing multi-resource fragmentation)?

The proposed research questions are intentionally broad and could lead to several investigations and outcomes. At the end of this study, at least one possible answer for each question is provided. Note that the answers in this thesis are supported by a substantial amount of results and analysis. However, this does not mean that different approaches could not accomplish similar results.

1.3 Proposals and Contributions

During the PhD, we proposed three approaches to address the challenge of performance interference in DCNs: IoNCloud, Predictor and Packer. For IoNCloud, we leverage the key observation that temporal bandwidth demands of cloud applications do not peak at exactly the same time (WANG et al., 2012; CHEN; SHEN, 2014) and propose a resource allocation strategy for reserving and isolating network resources in datacenters. It aims at minimizing resource underutilization while providing an efficient way to predictably share bandwidth among applications, with low management overhead. To show the benefits of the strategy, we developed IoNCloud (Isolation of Networks in the Cloud), a scheme for Infrastructure-as-a-Service (IaaS) datacenters that implements the proposed allocation strategy. IoNCloud employs the abstraction of attraction/repulsion among tenant applications[2] according to their temporal network usage and need of isolation, and groups them into virtual networks (VNs) with bandwidth guarantees. In doing so, we seek to explore the trade-off between high resource utilization (a common goal for providers to reduce operational costs) and strict network guarantees (desired by tenants).

Despite achieving predictable and guaranteed network performance, IoNCloud does not provide work-conserving sharing among VNs, which keeps reserved resources idle while some other applications could benefit by using them. Therefore, we developed Predictor, an evolution over IoNCloud, to provide network guarantees with work-conserving sharing.

Predictor dynamically programs and configures the network in Software-Defined Networking (SDN)-based DCNs. In this context, Predictor is designed with scalability of SDN-based DCNs in mind, taking into consideration the number of entries required in flow tables and flow setup time in DCNs with high turnover and millions of active flows. To achieve network guarantees and scalability in both full-bisection and oversubscribed SDN-based DCNs, it relies on two key observations: (i) providers do not need to control each flow individually, since they charge tenants based on the amount of resources consumed by applications; and (ii) congestion control in the network is expected to be proportional to the tenant's payment (BALLANI et al.,

[2] An application is represented by a set of VMs that consume computing and network resources (see Chapter 3.1.2 for more details).

2013; JANG et al., 2015).

With these insights, Predictor deals with performance interference and scalability of SDN in DCNs as follows. Performance interference is addressed by employing two SDN-based algorithms to dynamically program the network, improving resource sharing. By doing so, both tenants and providers have benefits. Tenants achieve predictable network performance by receiving minimum bandwidth guarantees for their applications. Providers, in turn, maintain high network utilization, essential to achieve economies of scale.

Scalability is improved in two ways. First, as we show through measurements (Section 2.3.2), reducing flow table size also decreases the time taken to install rules in flow tables (stored in Ternary Content-Addressable Memory – TCAM) of SDN-enabled switches[3]. In the proposed approach, flow table size is minimized by managing flows at application-level and by using wildcards (when possible). This setting allows providers to control traffic and gather statistics at application-level for each link and device in the network.

Second, we propose to proactively install rules for intra-application communication, guaranteeing bandwidth between VMs of the same application. By installing rules at application allocation time, flow setup time is reduced. Inter-application rules, in turn, may be either proactively installed in SDN-enabled switches (if tenants know other applications that their applications will communicate with (GROSVENOR et al., 2015) or if the provider employs some predictive technique (XIE et al., 2012; LACURTS et al., 2013)) or reactively installed according to demands. Proactively installing rules has both benefits and drawbacks: while flow setup time is eliminated, some flow table entries may take longer to expire (they might be removed only when their respective applications conclude and are deallocated). Our decision is motivated by the fact that intra-application traffic volume is expected to be the highest type of traffic (BALLANI et al., 2013).

In spite of achieving their goals, IoNCloud and Predictor (and most allocation strategies in the literature (JANG et al., 2015; BALLANI et al., 2013)) neglect resources other than the network at allocation time; as a matter of fact, CPU and memory are typically allocated according to slots (GHODSI et al., 2011). Slot-based allocation, unfortunately, often causes over-allocation of resources, which leads to wastage (as applications do not use all of their allocated resources) and fragmentation (GRANDL et al., 2014). In general, over-allocation of resources for applications results in less applications being accepted in the infrastructure and in lower datacenter utilization.

To address the aforementioned limitations, we leverage two observations: (i) extending the insight presented for IoNCloud, applications have complementary demands across time for multiple resources (GRANDL et al., 2014); and (ii) utilization of different resources peaks at different times (CHEN; SHEN, 2014). Following these observations, we design and evaluate Packer. Besides providing guaranteed network performance with work-conserving sharing (like

[3]We use the terms "SDN-enabled switches" and "forwarding devices" to refer to the same set of SDN-enabled network devices, that is, data plane devices that forward packets based on a set of rules.

Predictor), Packer aims at minimizing overall multi-resource fragmentation and, consequently, at increasing datacenter utilization for multiple types of resources (network, CPU, memory and disk I/O), without considering slots.

Packer is designed with four aspects in mind: application abstraction, multi-resource allocation, network sharing and resource monitoring. First, Packer utilizes a novel abstraction for applications, called Time-Interleaved Multi-Resource Abstraction (TI-MRA). Unlike previous abstractions (BALLANI et al., 2011; LEE et al., 2014; XIE et al., 2012; BALLANI et al., 2013), TI-MRA imposes no predefined structure for applications and allows the specification of requirements for multiple resources across time. Second, Packer employs a new allocation strategy that extends previous heuristics developed for multi-dimensional bin packing, in order to reduce multi-resource fragmentation. Third, Packer leverages SDN/OpenFlow to dynamically configure and enforce bandwidth guarantees for applications throughout the entire network (taking an approach similar to Predictor's). Fourth, Packer employs a resource monitoring mechanism to avoid resource wastage and to provide fast and up-to-date information upon unexpected events (e.g., if an application gets delayed due to a resource being congested).

Class of applications to maximize the benefits of each proposal. IoNCloud, Predictor and Packer were developed considering the most important classes of cloud applications (throughput-intensive and latency-sensitive ones). All three schemes provide network performance guarantees. Nonetheless, each scheme is better suited for certain types of applications, as follows.

IoNCloud and Packer use temporal (or the peak, at the cost of some underutilization) demands of network resources (IoNCloud) and multiple types of resources (Packer). Consequently, like Proteus (XIE et al., 2012) and CloudMirror (LEE et al., 2014), they are better suited for tenants that repeatedly run the same type of applications with similar input and data sets. This is common in iterative data processing (e.g., PageRank (LANGVILLE; MEYER, 2011; PAGE et al., 1999), Hypertext-Induced Topic Search (KLEINBERG, 1999), recursive relational queries (BANCILHON; RAMAKRISHNAN, 1986), social network analysis and network traffic analysis), where much of the data stay unchanged from iteration to iteration (XIE et al., 2012). In this case, applications could be profiled periodically or on each run[4]. Furthermore, other applications may also take advantage of Packer by specifying only peak demands for multiple resources and, at runtime, employing its monitoring mechanism to avoid wasting resources.

Predictor, in turn, does not require temporal resource requirements of applications; it only needs the bandwidth guarantees required by each application. In case the bandwidth guarantees of some applications are under- or over-provisioned, Predictor's work-conserving rate enforcement algorithm automatically adjusts the rate in order to avoid bandwidth wastage. Therefore, Predictor is suited for most classes of cloud applications, including MapReduce, machine learn-

[4]We discuss how to profile temporal resource requirements for IoNCloud and Packer in Chapters 3 and 5, respectively.

ing (ZHOU et al., 2008) and general user-facing web applications with strict latency requirements.

Contributions. Overall, the major contributions of this thesis are:

- IoNCloud, a scheme for large-scale cloud datacenters. IoNCloud (i) groups applications in virtual networks; (ii) maps them on the physical substrate; and (iii) provisions network resources at each link the VN was allocated on according to peak aggregate demands of applications in the same group that utilize the link (i.e., the bandwidth required at the period of time when the sum of network demands of applications belonging to the same group is the highest). Evaluation results show that IoNCloud (i) provides predictable network performance with guaranteed bandwidth for tenants; and (ii) reduces network underutilization, allocated bandwidth (which allows more applications to be admitted in the cloud) and management overhead;
- Predictor, a scheme that leverages SDN and employs two novel algorithms to provide predictable network performance. Specifically, Predictor offers bandwidth guarantees with work-conserving sharing for tenants and fine-grained network management for providers. It also addresses the scalability challenges of SDN-based DCNs by controlling flows at application-level and by proactively installing rules in forwarding devices. Results show that Predictor (i) provides guaranteed network performance with work-conserving sharing; (ii) significantly reduces the number of rules in flow tables; and (iii) requires small controller load;
- Packer, a scheme that, in addition to providing predictable and guaranteed network performance, minimizes multi-resource fragmentation. We achieve these goals by leveraging SDN and using novel algorithms to (i) allocate applications considering multiple resources; and (ii) periodically set the allowed rate for each VM. We evaluate Packer and show that it provides minimum bandwidth guarantees for applications and work-conserving sharing for providers, and improves datacenter utilization and provider revenue in comparison to related work, with the cost of taking more (yet acceptable) time to allocate applications.
- A novel abstraction for specifying application requests in Packer, called Time-Interleaved Muti-Resource Abstraction (TI-MRA). Unlike previous abstractions (BALLANI et al., 2011; XIE et al., 2012; BALLANI et al., 2013; LEE et al., 2014), TI-MRA allows the specification of temporal demands for multiple resources without a predefined structure for applications.

1.4 Organization

This thesis is outlined as follows. Chapter 2 defines basic concepts used throughout the thesis and examines datacenter networks (including topology, type of traffic and variable per-

formance) and software-defined networking (benefits and scalability challenges). Chapters 3, 4 and 5 introduce, respectively, IoNCloud, Predictor and Packer, with a thorough evaluation and discussion of their generality and limitations. Finally, Chapter 6 discusses related work and Chapter 7 presents concluding remarks and perspectives for future work.

2 BACKGROUND

In this chapter, we present the context in which our proposals (IoNCloud, Predictor and Packer) were built. In particular, we first define basic concepts used throughout the thesis (Section 2.1). Then, we take a look at datacenter networks (Section 2.2), highlighting the main aspects related to our proposals. Finally, we discuss the benefits and challenges (including results from experiments) of employing software-defined networking in DCNs for Predictor and Packer (Section 2.3).

2.1 Basic Concepts

In this section, we present the definition of basic concepts used throughout the thesis, as follows.

Application. An application is represented by a set of distributed computing entities (tasks or VMs) that communicate among themselves and/or with users through the network. These computing entities receive input data (from users, cloud services or other applications), perform some computation on the received data and produce a result, which is sent to users or to some other application/service. Furthermore, each application is executed by a tenant.

Management overhead in the network. Network management refers to three types of operations: (i) communication between data and control planes in SDN; (ii) routing and forwarding packets in both traditional and software-defined networks; and (iii) rate-limiting traffic in the intra-cloud network. Communication between the data and control planes in SDN is necessary for several types of operations, such as updating state in forwarding devices and getting information from the network. Routing refers to determining the set of paths used by each flow in the network, while forwarding represents the operation of sending packets along a specific path. Finally, rate-limiting refers to determining the share of bandwidth used by each flow of each tenant application in comparison to all other flows using the same link, for each link in the network. Consequently, management overhead happens when one or more of these operations are costly, both in terms of resource consumption (e.g., CPU, memory and bandwidth) or time taken to be peformed.

Quality of Service (QoS). It refers to the capability of a network to provide minimum service guarantees to (selected) tenants, according to their requests. In the context of this thesis, it refers to tenants getting minimum network performance guarantees for applications.

2.2 Datacenter Networks

Datacenters are the core of cloud computing, and their network is an essential component to allow distributed applications to run efficiently and predictably (BALLANI et al., 2011). However, not all datacenters provide cloud computing. In fact, there are two main types of

datacenters: production and cloud. Production datacenters are often shared by one tenant or among multiple (possibly competing) groups, services and applications, but with low rate of arrival and departure. They run data-analytics jobs with relatively little variation in demands, and their size varies from hundreds of servers to tens of thousands of servers. Cloud datacenters, in contrast, have high rate of tenant arrival and departure (churn) (SHIEH et al., 2011; ZHENG et al., 2015; DELIMITROU; SANCHEZ; KOZYRAKIS, 2015), run both user-facing applications and inward computation, require elasticity (since application demands are highly variable) and consist of tens to hundreds of thousands of physical servers (ABTS; FELDERMAN, 2012). Moreover, clouds can be composed of several datacenters spread around the world (SUNG et al., 2016). As an example, Google, Microsoft and Amazon (three of the biggest players in the market) have datacenters in four continents; and each company has over 900,000 servers (ANTHONY, 2013).

Providers typically have three main goals when designing a DCN (GUO et al., 2008): scalability, fault tolerance and agility. First, the infrastructure must scale to a large number of servers (and preferably allow incremental expansion with commodity equipment and little effort). Second, a DCN should be fault tolerant against failures of both computing and network resources. Third, a DCN ideally needs to be agile enough to assign any VM (which is part of a service or application) to any server (GREENBERG et al., 2009). As a matter of fact, DCNs should ensure that computations are not bottlenecked on communication (SINGLA; GODFREY; KOLLA, 2014).

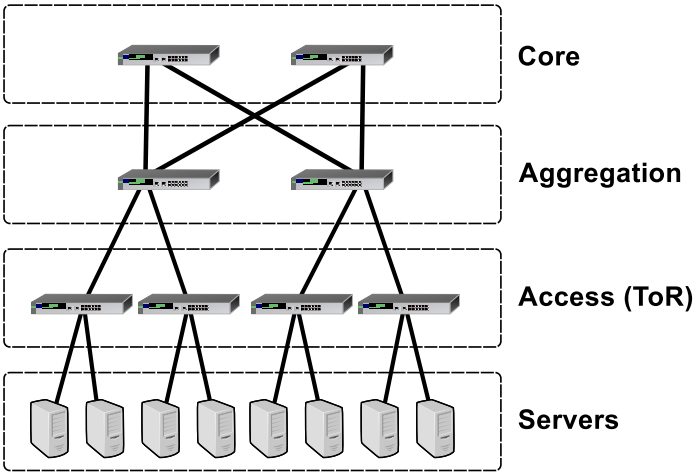
Currently, providers attempt to meet these goals by implementing the network as a multi-rooted tree (JANG et al., 2015), using LAN technology for VM addressing and two main strategies for routing: Equal-Cost MultiPath (ECMP) and Valiant Load Balancing (VLB) (MARCON et al., 2015). We structure our discussion of DCNs in the following manner. First, we examine the typical topology currently implemented by providers (Section 2.2.1). Then, we survey measurement studies regarding the types of traffic in DCNs (Section 2.2.2). Finally, we look into network performance variability (Section 2.2.3).

2.2.1 Topology

Figure 2.1 shows a canonical three-tiered multi-rooted tree-like topology, which is implemented in current datacenters (BENSON; AKELLA; MALTZ, 2010; BALLANI et al., 2011; AGACHE; DEACONESCU; RAICIU, 2015). The three tiers are: (i) the access (edge) layer, composed of Top-of-Rack (ToR) switches that connect servers mounted on every rack; (ii) the aggregation (distribution) layer, consisting of devices that interconnect ToR switches in the access layer; and (iii) the core layer, formed by switches that interconnect switches in the aggregation layer. Furthermore, every ToR switch may be connected to multiple aggregation switches for redundancy (usually 1+1 redundancy) and every aggregation switch is connected to multiple core switches. Typically, a three-tiered network is implemented in datacenters with

more than 8,000 servers (AL-FARES; LOUKISSAS; VAHDAT, 2008). In smaller datacenters, the core and aggregation layers are collapsed into one tier, resulting in a two-tiered datacenter topology (flat layer 2 topology) (BENSON; AKELLA; MALTZ, 2010).

Figure 2.1 – A canonical three-tiered tree-like datacenter network topology.



Source: by author (2015).

This multi-tiered topology has a significant amount of oversubscription, where servers attached to ToR switches have significantly more (possibly an order of magnitude) provisioned bandwidth between one another than they do with hosts in other racks (ABTS; FELDERMAN, 2012). Providers employ this technique in order to reduce costs and improve resource utilization, which are key properties to help them achieve economies of scale.

This topology, however, presents some drawbacks. First, the limited bisection bandwidth[1] constrains server-to-server capacity, and resources eventually get fragmented (limiting agility) (CURTIS; KESHAV; LOPEZ-ORTIZ, 2010; CURTIS et al., 2012). Second, multiple paths are poorly exploited (for instance, only a single path is used within a layer-2 domain by spanning tree protocol), which may potentially cause congestion on some links even though other paths exist in the network and have available capacity. Third, the rigid structure hinders incremental expansion (SINGLA et al., 2012). Fourth, the topology is inherently failure-prone due to the use of many links, switches and servers (LIU; MUPPALA, 2013).

To improve bisection bandwidth, some proposals follow a Clos-based (switch-oriented) design. A Clos topology is built up from multiple layers of switches (DALLY; TOWLES, 2003). Each switch in a layer is connected to all switches in the next layer, which provides extensive path diversity. Two proposals follow a Clos design: VL2 (GREENBERG et al., 2009; GREENBERG et al., 2011) and Fat-Tree (AL-FARES; LOUKISSAS; VAHDAT, 2008).

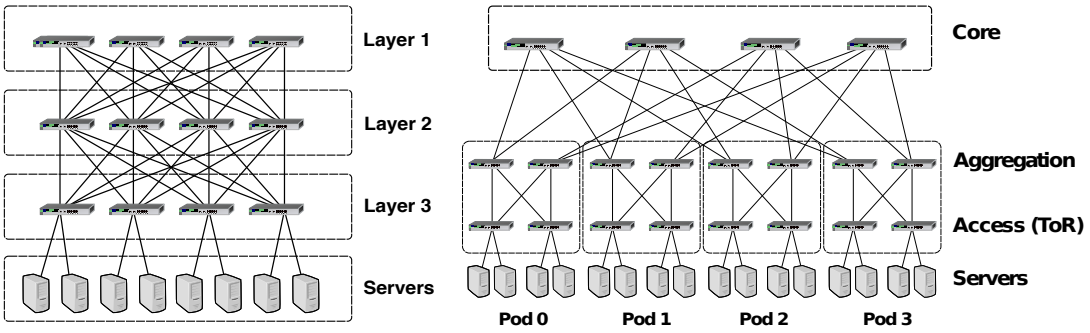
VL2 (GREENBERG et al., 2009; GREENBERG et al., 2011) is a Clos-based topology that

[1]The bisection bandwidth of the network is the worst-case segmentation (i.e., with minimum bandwidth) of the network in two equally-sized partitions (FARRINGTON; RUBOW; VAHDAT, 2009).

provides a larger number of paths between any two aggregation switches than the conventional topology. This means that, for N intermediate switches, the failure of one of them reduces the bisection bandwidth by only 1/N, which the authors call graceful degradation of bandwidth. An example of VL2 is shown is Figure 2.2(a).

Fat-Tree (AL-FARES; LOUKISSAS; VAHDAT, 2008), in turn, is a specific type of Clos topology (folded Clos topology). The topology, shown in Figure 2.2(b), is organized in a k-ary tree-like structure. There are k pods, each one of them has two layers (aggregation and access) of k/2 switches. Each k/2 switch from the access layer is connected to k/2 servers and the remaining ports are connected to k/2 aggregation switches. Each of the $(k/2)^2$ k-port core switches has one port connected to each of k pods. More specifically, the i-th port of any core switch is connected to pod i so that consecutive ports in the aggregation layer of each pod switch are connected to core switches on k/2 strides. In general, a fat-tree built with k-port switches supports $k^3/4$ hosts.

Figure 2.2 – Clos-based topologies.



(a) VL2. (b) Fat-Tree.

Source: by author (2015).

There are also other DCN topologies in the literature, such as switch-oriented (OSA (CHEN et al., 2012)), hybrid switch/server (DCell (GUO et al., 2008) and BCube (GUO et al., 2009)), server-only (CamCube (ABU-LIBDEH et al., 2010)) and random graph (SINGLA; GODFREY; KOLLA, 2014; SINGLA et al., 2012) topologies. We do not discuss these topologies because we follow related work (XIE et al., 2012; JANG et al., 2015) and focus on the topology implemented by most current datacenters (i.e., tree-like ones).

2.2.2 Traffic Properties

Traffic can be divided in two broad categories: north/south and east/west communication. North/south traffic (also known as extra-cloud) corresponds to the communication between a source and a destination host where one of the ends is located outside the cloud platform. In contrast, east/west traffic (also known as intra-cloud) is the communication in which both ends

are located inside the cloud. These types of traffic usually depend on the kind and mix of applications: user-facing applications (e.g., web services) typically exchange data with users and, thus, generate north/south communication, while inward computation (such as MapReduce) requires coordination among its VMs, generating east/west communication. Some studies (RECIO, 2012) indicate that north/south and east/west traffic correspond to around 25% and 75% of traffic volume, respectively. They also point out that both are increasing in absolute terms, but east/west is growing on a larger scale (RECIO, 2012).

Traffic in the cloud network is characterized by flows; each flow is identified by sequences of packets from a source to a destination node (i.e., a flow is defined by a set of packet header fields, such as source and destination addresses and ports and transport protocol). Typically, a bimodal flow classification scheme is employed, using elephant and mice classes. Elephant flows are composed of a large number of packets injected in the network, are usually long-lived and exhibit bursty behavior. In comparison, mice flows have a small number of packets and are short-lived (ABTS; FELDERMAN, 2012). Several measurement studies (BENSON; AKELLA; MALTZ, 2010; BODÍK et al., 2012; KANDULA et al., 2009; BENSON et al., 2011b; MENG; PAPPAS; ZHANG, 2010; GROSVENOR et al., 2015; BALLANI et al., 2013; LACURTS et al., 2013; JANG et al., 2015) were conducted to characterize network traffic and its flows. We summarize their findings as follows:

- **Traffic asymmetry.** Requests from users to cloud services are abundant, but small in most occasions. Cloud services, however, process these requests and typically send responses which are comparatively larger.
- **Nature of traffic**. Network traffic is highly volatile and bursty, with links running close to their capacity at several times during a day. Traffic demands change quickly, with some transient spikes and other longer ones (possibly requiring more than half the full-duplex bisection bandwidth) (LIU; KIND; LIU, 2013). Moreover, traffic is unpredictable at long time-scales (e.g., 100 seconds or more). However, it can be predictable on shorter timescales (at 1 or 2 seconds). Despite the predictability over small timescales, it is difficult for traditional schemes (such as statistical multiplexing) to make a reliable estimate of bandwidth demands for VMs (WANG; MENG; ZHANG, 2011).
- **General traffic location and exchange.** Most traffic generated by servers (on average 80%) stays within racks. Server pairs from the same rack and from different racks exchange data with a probability of only 11% and 0.5%, respectively. Probabilities for intra- and extra-rack communication are as follows: servers either talk with fewer than 25% or to almost all servers of the same rack; and servers communicate with less than 10% or do not communicate with servers located outside its rack.
- **Intra- and inter-application communication.** Most volume of traffic (around 70-80%) represents data exchange between VMs of the same application, with 18% of applications generating 99% of this traffic volume. In comparison, inter-application traffic vol-

ume represents around 20-30% of data exchange. However, the communication matrix between them is sparse; only 2% of application pairs exchange data, with the top 5% of pairs accounting for 99% of inter-application traffic volume. Consequently, communicating applications form several highly-connected components, with few applications connected to hundreds of other applications in star-like topologies.

- **Flow size, duration and number.** Mice (small) flows represent around 99% of the total number of flows in the network. They usually have less than 10 kilobytes and last only a few hundreds of milliseconds. Elephant (large) flows, in turn, represent only 1% of the number of flows, but account for more than half of the total traffic volume. They may have tens of megabytes and last for several seconds. With respect to flow duration, flows of up to 10 seconds represent 80% of flows, while flows of 200 seconds are less than 0.1% (and contribute to less than 20% of the total traffic volume). Further, flows of 25 seconds or less account for more than 50% of bytes. Finally, it has been estimated that a typical rack has around 10,000 active flows per second, which means that a network comprising 100,000 servers can have over 25,000,000 active flows.
- **Flow arrival patterns.** Arrival patterns can be characterized by heavy-tailed distributions with a positive skew. They best fit a log-normal curve having ON and OFF periods (at both 15 ms and 100 ms granularities). In particular, inter arrival times at both servers and ToR switches have periodic modes spaced apart by approximately 15 ms, and the tail of these distributions is long (servers may experience flows spaced apart by 10 seconds).
- **Link utilization.** Utilization is, on average, low in all layers but the core; in fact, in the core, a subset of links (up to 25% of all core links) often experience high utilization. In general, link utilization varies according to temporal patterns (time of day, day of week and month of year), but variations can be an order of magnitude higher at core links than at aggregation and access links. Due to these variations and the bursty nature of traffic, highly utilized links can happen quite often; 86% and 15% of links may experience congestion lasting at least 10 seconds and 100 seconds, respectively, while longer periods of congestion tend to be localized to a small set of links.
- **Hot-spots.** They are usually located at core links and can appear quite frequently, but the number of hot-spots never exceeds 25% of core links.
- **Packet losses.** Losses occur frequently even at underutilized links. Given the bursty nature of traffic, an underutilized network (e.g., with mean load of 10%) can experience lots of packet drops. Measurement studies found that packet losses occur usually at links with low average utilization (but with traffic bursts that go beyond 100% of link capacity); more specifically, such behavior happens at links of the aggregation layer and not at links of the access and core layers. Ideally, topologies with full bisection bandwidth (such as a Fat-Tree) should experience no loss, but the employed routing mechanisms (e.g., ECMP and VLB) cannot utilize the full capacity provided by the set of multiple paths

and, consequently, there is some packet loss in such networks as well (BENSON et al., 2011b).

2.2.3 Performance Variability

Currently, providers (for instance, Amazon EC2) offer VMs with guaranteed computing resources (JANG et al., 2015). However, the network is shared in a best-effort manner (LEE et al., 2014). Several recent measurement studies (GROSVENOR et al., 2015; JUDD, 2015; SCHAD; DITTRICH; QUIANÉ-RUIZ, 2010; WANG; NG, 2010; BALLANI et al., 2011; JANG et al., 2015; SHEA et al., 2014) concluded that, due to perfomance interference, the network throughput achieved by VMs can vary by a factor of five or more. For instance, Grosvenor et al. (GROSVENOR et al., 2015) show that variability can worsen tail performance by 50× for clock synchronization (PTPd) and 85× for key-value stores (Memcached). As the computation typically depends on the data received from the network (XIE et al., 2012; MITTAL et al., 2015) and the network is agnostic to application-level requirements (CHOWDHURY; ZHONG; STOICA, 2014), such variability often results in poor and unpredictable application performance (SHEN; LI, 2014). In this situation, tenants end up spending more money.

Performance variability is usually associated with three factors: type of traffic, network oversubscription and congestion control. First, the traffic in DCNs is remarkably different from other networks (GUO et al., 2014). Furthermore, the heterogeneous set of applications generates flows that are sensitive to either latency, throughput or both (CURTIS; KIM; YALAGANDULA, 2011; JANG et al., 2015); throughput-intensive flows are larger, creating contention in some links, which results in packets from latency-sensitive flows being discarded (adding significant latency) (GILL; JAIN; NAGAPPAN, 2011; ABTS; FELDERMAN, 2012).

Second, the conventional topology is typically oversubscribed, with more bandwidth available in servers than in the core (as discussed in Section 2.2.1). When periods of traffic bursts happen, the lack of bandwidth up the tree (i.e., at aggregation and core layers) results in contention and, thus, packet discards at congested links (leading to subsequent retransmissions). Since the duration of the timeout period is typically one or two orders of magnitude more than the round-trip time (RTT), latency is increased, becoming a significant source of performance variability (ABTS; FELDERMAN, 2012).

Third, TCP congestion control (used in such networks) cannot ensure performance isolation among applications; in fact, it only guarantees fairness among flows. Judd and Stanley (JUDD, 2015) show through measurements that many TCP design assumptions do not hold in datacenter networks, leading to inadequate performance. While TCP can provide high utilization, it does so very inefficiently. They conclude that the overall median throughput of the network is lower and that there is a large variation among flow throughput. This scenario is further exacerbated by UDP, which does not have any mechanism to control how distinct flows share the network.

Popa et al. (POPA et al., 2012) examines two main requirements for network sharing: (i)

bandwidth guarantees for tenants and their applications; and (ii) work-conserving sharing to achieve high network utilization for providers. In particular, these two requirements present a trade-off: strict bandwidth guarantees may reduce utilization, since applications have variable network demands over time (XIE et al., 2012); and a work-conserving approach means that, if there is residual bandwidth and some applications have demands, they should utilize it (even if the available bandwidth belongs to the guarantees of another application) (SHIEH et al., 2011).

2.3 Software-Defined Networking

In this section, we show a basic overview of SDN, as well as its benefits and challenges (including results from experiments) in DCNs.

Basic overview of SDN operation. Software-Defined Networking (SDN) seeks to decouple the control and data planes of the network, which enables the network to become programmable and the physical infrastructure to be abstracted from applications and services (JARRAYA; MADI; DEBBABI, 2014). Open Networking Foundation (ONF) presents a basic architecture (OPEN NETWORKING FOUNDATION, 2013) for SDN (depicted in Figure 2.3). This architecture is split in three main layers, as follows.

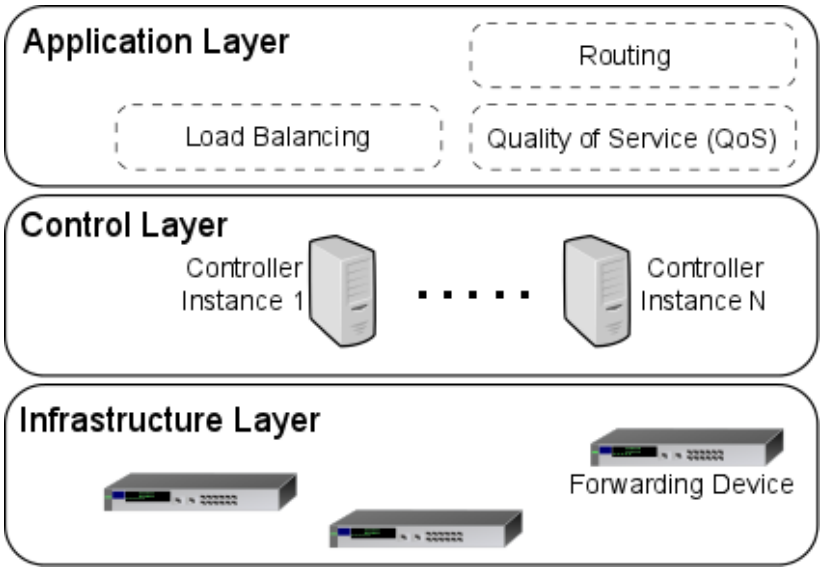
Infrastructure Layer. It is composed of the network's physical resources (e.g., SDN-enabled switches) that are part of the data plane. Forwarding devices are programmed by controllers (located in the control layer) via the OpenFlow (MCKEOWN et al., 2008) protocol (the most accepted standard by both academia and industry) and forward packets based on a set of rules;

Control Layer. This layer consists of one or more controllers providing a logically centralized control for the entire network. The control plane has a global view of the network, enabling decisions to be performed based on the current state of the whole network;

Application Layer. This layer is composed of control applications running on top of a logically centralized controller (control plane). These applications are responsible for the behavior of the network (including routing and quality of service).

In this paradigm, network devices (infrastructure layer) forward packets based on a set of rules dynamically generated by a controller (control layer) according to policies implemented by control applications (application layer). The operation of installing a rule (to handle a specific flow) in a SDN-enabled switch is called flow setup and works as follows. When a packet arrives, the forwarding device ASIC performs a lookup in the flow table(s). If there is a match, the packet is forwarded according to the matching rule(s). Otherwise (in case it is the first packet of a flow), the following steps are performed: (a) the ASIC sends the packet to the management CPU; (b) an operating system interruption is raised; (c) the packet is sent to the OpenFlow agent at the SDN-enabled switch; (d) the agent processes the packet and sends a request (containing the whole packet or the first 128 bytes, depending on the configuration) to the controller; (e) the controller receives and processes the request; (f) the controller sends the appropriate (set

Figure 2.3 – Basic SDN architecture defined by ONF.



Source: by author (2015).

of) rule(s) to handle the new flow; (g) the SDN-enabled switch receives the reply, installs the rule(s) and forwards the packet (and subsequent packets of the same flow) accordingly.

In general, SDN brings both benefits (Section 2.3.1) and scalability challenges (Section 2.3.2) for DCNs. After discussing them, we examine proposals that attempt to address these challenges (Section 2.3.3).

2.3.1 Benefits

SDN (with OpenFlow) offers several benefits for providers and tenants (KREUTZ et al., 2014). First, it does not rely on complex distributed protocols to synchronize information among forwarding devices and routers, thus becoming less error-prone. Second, this paradigm offers programmability to dynamically configure and manage the entire network, enabling providers to offer a base-level of network performance guarantees for tenants. Third, control logic centralization simplifies the development of complex networking functions, services and applications. Fourth, control applications (running on top of the controller) can seamlessly react to changes in order to maintain the correct operation of the network (following the specified policies, according to the current state of the network and the needs of tenants).

In general, administrators can efficiently apply a wide-range of policies (without requiring complex device by device configuration), including bandwidth guarantees (at application-level, which is required by several types of applications (CHOWDHURY; ZHONG; STOICA, 2014)), routing and load balancing (for instance, to address the limitations of ECMP (ALIZADEH et al., 2014; HU et al., 2015)). SDN also provides near-optimal traffic management, since

the controller can request and receive detailed information about network load on a low-level granularity (e.g., by device, link or specific flows traversing a link).

2.3.2 Scalability Challenges in DCNs

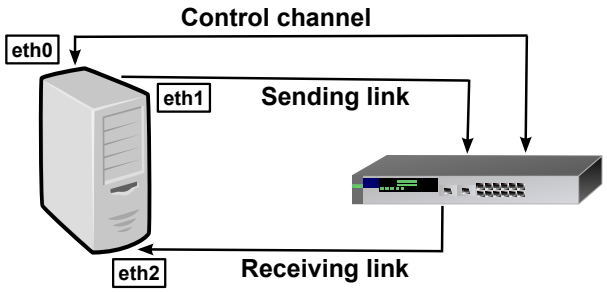
SDN-based networks involve the control plane more frequently than traditional networking (CURTIS et al., 2011). In the context of large-scale DCNs, this aspect leads to two scalability issues: flow setup time (the time taken to install new flow rules in forwarding devices) and flow table size in SDN-enabled switches (JARRAYA; MADI; DEBBABI, 2014; HU et al., 2015).

Flow setup time. It may add impractical delay for flows, especially for latency-sensitive ones (JANG et al., 2015) (as adding even 1 ms of latency to these flows is intolerable (ALIZADEH et al., 2010)). As SDN relies on the communication between network devices (data plane) and a logically centralized controller (control plane), it increases (i) control plane load and (ii) latency (sources for augmented delay). Control plane load is increased because a typical ToR switch will have to request rules to the controller for approximately more than 1,500 new flows per second (KANDULA et al., 2009) and the controller is expected to process and reply to all requests in, at most, a few milliseconds. Consequently, this may end up making both the communication with the controller and the controller itself bottlenecks. Latency is augmented because new flows are delayed at least two RTTs (i.e., communication between the ASIC and the management CPU and between that CPU and the controller) (CURTIS et al., 2011), so that the controller can install the appropriate rules at forwarding devices.

Experiments to measure flow setup time. We evaluated the time taken to perform the operation of inserting rules at a SDN-enabled switch's TCAM. Our measurement setup (shown in Figure 2.4) consists of one host with three 1 Gbps interfaces connected to an OpenFlow switch (Centec v350): eth0 interface is connected to the control port and eth1 and eth2 are connected to data ports on the switch. The switch uses OpenFlow 1.3 and has a TCAM that stores at most 2,000 rules. The host runs OpenFlow controller Ryu, which listens for control packets on eth0.

The experiment works as follows. The switch begins with a given number of rules installed in the TCAM (which represents its table occupancy). The host runs a packet generator to send a single UDP flow on eth1. This flow generates a table-miss event in the switch (i.e., the switch does not have an appropriate rule to handle the flow sent by the packet generator). Consequently, the switch sends a packet_in message to the controller. Upon receiving the packet_in, the controller processes the request and sends back a flow_mod message with the appropriate rule to be installed in the switch TCAM to handle the flow. Once the switch installs the rule, it forwards the matching packets to the link connected on the host eth2 interface. Like He et al. (HE et al., 2015), the latency of the operation is calculated as follows: (i) timestamp1 is recorded when the controller sends the flow_mod to the switch on eth0; and (ii) timestamp2 is

Figure 2.4 – Measurement setup.

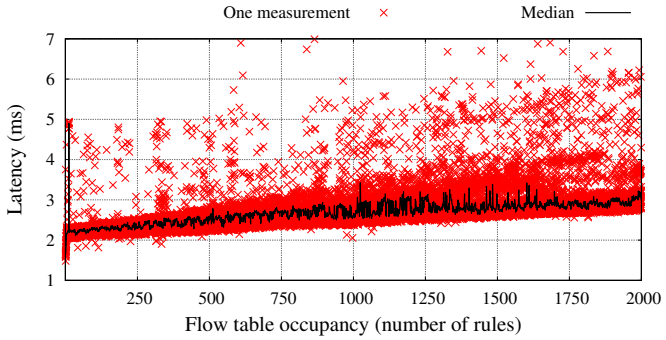


Source: by author (2016).

recorded when the first packet of the flow arrives on the host eth2 interface. Since the round-trip time (RTT) between the switch and host is negligible in our experiments[2], the latency is calculated by subtracting timestamp1 from timestamp2.

Figure 2.5 shows the latency of inserting new rules at the TCAM (y-axis) according to the number of rules already installed in the table (x-axis). The experiment was repeated 10 times; each point in the plot represents one measured value of one repetition and the line depicts the median value. Results show that median latency and variability increase according to flow table occupancy. These results are in line with previous measurements in the literature, such as He et al. (HE et al., 2015). Since adding even 1ms may be intolerable for some applications (e.g., latency-sensitive ones) (ALIZADEH et al., 2010), reduced flow table occupancy is highly desirable in DCNs because of flow setup time.

Figure 2.5 – Latency of inserting new rules according to flow table occupancy.



Source: by author (2016).

Flow table size. Flow tables are a restricted resource in commodity SDN-enabled switches, as TCAMs are typically expensive and power-hungry (COHEN et al., 2014; JYOTHI; DONG; GODFREY, 2015; HU et al., 2015). Such devices usually have a limited number of entries

[2]While the RTT is negligible in our experiments (as the switch is directly connected to the controller), it may not be the case in large-scale datacenters with hundreds of switches.

available for OpenFlow rules, which may not be enough when considering that large-scale datacenter networks have an elevated number of active flows per second (BENSON; AKELLA; MALTZ, 2010). That is, the number of entries required in flow tables can be significantly higher than the amount of resources available in commodity devices used in DCNs (CURTIS et al., 2011; COHEN et al., 2014)

Therefore, the design of a proposal must take both flow setup time and flow table size in SDN-enabled switches into account. We address them in Chapter 4, in which we introduce Predictor. Before that, we (i) analyze recent efforts that tackle these challenges (Section 2.3.3); and (ii) introduce IoNCloud (Chapter 3).

2.3.3 Improving Scalability in DCNs

Some proposals attempt to address the scalability challenges of SDN/OpenFlow in DCNs. DevoFlow (CURTIS et al., 2011) and DIFANE (YU et al., 2010) propose to reduce controller responsibilities in the network. The first one introduces new mechanisms for devolving control to forwarding devices (for small flows only; large flows are handled by the controller) and for efficient statistics gathering. Devolving control is accomplished through rule cloning and local actions. Rule cloning allows SDN-enabled switches to clone rules, replacing wildcarded fields of existing rules by values matching new flows, and all other aspects of the original rules are inherited. Local actions allow rules to be augmented with a set of routing actions that an SDN-enabled switch can take without involving the controller (e.g., multipath support and rapid rerouting). Statistics gathering, in turn, is performed through sampling (uniformly chosen packets are sent to the controller at a specific rate), triggers and reports (OpenFlow is extended with threshold-based triggers on counters) and approximate counters (maintained for small flows that match an existing rule).

The second one keeps all packets in the data plane by directing them through intermediate forwarding devices that store the appropriate rules. The controller is only responsible for partitioning the set of rules over the SDN-enabled switches. More specifically, DIFANE is built based on two main ideas, as follows. First, the controller generates and distributes the set of rules across a subset of the SDN-enabled switches in the network, which are called authority switches. Second, the SDN-enabled switches handle all packets in the data plane, diverting packets through authority switches according to the rules required.

Despite the benefits DevoFlow and DIFANE achieve in DCNs, they require more complex, customized hardware at forwarding devices. This requirement increases provider cost, which ends up being translated to more expensive resources for tenants.

Like DevoFlow, Hedera (AL-FARES et al., 2010) and Mahout (CURTIS; KIM; YALAGANDULA, 2011) also route elephant flows according to the rules defined by a centralized controller, while mice flows are handled by SDN-enabled switches. Hedera is a dynamic flow scheduling system for DCNs. It collects flow information from forwarding devices, computes

appropriate paths and instructs devices to reroute traffic accordingly. The system uses its global view of the network and knowledge of traffic demands to reduce bottlenecks when scheduling flows. Its main goal is to maximize network utilization with minimal impact on active flows.

Mahout proposes to monitor and detect large (elephant) flows by observing the end-hosts' socket buffers (through a shim layer in the OS). It manages elephant flows through an in-band signaling mechanism between the shim layer (at end-hosts) and the controller. At the forwarding device, mice flows are routed using the current load-balancing scheme (e.g., ECMP and VLB), while elephant flows are managed by rules defined by the controller.

To efficiently route large flows and utilize available resources through the multiple parallel paths, Hedera and Mahout require precise statistics from the network with at most 500 ms of interval between (RAICIU et al., 2010). Nevertheless, obtaining statistics with such frequency is impractical in dynamic networks such as large DCNs (CURTIS et al., 2011).

Lastly, Kandoo (YEGANEH; GANJALI, 2012) provides a logically distributed control plane for large-scale networks. It has two layers of controllers: the bottommost layer consists of a group of controllers with no interconnection and no knowledge of the network-wide state; and the topmost layer is composed of a logically centralized controller that maintains a global view of the network. While local controllers handle frequent events, the root controller handles only rare events. Nonetheless, it presents scalability issues to handle the high dynamic traffic patterns of DCNs, as the distributed set of controller instances need to maintain synchronized information (strong consistency) for the whole network. This is necessary in order to route traffic through less congested paths and to reserve resources for applications.

Having presented the context in which our proposals were built, we introduce IoNCloud, Predictor and Packer, respectively, in Chapters 3, 4 and 5.

3 IONCLOUD: PREDICTABLE NETWORK SHARING

IoNCloud implements our novel approach to allocate tenant applications in large-scale cloud platforms. The proposed strategy aims at providing performance predictability in the intra-cloud network while minimizing resource underutilization. To achieve this, unlike previous work (JANG et al., 2015; XIE et al., 2012; BALLANI et al., 2011; RODRIGUES et al., 2011), IoNCloud groups applications in virtual networks according to their temporal bandwidth demands.

This chapter is structured as follows. We first present the design of IoNCloud in Section 3.1. Then, we quantitatively evaluate it in Section 3.2. Finally, we discuss the generality and limitations of the proposed scheme in Section 3.3 and close the chapter with a brief summary in Section 3.4.

3.1 Design

In IoNCloud, all applications that belong to the same group are allocated on the same VN and share the same set of (virtual) network resources (i.e., they have shared bandwidth guarantees). Virtual networks, in turn, are completely isolated from one another, which means that each group has a guaranteed amount of network resources. An abstract view of IoNCloud is shown in Figure 3.1, which depicts application requests being received and allocated in two steps. The first step is responsible for application demand analysis and grouping, while the second maps VNs (groups composed of sets of applications) onto the physical substrate.

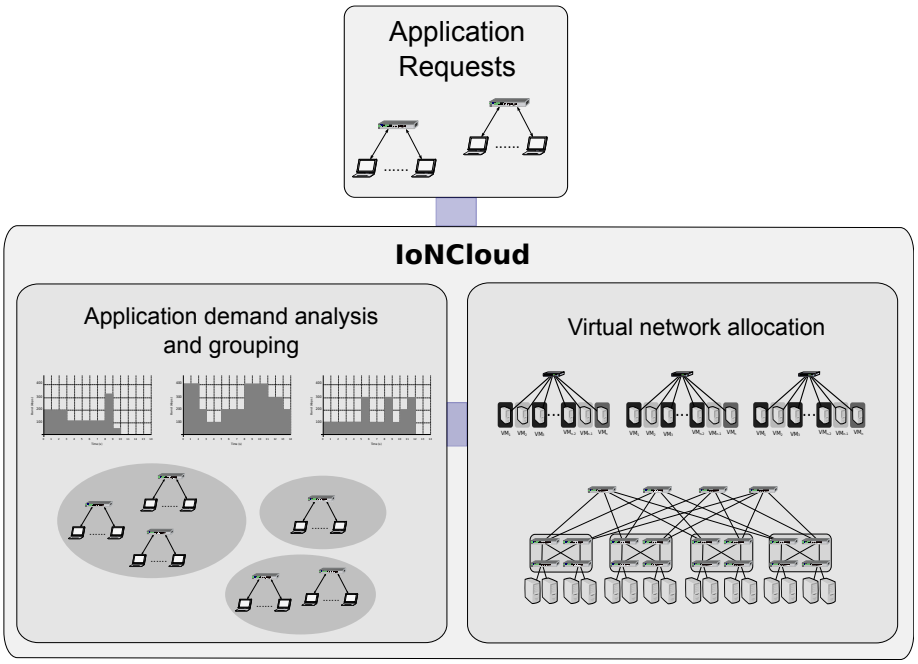
We first discuss how to obtain network profiles of applications to be allocated in the cloud platform (Section 3.1.1) and describe in detail the information contained in application requests (Section 3.1.2). Then, we present our novel strategy to group complementary applications in VNs according to their temporal network demands (Section 3.1.3) and to embed VNs on the cloud substrate (Section 3.1.4).

3.1.1 Network Profile of Applications

IoNCloud assumes that network profiles were previously generated (using techniques proposed in the literature, such as the ones described by Xie et al. (2012), LaCurts et al. (2013), Lee et al. (2014) and LaCurts et al. (2014)) and, thus, uses such information as input for incoming application requests. Like Choreo (LACURTS et al., 2013), IoNCloud considers, in each profile, the number of bytes sent by the application rather than the observed rate, as the former is independent of network congestion.

In particular, we highlight the feasibility of obtaining these profiles. In addition to the traffic properties discussed in Section 2.2.2, other properties related to the temporal demands of applications were verified by some studies (BALLANI et al., 2011; BENSON; AKELLA; MALTZ,

Figure 3.1 – IoNCloud model overview.



Source: by author (2015).

2010; XIE et al., 2012; LACURTS et al., 2013; CHEN; SHEN, 2014). These studies have conducted experiments on VM resource utilization during both short- and long-term periods of time. Their main findings are summarized as follows: (i) application traffic patterns are predictable; (ii) VMs of the same application (such as MapReduce) tend to have similar resource consumption; (iii) the same application running different datasets has similar patterns of resource usage; and (iv) periodical (e.g., daily) patterns of resource utilization can be observed for long-term applications.

These results enabled the proposal of some techniques to profile cloud applications. For instance, Xie et al. (2012) and LaCurts et al. (2013) use network monitoring tools (sFlow and tcpdump) to collect traffic traces (gathering application communication patterns), while Lee et al. (2014) discuss the utilization of application templates (provided as a library for users). Xie et al. (2012) also convert the output of these measurements into coarse-grained pulse functions. Such studies perform these profiling runs during a testing phase or in production environments, which allows them to collect sufficient information to create network profiles before running applications in the cloud. Therefore, such techniques can be used for IoNCloud, so that application profiles are known before allocation.

Unlike CloudMirror (LEE et al., 2014), IoNCloud also considers applications that cannot have their network profiles generated in advance (for instance, because the application requires an elevated amount of resources to run a profiling test or it concludes very quickly). In such situations, the time-varying function $B(t)$ (detailed in Section 3.1.2), which indicates the temporal

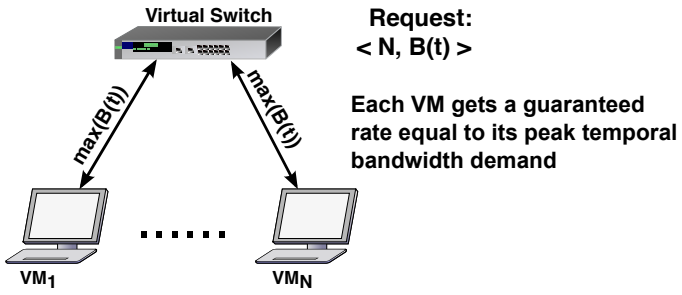
network demands of applications, is represented by a constant function (i.e., the same bandwidth requirement during the entire application lifetime). This constant value is specified by the tenant when submitting the request to the cloud. Such method enables the application to receive predictable and guaranteed network performance at the expense of some underutilization (since applications typically present variable traffic demands during their execution).

Note that some studies (e.g., Benson, Akella and Maltz (2010) and Kandula et al. (2009)) state that traffic in DCNs is unpredictable at long time-scales (e.g., 100 seconds or more) because they consider all traffic in the network. IoNCloud, like Proteus (XIE et al., 2012) and Choreo (LACURTS et al., 2013), requires temporal bandwidth demands of only the application being allocated. In other words, traffic is profiled for each application independently, not for the entire network.

3.1.2 Application Requests

Tenants request applications using the hose model (DUFFIELD et al., 1999a), similarly to prior work (POPA et al., 2013; POPA et al., 2012; JANG et al., 2015), as shown in Figure 3.2. In this abstraction, all VMs of an application are connected to a virtual switch through dedicated bidirectional links.

Figure 3.2 – Abstract view of an application's network topology.



Source: by author (2015).

More specifically, each application request is represented by its resource demand and formally defined by <N, $B(t)$>, with the terms specifying the number of VMs and the temporal bandwidth required by the application. The bandwidth demand is a time-varying function $B(t)$, similarly to Proteus (XIE et al., 2012). It represents the bandwidth required by the application at time "t". The amount of bandwidth reserved in each link connecting a VM to the application's virtual switch is represented by $\max(B(t))$, which denotes the peak temporal demand of the application's VM. This fine-grained specification allows IoNCloud to capture network requirements of applications in a precise manner.

Without loss of generality, we follow previous work (BALLANI et al., 2011; XIE et al., 2012; JANG et al., 2015) and make two assumptions. First, we abstract away computing re-

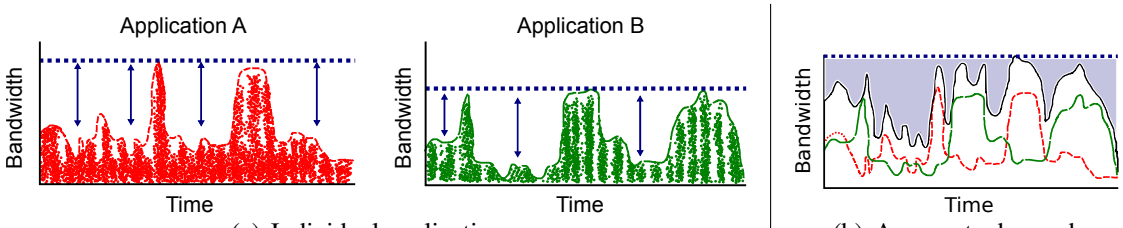
sources and assume a homogeneous set of VMs (i.e., equal in terms of CPU, memory and storage consumption). Second, we consider that all VMs of a given application follow the same network model[1]. Note that we make these assumptions only for the sake of notation simplicity. In practice, IoNCloud can easily support a heterogeneous set of VMs and different network models for VMs of the same application (see Section 3.3 for more details).

3.1.3 Application Demand Analysis and Grouping

This first step is responsible for analyzing network demands of applications and grouping them in virtual networks. This way, high resource utilization can be achieved without hurting predictability.

Figure 3.3 shows an example of how IoNCloud performs this process. In Figure 3.3(a), bandwidth requirements (dashed lines) of two applications (A and B) are fully guaranteed through a simple static reservation model (where the peak bandwidth is reserved, represented by dotted lines). However, this basic model causes underutilization of resources (shown by arrows in the figure), as unused bandwidth of one application (virtual network) cannot be used by any other application. IoNCloud, in contrast, enables applications with complementary bandwidth requirements (from either the same or different tenants) to share network resources. This is done by grouping them in the same VN and reserving the peak aggregate demand (i.e., the time when the sum of network demands of applications belonging to the same group is the highest), represented by the dotted line in Figure 3.3(b). Therefore, IoNCloud achieves better resource utilization, since periods of low demand from one application can be compensated by periods of high demand from other ones. Note that IoNCloud removes performance interference in the network by reserving the peak aggregate bandwidth of the applications (that belong to the same group) sharing a given link. Furthermore, it significantly reduces management overhead when compared to Proteus (XIE et al., 2012), since the latter must modify reservations as time passes by.

Figure 3.3 – Temporal network usage.



(a) Individual applications. (b) Aggregate demand.

Source: by author (2015).

[1]Many applications, such as MapReduce (which represents an important class of applications running in datacenters (GUO et al., 2013a)), have similar bandwidth demands among their VMs (XIE et al., 2012).

Note that, even though the applications in the same group may belong to different tenants and share bandwidth with one another, they receive minimum bandwidth guarantees. This happens because they have varying temporal network demands and the peak temporal aggregate demand of the group was reserved.

Algorithm. We design a time-varying network-aware algorithm to efficiently analyze the temporal bandwidth usage of applications and group them in virtual networks (see Algorithm 3.1). The key idea is based on minimizing the amount of unused bandwidth for each group created (i.e., reducing wasted bandwidth) and, therefore, enabling more applications to be admitted and allocated in the cloud platform. Formally, the objective is defined as reducing allocated (but unused) bandwidth:

$$Minimize \sum_{t \in T} \left(MaxAggBand - \sum_{app \in G} B(t) \right) \qquad \forall G \in Groups \quad (3.1)$$

where $MaxAggBand = \max(\sum_{app \in G} B(t) \quad \forall t \in T)$ denotes the maximum aggregate bandwidth of group G (i.e., the bandwidth that will be effectively reserved on links used for the group), T is the set of values representing the discrete time (in milliseconds) of the group's lifetime, G is a set that contains the applications in the group and $Groups$ is the set of all to-be-allocated groups. Thereby, the formula minimizes the unused bandwidth when choosing which applications will compose each group.

The algorithm works as follows. First, a bundle of incoming applications ($Applications$) is received and the set of groups ($GroupList$) is initialized as empty. The algorithm, then, retrieves one application (app) at a time from the set $Applications$ (lines 3 – 14) at no particular order and verifies three possibilities of grouping (lines 4 – 8): (i) creating a new group composed of app and another application from the set $Applications$ (i.e., trying all pairs of possible groupings of app with other incoming applications and selecting the one with least underutilization); (ii) inserting app into one of the existing groups; and (iii) creating a new group with app only. After verifying these possibilities, the best option is selected (line 9), app and possibly another application (if the first option was selected) are removed from $Applications$ (line 10), and the newly created group is inserted in the set $GroupList$, in case the first or third options were selected (lines 11 – 13). Finally, once the entire bundle of incoming applications has been analyzed and included into groups, the algorithm concludes, returns the set of groups (line 15) and the process of allocating each group (represented by a virtual network) onto the cloud infrastructure is started.

3.1.4 Virtual Network Allocation

This step is responsible for allocating each virtual network (group composed of applications that present complementary temporal bandwidth demands) created in the previous phase on the

Algorithm 3.1: Time-varying network-aware group creation.

Input : Bundle of applications to be allocated in the cloud
Output: List of application groups $GroupList$

```
Create a set Applications with all incoming applications;
Create an empty set GroupList;
foreach app ∈ Applications do
    Evaluate three possibilities of grouping:
        Creating new group containing app and a chosen application from Applications;
        Including app in existing group of the set GroupList;
        Creating new group with single application app;
    end
    Among the three above, select the option with least underutilization;
    Remove grouped applications from Applications;
    if new group was created then
        Include new group in the set GroupList;
    end
end
return the set GroupList;
```

physical infrastructure.

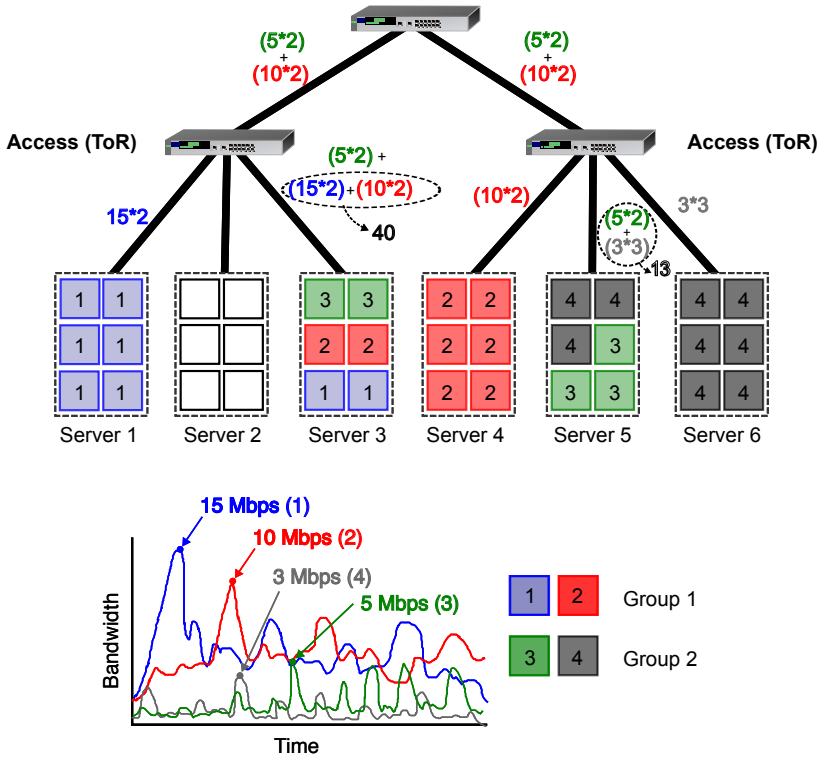
A simplified example of VN allocation in IoNCloud is shown in Figure 3.4. For the sake of simplicity, there are only two groups to be allocated, each one with two applications. For each group, IoNCloud prioritizes clustering VMs of the same application in the same physical server, since good locality reduces the amount of network resources used for intra-application communication[2]. For a single application, VMs located in the same server do not consume network resources when they communicate with each other. VMs allocated in different servers, in turn, need a certain amount of bandwidth to exchange data, which is given by the server with the lowest peak aggregate rate for an application. Consider "Server 1" (S_1) and "Server 3" (S_3) in the figure, which host application 1 (app_1). The bandwidth needed by VMs of this application for communication among themselves is given by $\min(|S_{1,app_1}|, |S_{3,app_1}|) \times \max(B(t))$, where $|S_{x,app_1}|$ represents the number of VMs of app_1 placed at the x^{th} server and $\max(B(t))$ denotes the peak bandwidth used by a single VM during its lifetime. In this example, $\min(6, 2) \times 15 = 30$ Mbps.

A group of applications, in contrast, requires the peak aggregate bandwidth demand of the group (instead of the sum of the peak demands of each application). Therefore, the allocated bandwidth on each physical link l of a VN corresponds to the peak aggregate demand of VMs in the group that use link l. This allocation is illustrated in Figure 3.4 in two situations: (i) when more than one application of the same group share a link, the aggregate peak bandwidth of the VMs of these applications is reserved (as we can see in Server 3, where 40 Mbps is reserved instead of 50 Mbps); and (ii) when a single application of a group uses a link, the bandwidth reserved corresponds to the amount needed by its VMs alone (other applications of the group do not need to use that link, as we can see in core links). Last but not least, note that VNs do not share bandwidth with one another (i.e., groups are completely isolated).

Algorithm. We developed a greedy algorithm to allocate virtual networks on the physical

[2]We follow related work (BALLANI et al., 2011; XIE et al., 2012; JANG et al., 2015) and consider only intra-application communication when allocating applications in the cloud platform, as this type of communication represents most of the traffic in the cloud (BALLANI et al., 2013).

Figure 3.4 – Shared bandwidth guarantees for applications.



Source: by author (2015).

substrate (see Algorithm 3.2). Because locality is key to make efficient use of resources, we address it with two granularities: (i) VMs of the same application are mapped on the infrastructure according to a VM placement objective (which is explained bellow); and (ii) VMs belonging to applications of the same group are allocated close to each other, because their bandwidth demands are complementary and, thus, network underutilization can be reduced (as determined by the grouping algorithm in the previous step). In general, the algorithm takes advantage of application affinity to allocate virtual networks in accordance with a given objective for virtual machine placement. IoNCloud can use any objective for VM placement, and it currently implements three different ones: minimizing bandwidth reservation (*MinBand*); minimizing energy consumption (*MinEnergy*); and maximizing fault tolerance (*MaxFT*). Since IoNCloud is agnostic about VM placement, these objectives will be detailed after the following overview of the allocation algorithm.

The algorithm allocates one VN at a time, with a coordinated node and link mapping, following insights provided by Chowdhury, Rahman and Boutaba (2009). The first step is the allocation of nodes (VMs) for each application in the group (lines 2 – 4), according to the VM placement policy defined. After all VMs of a VN are allocated, the algorithm starts the second step of the coordinated mapping, which is the allocation of bandwidth for the group (lines 5 – 7). This is performed through a bottom-up strategy, as follows. First, network resources are

Algorithm 3.2: Virtual network embedding.

Input : Physical infrastructure P, Set of groups $GroupList$
Output: Success/Failure array $allocated$

```
1  foreach Group g ∈ GroupList do
       // VM allocation
2      foreach Application app ∈ g do
3          Allocate VMs of app in the cloud infrastructure according to a predefined objective (e.g., minimum bandwidth, energy
           consumption or fault tolerance);
4      end
       // Bandwidth allocation
5      foreach Level lv from 0 to Height(P) do
6          Allocate bandwidth at lv according to demands of VMs at lower levels (similarly to Figure 3.4);
7      end
8      allocated[g] ← success/failure code for the allocation of group g;
9  end
10 return allocated;
```

reserved in racks (level 0). Then, they are reserved, in order, for each subsequent level of the topology[3] (according to the bandwidth needed by communication between VMs from distinct racks that belong to the same application, always aggregating the amount of bandwidth needed by VMs of the same group).

The algorithm returns a success code for each VN that was embedded on the substrate and a failure code otherwise (line 10). Application requests that belong to an unallocated VN are discarded (similarly to Amazon EC2 admission control (AMAZON, 2013)).

VM placement objectives. VM placement is often implemented as a multi-dimensional packing with constraints being defined according to a placement goal. As stated in the algorithm overview, IoNCloud currently supports three different goals, which are denoted: (i) *MinBand*; (ii) *MinEnergy*; and (iii) *MaxFT*. These placement goals are detailed next.

MinBand optimizes VM placement for minimal bandwidth reservation. It achieves this goal by clustering VMs of the same application and of the same group on the smallest subtree in the physical infrastructure (similarly to Ballani et al. (2011)). It takes two factors into account when searching for physical machines in the infrastructure to allocate VMs of a given application: (i) the number of VM slots available in each server; and (ii) the number of allocated VMs of the same group in each server. This way, it selects the server with the highest number of available VM slots (in case the application has more to-be-allocated VMs than the number of available VM slots per server) or the server with available slots holding the highest number of VMs of other applications of the same group. This behavior allows the algorithm to concentrate VMs of the same application and of the same group in the minimum number of servers, which tends to reduce the bandwidth needed by communication between VMs of the same application and the amount of network resources reserved for each VN, respectively.

MinEnergy follows insights from Mann et al. (2011) to consider energy consumption. VMs are consolidated on servers in order to reduce the number of servers turned on, thus minimizing the total amount of power consumed by these servers. This is achieved by a first-fit algorithm

[3]Like related work (RODRIGUES et al., 2011; BALLANI et al., 2011; XIE et al., 2012; MARCON et al., 2013; JANG et al., 2015), we assume the physical infrastructure topology in cloud datacenters is defined as a tree.

for packing VMs into servers. The only restriction is that VMs of the same application prioritize servers of the same rack first, as to avoid inter-rack communication.

MaxFT considers fault tolerance by spreading VMs on the cloud platform, so that applications can survive upon link, switch and/or rack failures (similarly to Bodík et al. (2012)). The key idea is to increase the number of servers used to allocate VMs in accordance to a given spreading factor (*sf*). In particular, the minimum number of servers is determined considering the servers with available resources and the number of VMs from the application each one of them can host. The new number of servers that will host these VMs is determined by choosing the minimum value between (i) the multiplication of the minimum number of servers required to allocate such VMs and *sf* and (ii) the number of VMs of the application: $\texttt{ExpectedNumSrvs} = \min(\texttt{MinNumSrvs}(\texttt{App}) \times sf, \texttt{NumVMs}(\texttt{App}))$. To illustrate the effect of the spreading factor, consider a very simple example where a single application with 6 VMs is being embedded on a homogeneous datacenter with 4 available VM slots per server. In this example, the minimum number of servers required to allocate the application is two ($\lceil$6 VMs / 4 slots $\rceil$). When *sf* is set to 1, the algorithm allocates all VMs in two servers. By increasing *sf* to 2, 3 and 4, the number of servers increases, respectively, to 4, 6 and 6. A high *sf* (resulting in, at most, one VM per server) is usually required for applications that need high availability (such as web services) (GILL; JAIN; NAGAPPAN, 2011), as failures potentially have high impact on user experience.

Allocation quality. Algorithm 3.2 was designed as a constructive heuristic with the focus of providing efficient allocation of resources for tenants. Therefore, it takes solution feasibility into account, but does not consider optimality. We made this choice for the following reason. Optimal techniques typically take a considerable amount of time to find the best solution for environments as big as cloud platforms (with hundreds of thousands of servers and hundreds of network devices) (MARCON et al., 2013). However, the allocation process must be performed as quickly as possible, since there are high rates of tenant arrival and departure (as discussed in Section 2.2). Therefore, it is computationally expensive to employ optimization strategies for these large-scale environments (YU et al., 2008).

3.2 Evaluation

In this section, we demonstrate the benefits of IoNCloud for both providers and tenants. Our evaluation focuses primarily on quantifying the advantage of grouping applications in virtual networks in terms of network predictability and resource utilization. Toward this end, we first describe the environment (in Section 3.2.1) and, then, present the main results (in Section 3.2.2).

3.2.1 Setup

Datacenter topology. We follow previous work (BALLANI et al., 2011; XIE et al., 2012; MARCON et al., 2013) and implement a discrete-event simulator that models a multi-tenant datacenter. We focus on tree-like topologies similar to multi-rooted trees used in current cloud platforms (BALLANI et al., 2013). The physical substrate is defined as a three-level tree topology with 8,000 servers at level 0, each with 4 VM slots (i.e., with a total amount of 32,000 available VMs in the cloud). Every machine is linked to a ToR switch (40 machines form a rack), and every 20 ToRs are connected to an aggregation switch. Finally, all aggregation switches are connected to a core switch. Link capacities are defined as follows: machines are connected to ToR switches with access links of 1 Gbps; links from racks up to aggregation switches are 10 Gbps; and aggregation switches are attached to a core switch with links of 50 Gbps. Thus, the default oversubscription of the network is 4.

Workload. To the best of our knowledge, there is no available source that provides realistic traces for cloud DCNs. Therefore, in line with related work (BALLANI et al., 2011; XIE et al., 2012; BALLANI et al., 2013), we generated the workload according to results obtained by measurement studies (BENSON; AKELLA; MALTZ, 2010; KANDULA et al., 2009; GREENBERG et al., 2011; SHIEH et al., 2011). More specifically, the workload is composed of requests of applications to be allocated in the cloud platform. Requests are formed by a heterogeneous set of applications (including MapReduce and Web Services), which is representative of applications running in public cloud platforms (ABTS; FELDERMAN, 2012). Each application is represented as a tuple <N, $B(t)$>, with N being the number of VMs and $B(t)$ a time-varying function to specify temporal network demands. The former is exponentially distributed around a mean of 49 VMs (following prior work). The latter was generated following results obtained by Kandula et al. (2009), Benson, Akella and Maltz (2010) and Greenberg et al. (2011). In fact, we used measurements related to inter-arrival flow time and size at servers to simulate application traffic.

Note that a few studies (BENSON et al., 2011b; KANDULA et al., 2009) indicate that traffic is unpredictable at long time-scales for the entire network. Therefore, we generate the traffic for each application (not the entire network), which is feasible (XIE et al., 2012).

3.2.2 Results

We compare IoNCloud, which employs shared bandwidth guarantees, with the approach adopted by most related work (BALLANI et al., 2011; RODRIGUES et al., 2011; GUO et al., 2010), which creates one virtual network per application. Ideally, we would have compared IoNCloud with Proteus (XIE et al., 2012). Proteus uses as input pulse functions obtained from the temporal network demands of applications. However, the generation of such pulse functions is addressed as a black-box in the paper and, thus, we cannot precisely develop a generator that

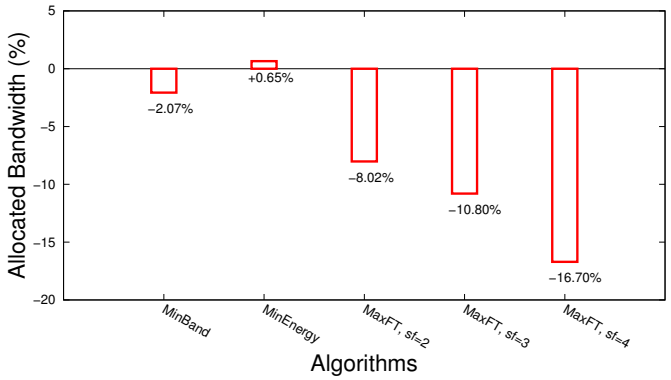
mimics its behavior.

As previously mentioned (Section 3.1.4), the algorithm used for virtual network allocation is agnostic in terms of VM placement. Hence, three VM placement algorithms are used in experiments: (i) *MinBand*, which minimizes the amount of bandwidth reserved for communication between VMs of the same application; (ii) *MinEnergy*, which minimizes energy consumption by reducing the number of used servers; and (iii) *MaxFT*, which maximizes fault-tolerance based on a given parameter (the desired ratio of extra servers used for spreading VMs).

For all experiments, we plot the percentile difference between both approaches given by the following equation: $(\frac{\texttt{IoNCloud}}{\texttt{One VN per App}} - 1) \times 100$. Hence, negative percentiles mean IoNCloud has achieved a lower value than traditional approaches, while positive percentiles mean IoNCloud has achieved a higher value than traditional approaches. In general lower values are better, with the sole exception being Figure 3.8.

Amount of reserved network resources. Figure 3.5 shows the total amount of reserved network resources according to the different placement algorithms. The y-axis represents the percentile difference between both approaches regarding the amount of bandwidth allocated, hence *the lower the value, the better*. We see that for any given approach, the amount of reserved resources increases in accordance with VM spreading. As expected, the shared bandwidth mechanism employed by IoNCloud outperforms the traditional methods when VMs are spread around the network, as it reduces the amount of reserved resources (up to 16.70%). This means that the provider can accept more applications in the cloud, improve resource utilization and, ultimately, increase datacenter throughput.

Figure 3.5 – Reserved network resources according to the placement algorithm employed.



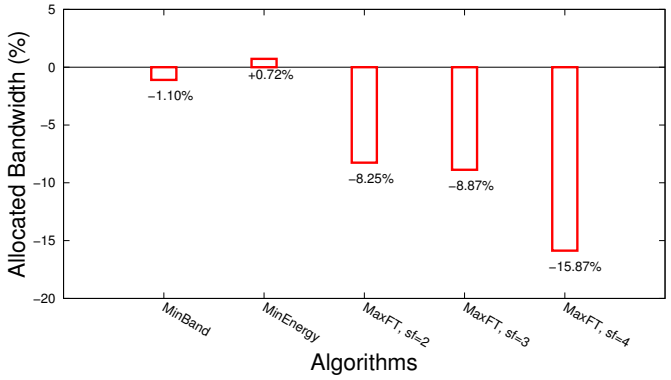
Source: by author (2015).

In contrast, IoNCloud is unable to achieve gains (in fact, with 0.65% of overhead in the worst-case) when there is no spreading, that is, when VMs are as packed as possible. This happens because the resource reservation employed by IoNCloud is performed per group, instead of per application (as traditional approaches). Therefore, the bandwidth allocated to each virtual link is only released after all applications in the respective group have finished. This design

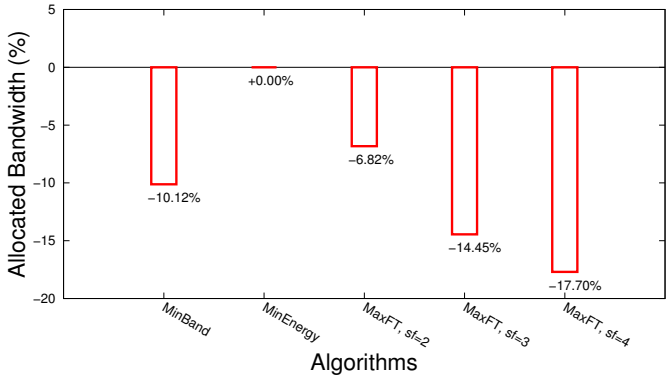
choice was deliberately chosen; such model can reduce the overhead of calculating the amount of bandwidth to be deallocated for each application that finishes its execution at each virtual link of the group. Moreover, we expect VM spreading to be norm in real cloud networks due to high churn of applications in these environments.

We further analyze bandwidth allocation by measuring the amount of reserved resources in access and aggregation layers[4] of the topology for all VM placement algorithms. We see in Figure 3.6 that IoNCloud allocates less resources in both layers. In particular, note that IoNCloud has better results in the aggregation. This effect also increases the chance of allocating virtual links, since network oversubscription at this level is higher than at the edge, and decreases the probability of packet discards in the network (which usually happens at this level (BENSON; AKELLA; MALTZ, 2010), as discussed in Section 2.2.2).

Figure 3.6 – Per-layer analysis of reserved network resources.



(a) At the access.



(b) At the aggregation.

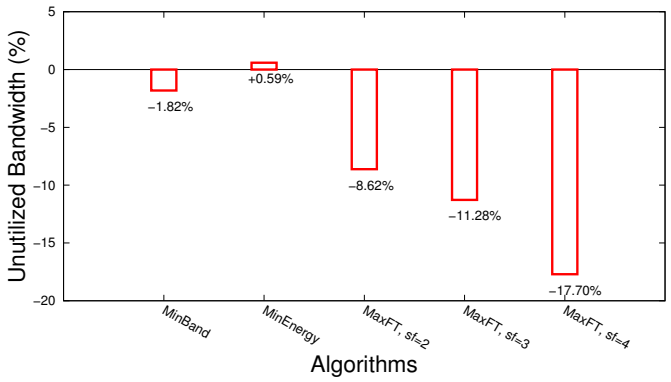
Source: by author (2015).

Underutilization in the network. Figure 3.7 depicts the percentile difference of unused bandwidth for different placement algorithms. Underutilization is quantified by measuring the unused bandwidth on each virtual link. *Lower values are better*, since they mean that the cloud

[4]In our experiments, there were no reserved resources in the core.

infrastructure is making better use of its reserved resources. As expected, IoNCloud achieves lower underutilization than current approaches. In fact, when compared to traditional schemes, IoNCloud is able to reduce waste, saving up to 18% of resources.

Figure 3.7 – Overall underutilization of resources.



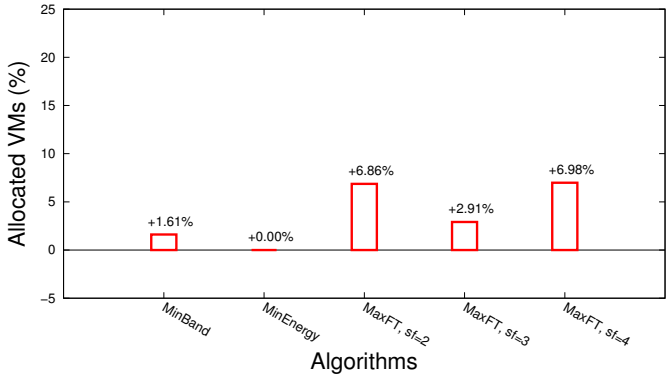
Source: by author (2015).

IoNCloud can reduce resource underutilization, but it still suffers from some underutilization. As mentioned in the previous experiment, this happens because the current implementation of IoNCloud performs bandwidth deallocation at group granularity (as opposed to application granularity).

Ratio of Allocated VMs. This metric shows the proportion of VMs that were allocated in servers. *Higher values are better*, as the revenue of the cloud provider is proportional to the number of VMs it allocates. Figure 3.8 shows the difference between VM allocation ratios. As observed, IoNCloud performs better for all algorithms except *MinEnergy*. Although the number of slots and VMs is the same, the allocation ratio differs depending on the allocation goal. This is because VMs can only be allocated if there is enough bandwidth for guaranteeing the setup of virtual links. Hence, reducing the amount of allocated bandwidth (as seen in Figure 3.5) increases the acceptance ratio of VMs in the cloud platform (since bandwidth is the bottleneck resource).

To understand the behavior of VM rejection, we perform experiments in a scenario where all datacenter links have unlimited bandwidth. Table 3.1 shows a comparison between both scenarios: normal and unlimited bandwidth. As can be observed, the assumption that bandwidth consumption interferes in VM allocation is verified, since all methods achieve 100% allocation with unlimited bandwidth. Note that *MinEnergy* is the only algorithm that achieves 100% VM allocation ratio under normal conditions. This is because VMs are packed together and fragmentation is minimal, thus, the majority of VMs will be closer. When minimizing bandwidth (*MinBand*), VMs may be allocated on free slots that are far from each other, which means that virtual links have a higher probability of reaching a bottlenecked physical link. *MaxFT* worsens this behavior, as it explicitly allocates VMs farther from each other.

Figure 3.8 – Ratio of VMs that were placed in physical servers.



Source: by author (2015).

Table 3.1 – VM allocation ratio with different VM placement goals for scenarios with normal and unlimited bandwidth capacity on links.

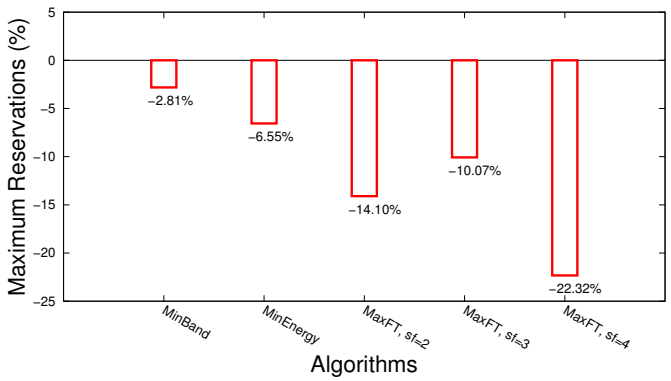
VM placement goal	Bandwidth	
	Normal	Unlimited
MinBand	0.929	1
MinEnergy	1	1
MaxFT, sf=2	0.845	1
MaxFT, sf=3	0.892	1
MaxFT, sf=4	0.888	1

Source: by author (2015).

Link Sharing and Management Overhead. We also measure the number of reservations over each link in the datacenter network. Figure 3.9 shows the percentile difference of the maximum number of virtual links allocated in the network. We find that IoNCloud results in a significantly lower number of reservations to be managed (which can be as high as 22.32% less). In an environment as large and dynamic as a cloud platform, where network devices are limited in terms of the amount of control state and the rate at which these states can be updated, this typically results in a reduced reservation management overhead. Furthermore, during the experiments, we observed relatively small absolute values (an overall value of less than 10,000) for the number of reservations for all strategies. This reflects the spatial locality applied by the allocation algorithms and suggests that the bandwidth reservation schemes can be accomplished using technologies already available in current datacenters (e.g., using rate-limiters in off-the-shelf switches or programmability of hypervisors (XIE et al., 2012)).

Generality of the results. We believe the obtained results are generalizable to most workloads and real cloud platforms (e.g., Amazon EC2 and Microsoft Azure). First, even though we could not obtain real traces from public cloud networks, we reversed engineered the traces used

Figure 3.9 – Maximum number of allocated virtual links.



Source: by author (2015).

by Kandula et al. (2009) and Benson, Akella and Maltz (2010), generating applications with heterogeneous traffic (e.g., MapReduce and web service) to share bandwidth. Second, we used a tree-topology in our experiments to mimic current datacenter networks (JANG et al., 2015) and to follow related work (BALLANI et al., 2013; BALLANI et al., 2011; XIE et al., 2012), so that the benefits and overheads of IoNCloud (i) can be closer to what would happen in current DCNs; and (ii) follow the same pattern of prior work, respectively.

However, note that the used workload represents neither the best nor the worst-case scenarios, both of which are not expected to happen (i.e., they are unrealistic). The best-case is when all applications have complementary demands with other applications and the grouping would be formed in a way that no bandwidth would be wasted in any period of time. This would further maximize the benefits of IoNCloud presented in our evaluation. In contrast to the best-case, the worst-case scenario happens when no application has complementary demands with other applications. This would result in one virtual network per application and the benefits of IoNCloud would be reduced to the advantages of related proposals that allocate one application per VN (e.g., Oktopus (BALLANI et al., 2011)).

3.3 Discussion

Datacenter network topology. Current datacenters are typically implemented through a multi-rooted tree topology (JANG et al., 2015). Therefore, we focus on this kind of topology to show the benefits of IoNCloud. However, IoNCloud can be easily adapted for other topologies, such as random graphs (SINGLA; GODFREY; KOLLA, 2014; SINGLA et al., 2012). In particular, it can be applied to multipath topologies, both where load balancing is uniform across paths and where it is not uniform. For the first case (e.g., Fat-Tree), a single aggregate link can be used as a representation for a set of parallel links for bandwidth reservation (BALLANI et al., 2011; POPA et al., 2013). For the latter, IoNCloud would have to use an additional layer at

hypervisor-level to control each path and its respective bandwidth for communication between VMs.

Online allocation of applications. IoNCloud uses the principle of allocating groups of applications in order to increase datacenter resource utilization. In this context, there is, at least, two ways of robustly providing online allocation for incoming application requests: (i) by allocating an incoming application to an existing group; and (ii) by allocating requests according to time slots. The first approach is straightforward, but may introduce some overhead to manage network resources when expanding and shrinking existing groups. The second one (which we employed in our evaluation) takes advantage of high churn in cloud environments (SHIEH et al., 2011; ZHENG et al., 2015). Thus, for each time slot (i.e., a predefined period of time), IoNCloud can allocate the set of incoming requests by grouping them according to their bandwidth demands, without modifying previously allocated groups (less overhead).

Generality of the network model. Currently, IoNCloud adopts a single network model for all VMs of the same application. Nonetheless, it requires no modification when considering VMs of the same application with distinct network profiles. However, it may add some complexity to the resource allocation process. Another option is to extend IoNCloud to enforce per-VM traffic models by reserving bandwidth on links according to the VM with the highest demand in each application (at the cost of some underutilization).

Network virtualization implementation. In practice, there are two ways of implementing VNs on the cloud: (i) in hypervisors; and (ii) in network devices. The first option is typically employed by state-of-the-art proposals (BALLANI et al., 2013; BALLANI et al., 2011; RODRIGUES et al., 2011; POPA et al., 2013; JEYAKUMAR et al., 2013; JANG et al., 2015), since it is easier to scale to large datacenters. The second is performed using SDN and OpenFlow. IoNCloud is agnostic about the technique utilized for network virtualization; in fact, it can use any of these approaches.

3.4 Summary

Cloud providers seek to increase datacenter resource utilization in order to reduce operational costs and achieve economies of scale. However, they lack efficient mechanisms to control how the intra-cloud network is shared among tenant applications. Therefore, they offer no real bandwidth guarantees for tenants, resulting in unpredictable network performance. We have introduced IoNCloud, a scheme that provides network predictability while minimizing resource underutilization and management overhead. To achieve this, IoNCloud groups applications in VNs according to their temporal bandwidth usage. Evaluation results show the benefits of our strategy, which is able to use available bandwidth more efficiently, reducing allocated bandwidth, network underutilization and management overhead.

4 PREDICTOR: PREDICTABLE, WORK-CONSERVING NETWORK SHARING

IoNCloud does not provide work-conserving sharing among VNs, hurting provider revenue. Predictor, in turn, implements a novel and low-overhead strategy to provide fine-grained management and predictable performance with work-conserving sharing in both full-bisection and oversubscribed SDN-based cloud DCNs.

Predictor is designed taking four requirements into consideration: (i) scalability, (ii) resiliency, (iii) predictable and guaranteed network performance, and (iv) high network utilization. First, any design for network sharing must scale to hundreds of thousands of VMs and deal with heterogeneous workloads of applications (traffic properties of DCNs were discussed in Section 2.2.2). Second, it needs to be resilient to churn both at flow-level (because of the rate of new flows/s (BENSON; AKELLA; MALTZ, 2010)) and at application-level (given the rates of application allocation/deallocation observed in datacenters (SHIEH et al., 2011)). Third, it needs to provide predictable and guaranteed network performance, allowing applications to maintain a base-level of performance even when the network is congested. Finally, any design should achieve high network utilization, so that spare bandwidth can be used by applications with more demands than their guarantees.

We describe Predictor as follows. First, we present its design in Section 4.1. Then, we quantitatively show its benefits and overheads in Section 4.2. Finally, we discuss its generality and limitations in Section 4.3 and close the chapter with a brief summary in Section 4.4.

4.1 Design

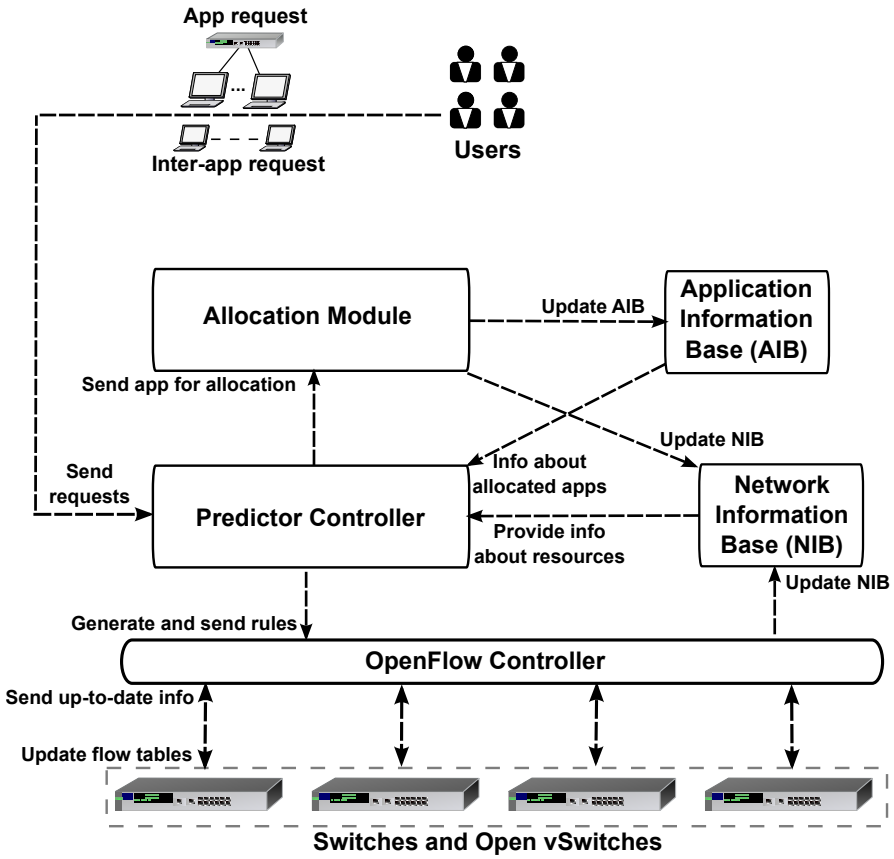
Predictor is designed to fulfill the above requirements. While providers can reduce operational costs and achieve economies of scale, tenants can run their applications predictably (possibly faster, reducing costs). Figure 4.1 shows an overview of Predictor, which is composed of five components: Predictor controller, allocation module, application information base (AIB), network information base (NIB) and OpenFlow controller. They are discussed next.

Predictor Controller. It receives requests from tenants. A request can be either an application to be allocated (whose resources to be used are determined by the allocation module) or a solicitation for inter-application bandwidth guarantees (detailed in Sections 4.1.1 and 4.1.2). In case of an incoming application, it sends the request to the allocation module. Once the allocation is completed (or if the request is for inter-application communication), the Predictor controller generates and sends appropriate flow rules to the OpenFlow controller. The OpenFlow controller, then, updates the tables (of forwarding devices) that need to be modified.

Note that the controller installs rules to identify flows at application-level[1] (more details in

[1]The granularity of rules (at application-level) hinders neither network controllability for providers nor network sharing among tenants and their applications because (a) providers charge tenants based on the amount of resources consumed by applications; and (b) congestion control in the network is expected to be performed at application-level (BALLANI et al., 2013).

Figure 4.1 – Predictor overview.



Source: by author (2016).

Sections 4.2 and 4.3). Predictor can also take advantage of flow management at lower levels (for instance, by matching source and destination MAC and IP fields), since it uses the OpenFlow protocol. Nonetheless, given the amount of resources available in commodity SDN-enabled switches and the number of flows that come and go in a small period of time, low-level rules are kept to a minimum.

Allocation Module. This component is responsible for allocating incoming applications in the datacenter infrastructure, according to available resources. It receives requests from the Predictor controller, determines the set of resources to be allocated for each new request and updates the AIB and NIB. We detail the allocation logic in Section 4.1.2.2.

Application Information Base (AIB). It keeps detailed information regarding each allocated application, including its identifier (ID), VM-to-server mapping, IP addresses, bandwidth guarantees, network weight (for work-conserving sharing), links being used and other applications it communicates with. It provides information for the Predictor controller to compute flow rules that need to be installed in SDN-enabled switches.

Network Information Base (NIB). It is composed of a database of resources, including hosts, switches, links and their capabilities (such as link capacity and latency). In general, it

keeps information about computing and network state, received from the OpenFlow controller (current state) and the allocation module (resources used for newly allocated applications). The Predictor controller uses information stored in the NIB to map logical actions (e.g., intra- or inter-application communication) into the physical network. While the AIB maintains information at application granularity, the NIB keeps information at network layer. The design of the NIB was inspired by Onix (KOPONEN et al., 2010) and PANE (FERGUSON et al., 2013).

OpenFlow Controller. It is responsible for communication to/from forwarding devices and Open vSwitches (PFAFF et al., 2015) in hypervisors, in order to update network state and get information from the network (e.g., congested links and failed resources). It receives information from the Predictor controller to modify flow tables in forwarding devices and updates the NIB upon getting information from the network.

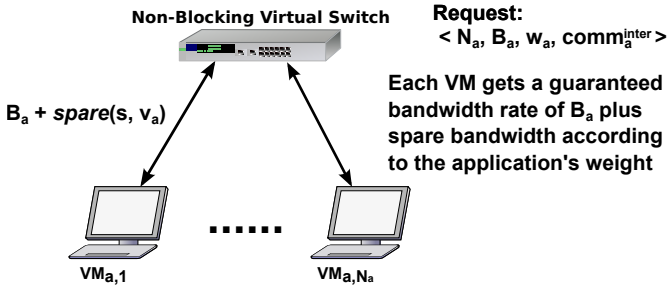
We explain Predictor in detail by describing application requests (Section 4.1.1), the mechanisms employed for resource sharing (Section 4.1.2) and the control plane design (Section 4.1.3).

4.1.1 Application Requests

Like in IoNCloud (Chapter 3) and similarly to past proposals (POPA et al., 2013; BALLANI et al., 2011; XIE et al., 2012; BALLANI et al., 2013; JANG et al., 2015), tenants request applications using the hose model to capture the semantics of the guarantees being offered, as shown in Figure 4.2. Recall that, in this model, all VMs of an application are connected to a non-blocking virtual switch through dedicated bidirectional links.

In Predictor, each application a is represented by its resource demands and network weight, or more formally, $\langle N_a, B_a, w_a, comm_a^{inter} \rangle$. Its terms are: $N_a \in \mathbb{N}^*$ specifies the number of VMs; $B_a \in \mathbb{R}^+$ represents the bandwidth guarantees required by each VM; $w_a \in [0, 1]$ indicates the network weight; and $comm_a^{inter}$ is an optional field that contains information about inter-application communication for application a.

Figure 4.2 – Virtual network topology of a given application.



Source: by author (2016).

The network weight is defined by the provider according to the tenant's payment. It enables residual bandwidth (unallocated or reserved bandwidth for an application and not cur-

rently being used) to be proportionally shared among applications with more demands than their guarantees (work-conservation). Therefore, the total amount of bandwidth available for each VM of application a at a given period of time, following the hose model, is denoted by $B_a + spare(s, v_a)$, where $spare(s, v_a)$ identifies the share of spare bandwidth assigned to VM v of application a located at server s:

$$\texttt{spare}(s, v_a) = \frac{w_a}{\sum_{v \uparrow V_s | v \in V_s} w_v} \times \texttt{SpareCapacity} \tag{4.1}$$

where V_s denotes the set of all co-resident VMs (i.e., VMs placed at server s), $v \uparrow V_s \mid v \in V_s$ represents the subset of VMs at server s that need to use more bandwidth than their guarantees and `SpareCapacity` indicates the residual capacity of the link that connects server s to the ToR switch.

The term $comm_a^{inter}$ is optional and allows tenants to proactively request guarantees for inter-application communication (since it would be infeasible to provide all-to-all communication between VMs in the datacenter (BALLANI et al., 2013)). It is a set composed of elements in the form of $\langle \texttt{srcVM}_\texttt{a}, \texttt{dstVMs}, \texttt{begTime}, \texttt{endTime}, \texttt{reqRate} \rangle$, where: $\texttt{srcVM}_\texttt{a}$ denotes the source VM of application a; `dstVMs` is the set of destination VMs (i.e., unicast or multicast communication from the source VM to each destination VM); `begTime` and `endTime` represent, respectively, the time that the communication starts and ends; and `reqRate` indicates the total amount of bandwidth per second needed by flows belonging to traffic from this (these) communication(s).

By providing optional specification of inter-application communication, Predictor allows requests from tenants with and without knowledge of application communication patterns and desired resources. Application traffic patterns are often known (GROSVENOR et al., 2015; CHOWDHURY; ZHONG; STOICA, 2014) or can be estimated by employing the techniques described by Lee et al. (LEE et al., 2014), Xie et al. (XIE et al., 2012) and LaCurts et al. (LACURTS et al., 2013). Note that, even if tenants do not proactively request resources for communication with other applications/services (i.e., they do not use $comm_a^{inter}$), their applications will still be allowed to reactively receive guarantees for communication with others (as detailed in Section 4.1.2.1).

In line with past proposals (BALLANI et al., 2011; XIE et al., 2012; BALLANI et al., 2013; JANG et al., 2015), two assumptions are made. First, we abstract away non-network resources and consider all VMs with the same amount of CPU, memory and storage. Second, we consider that all VMs of a given application receive the same bandwidth guarantees (B_a). Nonetheless, note that our strategy needs no modification when removing these assumptions.

4.1.2 Resource Sharing

In this section, we discuss how resources are shared among applications. In particular, we first examine how bandwidth guarantees are provided. Then, we take a look at the process of resource allocation and, finally, we present the logic behind the work-conserving mechanism employed by Predictor.

4.1.2.1 Bandwidth Guarantees

Predictor provides bandwidth guarantees for both intra- and inter-application communication. We discuss each one next.

Intra-application network guarantees. Typically, this type of communication represents most of the traffic in DCNs (BALLANI et al., 2013). Thus, Predictor allocates and ensures bandwidth guarantees at application allocation time[2] by proactively installing flow rules and rate-limiters in the network through OpenFlow.

Each VM of a given application a is assigned a bidirectional rate of B_a (as detailed in Section 4.1.1). Limiting the communication between VMs located in the same server or in the same rack is straightforward, since it can be done locally by the Open vSwitch at each hypervisor.

In contrast, for inter-rack communication, bandwidth must be guaranteed throughout the network, along the path used for such communication. Predictor provides guarantees for this traffic by employing the concept of *VM clusters*[3]. To illustrate this concept, Figure 4.3 shows a simplified scenario where a given application a has four clusters: $c_{a,1}$, $c_{a,2}$, $c_{a,3}$ and $c_{a,4}$. Since each VM of a cannot send or receive data at a rate higher than B_a, traffic between a pair of clusters $c_{a,x}$ and $c_{a,y}$ is limited by the smallest cluster: $rate_{c_{a,x},c_{a,y}} = \min(|c_{a,x}|, |c_{a,y}|) \times B_a$, where $rate_{c_{a,x},c_{a,y}}$ represents the calculated bandwidth for communication between clusters $c_{a,x}$ and $c_{a,y}$ (for $x, y \in \{1, 2, 3, 4\}$ and $x \neq y$), and $|c_{a,i}|$ denotes the number of VMs in cluster i of application a. In this case, $rate_{c_{a,x},c_{a,y}}$ is guaranteed along the path used for communication between these two clusters by rules and rate-limiters configured in forwarding devices through OpenFlow.

We apply this strategy at each level up the topology (reserving the minimum rate required for the communication among clusters). In general, the bandwidth required by one VM cluster to communicate with all other clusters of the same application is given by the following

[2]While Predictor may overprovision bandwidth at the moment applications are allocated, it does not waste bandwidth because of its work-conserving strategy (explained in Section 4.1.2.3). Without overprovisioning bandwidth at first, it would not be feasible to provide bandwidth guarantees for applications (as DCNs are typically oversubscribed).

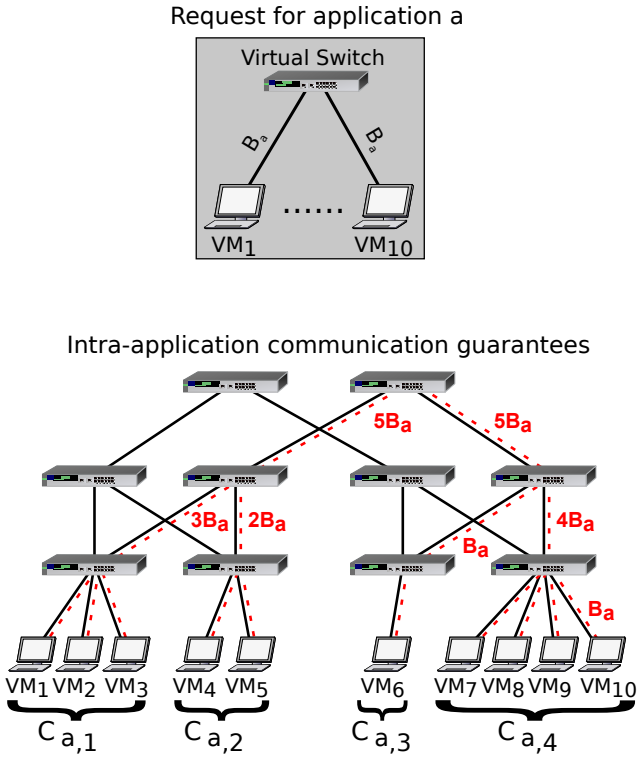
[3]A VM cluster is a set of VMs of the same application located in the same rack.

expression:

$$rate_{c_{a,x}} = \min\left(|c_{a,x}| \times B_a, \sum_{c \in C_a, c \neq c_{a,x}} |c| \times B_a\right) \qquad \forall c_{a,x} \in C_a \quad (4.2)$$

where $rate_{c_{a,x}}$ denotes the bandwidth required by cluster x to communicate with other clusters associated with application a and C_a indicates the set of all clusters of application a.

Figure 4.3 – Example of intra-application bandwidth guarantees.



Source: by author (2016).

Inter-application communication. Applications in datacenters may exhibit complex communication patterns. However, providing them with static hose guarantees does not scale for DCNs (BALLANI et al., 2013), since bandwidth guarantees would have to be enforced between all pairs of VMs. Furthermore, tenants may not know in advance all applications/services that their applications will communicate with.

Predictor can dynamically set up guarantees for inter-application communication according to the needs of applications and residual bandwidth in the network. In case guarantees were not requested using the field $comm_a^{inter}$ (as described in Section 4.1.1), the Predictor controller provides two ways of establishing guarantees for communication between VMs of distinct applications and services, as follows.

Reacting to new flows in the network. When a VM needs to exchange data with one or more VMs of another application, it can simply send packets to those VMs. The hypervisor (through

its Open vSwitch) of the server hosting the source VM receives such packets and, since they do not match any rule, sends them to the controller. The Predictor controller, then, determines the rules needed by the new flows and installs the set of rules along the appropriate path(s) in the network.

Receiving communication requests from applications. Prior to initiating the communication with VMs belonging to other applications, the source VM can send a request to the Predictor controller for communication with VMs from other application(s). This request is composed of the set of destination VMs, the bandwidth needed and the expected amount of time the communication will last. Upon receiving the request, the Predictor controller verifies residual resources in the network, sends a reply and, in case there are enough available resources, generates and installs the appropriate set of rules and rate-limiters for this communication. This approach is similar to providing an API for applications to request network resources, like PANE (FERGUSON et al., 2013).

4.1.2.2 Resource Allocation

The allocation process is responsible for performing admission control and mapping application requests in the datacenter infrastructure. An allocation can only be made if there are enough computing and network resources available (MOENS et al., 2014). That is, VMs must only be mapped to servers with available resources, and there must be enough residual bandwidth for communication between VMs (as specified in the request). For simplicity, we follow related work (BALLANI et al., 2011; XIE et al., 2012; JANG et al., 2015) and discuss Predictor and its allocation component in the context of traditional tree-based topologies implemented in current datacenters.

We design a location-aware heuristic to efficiently allocate tenant applications in the infrastructure. The key principle is minimizing bandwidth for intra-application communication (thus allocating VMs of the same application as close as possible to each other), since this type of communication generates most of the traffic in the network (as discussed before) and DCNs typically have scarce resources (XIE et al., 2012).

Algorithm 4.1 allocates one application at a time, as requests are received. It receives as input the physical infrastructure P (composed of servers, racks, switches and links) and the incoming application request a (formally defined in Section 4.1.1 as $\langle N_a, B_a, w_a, comm_a^{inter} \rangle$), and works as follows. First, it searches for the best placement in the infrastructure for the incoming application via dynamic programming (lines 1 – 17). To this end, $N_s^P(l-1)$ represents the set of neighbors (directly connected switches) of switch s at level $l-1$. Furthermore, three data structures are defined and dynamically initialized for each request: (i) set R^a stores subgraphs with enough computing resources for application a; (ii) V_s^a stores the total number of VMs of application a the s-rooted subgraph can hold; and (iii) C_s^a stores the number of VM clusters that can be formed in subgraph s. The algorithm traverses the topology starting at rack

level (level 1), up to the core, and determines subgraphs with enough available resources to allocate the incoming request.

Algorithm 4.1: Location-aware algorithm.

Input : Physical infrastructure P (composed of servers, racks, switches and links), Application a=$\langle N_a, B_a, w_a, comm_a^{inter} \rangle$
Output: Success/Failure code $allocated$

```
   // Search for the best placement in the infrastructure
1  R^a ← ∅;
2  foreach level l of P do
3      if l == 1 then // Top-of-Rack switches
4          foreach ToR r do
5              V_r^a ← num. available VMs in the rack;
6              C_r^a ← 1;
7              if V_r^a ⩾ N^a then R^a ← R^a ∪ {r} ;
8          end
9      end
10     else // Aggregation and core switches
11         foreach Switch s at level l do
12             V_s^a ← Σ_{w∈{N_s^P(l−1)}} V_w^a;
13             C_s^a ← Σ_{w∈{N_s^P(l−1)}} C_w^a;
14             if V_s^a ⩾ N^a then R^a ← R^a ∪ {s} ;
15         end
16     end
17 end
   // Proceed to the allocation
18 allocated ← failure code;
19 while Application a not allocated and R_a not empty do
20     r ← Select subgraph from R^a;
21     R^a ← R^a \ {r};
       // VM placement
22     Allocate VMs of application a at r;
       // Bandwidth allocation
23     foreach Level l from 0 to Height(r) do
24         Allocate bandwidth at l according to Section 4.1.2.1 and Equation 4.2;
25     end
26     foreach Inter-application communication c ∈ comm_a^inter do
27         Allocate bandwidth for inter-application communication c specified at allocation time (as defined in Section 4.1.1);
28     end
29     if Application was successfully allocated at r then
30         allocated ← success code
31     end
32     else allocated ← failure code;
33 end
34 return allocated;
```

After verifying the physical infrastructure and determining possible placements, the algorithm starts the allocation phase (lines 18 - 33). First, it selects one subgraph r at a time from the set R^a to allocate the application (line 20). The selection of a candidate subgraph takes into account the number of VM clusters. Therefore, the selected subgraph is the one with the minimum number of VM clusters, so that VMs of the same application are allocated close to each other, reducing the amount of bandwidth needed for communication between them (as the network often represents the bottleneck when compared to computing resources (CHEN et al., 2014)).

When a subgraph is selected, the algorithm allocates the application with a coordinated node (VM-to-server, in line 22) and link (bandwidth reservation, in lines 23 – 28) mapping, similarly to the virtual network embedding problem (CHOWDHURY; RAHMAN; BOUTABA,

2009). In particular, bandwidth for intra-application communication (lines 23 – 25) is allocated through a bottom-up strategy, as follows. First, it is reserved at servers (level 0). Then, it is reserved, in order, for each subsequent level of the topology, according to the bandwidth needed by communication between VMs from distinct racks that belong to the same application (as explained in Section 4.1.2.1 and in Equation 4.2, and exemplified in Figure 4.3). After that, bandwidth for inter-application communication (that was specified at allocation time in field $comm_a^{inter}$) is allocated in lines 26 – 28 (recall that $comm_a^{inter}$ was defined in Section 4.1.1).

Finally, the algorithm returns a success code if application a was allocated or a failure code otherwise (line 34). Applications that could not be allocated upon arrival are discarded, similarly to Amazon EC2 (AMAZON, 2014).

4.1.2.3 Work-Conserving Rate Enforcement

Predictor provides bandwidth guarantees with work-conserving sharing. This is because only enforcing guarantees through static provisioning leads to underutilization and fragmentation (POPA et al., 2013), while offering work-conserving sharing only does not provide strict guarantees for tenants (BALLANI et al., 2013). Therefore, in addition to ensuring a base-level of guaranteed rate, Predictor proportionally shares available bandwidth among applications with more demands than their guarantees, as defined in Equation 4.1.

We design an algorithm to periodically set the allowed rate for each co-resident VM on a server. In order to provide smooth interaction with TCP, we follow ElasticSwitch (POPA et al., 2013) and execute the work-conserving algorithm between periods of time one order of magnitude larger than the network round-trip time (RTT), e.g., 10 ms instead of 1 ms.

Algorithm 4.2 aims at enabling smooth response to bursty traffic (since traffic in DCNs may be highly variable over short periods of time (ABTS; FELDERMAN, 2012; NAGARAJ et al., 2016)). It receives as input the list of VMs (V_s) hosted on server s, their current demands (which are determined by monitoring VM socket buffers, similarly to Mahout (CURTIS; KIM; YALAGANDULA, 2011)), their bandwidth guarantees and their network weight (specified in the application request and defined in Section 4.1.1).

First, the rate for each VM is calculated based on their demands and the guaranteed bandwidth $B[v]$ (lines 1 – 4). In case the demand of a VM is equal or lower than its bandwidth guarantees (represented by $v \downarrow V_s \mid v \in V_s$), the rate is assigned and enforced (line 2), so that the exact amount of bandwidth needed for communication is used (wasting no network resources). In contrast, the guarantee $B[v]$ is initially assigned to $nRate[v]$ for each VM $v \in V_s$ with higher demands than its guarantees (represented by $v \uparrow V_s \mid v \in V_s$), in line 3. Then, the algorithm calculates the residual bandwidth of the link connecting the server to the ToR switch (line 5). The residual bandwidth is calculated by subtracting from the link capacity the guarantees of VMs with higher demands than their guarantees and the rate of VMs with equal or lower demands than their guarantees.

Algorithm 4.2: Work-conserving rate allocation.

Input : Set of VMs V_s allocated on server s, Current demands of VMs $demand$, Bandwidth guarantees B for each VM, Network weight w for each VM
Output: Rate $nRate$ for each co-resident VM

1 **foreach** $v \in V_s$ **do**
2 **if** $v \downarrow V_s$ **then** $nRate[v] \leftarrow demand[v]$;
3 **else** $nRate[v] \leftarrow B[v]$;
4 **end**
5 $residual \leftarrow LinkCapacity - \left(\sum_{v \uparrow V_s} B[v] + \sum_{v \downarrow V_s} demand[v]\right)$;
6 hungryVMs $\leftarrow v \uparrow V_s \mid v \in V_s$;
7 **while** $residual > 0$ *and* hungryVMs *not empty* **do**
8 **foreach** $v \in$ hungryVMs **do**
9 $nRate[v] \leftarrow nRate[v] + \min\left(demand[v] - nRate[v], \left(\frac{w[v]}{\sum_{u \uparrow V_s} w[u]} \times residual\right)\right)$;
10 **if** $nRate[v] == demand[v]$ **then**
11 hungryVMs $\leftarrow$ hungryVMs $\setminus \{v\}$;
12 **end**
13 **end**
14 **end**
15 **return** $nRate$;

The last step establishes the bandwidth for VMs with higher demands than their guarantees (line 6 - 14). The rate (line 9) is determined by adding $nRate[v]$ (initialized in line 3) and the minimum bandwidth between (i) the difference of the current demand ($demand[v]$) and the rate ($nRate[v]$); and (ii) the proportional share of residual bandwidth the VM would be able to receive according to its weight $w[v]$. Note that there is a "while" loop (lines 7 – 14) to guarantee that all residual bandwidth is used or all demands are satisfied. If this loop were not used, there could be occasions when there would be unsatisfied demands even though some bandwidth would be available.

With this algorithm, Predictor guarantees that VMs will not receive more bandwidth than they need (which would waste network resources) and bandwidth will be fully utilized if there are demands (work-conservation). Moreover, the algorithm has fast convergence on bandwidth allocation and can adapt to the significant variable communication demands of cloud applications. Therefore, if there is available bandwidth, VMs can send traffic bursts at a higher rate (unlike Silo (JANG et al., 2015), Predictor allows traffic bursts with complete work-conservation).

In summary, if the demand of a VM exceeds its guaranteed rate, data can be sent and received at least at the guaranteed rate. Otherwise, if it does not, the unutilized bandwidth will be shared among co-resident VMs whose traffic demands exceed their guarantees. We provide an extensive evaluation in Section 4.2 to verify the benefits of the algorithm.

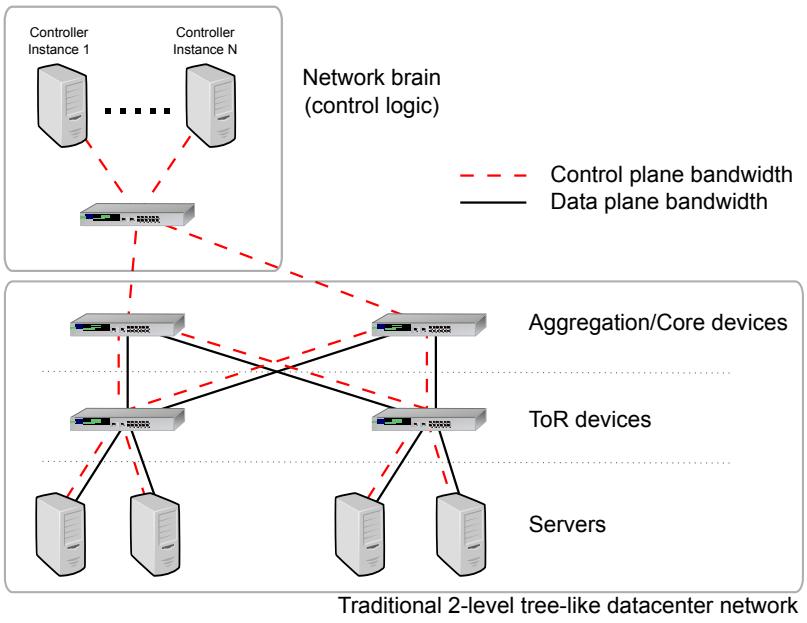
4.1.3 Control Plane Design

The control plane design of the network is an essential part of software-defined networks, as disconnection between the control and data planes may lead to severe packet loss and performance degradation (forwarding devices can only operate correctly while connected to a controller) (MULLER et al., 2014; HU et al., 2013). Berde et al. (BERDE et al., 2014) define

four requirements for the control plane: (i) high availability (usually five nines (ROS; RUIZ, 2014)); (ii) global network state, as the control plane must be aware of the entire network state to provide guarantees for tenants and their applications; (iii) high throughput, to guarantee performance in terms of satisfying requests even at periods of high demands; and (iv) low latency, so that end-to-end latency for control plane communication (i.e., updating network state in response to events) is small.

Based on these requirements, Figure 4.4 shows the control plane design for Predictor. In this figure, we show, as a basic example, a typical 2-layer tree-like topology with decoupled control and data planes. We can see two major aspects: (i) the placement of controller instances (control plane logic) as a cluster in one location of the network (connected to all core switches); and (ii) the separation between resources for both planes (represented by different line styles and colors for link bandwidth), indicating out-of-band control plane communication. We discuss them next.

Figure 4.4 – Design of Predictor's control plane.



Source: by author (2016).

Cluster of controller instances. Following WL2 (CHEN et al., 2015), the control plane logic is composed of a cluster of controller instances. There are two reasons for this. First and most important, Predictor needs strong consistency among the state of its controllers to provide network guarantees for tenants and their applications. If instances were placed at different locations of the topology, the amount of synchronization traffic would be unaffordable, since DCNs typically have highly dynamic traffic patterns with variable demands (ABTS; FELDERMAN, 2012; GUO et al., 2014; NAGARAJ et al., 2016). Moreover, DCNs are typically oversubscribed with scarce bandwidth (XIE et al., 2012).

Second, the control plane is expected to scale-out (periodically grow or shrink the number of active controller instances) according to its load, needed for high availability and throughput. Since DCNs usually count with multiple paths (GILL; JAIN; NAGAPPAN, 2011), one controller location is often sufficient to meet existing requirements (HELLER; SHERWOOD; MCKEOWN, 2012). Furthermore, if controllers were placed at several locations, a controller placement algorithm (e.g., Survivor (MULLER et al., 2014)) would have to be executed each time the number of instances were adjusted, which would delay the response to data plane requests (as this is a NP-Hard problem (HELLER; SHERWOOD; MCKEOWN, 2012)).

Out-of-band control. Predictor uses out-of-band control to manage the network. As the network load may change significantly over small periods of time (BENSON; AKELLA; MALTZ, 2010) and some links may get congested (ABTS; FELDERMAN, 2012) (due to the high oversubscription factor (ADAMI et al., 2013)), the control and data planes must be kept isolated from one another, so that traffic from one plane does not interfere[4] with the other. In other words, control plane traffic should not be impacted by rapid changes in data plane traffic patterns (e.g., bursty traffic). Using out-of-band control, some bandwidth of each link shared with the data plane (or all bandwidth from links dedicated to control functions) is reserved for the control plane (represented in Figure 4.4 as red dotted lines). In the next section, we show how the amount of bandwidth reserved for the control plane affects efficiency of Predictor.

4.2 Evaluation

Below, we evaluate the benefits and overheads of Predictor. We focus on showing that Predictor (i) can scale to large SDN-based DCNs; (ii) provides both predictable network performance (with bandwidth guarantees) and work-conserving sharing; and (iii) outperforms existing schemes for DCNs (the baseline SDN/OpenFlow controller and DevoFlow (CURTIS et al., 2011)). Towards this end, we first describe the environment and workload used (in Section 4.2.1). Then, we examine the main aspects of the implementation of Predictor (in Section 4.2.2). Finally, we present the results in Section 4.2.3.

4.2.1 Setup

Environment. We have implemented a simulator that models an IaaS multi-tenant, SDN-based datacenter. The network is defined as a tree-like topology, similar to current DCNs and related work (XIE et al., 2012; BALLANI et al., 2013; JANG et al., 2015). It is composed of a three-tier topology with 16,000 servers at level 0. We follow current schedulers and related work (GRANDL et al., 2014) and divide computing resources of servers (corresponding to

[4] In oversubscribed networks, such as DCNs, where traffic may exceed link capacities in some occasions, in-band control may result in network inconsistencies, as control packets may not (or take a long time to) reach the destination.

some amount of CPU, memory and storage) into slots for hosting VMs; each server is divided into 4 slots, resulting in a total amount of 64,000 available VMs in the datacenter. Every 40 machines form a rack, and every 10 ToRs are connected to an aggregation switch. Finally, all aggregation switches are connected to a core switch. The capacity of each link is defined as follows: 1 Gbps for server-ToR links, 10 Gbps for ToR-aggregation links and 50 Gbps for aggregation-core links.

Workload. The workload is composed of incoming application requests (to be allocated in the datacenter) arriving over time. In particular, we consider a heterogeneous set of applications, including MapReduce and Web Services. As defined in Section 4.1.1, each application a is represented as a tuple $\langle N_a, B_a, w_a, comm_a^{inter} \rangle$. Given the lack of publicly available traces for DCNs, the workload was generated in line with related work (XIE et al., 2012; BALLANI et al., 2013). N_a is exponentially distributed around a mean of 49 VMs (following measurements from prior work (SHIEH et al., 2011)). B_a was generated by reverse engineering the traces used by Benson et al. (BENSON; AKELLA; MALTZ, 2010) and Kandula et al. (KANDULA et al., 2009). More specifically, we used their measurements related to inter-arrival flow-time and flow-size at servers to generate and simulate network demands of applications. Unless otherwise specified, of all traffic, 20% of flows are destined to other applications (BALLANI et al., 2013) and 1% is classified as large flows (ABTS; FELDERMAN, 2012). We pick the destination of each flow by first determining whether it is an intra- or inter-application flow and then uniformly selecting a destination. The weight w_a is uniformly distributed in the interval $[0, 1]$.

Like in IoNCloud, we generate the traffic used in the experiments for each application (not the entire network), since a few studies (BENSON et al., 2011b; KANDULA et al., 2009) indicate that traffic is unpredictable at long time-scales for the entire network.

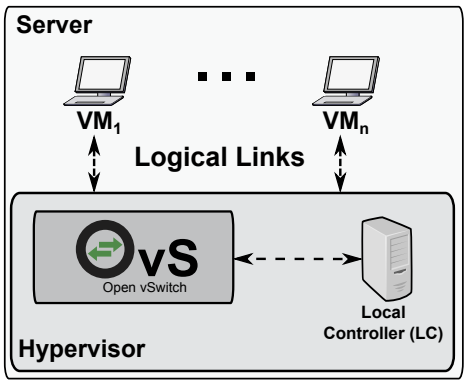
4.2.2 Aspects of Predictor

Figure 4.5 shows the server-level architecture of Predictor. As described by Pfaff et al. (PFAFF et al., 2015), the virtual machines allocated on the server send and receive packets to/from the network through the hypervisor, using an Open vSwitch. We implemented a local controller which directly communicates with the Open vSwitch. Together, the Open vSwitch and the local controller are responsible for handling all traffic to/from local virtual machines.

This architecture leverages the relatively large amount of processing power at end-hosts (MOSHREF et al., 2016) in the datacenter to implement two key aspects of Predictor (following the description presented in the previous sections): (i) identifying flows at application-level; and (ii) providing network guarantees and dynamically enforcing rates for VMs. Both aspects are discussed next.

First, to perform application-level flow identification, Predictor utilizes Multiprotocol Label Switching (MPLS). More specifically, applications are identified in OpenFlow rules (at the

Figure 4.5 – Server-level architecture of Predictor.



Source: by author (2016).

Open vSwitch) through the label field in the MPLS header. The MPLS label is composed of 20 bits, which allows Predictor to identify 1,048,576 different applications. The complete operation of identifying and routing packets at application-level works as follows. For each packet received from the source VM, the Open vSwitch (controlled via the OpenFlow protocol) in the source hypervisor pushes a MPLS header (four bytes) with an ID in the label field (the application ID of the source VM for intra-application communication or a composite ID for inter-application communication). Subsequent switches in the network use MPLS label and IP source and destination addresses (which may be wildcarded, depending on the possibilities of routing) matching fields to choose the correct output port to forward incoming packets. When packets arrive at the destination hypervisor, the Open vSwitch pops the MPLS header and forwards the packet to the correct VM.

Second, the local controller at each server performs rate-limiting of VMs. More precisely, the local controller dynamically sets the allowed rate for each hosted VM by installing the appropriate rules and rate-limiters at the Open vSwitch. The rate is calculated by Algorithm 4.2, discussed in Section 4.1.2.3. Note that Predictor also reduces rate-limiting overhead when compared to previous schemes (e.g., Silo (JANG et al., 2015), Hadrian (BALLANI et al., 2013), CloudMirror (LEE et al., 2014) and ElasticSwitch (POPA et al., 2013)), for it only rate-limits the source VM while other schemes rate-limit each pair of source-destination VMs.

4.2.3 Results

Next, we explain the behavior of the three schemes we are comparing against each other (Predictor, DevoFlow and the baseline). Then, we show the results of the evaluation: (i) we examine the scalability of employing Predictor on large SDN-based DCNs; and (ii) we verify bandwidth guarantees and predictability.

Comparison. We compare Predictor with the baseline SDN/OpenFlow controller and the

state-of-the-art controller for DCNs (DevoFlow (CURTIS et al., 2011)). Before showing the results, we briefly explain the behavior of Predictor, the baseline and DevoFlow.

In Predictor, bandwidth for intra-application communication is guaranteed at allocation time. For inter-application communication guarantees, we consider two modes of operation, as follows. The first one is called Proactive Inter-Application Communication (PIAC), in which tenants specify in the request all other applications that their applications will communicate with (by using the field $comm_a^{inter}$, as explained in Section 4.1.1). The second one is called Reactive Inter-Application Communication (RIAC), in which rules for inter-application traffic are installed by the controller by either reacting to new flows in the network or receiving communication requests from applications, as defined in Section 4.1.2.1. Note that both modes correspond to the extremes for inter-application communication: while PIAC considers that all inter-application communication is specified at allocation time, RIAC considers the opposite. Furthermore, we highlight that both modes result in the same number of rules in devices, but differ in controller load and flow setup time (results are shown below).

In the baseline, SDN-enabled switches forward to the controller packets that do not match any rule in the flow table (we consider the default behavior of OpenFlow versions 1.3 and 1.4 upon a table-miss event). The controller, then, responds with the appropriate set of rules specifically designed to handle the new flow.

DevoFlow considers flows at the same granularity than the baseline, thus generating a similar number of rules in forwarding devices. However, forwarding devices rely on more powerful hardware and templates to generate rules for small flows without involving the controller. For large flows, DevoFlow has two modes of operation. DevoFlow Triggers requires SDN-enabled switches to identify large flows and ask the controller for appropriate rules for these flows (i.e., only packets of large flows are forwarded to the controller). DevoFlow Statistics, in turn, requires forwarding devices to send the controller uniformly chosen samples (packets), typically at a rate of 1/1000 packets, so that the controller itself identifies and generates rules for large flows. In summary, both DevoFlow modes generate the same number of rules in devices, but differ in controller load and flow setup time.

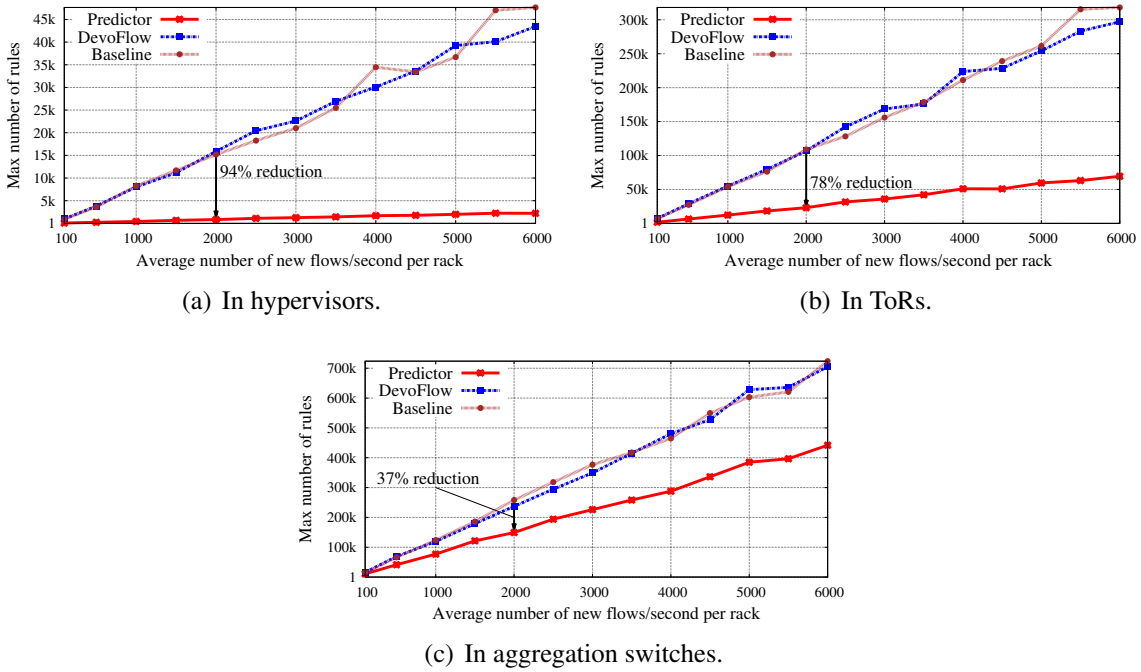
Scalability metrics. We use four metrics to verify the scalability of Predictor in SDN-based datacenter networks: number of rules in flow tables, controller load, impact of reserved control plane bandwidth and flow setup time. These are typically the factors that restrict scalability the most (JARRAYA; MADI; DEBBABI, 2014; POPA et al., 2013).

Reduced number of flow table entries. Figure 4.6 shows how network load (measured in new flows/second per rack) affects flow table occupancy in forwarding devices. More precisely, the plots in Figures 4.6(a), 4.6(b) and 4.6(c) show, respectively, the maximum number of entries observed in our experiments[5] that are required in any hypervisor, ToR and aggregation switch

[5]Since flow table capacity of current available OpenFlow switches ranges from one thousand (NAKAGAWA et al., 2013) to around one million entries (NOVIFLOW, 2015), the observed values during the experiments are within acceptable ranges.

for a given average rate of new flows at each rack (results for core devices are not shown, as they are similar for all three schemes).

Figure 4.6 – Maximum number of rules (that were observed in the experiments) in forwarding devices.



(a) In hypervisors.

(b) In ToRs.

(c) In aggregation switches.

Source: by author (2016).

In all three plots, we see that the average number of arriving flows during an experiment affects directly the number of rules needed in devices. These results are explained by the fact that the number of different flows that pass through forwarding devices is large and may quickly increase due to the elevated number of end-hosts (VMs) and arriving flows in the network. Overall, the increase of the total number of flows requires more rules for the correct operation of the network (according to the needs of tenants) and enables richer communication patterns (representative of cloud datacenters (BALLANI et al., 2013)). Note that the number of rules for the baseline and DevoFlow is similar because (i) they consider flows at the same granularity; and (ii) the same default timeout for rules was adopted for all three schemes.

The results show that Predictor substantially outperforms DevoFlow and the baseline (especially for realistic numbers of new flows in large-scale DCNs, i.e., higher than 1,500 new flows/second per rack (POPA et al., 2013)). More importantly, the curves representing Predictor have a smaller growing factor than the ones for DevoFlow and the baseline. The observed improvement happens because Predictor manages flows at application-level and also wildcards the source and destination addresses in rules when possible (as explained in Section 4.2.2). Predictor reduces the number of rules up to 94% in hypervisors, 78% in ToRs and 37% in aggregation devices. In particular, the reduction in aggregation switches is smaller than in hypervisors and ToRs because more rules need to be installed with destination IP addresses (i.e., they cannot be

installed with wildcards in the IP destination field). In core devices, the reduction is negligible (around 1%), because (a) a high number of flows does not need to traverse core links to reach their destinations, thus the baseline and DevoFlow do not install many rules in core devices, while Predictor installs application-level rules; and (b) Predictor proactively installs rules for intra-application traffic (while other schemes install rules reactively).

Since Predictor considers flows at application-level and inter-application flows may require rules at a lower granularity (e.g., by matching MAC and IP fields), we now analyze how the number of inter-application flows affects the number of rules in forwarding devices for Predictor (previous results considered 20% of inter-application flows, a realistic percentage according to the literature (BALLANI et al., 2013)). Note that we only show results for Predictor because the percentage of inter-application flows does not impact the number of rules in forwarding devices for the baseline and DevoFlow.

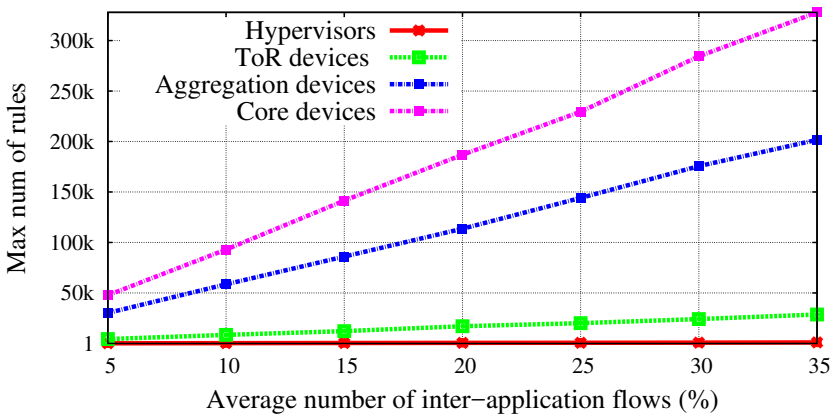
Figure 4.7 shows the maximum number of entries in hypervisors, ToR, aggregation and core devices observed in our experiments (y-axis) for a given percentage of inter-application flows (x-axis), considering an average of 1,500 new flows/second per rack. As expected, we see that the number of rules in devices increases according to the number of inter-application flows. This happens because this type of communication often involves a restricted subset of VMs from different applications. Therefore, Predictor may not install application-level rules for these flows and may end up installing lower-granularity ones (e.g., by matching the IP field). Nonetheless, application-level rules address most of the traffic in the DCN.

Moreover, the number of rules in aggregation and, in particular, in core switches is higher than in ToR devices and in hypervisors. It is so because core switches interconnect several aggregation switches and, as time passes, the arrival and departure of applications lead to dispersion of available resources in the infrastructure. In this context, VMs from different applications (allocated in distinct ToRs) communicate with each other through paths that use aggregation and core switches.

In general, Predictor reduces the number of rules installed in forwarding devices, which can (i) improve hypervisor performance (as measured by LaCurts et al. (LACURTS et al., 2013)); (ii) minimize the amount of TCAM occupied by rules in SDN-enabled switches (TCAMs are a very expensive resource (COHEN et al., 2014) and consume a high amount of power (KREUTZ et al., 2014)); and (iii) minimize the time needed to install new rules in TCAMs, as measured in Section 2.3.2.

Low controller load. As DCNs typically have high load, the controller should handle flow setups efficiently. Figure 4.8 shows the required capacity in number of messages/s for the controller. For better visualization, the y-axis is represented in logarithmic scale, as the values differ significantly for different schemes. As expected, the number of messages sent to the controller increases according to the average number of new flows/s per rack (except for Predictor PIAC and DevoFlow Statistics). The controller must set up network paths and allocate resources according to arriving flows (flows without matching rules in forwarding devices).

Figure 4.7 – Maximum number of rules in forwarding devices for different percentages of inter-application flows for Predictor.



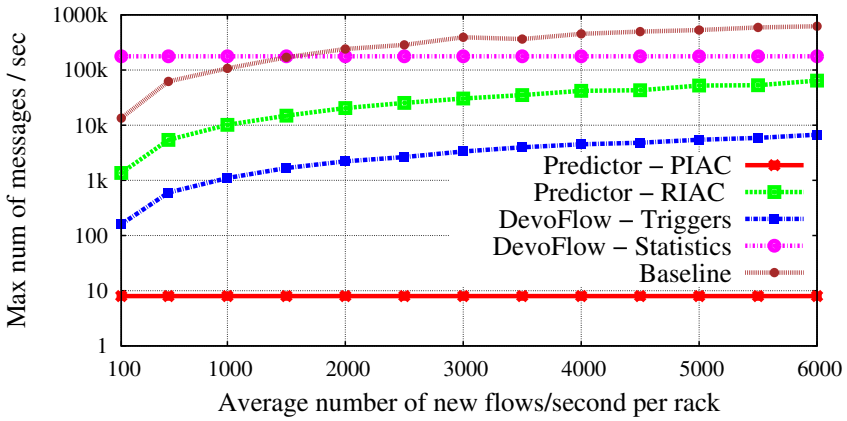
Source: by author (2016).

The baseline imposes a higher load to its controller than other schemes. DevoFlow Statistics requires a regular load to its controller, independently of the number of flows, as the number of messages sent by forwarding devices to the controller depends only on the amount of traffic in the network; in this scheme, devices send to the controller randomly chosen packet samples at a rate of 1/1000 packets. DevoFlow Triggers, in turn, only needs controller intervention to install rules for large flows (at the cost of more powerful hardware at forwarding devices). Thus, it significantly reduces controller load, but may also reduce controller knowledge of (i) network load and (ii) flow table state in SDN-enabled switches. Predictor RIAC proactively installs application-level rules for intra-application communication at allocation time and reactively sends rules for inter-application traffic upon receiving communication requests, which reduces the number of flow requests when compared to the baseline ($\approx 91\%$) but increases it in comparison to DevoFlow Triggers ($\approx 8\%$). Finally, Predictor PIAC receives fine-grained information about intra- and inter-application communication at application allocation time, proactively installing the respective rules when needed. Therefore, controller load can be significantly reduced (i.e., the controller receives requests only when applications are allocated) without hurting knowledge of network state, but at the cost of some burden on tenants (as they need to specify inter-application communication at allocation time for Predictor PIAC).

Recall that the Predictor modes under evaluation correspond to extremes. Therefore, in practice, we expect that Predictor controller load will be between the results shown for PIAC and RIAC. Moreover, we do not show results for controller load varying the number of large flows because results are the same for both modes (and also for the baseline and DevoFlow Statistics). DevoFlow Triggers, however, imposes a higher load to its controller as the number of large flows increases (Figure 4.8 depicted results for a realistic value of 1% of large flows).

So, in both modes, the Predictor controller is aware of most of the traffic (at application-

Figure 4.8 – Controller load.



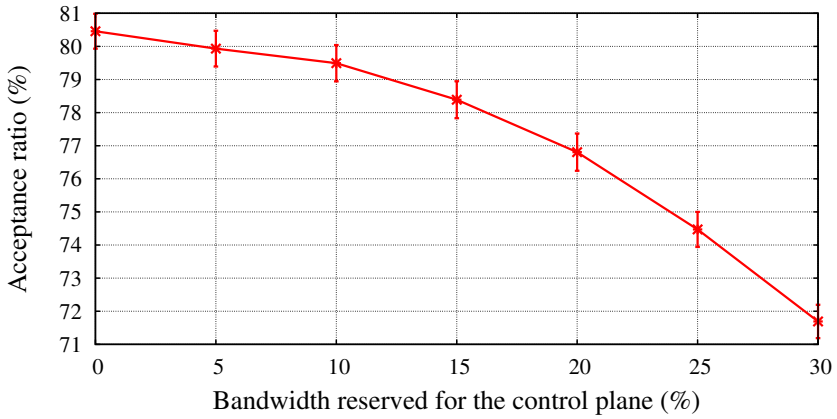
Source: by author (2016).

level) and performs fine-grained control. In contrast, DevoFlow Triggers has knowledge of only large flows (approximately 50% of the total traffic volume (BENSON; AKELLA; MALTZ, 2010)) and DevoFlow Statistics has partial knowledge of network traffic with a high number of messages sent to the controller.

Impact of control plane bandwidth. SDN separates the control and data planes. Ideally, control plane communication is expected to be isolated from data plane traffic, avoiding cross-interference. In this context, the bandwidth it requires varies: the more dynamic the network, the more control plane traffic may be required for updating network state and getting information from forwarding devices. We evaluate the impact of reserving some amount of bandwidth (5%, 10%, 15%, 20%, 25% and 30%) on data plane links to the control plane and compare it with a baseline value of 0% (which represents no bandwidth reservation for the control plane). In other words, we want to verify how the acceptance ratio of applications (y-axis) is affected according to the amount of bandwidth reserved for the control plane (x-axis), since the network is the bottleneck in comparison to computing resources (XIE et al., 2012; CHEN et al., 2014). Figure 4.9 confirms that acceptance ratio of requests decreases according to the amount of bandwidth available for the control plane (clearly, more bandwidth for the control plane means less bandwidth for the data plane). Nonetheless, this reduction is small, even for a worst-case scenario: reserving 30% of bandwidth on data plane links for the control plane results in accepting around 9% fewer requests (regarding the total number of requests).

Therefore, depending on the configuration, SDN may affect DCN resource utilization and, consequently, provider revenue. There are two main reasons: (i) it involves the control plane more frequently (CURTIS et al., 2011); and (ii) SDN-enabled switches are constantly exchanging data with the controller (for both flow setup and the controller to get updated information about network state). In this context, the amount of bandwidth required for the control plane for flow setup is directly proportional to the number of requests to the controller (Figure 4.8). In

Figure 4.9 – Impact of reserved bandwidth for the control plane on acceptance ratio of requests (error bars show 95% confidence interval).



Source: by author (2016).

our experiments, SDN-enabled switches were configured to send the first 128 bytes of the first packet of new flows to the controller (instead of sending the whole packet). With this configuration and for a realistic number of new flows/s per rack (i.e., 1,500 new flows), the bandwidth required by the controller for flow setup for each scheme was at most the following: 1 Mbps for Predictor PIAC, 15 Mbps for Predictor RIAC, 2 Mbps for DevoFlow Triggers, 173 Mbps for DevoFlow Statistics and 166 Mbps for the baseline. Even though Predictor may require more bandwidth for its control plane than DevoFlow in some occasions, it has better knowledge of current network state and does not need customized hardware at forwarding devices.

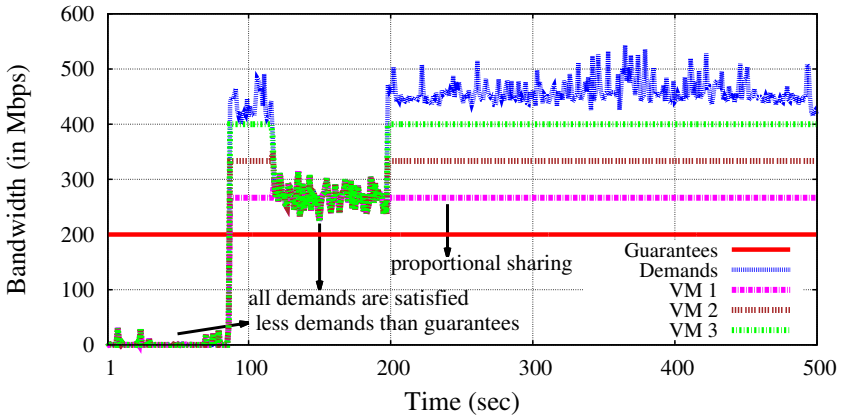
After verifying the feasibility of employing Predictor on large-scale, SDN-based DCNs (i.e., the benefits provided by Predictor, as well as the overheads), we turn our focus to the challenge of bandwidth sharing unfairness. In particular, we show that Predictor (i) proportionally shares available bandwidth; (ii) provides minimum bandwidth guarantees for applications; and (iii) provides work-conserving sharing under worst-case scenarios, achieving both predictability for tenants and high utilization for providers.

Impact of weights on proportional sharing. Before demonstrating that Predictor provides minimum guarantees with work-conserving sharing, we evaluate the impact of weights when proportionally sharing available bandwidth. More specifically, we first want to confirm that available bandwidth is proportionally shared according to the weights assigned to applications and their VMs.

Toward this end, Figure 4.10 shows, during a predefined period of time, three VMs from different applications allocated on a given server with same demands (the demand of each VM is indicated by the blue line in the plot) and guarantees (red line), but different weights (0.2, 0.4 and 0.6, respectively). We verify that, in case that the sum of all three VM demands do not exceed the link capacity (1 Gbps), all VMs have their demands satisfied (e.g., between 1s – 86s

and 118s – 197s), independently of their guarantees. In contrast, if the sum of demands exceed the link capacity, each VM gets a share of available bandwidth (i.e., more than its guarantees) according to its weight (the higher the weight, the more bandwidth it gets). Note that, in this case, the rate of each VM stabilizes (between 87s – 117s and 197s – 500s) because, as the sum of demands exceed the link capacity (and VMs have the same demands and guarantees), the only factor that impacts available bandwidth sharing is the weight. In general, the results show that the use of weights enables proportional sharing.

Figure 4.10 – Proportional sharing according to weights (VM 1: 0.2; VM 2: 0.4; and VM 3: 0.6), considering the same guarantees (200 Mbps) and the same demands for all three VMs allocated on a given server connected through a link of 1 Gbps.

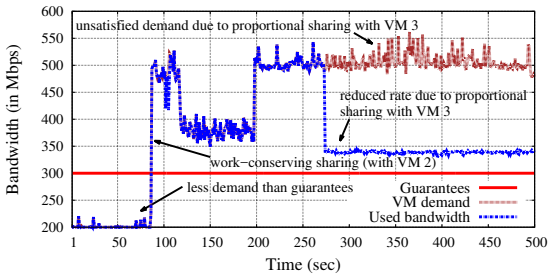


Source: by author (2016).

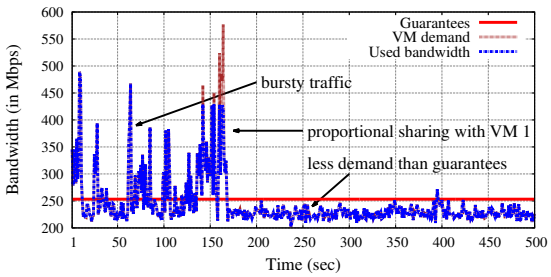
Minimum bandwidth guarantees for VMs. We define it as follows: the VM rate should be (a) at least the guaranteed rate if the demand is equal or higher than the guarantees; or (b) equal to the demand if it is lower than the guarantees. To illustrate this point, we show, in Figure 4.11, the set of VMs (in this case, three VMs from different applications) allocated on a given server during a predefined time period of an experiment. Note that VM 1 [Figure 4.11(a)] and VM 3 [Figure 4.11(c)] have similar guarantees, but receive different rates ("used bandwidth") when their demands exceed the guarantees (e.g., after 273s). This happens because they have different network weights (0.17 and 0.59, respectively), and the rate is calculated considering the demands, bandwidth guarantees, network weight and residual bandwidth. Moreover, we see (from Figures 4.10 and 4.11) that VMs may not get the desired rate to satisfy all of their demands instantaneously (when their demands exceed their guarantees) because (i) the link capacity is limited; and (ii) available bandwidth is proportionally shared among VMs.

In summary, we see that Predictor provides minimum bandwidth guarantees for VMs, since the actual rate of each VM is always equal or higher than the minimum between the demands and the guarantees. Therefore, applications have minimum bandwidth guarantees and, thus, can achieve predictable network performance.

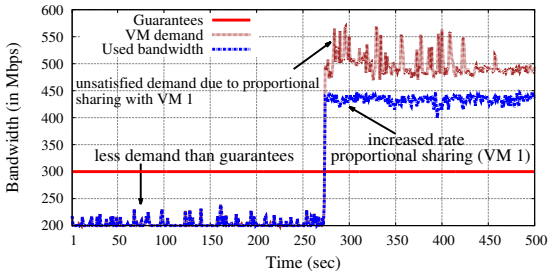
Figure 4.11 – Bandwidth rate achieved by the set of VMs allocated on a given server during a predefined period of time.



(a) VM 1.



(b) VM 2.



(c) VM 3.

Source: by author (2016).

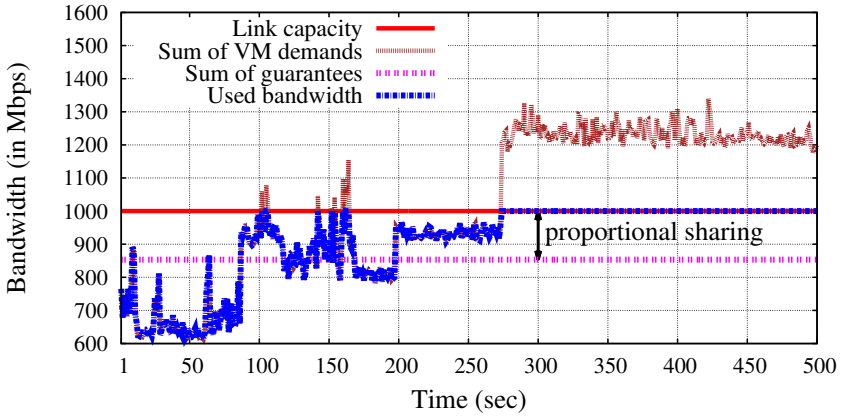
Work-conserving sharing. Bandwidth which is not allocated, or allocated but not currently used, should be proportionally shared among other VMs with more demands than their guarantees (according to the weights of each application, using Algorithm 4.2). Figure 4.12 shows the aggregate bandwidth[6] on the server holding the set of VMs in Figure 4.11. In these two figures, we verify that Predictor provides work-conserving sharing in the network, as VMs can receive more bandwidth (if their demands are higher than their guarantees) when there is spare bandwidth. Thus, providers can achieve high network utilization. Furthermore, by providing work-conserving sharing, Predictor offers high responsiveness[7] to changes in bandwidth

[6]Note that Predictor considers only bandwidth guarantees when allocating VMs (i.e., it does not take into account temporal demands). Therefore, even though the sum of temporal demands of all VMs allocated on a given server may exceed the server link capacity, the sum of bandwidth guarantees of these VMs will not exceed the link capacity.

[7]Responsiveness is a critical aspect of cloud guarantees (MOGUL; POPA, 2012).

requirements of applications.

Figure 4.12 – Work-conserving sharing on the server holding the set of VMs from Figure 4.11.



Source: by author (2016).

In general, Predictor provides significant improvements over DevoFlow, as it allows high utilization in the network for providers and predictability with guarantees for tenants and their applications. As a side effect, Predictor may have higher controller load than DevoFlow (the cost of providing fine-grained management in the network without imposing to tenants the burden of specifying inter-application communication at allocation time).

Generality of the results. Like in IoNCloud, we believe that the obtained results are generalizable to most workloads and real cloud platforms (e.g., Amazon EC2 and Microsoft Azure). Despite the fact that we could not obtain real traces from public clouds for pragmatic reasons, we reversed engineered the traces used by Kandula et al. (2009) and Benson, Akella and Maltz (2010), generating a mixed workload (from different types of applications, such as MapReduce and web services). Moreover, we chose a tree-topology to evaluate Predictor because we wish to mimic current datacenter networks (JANG et al., 2015) and to follow related work (BALLANI et al., 2013; BALLANI et al., 2011; XIE et al., 2012).

We highlight that the used workload represents a middle-case (a realistic demand). We did not evaluate the best- and worst-case scenarios because they are unrealistic, i.e., they are not expected to happen in practice. Nonetheless, we briefly discuss them in the following manner. The best-case happens when all VMs generate only intra-application communication. Since Predictor manages flows at application-level, it would result in (i) the minimum number of rules and rate-limiters necessary to correctly forward traffic and ensure bandwidth guarantees for applications; and (ii) the lowest controller load. In contrast, the worst-case occurs when all VMs generate only inter-application communication with a specific traffic pattern that requires rules to be installed at VM-to-VM granularity (not at application-level). This would result in flow table size being similar to DevoFlow and the baseline and in controller load being similar to the baseline. In this case, Predictor would still provide minimum bandwidth guarantees and

work-conserving sharing (as opposed to DevoFlow and the baseline).

4.3 Discussion

After evaluating Predictor, we discuss its generality and limitations.

Application-level flow identification. In our proof-of-concept implementation, Predictor identifies flows at application-level through the MPLS label (application ID with 20 bits). Therefore, it needs a MPLS header in each packet (adding four bytes of overhead). In practice, there are at least two other options to provide such functionality. First, when considering the matching fields defined by OpenFlow, application-level flows could also be identified by utilizing IEEE standard 802.1ad (Q-in-Q) with double VLAN tagging. The advantage of double tagging is a higher number of IDs available (24 bits), while the drawback is an overhead of eight bytes (two VLAN headers) per packet. Second, application-level flows could be identified by using OpenFlow Extensible Match (OXM)[8] to define a unique match field for this purpose. Nonetheless, this method is less flexible, as it requires (i) switch support for OXM; and (ii) programming to add a new matching field in forwarding devices.

Topology-awareness. Even though Algorithm 4.1 was specifically designed for tree-like topologies, the proposed strategy is topology-agnostic. Therefore, we simply need to replace Algorithm 4.1 to employ Predictor in DCNs with other types of interconnections. We used a tree-like placement algorithm in this paper for three reasons. First, currently most providers implement DCNs as (oversubscribed) trees, since they can control the oversubscription factor more easily with this type of structure (in order to achieve economies of scale). Second, by using an algorithm specially developed for a particular structure, we can enable better use of resources. Thus, we show more clearly the benefits and overheads of the proposed strategy. Third, we used tree topologies for the sake of explanation, as it is easier to explain and to understand how bandwidth is allocated and shared among VMs of the same application in this kind of topology (e.g., in Figure 4.3) than, for example, in random graphs.

Dynamic rate allocation with feedback from the network. The designed work-conserving algorithm does not take into account network feedback provided by the OpenFlow module. This design choice was deliberately made; we aim at reducing management traffic in the network, since DCNs are typically oversubscribed networks with scarce resources (XIE et al., 2012). Nonetheless, the algorithm could be extended to consider feedback, which would further help controlling the bandwidth used by flows traversing congested links.

Application ID management. Predictor controller assigns IDs for applications (in order to identify flows at application-level) upon allocation and releases IDs upon deallocation. Therefore, ID management is straightforward, as Predictor has full control over which IDs are in use at each period of time.

[8] OXM was introduced in OpenFlow version 1.2 and currently is supported by several commercial forwarding devices.

Application request abstraction. Currently, Predictor only supports the hose model (DUFFIELD et al., 1999b). Nonetheless, it can use extra control applications (one for each abstraction) (i) to parse requests specified with other models (e.g., TAG (LEE et al., 2014) and hierarchical hose model (BALLANI et al., 2013)); and (ii) to install rules accordingly. With other abstractions, Predictor would employ the same sharing mechanism (Section 4.1.2). Thus, it would provide the same level of guarantees.

4.4 Summary

Datacenter networks are typically shared in a best-effort manner, resulting in interference among applications. SDN may enable the development of a robust solution for interference. However, the scalability of SDN-based proposals is limited, because of flow setup time and the number of entries required in flow tables.

We have introduced Predictor in order to scalably provide predictable and guaranteed performance for applications in SDN-based DCNs. Performance interference is addressed by using two novel SDN-based algorithms. Scalability is tackled as follows: (i) flow setup time is reduced by proactively installing rules for intra-application communication at allocation time (since this type of communication represents most of the traffic in DCNs); and (ii) the number of rules in forwarding devices is minimized by managing flows at application-level. Evaluation results show the benefits of Predictor. First, it provides minimum bandwidth guarantees with work-conserving sharing (successfully solving performance interference). Second, it eliminates flow setup time for most traffic in the network and significantly reduces flow table size (up to 94%), while keeping low controller load (successfully dealing with scalability of SDN-based DCNs). In future work, we intend to evaluate Predictor on a testbed (such as CloudLab (CloudLab, 2016)).

5 PACKER: MINIMIZING MULTI-RESOURCE FRAGMENTATION

IoNCloud and Predictor provide predictable and guaranteed network performance. However, they neglect non-network resources (e.g., CPU, memory and disk), resulting in fragmentation of these resources. As a matter of fact, applications perform rich and complex tasks (DOGAR et al., 2014), with different demands for each resource type (GHODSI et al., 2011). Most applications running on datacenters have multiple stages, where a subsequent stage can only start when the previous stage finishes (XIE et al., 2012). Consequently, any resource that becomes a bottleneck and delays a stage (especially performance interference in the network (BALLANI et al., 2013)) may slow down the entire application.

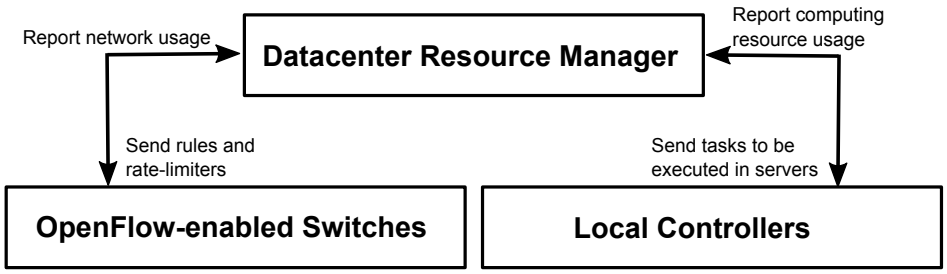
To address the aforementioned challenge, we propose Packer, a scheme that aims at (i) providing the same level of network guarantees than Predictor; and (ii) minimizing multi-resource fragmentation and, consequently, increasing datacenter utilization. To achieve these goals, Packer considers multiple types of resources to admit (allocate) applications in the datacenter, so that tenants can request and receive the necessary amount of resources that their applications need to correctly execute and finish without delay. Furthermore, by considering multiple resources, providers can avoid over-allocation (GRANDL et al., 2014) and make better use of the whole infrastructure, allocating more applications and increasing revenue.

This chapter is structured as follows. We first present the design of Packer in Section 5.1. Then, we describe its evaluation in Section 5.2. Finally, we discuss the generality and limitations in Section 5.3 and close the chapter with a brief summary in Section 5.4.

5.1 Design

Packer implements a novel strategy for minimizing multi-resource fragmentation and for providing predictable and guaranteed network performance in large-scale cloud datacenters. Figure 5.1 shows an overview of Packer. The scheme is composed of three components: datacenter resource manager (DRM), OpenFlow-enabled switches and one local controller per server. They are discussed next.

Figure 5.1 – Packer overview.



Source: by author (2016).

Datacenter resource manager (DRM). It is responsible for (i) allocating applications in the datacenter; and (ii) handling global events (e.g., bandwidth enforcement throughout the entire network for applications). More specifically, it receives an application request (in the form of a TI-MRA specification) and employs a novel resource allocation strategy (described in Section 5.1.2) to determine the set of resources to be used by the application. Then, it sends the application tasks to the servers that will execute them (as determined by the allocation strategy) and, via OpenFlow, configures switches (with rules and rate-limiters) to provide connectivity and network performance guarantees for the application. Furthermore, the DRM periodically receives up-to-date information regarding resource usage from local controllers at servers and from OpenFlow switches.

OpenFlow switches. These devices are responsible for forwarding traffic according to the instructions received from the DRM. They receive rules and rate-limiters to correctly handle traffic and enforce bandwidth for applications. Moreover, they periodically report resource usage statistics to the DRM.

Local controllers (LCs) at servers. They are part of the resource monitoring mechanism utilized in Packer (described in Section 5.1.4). LCs are responsible for (i) monitoring multi-resource usage at servers and reporting it to the DRM; (ii) enforcing allocations; and (iii) reacting to local events (e.g., dealing with congested resources inside their respective server).

We detail Packer in the following manner. We first present the novel abstraction (called Time-Interleaved Multi-Resource Abstraction – TI-MRA) used for applications in Section 5.1.1. Then, we utilize TI-MRA as input for the new allocation algorithm (Section 5.1.2) and describe the strategy used for providing predictable and guaranteed network performance (Section 5.1.3). Finally, we detail the resource monitoring mechanism in Section 5.1.4. The notations used throughout this chapter are presented in Table 5.1.

5.1.1 Time-Interleaved Multi-Resource Abstraction (TI-MRA)

Prior work has designed abstractions expressed as physical network models (i.e., the hose model) (BALLANI et al., 2011; XIE et al., 2012; JANG et al., 2015), two-level trees (hierarchical hose) (BALLANI et al., 2011; BALLANI et al., 2013) or based on communication patterns (TAG) (LEE et al., 2014). However, they focus on the network and neglect other resources. In particular, the hose model (used by most related work) does not accurately capture the network requirements of applications with complex traffic interactions (LEE et al., 2014).

An effective abstraction is expected to consider two purposes. The first is to allow tenants to specify their application requirements in a simple and accurate manner. The second is to allow providers to minimize over-allocation (i.e., allocating the correct amount of resources required by applications), which may increase the percentage of allocated applications and, consequently, may improve datacenter throughput.

Based on these purposes and the limitations of prior work, we propose a novel abstraction

Table 5.1 – Notations adopted throughout this chapter.

Symbol	Description
A	Set of application requests
G^a_{TI-MRA}	TI-MRA graph of application $a \in A$
V^a	Set of nodes of application $a \in A$ ($V^a = K^a \cup C^a$)
K^a	Set of tasks of application $a \in A$ ($K^a \subseteq V^a$)
C^a	Set of cloud services used by app $a \in A$ ($C^a \subset V^a$)
E^a	Set of edges (dependencies between nodes) of app $a \in A$
T^a	Discrete time instants of application $a \in A$ ($T^a \subseteq \mathcal{T}$)
w^a	Weight of application $a \in A$
N	Set of all infrastructure nodes ($N = S \cup \mathcal{J}$)
S	Set of servers in the datacenter infrastructure ($S \subseteq N$)
$\mathcal{J}$	Set of services available in the datacenter ($\mathcal{J} \subset N$)
$\mathcal{L}$	Set of links in the datacenter network
$\mathcal{T}$	Discrete time instants of the infrastructure
$\mathcal{P}$	Set of all paths available in the network
$\mathcal{P}(n_1, n_2)$	Set of paths from src node $n_1 \in N$ to dest node $n_2 \in N$
$w^{\mathtt{r}}$	Weight of $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}, \mathtt{BAND}\}$
$\delta(v, \mathtt{r}, t)$	Amount of resource $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$ required by node $v \in V^a$ at time $t \in T^a$ for app $a \in A$
$\sigma(e = (u, v), t)$	Bandwidth for communication between nodes $u, v \in V^a \mid e = (u, v) \in E^a$ at time $t \in T^a$ for application $a \in A$
$\mathcal{M}_n(v)$	Node $n \in N$ that holds the node $v \in V^a$ of app $a \in A$
$\mathcal{M}_e(u, v)$	Infrastructure path ($\mathcal{P}(\mathcal{M}_n(u), \mathcal{M}_n(v))$) used for communication between nodes $u, v \in V^a$ of app $a \in A$
$\mathcal{A}(n)$	Tasks running at infrastructure node $n \in N$
$\mathcal{B}(l)$	Total capacity (bandwidth) of link $l \in \mathcal{L}$
$\mathcal{C}(l)$	Communications (edges in the TI-MRA) using link $l \in \mathcal{L}$
$\mathcal{D}(u)$	Application that task u belongs to
$\mathcal{E}(v)$	Nodes that node $v \in V^a \mid a \in A$ depends on
$\mathcal{Q}(n, \mathtt{r}, t)$	Amount of available resource $\mathtt{r}$ on $n \in N$ at time $t \in \mathcal{T}$
$\mathcal{R}(n, \mathtt{r})$	Capacity of infrastructure node $n \in N$ for resource type $\mathtt{r}$

Source: by author (2016).

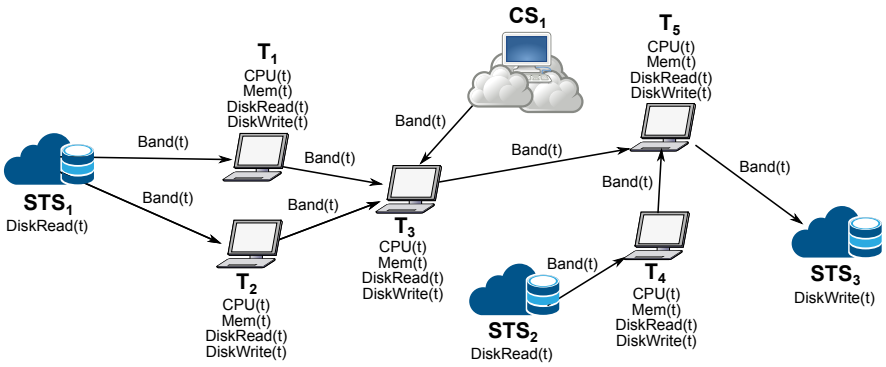
for applications, called Time-Interleaved Multi-Resource Abstraction (TI-MRA). TI-MRA allows the specification of not only network demands but also other types of resources. TI-MRA leverages tenants' knowledge of their applications to yield a flexible representation of the applications' resource consumption. It uses the same principle of (a) temporal bandwidth requirements in TIVC (XIE et al., 2012), but extends it to all kinds of resources; and (b) communication patterns in TAG (LEE et al., 2014). Furthermore, it also takes into account dependencies other than among tasks (such as between tasks and cloud services), in order to optimize the use of resources. The intuition is that TI-MRA allows a flexible representation of application requirements rather than imposing a predefined abstraction (e.g., the hose model) for applications to map their requirements to.

TI-MRA extends the concept of time-varying graphs (TVGs) (WEHMUTH; ZIVIANI; FLEURY, 2015) to represent temporal demands of multiple resources. A TI-MRA graph of application $a \in A$ is represented as $G^a_{\mathtt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$, with the terms being defined as follows: $V^a = K^a \cup C^a$ is the set of application nodes, composed of tasks (K^a) and cloud services required by the application (C^a); E^a is the set of edges, representing the dependencies between nodes; T^a is the set of discrete time instants, from the time the first node of

application a begins its computation to the time the last node finishes; $w^a \in [0, 1]$ is defined by the provider (based on the tenant's payment) and indicates the weight of application a, so that residual resources (unallocated, or reserved resources for an application and not currently being used) can be proportionally shared among applications with more demands than their guarantees (work-conservation); and $\delta(v, \mathtt{r}, t) \in \mathbb{R}^+$ returns the demand of node $v \in V^a$ at time $t \in T^a$ for resource $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$. The last function, $\sigma(e = (u, v), t) \in \mathbb{R}^+$, denotes the bandwidth necessary for communication between nodes $u \in V^a$ and $v \in V^a \mid u \neq v$ at time $t \in T^a$, for $e = (u, v) \in E^a$. Note that we do not consider moving nodes and edges across time. This does not impact the generality of TI-MRA because when a node or edge has no demand for a given resource, the call for the respective function ($\delta(v, \mathtt{r}, t)$ or $\sigma(e = (u, v), t)$) returns zero.

An example of TI-MRA is shown in Figure 5.2. The figure depicts a simple application composed of five tasks and temporal resource requirements for CPU, memory, disk I/O write, disk I/O read and bandwidth. In this example, tasks T_1 and T_2 get their input data from storage service STS_1; T_3 depends on tasks T_1 and T_2 and on data sent from cloud service CS_1; T_4 reads data from storage service STS_2 to perform its computation; and T_5 depends on tasks T_3 and T_4 and stores the final result in STS_3. Moreover, note that edges (links representing the exchange of data) are unidirectional (different amounts of bandwidth for sending and receiving data). Having two links instead of a single bidirectional link avoids over-allocation and bandwidth wastage.

Figure 5.2 – TI-MRA of a simple application composed of five tasks (T_1 to T_5), where tasks read and write data from/to storage services (STS_1, STS_2 and STS_3) and use cloud service CS_1.



Source: by author (2016).

Producing TI-MRA models. TI-MRA can be used not only by tenants who have a deep understanding of their application demands, but also by users who do not know it in advance. The former can tune resource demands according to the application requirements, possibly reducing costs (avoiding over-allocation) without impact on performance. The latter, in turn, can specify only peak demands for resources (i.e., a constant temporal function). This would be similar to the hose model specification. Alternatively, the same strategy employed in CloudMirror (LEE

et al., 2014) for generating TAG models could be used here: application templates for TI-MRA could be provided as a library for users through the extension of cloud orchestration systems like OpenStack Heat and AWS CloudFormation.

5.1.2 Allocation Strategy

The problem of allocating applications considering multiple types of resources in datacenters can be reduced to multi-dimensional bin packing (also called vector bin packing – VBP) (GRANDL et al., 2014), which is NP-Hard for every dimension d and APX-Hard for $d \geqslant 2$ (PANIGRAHY et al., 2011). Given balls (application tasks) and bins (servers) with sizes for each property (resource) considered, VBP assigns the balls to bins according to an optimization objective. In our case, the goal is to reduce fragmentation and over-provisioning and, consequently, improve the percentage of allocated applications and their tasks.

In case applications are constrained by a single resource (e.g., network), the problem becomes, in essence, a one dimensional bin packing (PANIGRAHY et al., 2011). However, cloud applications are typically constrained by multiple resources, including network (JANG et al., 2015) and CPU (OUSTERHOUT et al., 2015). Moreover, the network is a distributed resource (composed of several links); therefore, the amount of resources consumed depends on bandwidth demands of tasks as well as their location in the infrastructure (the whole path used for communication must be taken into account), which increases the difficulty in efficiently optimizing the use of resources.

Problem definition. The process of multi-resource allocation is formally defined as follows. The TI-MRA specification of application $a \in A$ is given by $G^a_{TI-MRA} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$. A node $v \in V^a$ can be either a task or a cloud service and is mapped to an infrastructure node $n \in N$. Each task $k \in K^a \mid K^a \subseteq V^a$ is assigned to an infrastructure server ($s \in S \mid S \subseteq N$) by mapping $\mathcal{M}_n : K^a \rightarrow S,\ \forall a \in A$ (Equation 5.1). Each cloud service $c \in C^a \mid C^a \subset V^a$ required by application $a \in A$, in turn, is part of the set of services ($\mathcal{J} \mid \mathcal{J} \subset N$) available in the platform (offered by either the provider or a tenant) and is assigned to a node $j \in \mathcal{J}$ that runs the requested service by mapping $\mathcal{M}_n : C^a \rightarrow \mathcal{J},\ \forall a \in A$ (Equation 5.2).

$$\mathcal{M}_n(k) \in S \mid k \in K^a \text{ or} \tag{5.1}$$

$$\mathcal{M}_n(c) \in \mathcal{J} \mid c \in C^a \tag{5.2}$$

Each dependency between nodes (i.e., edges in the TI-MRA graph, specified in set E^a) is mapped to a single path between the corresponding infrastructure nodes (servers for tasks and cloud services for services needed by the application). The assignment is defined by mapping $\mathcal{M}_e : E^a \rightarrow \mathcal{P},\ \forall a \in A$, where $\mathcal{P}$ denotes the set of all available paths in the network, such that

for all $e = (u, v) \in E^a,\ \forall a \in A$:

$$\mathcal{M}_e(u, v) \in \mathcal{P}\left(\mathcal{M}_n(u), \mathcal{M}_n(v)\right) \tag{5.3}$$

The nodes in V^a run for at most T^a discrete time units to perform the computation required by application $a \in A$ and communicate among themselves using links $e \in E^a$. Each node $v \in V^a$ presents temporal demands for computing resource $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$ ($\delta(v, \mathtt{r}, t)$), $\forall t \in T^a$. Furthermore, each edge $e = (u, v) \in E^a$ between communicating nodes u and v (such that $u \neq v$) presents temporal bandwidth demands: $\sigma(e = (u, v), t), \forall t \in T^a$.

A valid allocation is constrained by the amount of available resources. More specifically, a cloud service $j \in \mathcal{J}$ will only perform the task required by node $c \in C^a \mid a \in A$ if it has enough available resources to satisfy the demands. Similarly, a task can only be allocated in a server if the server has enough available capacity for each type of resource being considered. Given $\mathcal{A}(n)$ a function that returns all tasks running at infrastructure node $n \in N$, $\mathcal{R}(n, \mathtt{r})$ a function that returns the capacity of infrastructure node $n \in N$ for resource $\mathtt{r}$ and $\mathcal{T}$ the discrete time instants of the infrastructure, the constraint for computing resources is defined as follows:

$$\sum_{v \in \mathcal{A}(n)} \delta(v, \mathtt{r}, t) \leqslant \mathcal{R}(n, \mathtt{r}) \qquad \forall n \in N,\ \forall \mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\},\ \forall t \in \mathcal{T} \tag{5.4}$$

The network is a special case, since it is a distributed resource. Specifically, bandwidth must be taken into account at the entire path used for each communication between application nodes and the amount of allocated bandwidth at a link l must not exceed the total capacity of l (Equation 5.5).

$$\sum_{e=(u,v) \in \mathcal{C}(l)} \sigma(e, t) \leqslant \mathcal{B}(l) \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T} \tag{5.5}$$

where $\mathcal{L}$ represents the set of links in the datacenter network, $\mathcal{C}(l)$ returns the set of communications (edges in TI-MRA graphs of applications) using link l and $\mathcal{B}(l)$ returns the total capacity (bandwidth) of link l.

Note that the above constraints are non-linear, in particular functions $\mathcal{A}$ and $\mathcal{C}$ (since they depend on the allocation of application nodes in infrastructure nodes). Efficient solvers are known only for some non-linear problems, such as the quadratic assignment problem. However, even when placement considerations are eliminated, the problem of VBP is APX-Hard (WOEGINGER, 1997) and re-solving it whenever new applications arrive worsens the process. Consequently, finding the optimal solution is expensive and requires a lot of time. Unfortunately, large-scale cloud datacenters require the allocation process to be performed as fast as possible, since they typically have high rate of application arrival and departure.

Algorithm. VBP has several proposed heuristics (PANIGRAHY et al., 2011). However, those heuristics cannot be used in datacenters without substantial modification, as they (i) assume that all input (i.e., applications) is known a priori, whereas we need to cope with online arrival of applications; (ii) consider "balls" of a fixed size, while applications have time-varying demands; and (iii) do not consider a distributed resource such as the network (with paths composed of multiple links). Therefore, we design a novel algorithm to efficiently allocate applications in cloud datacenters. The key principle is minimizing multi-resource fragmentation, thus improving the ratio of allocated applications and, consequently, maximizing datacenter utilization and provider revenue.

Algorithm 5.1 allocates one application at a time, as requests arrive. It receives as input the datacenter infrastructure ($\langle N, \mathcal{L}, \mathcal{T}, \mathcal{P} \rangle$) and the TI-MRA specification of an application $a \in A$ ($G^a_{\texttt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$). First, it calculates available resources (CPU, memory, disk I/O write and disk I/O read) in servers (lines 1 – 3) and available bandwidth in links (lines 4 – 6). Since each node of application a may have a different duration, available resources in the infrastructure are calculated for T^a time units (the duration of a).

After that, nodes in V^a are sorted sequentially according to their initial execution time and the nodes they depend on (line 7). Based on the sorted list of nodes (`sNodes`), the algorithm gets one node v at a time (line 8), initializes as empty the list of infrastructure nodes with enough available resources to hold node v (line 9) and calculates, based on our novel metric shown in Equation 5.6, the score of v on infrastructure nodes (lines 10 – 15). The metric works as follows. Equation 5.6 seeks to maximize the score achieved by both computing and network resources according to their current utilization level (calculated in Equations 5.7 and 5.8, respectively) in case there are enough available resources (i.e., node $n \in N$ and link $l \in \mathcal{L}$, which v would use for communication with the application nodes it depends on if it were allocated on n, have enough available resources at all times $t \in T^a$). Otherwise, it returns $-\infty$. For each computing resource $R = \{\texttt{CPU}, \texttt{MEM}, \texttt{IO\_WRITE}, \texttt{IO\_READ}\}$ (Equation 5.7) and for bandwidth `BAND` (Equation 5.8), the metric subtracts the resource demand from the respective available resource, raises the resulting value by the power of 3, multiplies it by the amount of the respective available resource and multiplies it again by the weight associated to the resource ($w^{\texttt{r}}$). In particular, $w^{\texttt{r}}$ is dynamically defined as being inversely proportional to the current utilization level of $\texttt{r}$ and is calculated as $w^{\texttt{r}} = 1 - (\frac{\texttt{util(r)}}{\sum_{s \in R \cup \{\texttt{BAND}\}} \texttt{util(s)}}), \forall \texttt{r} \in R \cup \{\texttt{BAND}\}$. That is, the higher the current utilization of $\texttt{r}$, the lower its weight. This way, the metric prioritizes resources with lower utilization.

$$\texttt{S}[n,v] = \begin{cases} \texttt{S}_\texttt{C}[n,v] + \texttt{S}_\texttt{B}[n,v] & \delta(v,\texttt{r},t) \leqslant \mathcal{Q}(n,\texttt{r},t) \text{ and } \sigma((u,v),t) \leqslant \mathcal{Q}(l,\texttt{BAND},t), \\ & \forall \texttt{r} \in R, \forall u \in \mathcal{E}(v), \forall t \in T^a, \forall l \in \mathcal{M}_e(u,v); \\ -\infty & \text{otherwise.} \end{cases} \tag{5.6}$$

$$\mathtt{S_C}[n,v] = \sum_{t \in T^a} \sum_{\mathtt{r} \in R} w^{\mathtt{r}} \times (\mathcal{Q}(n,\mathtt{r},t) - \delta(v,\mathtt{r},t))^3 \times \mathcal{Q}(n,\mathtt{r},t) \tag{5.7}$$

$$\mathtt{S_B}[n,v] = \sum_{t \in T^a} \sum_{u \in \mathcal{E}(v)} \sum_{l \in \mathcal{M}_e(u,v)} w^{\mathtt{BAND}} \times (\mathcal{Q}(l,\mathtt{BAND},t) - \sigma((u,v),t))^3 \times \mathcal{Q}(l,\mathtt{BAND},t) \tag{5.8}$$

Note that the metric described here uses normalized values (for resource requirements of applications as well as residual resources in the datacenter infrastructure) by the capacity of the node/link being considered.

The next step is the allocation of v and its communication dependencies (edges with v as the destination node in the TI-MRA graph) in lines 16 – 23, according to Equations 5.1 – 5.5. Function `GetNodeWithBestScore` returns the infrastructure node with the best (maximum) score in the list `FeasibleNodes` for holding node v (line 16). In case no infrastructure node has enough available resources to hold v, the algorithm returns a failure code and application a is discarded (line 17). Otherwise, node v is allocated at infrastructure node n (line 19) and, since nodes that v depends on are already allocated (i.e., dependencies are allocated first, according to the sorted list of nodes), bandwidth for communication between v and its dependencies is also allocated (lines 20 – 22). When all nodes and edges in the TI-MRA graph of application a are successfully allocated, the algorithm returns a success code and finishes (line 25).

Algorithm 5.1: Multi-Resource Allocation Algorithm.

Input : Datacenter infrastructure $\langle N, \mathcal{L}, \mathcal{T}, \mathcal{P} \rangle$, Application a represented by $G^a_{\mathtt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$
Output: Success/Failure code

1 **foreach** *Infrastructure node* $n \in N$ **do**
2 $\quad \mathcal{Q}[n,\mathtt{r},t] \leftarrow \mathcal{R}(n,\mathtt{r}) - \sum_{v \in \mathcal{A}(n)} \delta(v,\mathtt{r},t),\ \forall \mathtt{r} \in \{\mathtt{CPU},\mathtt{MEM},\mathtt{IO\_WRITE},\mathtt{IO\_READ}\},\ \forall t \in T^a \mid T^a \subseteq \mathcal{T}$;
3 **end**
4 **foreach** *Infrastructure link* $l \in \mathcal{L}$ **do**
5 $\quad \mathcal{Q}[l,\mathtt{BAND},t] \leftarrow \mathcal{B}(l) - \sum_{e=(u,v) \in \mathcal{C}(l)} \sigma(e,t),\ \forall t \in T^a \mid T^a \subseteq \mathcal{T}$;
6 **end**
7 `sNodes` $\leftarrow$ `SortNodes` (V^a);
8 **foreach** $v \in$ `sNodes` **do**
9 $\quad$ `FeasibleNodes` $\leftarrow \emptyset$;
10 $\quad$ **foreach** $n \in N$ **do**
11 $\quad\quad$ `S`$[n,v] \leftarrow$ calculate score according to Equations 5.6, 5.7 and 5.8;
12 $\quad\quad$ **if** `Score`$[n,v] \neq -\infty$ **then**
13 $\quad\quad\quad$ `FeasibleNodes` $\leftarrow$ `FeasibleNodes` $\cup \{n\}$;
14 $\quad\quad$ **end**
15 $\quad$ **end**
16 $\quad n \leftarrow$ `GetNodeWithBestScore(FeasibleNodes)`;
17 $\quad$ **if** *n is null* **then return** failure code;
18 $\quad$ **else**
19 $\quad\quad \mathcal{M}_n(v) \leftarrow n$;
20 $\quad\quad$ **foreach** $u \in \mathcal{E}(v)$ **do**
21 $\quad\quad\quad \mathcal{M}_e(u,v) \leftarrow p \mid p \in \mathcal{P}(\mathcal{M}_n(u), \mathcal{M}_n(v))$;
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 **end**
25 **return** success code;

5.1.3 Network Sharing Strategy

The network sharing strategy has two objectives: (i) providing predictable and guaranteed network performance for applications, in order to avoid performance interference (JANG et al., 2015); and (ii) achieving work-conserving sharing, so that applications have the possibility of using more bandwidth than their guarantees when needed and providers can achieve high network utilization.

To achieve these goals, we leverage the paradigm of SDN to dynamically configure the network, in order to enforce bandwidth guarantees and to provide work-conserving sharing. The strategy works as follows. According to the output of Algorithm 5.1 for application $a \in A$, the DRM performs two actions. First, it sends each task of a to the local controller of its selected server (as LCs manage and enforce resource allocation at servers). Second, it installs rules and rate-limiters in forwarding devices to guarantee connectivity and bandwidth for communication between these nodes.

In addition to ensuring a base level of guaranteed rate for applications, the strategy can proportionally share available bandwidth among applications with more demands than their guarantees. Towards this end, local controllers run an algorithm to periodically set the allowed rate for each allocated application node. Algorithm 5.2 aims at enabling smooth response to bursty traffic (since traffic in DCNs may be highly variable over short periods of time (ABTS; FELDERMAN, 2012)). It receives as input the infrastructure node $n \in N$ that it belongs to, the current time $t \in \mathcal{T}$, current bandwidth demands of application nodes allocated at n (which are determined by monitoring socket buffers, similarly to Mahout (CURTIS; KIM; YALAGANDULA, 2011)) and temporal bandwidth requirements of these nodes (specified in the request). First, the algorithm initializes as empty the list of application nodes with more bandwidth demands than the value specified in the request (line 1). Then, for each application node v allocated at n (line 2), the minimum rate between (i) the specified demand at time t ($\sigma(\sum(v, *), t)$, which represents the sum of bandwidth required by node v for communication with all nodes that depend on v at time t) and (ii) the current demand of v ($d[v]$) is assigned to `nRate` (line 3). If the current demand is higher than the specified demand, the node is added to the list of nodes with more demands than their guarantees (called `hungryNodes`, in line 4).

Then, the algorithm calculates the residual bandwidth ($\mathcal{Q}(n, \texttt{BAND}, t)$) of the link connecting server n to its top-of-rack (ToR) switch at time t (line 6). The residual bandwidth is calculated by subtracting from the link capacity the rate assigned to the application nodes (in line 3). The last step establishes the bandwidth rate for application nodes with more demands than their guarantees, if there is available bandwidth (lines 7 – 14). The rate of each node $v \in$ `hungryNodes` (in line 10) is determined by adding `nRate[v]` (initialized in line 3) and the minimum bandwidth between (i) the difference of the current demand ($d[v]$) and the rate (`nRate[v]`); and (ii) the proportional share of residual bandwidth the application node can receive according to its weight $w^{\mathcal{D}(v)}$ (calculated in line 9), where $\mathcal{D}(v)$ indicates the application that node v belongs to.

Algorithm 5.2: Work-conserving algorithm.

Input : Infrastructure node n, Time $t \in \mathcal{T}$, Current bandwidth demands of applications nodes d, Temporal bandwidth requirements of application nodes σ
Output: Rate `nRate` for each application node

1 `hungryNodes` $\leftarrow \varnothing$;
2 **foreach** $v \in \mathcal{A}(n)$ **do**
3 $\quad$ `nRate[v]` $\leftarrow \min\left(\sigma(\sum(v,*),t), d[v]\right)$;
4 $\quad$ **if** $\sigma(\sum(v,*),t) < d[v]$ **then** `hungryNodes` $\leftarrow$ `hungryNodes` $\cup\ v$;
5 **end**
6 $\mathcal{Q}(n, \texttt{BAND}, t) \leftarrow \mathcal{B}(link) - \sum_{v \in \mathcal{A}(n)}$ `nRate[v]`, at time t;
7 **while** $\mathcal{Q}(n, \texttt{BAND}, t) > 0$ *and* `hungryNodes` *not empty* **do**
8 $\quad$ **foreach** $v \in$ `hungryNodes` **do**
9 $\quad\quad$ `value` $\leftarrow \min\left(d[v] - \texttt{nRate[v]}, \left(\frac{w^{\mathcal{D}(v)}}{\sum_{u \in \texttt{hungryNodes}} w^{\mathcal{D}(u)}} \times \mathcal{Q}(n, \texttt{BAND}, t)\right)\right)$;
10 $\quad\quad$ `nRate[v]` $\leftarrow$ `nRate[v]` $+$ `value`;
11 $\quad\quad$ $\mathcal{Q}(n, \texttt{BAND}, t) \leftarrow \mathcal{Q}(n, \texttt{BAND}, t) -$ `value`;
12 $\quad\quad$ **if** `nRate[v]` $== d[v]$ **then** `hungryNodes` $\leftarrow$ `hungryNodes` $\setminus \{v\}$;
13 $\quad$ **end**
14 **end**
15 **return** `nRate`;

The residual bandwidth is updated in line 11 and, in case the demands of node v were satisfied, it is removed from the list `hungryNodes` (line 12). Note that there is a "while" loop (lines 7 – 14) to guarantee that all residual bandwidth is used or all demands are satisfied. If this loop were not used, there could be occasions when there would be unsatisfied demands even though some bandwidth would be available.

In summary, if the demand of an application node exceeds its guaranteed rate (the rate specified in the request – σ), data can be sent and received at least at the guaranteed rate. Otherwise, if it does not, the unutilized bandwidth will be shared among co-resident application nodes whose traffic demands exceed their guarantees (work-conservation).

Finally, as discussed in Chapter 2, note that SDN has scalability challenges on DCNs (JARRAYA; MADI; DEBBABI, 2014): (i) elevated flow setup time, as forwarding devices ask the controller for appropriate rules when they receive the first packet of a new flow; and (ii) large flow tables in switches, since DCNs may have millions of flows per second (BENSON; AKELLA; MALTZ, 2010) and, thus, the number of entries needed in TCAMs may be significantly higher than the amount of resources available in commodity switches. We adopt the strategy proposed in Predictor (Chapter 4) to address these challenges.

5.1.4 Resource Monitoring Mechanism

Packer is designed with scalability and high multi-resource utilization (i.e., minimizing fragmentation of multiple resources) in mind. This implies that the resource monitoring mechanism (i) should not incur significant overhead (especially to scarce resources such as the network (XIE et al., 2012)); and (ii) needs to be able to acquire real-time information about resource usage, so that idle resources can be allocated to applications that need them. Moreover, the mechanism is expected to provide fast and up-to-date information upon unexpected events

(e.g., in case an application gets delayed due to a resource being congested).

We designed a two-level strategy for resource monitoring, composed of (i) the datacenter resource manager (DRM) and (ii) local controllers at servers and OpenFlow switches. First, a local controller runs at each server and coordinates the allocation of the server's resources to application tasks. LCs have two objectives, described as follows. The first objective is to observe aggregate resource usage and periodically report it to the DRM (so that the DRM gets updated information about the infrastructure utilization). The second objective is related to handling local events: since LCs have no interconnection among themselves (in order to reduce management traffic in the network) and no knowledge of infrastructure-wide state, they are allowed to handle only local events (e.g., dealing with local congested resources and enforcing allocations to tasks). This is important for relieving the load on the DRM and for reducing the amount of bandwidth used for communication between LCs and the DRM.

Second, the DRM maintains infrastructure-wide state, as it periodically receives resource usage statistics from local controllers at servers and from switches (via the OpenFlow protocol). With the information received from servers and switches, it reacts to global events such as the allocation of applications and bandwidth enforcement throughout the entire network for applications.

5.2 Evaluation

In this section, we focus on showing that Packer (i) minimizes multi-resource fragmentation; (ii) improves provider revenue; (iii) incurs acceptable overhead; (iv) provides predictable and guaranteed network performance with work-conserving sharing; and (v) outperforms existing state-of-the-art schemes (Tetris (GRANDL et al., 2014) and slot-based allocation (THE APACHE SOFTWARE FOUNDATION, 2014c; THE APACHE SOFTWARE FOUNDATION, 2015)).

5.2.1 Setup

Environment. We have implemented a simulator that models computing and network resources of a multi-tenant datacenter. For computing resources, we follow Tetris (GRANDL et al., 2014) and use a similar server configuration: 16 CPU cores, 32 GB of memory, 4 disks operating at 50 MBps each for read and write operations and a 1 Gbps NIC. For the slot-based scheme, we follow related work (BALLANI et al., 2011) and divide each server into four equal slots. The network, in turn, is defined as a tree-like topology, similar to current DCNs and related work (JANG et al., 2015). It is composed of a three-tier topology with 1,200 servers at level 0. Every 40 machines form a rack, and every 10 ToRs are connected to an aggregation switch. Finally, all aggregation switches are connected to a core switch. Unless otherwise specified, the capacity of each link is defined as follows: 1 Gbps for server-ToR links, 10 Gbps for

ToR-aggregation links and 100 Gbps for aggregation-core links.

Workload. We built a workload suite composed of incoming application requests (to be allocated in the datacenter) arriving over time. We consider a heterogeneous set of applications, including MapReduce and Web Services. As defined in Section 5.1.1, each application a is represented by a TI-MRA graph $G^a_{\texttt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$. Given the lack of publicly available traces for DCNs, the workload was generated in line with related work (GRANDL et al., 2014; SHIEH et al., 2011; BENSON; AKELLA; MALTZ, 2010; KANDULA et al., 2009; XIE et al., 2012). First, like Tetris (GRANDL et al., 2014), computing resources of tasks were picked uniformly at random between 0 and the maximum value of a slot. Note that we limit the demand for computing resources of each task from each application to the maximum size of a slot in order to provide a fair and accurate comparison (otherwise, since we use the same workload for all schemes, some tasks would never be allocated with the slot-based approach). Second, like Predictor (in Chapter 4), bandwidth demands were generated based on the measurements from Benson et al. (BENSON; AKELLA; MALTZ, 2010) and Kandula et al. (KANDULA et al., 2009). Finally, the weight w^a of each application a is uniformly distributed in the interval $[0, 1]$.

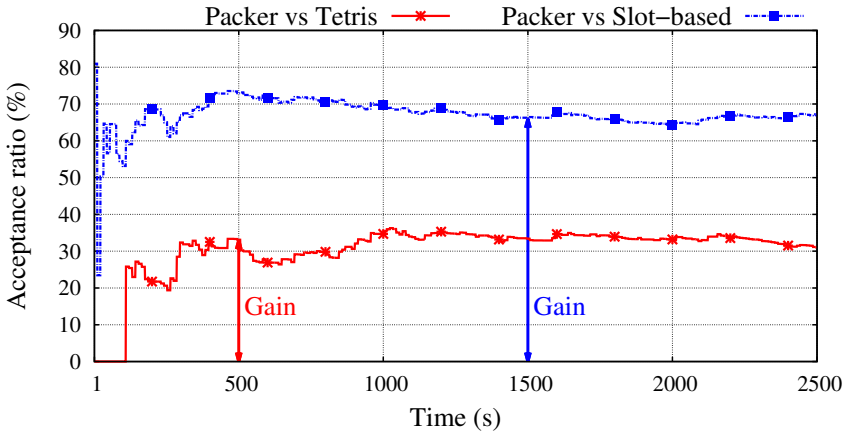
5.2.2 Results

We compare Packer with Tetris (GRANDL et al., 2014) and the slot-based allocation (THE APACHE SOFTWARE FOUNDATION, 2014c; THE APACHE SOFTWARE FOUNDATION, 2015). For all experiments comparing different strategies, we plot the percentage difference between Packer and the related work being compared as $\frac{\texttt{Packer} - \texttt{related\_work}}{\texttt{Packer}} \times 100$. Hence, positive values mean Packer has achieved a higher value than the approach being compared, while negative values mean Packer has achieved a lower value. In general higher values are better, with the sole exception being the overhead of the allocation algorithm (Figure 5.6).

Increased acceptance ratio. Figure 5.3 shows the proportion of application tasks that were allocated between Packer and Tetris and Packer and slot-based according to the time. *Higher values are better*, as they mean that Packer allocates more tasks than the respective proposal being compared. At first, the gains of Packer in comparison to the other proposals have high variability because there are ample resources and, therefore, most incoming applications are allocated. As time passes and the cloud-load increases (less available resources), the gains tend to stabilize (around time 1,000), because new applications are allocated only when already allocated applications conclude their execution and are deallocated (which releases resources). In general, we observe that Packer consistently outperforms Tetris ($\approx$30%) and slot-based allocation ($\approx$67%). Although the amount of available resources in the infrastructure is the same, the allocation ratio differs for each approach. This happens because each scheme uses a different allocation strategy. More specifically, Tetris seeks to minimize computing resource fragmentation, while only penalizing the bandwidth used. This may not result in good choices for

allocation because of network fragmentation (as the network is an important bottleneck in data-centers (XIE et al., 2012)). The slot-based allocation, in turn, is constrained by the static number and size of slots in the servers, which limit the feasible choices for allocating tasks to servers. In contrast to both proposals, Packer employs our novel algorithm described in Section 5.1.2 and better explores the trade-off between using local computing resources (CPU, memory and disk I/O) and remote distributed resources (the network).

Figure 5.3 – Acceptance ratio of application tasks.

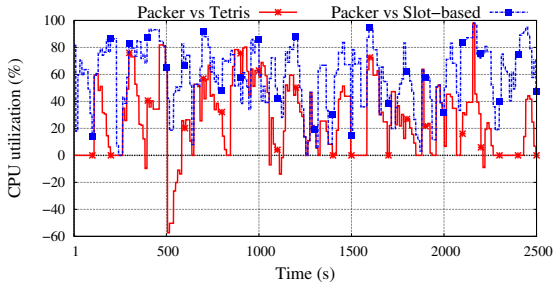


Source: by author (2016).

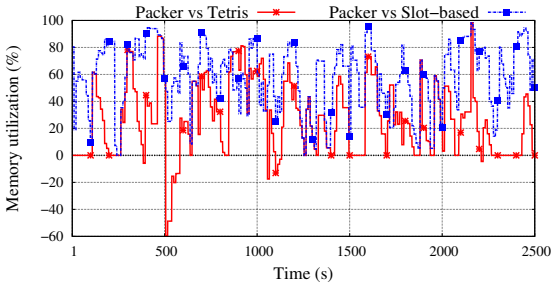
Maximized resource utilization. Figure 5.4 depicts the percentage difference between Packer and Tetris and Packer and slot-based allocation for the utilization of different types of resources. Positive values indicate that Packer achieves better utilization than the respective proposal being compared (*the higher the value, the better*), while negative values denote that the proposal being compared achieves better results than Packer. Figures 5.4(a), 5.4(b), 5.4(c) and 5.4(d) show results for CPU, memory, disk I/O write and disk I/O read, respectively. We see that, during most of the time, Packer allows better use of available resources, improving utilization by a significant percentage. Nonetheless, Packer has lower utilization of some types of resources at some periods of time (negative values in the plots). This happens because, with different allocation strategies, different applications are accepted and rejected in each scheme. Therefore, despite allocating significantly more application tasks than Tetris and slot-based (Figure 5.3), there are some small periods of time when the tasks allocated in Packer consume less resources.

Figure 5.4(e), in turn, shows the percentage difference of bandwidth utilization. We verify that Packer can maintain significantly higher utilization of the network than Tetris ($\approx$76% more) and slot-based ($\approx$87% more). Moreover, during the experiments, Packer never achieved lower network utilization than the proposals being compared. In general, Figures 5.3 and 5.4 show that Packer accepts more applications and, consequently, maximizes utilization, which indicates that fragmentation is minimized.

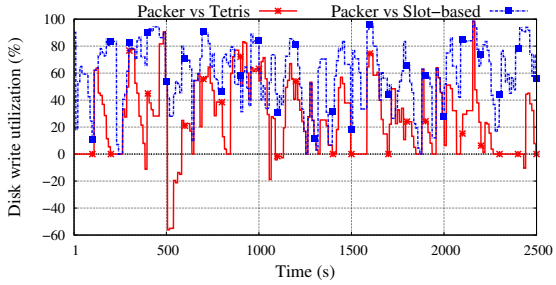
Figure 5.4 – Resource utilization (positive values in the y-axis indicate that Packer achieves better utilization than the respective proposal being compared, while negative values denote that the proposal being compared achieves better utilization than Packer).



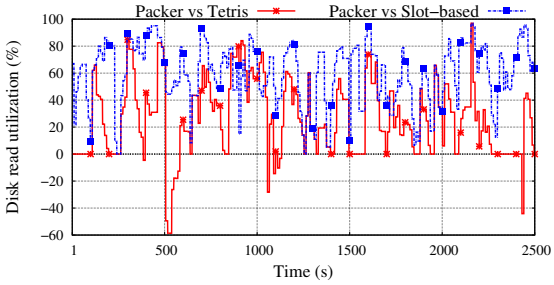
(a) CPU.



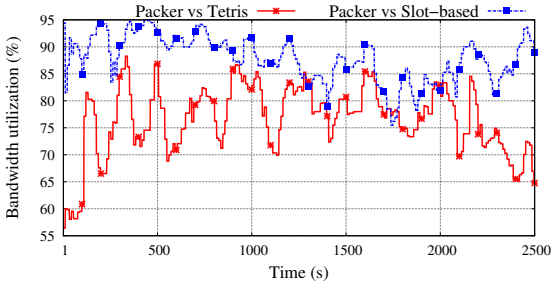
(b) Memory.



(c) Disk I/O write.



(d) Disk I/O read.

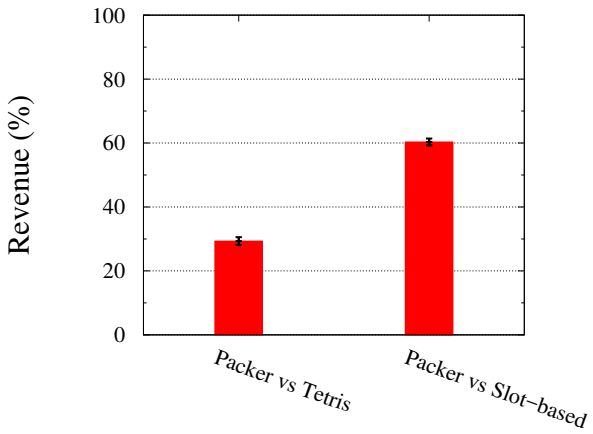


(e) Bandwidth.

Source: by author (2016).

Increased provider revenue. We follow related work (BALLANI et al., 2011; XIE et al., 2012) and adopt a simple pricing model to quantify provider revenue for Packer, Tetris and slot-based allocation, which effectively charges both computation and networking. A tenant running application a pays: $\sum_{t \in T^a} \sum_{v \in V^a} \left(\sum_{\mathtt{r}} \delta(v, \mathtt{r}, t) \times k_v + \sum_{u \in \mathcal{E}(v)} \sigma((u, v), t) \times k_b \right)$, where $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$, k_v is the unit-time computing resource cost and k_b is the unit-volume bandwidth cost. Figure 5.5 depicts the revenue of Packer in comparison to Tetris and slot-based allocation (error bars show 95% confidence interval). *Higher values are better*, as they mean that Packer provides more revenue than the respective proposal being compared. We see that, by improving the allocation ratio of application tasks (Figure 5.3) and resource utilization (Figure 5.4), Packer can significantly increase provider revenue ($\approx$29% and $\approx$60% in comparison to Tetris and slot-based, respectively).

Figure 5.5 – Revenue.



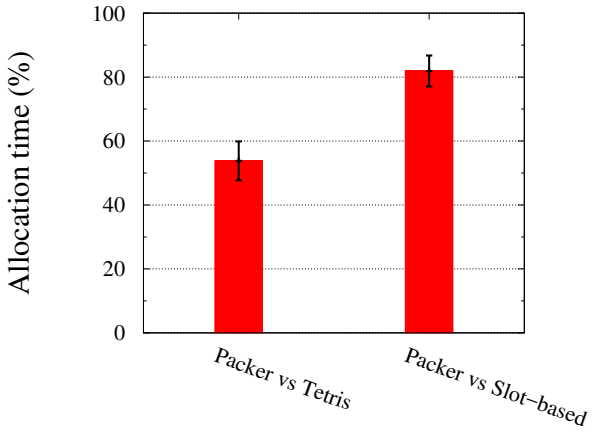
Source: by author (2016).

Acceptable overhead. Figure 5.6 quantifies the overhead introduced by Packer in comparison to Tetris and slot-based (error bars show 95% confidence interval). The overhead is given by the mean time taken to allocate an incoming application in the infrastructure. Here, positive values indicate that Packer takes more time to allocate applications than the respective proposal being compared, while negative values would indicate that Packer takes less time (i.e., *lower values are better*). We see that Packer takes more time to allocate applications than the other two proposals (≈53% more time than Tetris and ≈81% more than slot-based). This is justified by three factors (i) the complexity of the allocation metric (Section 5.1.2); (ii) the fact that Packer considers the whole network (as opposed to Tetris that only penalizes network use); and (iii) the fact that Packer verifies each computing resource (CPU, memory and disk I/O) according to the applications' requirements (as opposed to slot-based that statically divides computing resources into slots). Nonetheless, while the percentage is high, the median time taken to allocate applications (observed in our experiments) is small for all three proposals: ≈15.4s in Packer, ≈3.5s in Tetris and ≈0.3s in slot-based allocation. Thus, considering the benefits provided by Packer (shown in Figures 5.3, 5.4 and 5.5), it is acceptable to take some additional seconds to allocate applications.

Now, we turn our focus to the challenge of performance interference. In particular, we show that Packer provides (i) minimum bandwidth guarantees for applications; and (ii) work-conserving sharing, achieving both predictability for tenants and high utilization for providers. To show the results in a clear way, here we consider the temporal bandwidth guarantees of application tasks (σ) as a constant function (while the actual requirements vary over time).

Minimum bandwidth guarantees for applications. Packer adopts the following definition of minimum bandwidth guarantees: the task rate should be (a) at least the guaranteed rate if the demands are equal or higher than the guarantees; or (b) equal to the demands if they are lower than the guarantees. To illustrate it, we show, in Figure 5.7, a task allocated on a given server

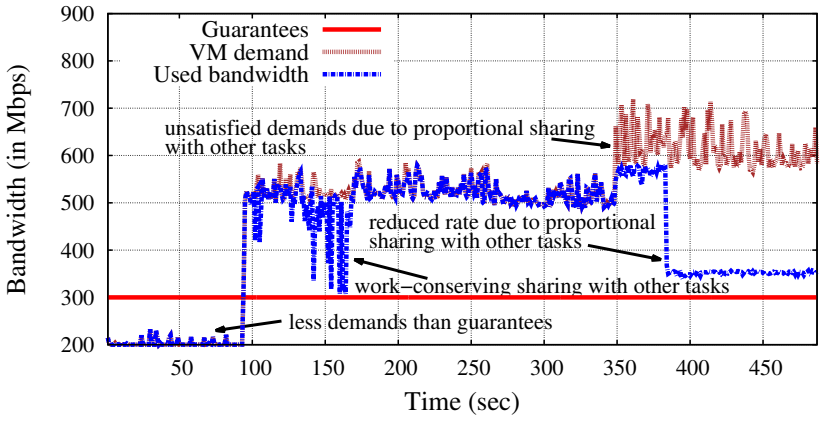
Figure 5.6 – Allocation time.



Source: by author (2016).

during a predefined time period of an experiment. We see that the task may not get the desired rate to satisfy all of its demands instantaneously (when its demands exceed its guarantees) because (i) the link capacity is limited; and (ii) available bandwidth is proportionally shared among tasks. In summary, we verify that Packer provides minimum bandwidth guarantees for tasks, since the actual rate is always equal or higher than the minimum between the demands and the guarantees. Therefore, applications have minimum bandwidth guarantees and, thus, can achieve predictable network performance.

Figure 5.7 – Bandwidth allocation for a task on a given server.

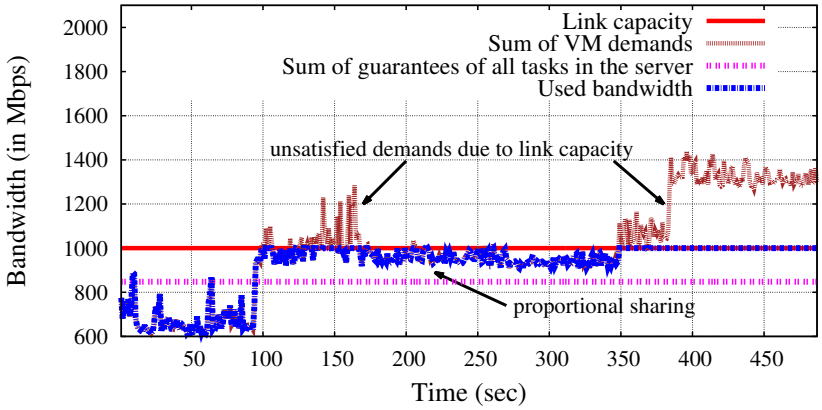


Source: by author (2016).

Work-conserving sharing. Work-conservation is the ability to use more bandwidth if the task has higher demands than its guarantees and there is available bandwidth in the network. In other words, bandwidth which is not allocated, or allocated but not currently used, should be proportionally shared among tasks with more demands than their guarantees (according to the

weights of each application – w^a, using Algorithm 5.2). Figure 5.8 shows the aggregate bandwidth[1] on the server holding the task in Figure 5.7. In these two figures, we verify that Packer provides work-conserving sharing in the network, as tasks can receive more bandwidth (if their demands are higher than their guarantees) when there is spare bandwidth. Thus, providers can achieve high utilization.

Figure 5.8 – Work-conserving sharing on a given server.



Source: by author (2016).

Generality of the results. We believe that the obtained results are generalizable to most real workloads and cloud platforms. However, recall that Packer takes into account multiple types of resources. Consequently, in case most applications have more demand for the same type of resource (even if this is not a realistic scenario), the results may vary. For instance, consider an extreme example of a datacenter with high network utilization (e.g., ≈99%), but low CPU, memory and disk I/O utilization. In this situation, acceptance ratio of applications will be determined only by the network and, thus, may be different in comparison to the obtained results. We defer a detailed evaluation of this type of scenario to future work.

5.3 Discussion

We discuss here some questions that have arisen during the design of Packer.

TI-MRA pros and cons. This abstraction considers temporal demands of resources. There are advantages and drawbacks of adopting this approach. The main advantage is to minimize multi-resource underutilization, which significantly improves datacenter utilization (i.e., reduces wastage). The main drawback is the need for fine-grained specification of applications, which may be a burden on some tenants (the ones with less knowledge of their applications).

[1]Note that Packer considers the temporal bandwidth guarantees requested (σ) when allocating tasks. Therefore, although the sum of the actual demands of all tasks allocated on a given server may exceed the server link capacity, the sum of bandwidth guarantees of these tasks will not exceed the link capacity.

Hence, TI-MRA also allows the specification of only peak demands for multiple resources, minimizing the burden of application specification. While this may reduce infrastructure utilization, it does not impact provider revenue, as tenants are allocating such resources and paying for them.

Profiling application demands for specifying TI-MRA. Datacenter application demands are often known (GROSVENOR et al., 2015) or can be obtained from tenants (BALLANI et al., 2011; JANG et al., 2015). Alternatively, we employ the techniques described by Grandl et al. (GRANDL et al., 2014), Chen and Shen (CHEN; SHEN, 2014) and Lee et al. (LEE et al., 2014). First, according to Chen and Shen (CHEN; SHEN, 2014), the same task (i.e., the same program with the same options) running on different servers tends to have similar resource utilization patterns. In this context, recurring applications are common in datacenters (AGARWAL et al., 2012); for instance, analytic applications repeat hourly or daily to perform the same computation on new data (GRANDL et al., 2014). Therefore, Packer can use statistics measured in prior runs of the same application. Second, according to Lee et al. (LEE et al., 2014), orchestration systems like OpenStack Heat and AWS CloudFormation could be used to generate abstract models. They use templates (provided as a library for tenants) that explicitly describe the structure of applications and their resource demands. In this sense, these systems could be extended with temporal multi-resource requirements to generate TI-MRAs. Third, Packer can use the pattern detection algorithm for resource demands developed by Chen and Shen (CHEN; SHEN, 2014). The algorithm utilizes logs of resource usage recorded by the cloud datacenter from previous runs of the same application and, thereby, can estimate utilization patterns for the requested application. Fourth, in case none of the previous methods can be used, we follow Grandl et al. (GRANDL et al., 2014) and over-estimate resource demands (by considering a constant temporal function). Note that over-estimation is better than under-estimation, as the former does not slow down applications. Furthermore, Packer's resource monitoring mechanism verifies idle resources and reports them to the DRM, so that they can be allocated to applications.

Employing existing abstractions in the literature for Packer. Packer could use existing abstractions for application specification, with the constraint that those abstractions take into account multiple types of resources. Adapting other abstractions for Packer could be performed with the development of a module that reads the specification and converts it to a TI-MRA, so that the use of other abstractions would be seamless to Packer's allocation process.

5.4 Summary

We presented Packer, a scheme that addresses the challenges of multi-resource allocation and performance interference in the network. It employs a novel abstraction called Time-Interleaved Multi-Resource Abstraction (TI-MRA) and a new algorithm for allocating multiple types of resources with reduced fragmentation. Furthermore, Packer uses (a) SDN to dynam-

ically configure and manage the network according to available resources and requirements of applications; and (b) a monitoring mechanism to avoid wastage and congested resources. Evaluation results show that (i) acceptance ratio of applications is increased; (ii) datacenter utilization is maximized (i.e., fragmentation is minimized); (iii) provider revenue is augmented; and (iv) applications achieve predictable and guaranteed network performance with work-conserving sharing.

In future work, we intend to propose novel allocation strategies to consider other aspects (such as availability and energy consumption) and to evaluate Packer in a testbed (e.g., CloudLab (CloudLab, 2016)).

6 RELATED WORK

Providers require efficient and robust mechanisms to manage DCNs. Ideally, they should enforce isolation among tenants and quality of service (QoS). Currently, DCNs are agnostic to network demands of applications (JANG et al., 2015). More specifically, DCNs are implemented with high utilization in mind (which allows providers to achieve economies of scale), and VMs may not have bandwidth guarantees.

Researchers have proposed several schemes to address performance interference among tenants and to minimize multi-resource fragmentation. Related work can be divided in two categories: network performance (Section 6.1), related to IoNCloud, Predictor and Packer; and multi-resource allocation (Section 6.2), related to Packer.

6.1 Network Performance

There is an extensive body of literature that addresses network performance in DCNs. These approaches can provide deterministic bandwidth guarantees (Section 6.1.1) and non-deterministic bandwidth guarantees (Section 6.1.2). Each one of these proposals presents pros and cons, discussed in the following sections.

6.1.1 Deterministic Bandwidth Guarantees

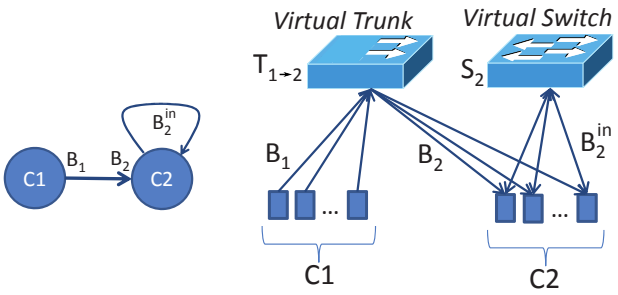
These schemes take advantage of rate-limiting at hypervisors, VM placement and virtual network embedding in order to increase their robustness.

Silo (JANG et al., 2015) builds upon the key insight that controlling tenant bandwidth leads to deterministic bounds on network queuing delay and offers tenants guaranteed bandwidth, delay and burst allowance for traffic between VMs of the same application. The system is composed of two components: (i) an admission control and VM placement algorithm to efficiently map tenants' multi-dimensional network guarantees; and (ii) a software packet pacer for fine-grained rate-limiting of VMs.

CloudMirror (LEE et al., 2014) utilizes new network abstraction and placement algorithm to provide bandwidth guarantees for applications. First, the novel network abstraction, called Tenant Application Graph (TAG), is based on applications' communication patterns. An example of TAG is depicted in Figure 6.1. Figure 6.1(a) illustrates the model of a simple application, composed of two components (C_1 and C_2), and Figure 6.1(b) shows an alternative way of visualizing the model expressed in Figure 6.1(a), highlighting the communication between VMs in the same components and also in different components. Second, the placement algorithm meets requirements specified by TAGs while considering the high availability required by applications.

SecondNet (GUO et al., 2010) is a DCN virtualization architecture that uses the concept

Figure 6.1 – TAG model.



(a) Example. (b) Explanation.

Source: Lee et al. (2014).

of virtual datacenter (VDC) as the unit of resource allocation. It distributes all the virtual-to-physical mapping, routing and bandwidth guarantees in server hypervisors and uses port-switching based source routing (PSSR) to allow efficient routing in arbitrary topologies utilizing commodity switches.

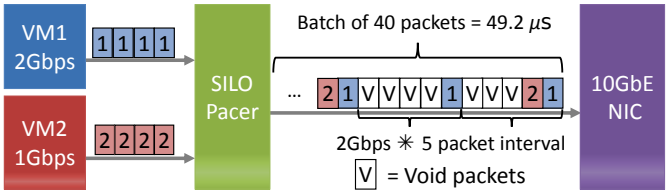
Oktopus (BALLANI et al., 2011) explores the trade-off between the guarantees offered to tenants, the tenant cost and provider revenue. It offers two novel abstractions for tenants: virtual cluster (VC) and virtual oversubscribed cluster (VOC). VC ensures the virtual network has no oversubscription and, thus, the maximum rate at which VMs of the same application can exchange data is their guarantees. VOC, in turn, is designed for applications that can deal with some level of oversubscription, in order to minimize the bandwidth allocated on the physical infrastructure (which can reduce tenant costs).

Gatekeeper (RODRIGUES et al., 2011) enables network performance isolation among applications using a distributed mechanism implemented at the virtualization layer of each hypervisor. It provides a minimum and a maximum rate for VMs, thus achieving deterministic bandwidth guarantees for applications. Moreover, the maximum rate can be set to infinity in order to maximize network utilization (at the cost of non-deterministic guarantees).

CloudNaaS (BENSON et al., 2011a) leverages SDN to provide increased control over network resources, providing bandwidth guarantees for tenants. It allows applications to be deployed with a rich and extensible set of network functions (e.g., isolation, custom addressing, service differentiation and flexible interposition of middleboxes).

In general, Silo, CloudMirror, SecondNet, Oktopus, Gatekeeper and CloudNaaS provide predictable network performance. However, such predictability presents a high cost: these efforts may result in network underutilization, as they statically reserve resources for applications based on their peak bandwidth demands. For instance, Silo uses packet pacing (shown in Figure 6.2) to control bandwidth of VMs and latency of traffic. This allows the system to achieve its goals, but may hurt utilization. Furthermore, these proposals address only intra-application communication.

Figure 6.2 – Example of bandwidth guarantees in Silo. Guarantees for VM_1 and VM_2 are 2 Gbps and 1 Gbps, respectively. Void packets are used to achieve packet pacing, which may result in underutilization of resources.



Source: Jang et al. (2015).

Proteus (XIE et al., 2012), in turn, uses a fine-grained virtual network abstraction, called Time-Interleaved Virtual Cluster (TIVC), that models the temporal network demands of cloud applications. With TIVC, the system can provide bandwidth guarantees and reduce bandwidth waste, allowing more applications to be allocated into the datacenter. Proteus also provides a novel allocation algorithm that considers the spatial and temporal demands of resources to map applications in the infrastructure. The system, however, has a complex allocation scheme with significant management overhead, since it reserves bandwidth for each application at each period of time according to the network profile of the respective application. Moreover, it is not work-conserving (i.e., it only reserves bandwidth according to the profile, which does not take into account possible delays - or unpredictable changes - that may happen during execution).

EyeQ (JEYAKUMAR et al., 2013) attempts to provide bandwidth guarantees with work-conservation. The system is based on the key insight that, by relieving the network's core of persistent congestion, bandwidth can be partitioned in a distributed manner at the edge. It also leverages the high bisection bandwidth in DCNs and enforces admission control based on traffic, regardless of the transport protocol. This design pushes bandwidth contention to the edge, enabling EyeQ, in theory, to support end-to-end minimum bandwidth guarantees to VMs. Despite providing strict guarantees in full bisection networks, the system does not completely eliminate congestion at core links in oversubscribed networks and, upon core-link congestion, EyeQ cannot provide bandwidth guarantees (GUO et al., 2013b).

HUG (CHOWDHURY et al., 2016) aims at offering bandwidth guarantees for applications without hurting network utilization. While it achieves its objective for full-bisection networks, it cannot offer bandwidth guarantees for applications in oversubscribed networks. Moreover, it may also introduce high management overhead, as the scheme needs to determine a correlation vector for each tenant that quantifies, for each bit sent on a given link, the amount of bits the tenant sends on other links.

Finally, Hadrian (BALLANI et al., 2013) introduces a strategy that considers two factors. First, it addresses communication dependencies between VMs of the same application as well as VMs of different applications, guaranteeing bandwidth for expected communication (i.e., for communication specified when submitting the request). Second, it takes into account upper-

bound proportionality (i.e., the maximum bandwidth a tenant can acquire is in proportion to its payment). Despite the benefits, it (i) needs a larger, custom packet header (hindering its deployment); (ii) does not ensure complete work-conservation, as the maximum allowed bandwidth is limited according to the tenant's payment; and (iii) requires switches to dynamically perform rate calculation (and enforce such rate) for each flow in the network.

6.1.2 Non-Deterministic Bandwidth Guarantees

Seawall (SHIEH et al., 2011) and NetShare (LAM et al., 2012) share the network proportionally according to weights assigned to VMs and tenants. Seawall (SHIEH et al., 2010; SHIEH et al., 2011) is a bandwidth allocation scheme that divides network capacity based on an administrator-specified policy. The key idea is to assign weights to any entity that generates traffic (such as a VM or a process) and to allocate bandwidth according to these weights in a proportional way. It uses congestion control and point-to-multipoint tunnels to enforce bandwidth sharing.

NetShare (LAM et al., 2012) proposes a statistical multiplexing mechanism to allocate bandwidth for tenants in a proportional way, to achieve high utilization and to provide weighted hierarchical max-min fair sharing (bandwidth unused by an application is shared proportionally by other applications). The weights are either specified by a manager or automatically assigned at each switch port based on a virtual machine heuristic.

FairCloud (POPA et al., 2012), in turn, explores the trade-off among network proportionality (i.e., network resources should be divided among tenants in proportion to their payments), minimum guarantees and high utilization. To better navigate among the factors of this trade-off, three resource allocation policies are proposed: Proportional Sharing at Link-level (PS-L), Proportional Sharing at Network-level (PS-N) and Proportional Sharing on Proximate Links (PS-P). PS-L achieves link proportionality, but does not offer bandwidth guarantees. PS-N provides better proportionality at network level (congestion proportionality), but also offers no minimum guarantees. Lastly, PS-P provides minimum bandwidth guarantees in tree-based topologies, but does not offer proportionality.

NumFabric (NAGARAJ et al., 2016) allows operators to specify different bandwidth allocation objectives and achieves them using distributed mechanisms at both switches and end-hosts. More specifically, it (i) is based on the Network Utility Maximization (NUM) (KELLY; MAULLOO; TAN, 1998) to allow per-flow allocation policies to be expressed using different utility functions; and (ii) utilizes, as objective, the maximization of the sum of utility functions.

Seawall, NetShare, FairCloud and NumFabric, however, may result in substantial management overhead (since bandwidth consumed by each flow at each link is dynamically calculated according to the flow weight, and large DCNs can have millions of flows per second). In particular, NumFabric and FairCloud require changes in switches, and NetShare presents two limitations. First, scalability is compromised because queues have to be configured at each

switch port for each application. Second, it relies on specific features of Fulcrum switches to implement its mechanisms, which reduces its deployability (BARI et al., 2013).

Varys (CHOWDHURY; ZHONG; STOICA, 2014), Aalo (CHOWDHURY; STOICA, 2015), CODA (ZHANG et al., 2016), Baraat (DOGAR et al., 2014), PIAS (BAI et al., 2015) and Karuna (CHEN et al., 2016) seek to improve application performance by minimizing average and tail flow completion time. Varys is a system that allows data-intensive frameworks to express their communication requirements as coflows[1], guaranteeing starvation freedom and maintaining high network utilization. The system implements two algorithms to schedule and allocate bandwidth for coflows. First, Smallest-Effective-Bottleneck-First (SEBF) greedily schedules a coflow based on the flow which the completion time is the bottleneck. Second, Minimum-Allocation-for-Desired-Duration (MADD) allocates rates to individual flows inside the coflow. MADD slows down all the flows in a coflow to match the completion time of the flow that will take the longest time to finish. This way, other coflows can make progress and the average coflow completion time decreases.

Like Varys, Aalo and CODA also use the coflow abstraction to improve network performance of applications. However, unlike Varys, Aalo seeks to schedule coflows without prior knowledge. It implements a multi-level scheduler, called Discretized Coflow-Aware Least-Attained Service (D-CLAS), to separate coflows into several priority queues based on the number of bytes already sent by each coflow. CODA argues that existing coflow-based solutions rely on modifying applications to extract coflows, which may be unrealistic for many practical scenarios. Consequently, it aims at automatically identifying and scheduling coflows without any application modification. It uses an incremental clustering algorithm to perform application-transparent coflow identification and an error-tolerant scheduler to mitigate identification errors.

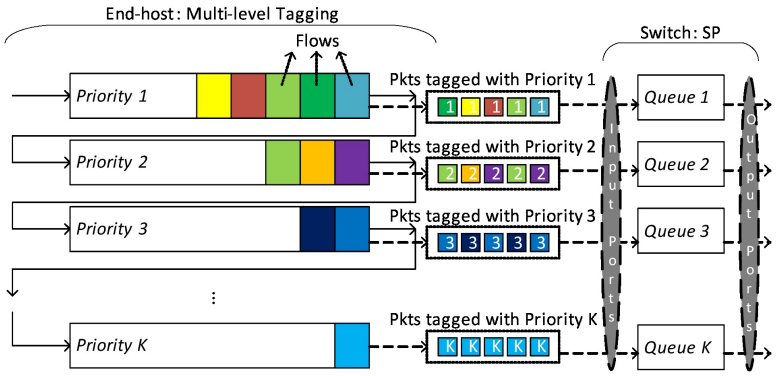
Baraat is a decentralized task-aware scheduling system for datacenters. Baraat addresses the problems associated with centralized scheduling (e.g., scalability and fault tolerance) and makes coordinated scheduling decisions while incurring low overhead. To enable decentralized scheduling, it assigns a globally unique priority for each task (all flows of a task share the task's priority). Moreover, the system uses an algorithm called FIFO with limited multiplexing (FIFO-LM), which schedules tasks based on their arrival order, but dynamically changes the level of multiplexing when heavy tasks are encountered. This way, FIFO-LM ensures that small tasks are not blocked by heavy tasks that, in turn, are not starved.

PIAS aims at minimizing flow completion time by implementing Shortest Job First (SJF) with the assumption that flow size is not known beforehand (no prior information about applications' network demands). The system uses multiple priority queues available in commodity switches to implement a Multiple Level Feedback Queue (MLFQ). MLFQ gradually demotes flows from higher-priority queues to lower-priority ones based on the number of bytes the flow has sent. Figure 6.3 shows a general view of PIAS. End-hosts tag packets according to prior-

[1] Coflow (CHOWDHURY; STOICA, 2012) is defined as a collection of flows that share a common performance goal (e.g., minimizing the completion time of the latest flow or ensuring that flows meet a common deadline).

ities. In switches, packets in different queues are scheduled with strict priority, while packets in the same queue are scheduled in a FIFO order. With this setting, mice flows are expected to conclude while being in the first few high-priority queues, being prioritized over elephant flows.

Figure 6.3 – PIAS overview.



Source: Bai et al. (2015).

Karuna (CHEN et al., 2016) addresses the challenge of scheduling flows with and without deadlines. It aims at maximizing deadline meet rate for deadline flows and at minimizing average flow completion time (FCT) for non-deadline flows. Towards this end, it implements a protocol called Minimal-impact Congestion control Protocol (MCP) to give deadline flows as little bandwidth as possible and extends PIAS for non-deadline flows.

QJUMP (GROSVENOR et al., 2015) addresses network interference by reducing switch queueing, since placing a finite bound on queueing allows better control over interference. In QJUMP, every application is assigned to a class. Flows from higher classes are rate-limited in the end-host, but can jump the queue over packets from lower classes once allowed into the network. The system explores the trade-off between throughput and latency. Thus, it can provide some level of guarantees ranging from strictly bounded latency with low rate to line-rate throughput with high latency variance.

Overall, these schemes (Varys, Aalo, CODA, Baraat, PIAS, Karuna and QJUMP) are able to improve network performance. However, none of them offers deterministic (strict) bandwidth guarantees. In particular, QJUMP may provide some level of guarantees, but is restricted to a specific system configuration (the maximum allowed rate is low and, thus, network utilization is significantly reduced). Furthermore, Karuna cannot guarantee that deadline flows will complete within their deadline upon a bursty arrival of such flows.

6.2 Multi-Resource Allocation

Proposals for multi-resource allocation can be classified in two categories: slot-based allocation and dynamic allocation of resources. We detail each one of them as follows.

Slot-based allocation. These approaches (JANG et al., 2015; BALLANI et al., 2013; BALLANI et al., 2011; THE APACHE SOFTWARE FOUNDATION, 2014b; THE APACHE SOFTWARE FOUNDATION, 2014a; THE APACHE SOFTWARE FOUNDATION, 2015) allocate resources based on slots for VMs. A slot-based allocation typically consider one resource during the process (e.g., CPU, memory or network) and predefined amounts of other resources, and allocates them. This leads to wastage and fragmentation, as only the resource considered in the process is optimized. This scenario is further exacerbated by the churn of applications in cloud datacenters.

Dynamic allocation of resources. Tetris (GRANDL et al., 2014), dominant resource fairness (DRF) (GHODSI et al., 2011) and a spatial/temporal strategy (CHEN; SHEN, 2014) aim at minimizing overall fragmentation of resources. Tetris is a multi-resource cluster scheduler that packs applications to machines according to their requirements. However, the system (i) may result in starvation (depending on the workload); and (ii) may not provide guarantees in oversubscribed networks. DRF, in turn, is a generalization of max-min fairness[2], where the allocation of an application is determined by the application's dominant share (i.e., the maximum share that the application has been allocated of any resource). Since DRF focus on fairness, it may result in fragmentation (GRANDL et al., 2014). Furthermore, DRF does not consider the network in the allocation process. Finally, Chen and Shen (2014) propose a VM allocation strategy that consolidates complementary VMs across space and time. This strategy focuses only on CPU and memory and, consequently, does not address performance interference in the network.

6.3 Summary

Table 6.1 shows an overview of the proposals discussed in this chapter. We compare them with IoNCloud, Predictor and Packer considering the following properties: (i) allocation method, in order to verify if they handle admission control in the cloud; (ii) minimum bandwidth guarantees (which is desired by tenants); (ii) work-conserving sharing and high utilization (which is desired by providers); and (iv) the management overhead they introduce.

In particular, we highlight some important aspects. First, IoNCloud, Gatekeeper and Hadrian provide limited work-conservation because of the following reason. IoNCloud allows work-conserving sharing only inside virtual networks (i.e., among applications of the same group, not among groups), while Gatekeeper allows it only up to a maximum value for each application (specified by the tenant) and Hadrian limits the maximum bandwidth of an application according to the tenant's payment.

Second, HUG and EyeQ provide limited minimum guarantees because the network needs full-bisection bandwidth. In other words, they do not provide guarantees in oversubscribed networks. Third, some proposals (such as PIAS and QJump) assume there exists an allocation

[2]Max-min fairness maximizes the minimum allocation received by an application in the infrastructure.

mechanism to admit applications in the cloud datacenter (i.e., applications are already allocated). Therefore, they only manage network resources at application runtime.

Fourth, management overhead is typically high when approaches need to manage bandwidth at a fine granularity (e.g., per-flow). This happens because DCNs have hundreds of thousands of servers (and VMs), hundreds of switches and hundreds of links, with millions of active flows per second (as shown by Benson, Akella and Maltz (2010)). For instance, consider a large DCN (with 100,000 VMs) where each flow is rate-limited at each link according to the requested guarantees and with the possibility of receiving more bandwidth (due to work-conserving sharing). In this case, overhead would be high to manage traffic in the network.

In general, we see that IoNCloud, Predictor and Packer cover more aspects than related proposals. IoNCloud provides minimum bandwidth guarantees for applications with work-conserving sharing inside each virtual network. This enables IoNCloud to achieve high network utilization and low management overhead (since providers only need to manage virtual networks). Predictor, in turn, provides minimum bandwidth guarantees with work-conservation and achieves high utilization with low management overhead (as flows are managed at application-level). Finally, Packer, in contrast to IoNCloud and Predictor, considers multiple types of resources at allocation time. Packer also provides minimum guarantees and work-conservation, allowing the network to be highly utilized. Furthermore, like Predictor, it manages flows at application-level, thus achieving low management overhead.

Table 6.1 – Comparison of IoNCloud, Predictor and Packer with related work.

Proposals	Allocation method	Min guarantees	Work-conservation	High utilization	Management overhead
IonCloud	Network-only	✓	Limited	✓	Low
Predictor	Network-only	✓	✓	✓	Low
Packer	Multi-resource	✓	✓	✓	Low
HUG	Network-only	Limited	✓	✓	High
Silo	Network-only	✓	✗	✗	Low
CloudMirror	Network-only	✓	✗	✗	Low
SecondNet	Network-only	✓	✗	✗	Low
Oktopus	Network-only	✓	✗	✗	Low
Gatekeeper	✗	✓	Limited	✗	Low
CloudNaaS	Network-only	✓	✗	✗	Low
Proteus	Network-only	✓	✗	✓	High
EyeQ	✗	Limited	✓	✓	High
Hadrian	Network-only	✓	Limited	✗	High
Seawall	✗	✗	✓	✓	High
NetShare	✗	✗	✓	✓	High
FairCloud (PS-L)	✗	✗	✓	✓	High
FairCloud (PS-N)	✗	✗	✓	✓	High
FairCloud (PS-P)	✗	✓	✓	✓	High
NumFabric	✗	✓	✓	✓	High
Varys	✗	✗	✓	✓	High
Aalo	✗	✗	✓	✓	High
CODA	✗	✗	✓	✓	High
Baraat	✗	✗	✓	✓	Low
PIAS	✗	✗	✓	✓	Low
Karuna	✗	✗	✓	✓	Low
QJump	✗	✓	✗	✗	Low
Tetris	Multi-resource	✗	✓	✓	Low
DRF	Multi-resource	✓	✗	✗	Low

Source: by author (2016).

7 CONCLUDING REMARKS

In this thesis, we addressed the challenge of performance interference in cloud datacenters, in order to provide predictable and guaranteed network performance with work-conserving sharing. In particular, current DCNs are subject to interference because of three main factors (as detailed in Sections 2.2.2 and 2.2.3): type of traffic, network oversubscription and congestion control. First, traffic in DCNs is complex, bursty and presents different patterns than traditional networks (GUO et al., 2014). Second, the topology is typically oversubscribed, with more bandwidth available in servers than in the core (SINGH et al., 2015). Providers use oversubscription in order to increase average network utilization, so that operational costs can be reduced. However, the high rate at which applications arrive ends up fragmenting available resources, and VMs of the same application may need to be allocated on distant servers (SHIEH et al., 2011). In this context, they need to use high-utilized and oversubscribed links to communicate among themselves (which may increase performance interference). Third, despite providing high utilization, TCP congestion control does not offer isolation at application-level (ABTS; FELDERMAN, 2012). In fact, it only ensures some fairness at flow-level. Overall, the median throughput of the network is low and there is a large variation among flow throughput (JUDD, 2015).

To cope with performance interference, we proposed IoNCloud, Predictor and Packer. **IoNCloud**, the *first contribution* of this thesis, leverages the key insight that temporal bandwidth demands of cloud applications do not peak at exactly the same time. We showed that, unlike related work, it can provide bandwidth guarantees for applications while maintaining high DCN utilization. More specifically, results indicate that IoNCloud (i) provides predictable network performance with guaranteed bandwidth for tenants; and (ii) reduces network underutilization, allocated bandwidth (which allows more applications to be admitted in the cloud) and management overhead. However, while IoNCloud provides predictable network performance for tenants and their applications, it does not offer work-conserving sharing among virtual networks (only inside VNs).

To improve network sharing and provide complete work-conservation, we leveraged SDN and designed **Predictor** (the *second contribution* of this thesis), a scheme that addresses performance interference by utilizing two novel algorithms. Predictor is also designed with scalability of SDN-based DCNs in mind, taking into consideration the number of entries required in flow tables and flow setup time in DCNs with millions of active flows. More specifically, Predictor considers flows at application-level (reducing flow table size) and proactively installs rules and rate-limiters at allocation time (minimizing flow setup time). Results show that Predictor provides guaranteed network performance, offers complete work-conserving sharing, significantly reduces the number of rules in flow tables and requires small controller load.

Despite achieving predictable and guaranteed network performance, IoNCloud and Predictor neglect non-network resources, which may result in fragmentation of such resources (i.e.,

reduced datacenter utilization). **Packer**, the *third contribution* of this thesis, addresses the challenges of performance interference and multi-resource allocation. In particular, it aims at minimizing fragmentation (consequently, increasing utilization) of multiple types of resources while guaranteeing network performance for applications. To enable Packer to achieve its goals, we developed a novel abstraction for applications in cloud datacenters, called **Time-Interleaved Multi-Resource Abstraction (TI-MRA)**, the *fourth contribution* of this thesis. TI-MRA allows the specification multi-resource requirements for applications through the use of temporal functions. While more complex than the hose model, it allows finer-grained specification of resource demands for complex applications and, at the same time, provides a simple interface for requesting applications that either do not present complex interactions or that execution patterns are unknown (by specifying only peak demands for resources).

In Chapter 1, we presented the following hypothesis (based on the limitations of state-of-the-art proposals in the context of network performance in cloud datacenters):

Hypothesis: a datacenter allocation strategy that considers multiple types of resources (CPU, memory, disk I/O and, particularly, the entire – traditional or SDN-based – network) can scalably provide predictable and guaranteed network performance for cloud applications, without hurting multi-resource utilization and provider revenue.

The research conducted and the proposal of IoNCloud, Predictor and Packer set a clear path towards supporting the proposed hypothesis. With IoNCloud, we confirmed that, in case the entire network is taken into account at allocation time, cloud applications can receive predictable and guaranteed network performance. Moreover, IoNCloud also enabled (i) higher network utilization (with limited work-conservation) in comparison to proposals in the literature; and (ii) more applications to be accepted in the datacenter (increasing provider revenue). With Predictor, we showed that it is possible to provide complete work-conservation while offering guaranteed network performance in SDN-based DCNs. Finally, we designed Packer and showed that, in addition to the benefits offered by Predictor, it is possible to minimize multi-resource fragmentation, which further maximizes resource utilization and provider revenue.

Based on the work presented in this thesis, it is possible to identify evidences to answer the research questions associated with the hypothesis that have been posed to guide this study. The answer to each question is detailed as follows.

RQ1: How can a distributed resource such as the network be efficiently managed, given the large scale and high dynamicity of cloud platforms?

Answer. In this thesis, we developed novel algorithms to efficiently utilize network virtualization (IoNCloud) and SDN (Predictor and Packer) in order to manage resources and achieve predictable and guaranteed performance in large-scale networks (as evaluated in Sections 3.2, 4.2 and 5.2). First, the use of network virtualization and the proposal of two new algorithms enable physical resources to be divided into slices, allowing a certain amount of bandwidth to be reserved for one or more applications in a flexible and precise

manner (e.g., independence between virtual and physical topologies). Second, the development of novel SDN-based algorithms enables network resources to be dynamically configured and managed according to application requirements and available resources in SDN-based DCNs.

RQ2: How can bandwidth be reserved for applications without hurting network utilization?

Answer. We developed two approaches to maintain high network utilization. First, in IoNCloud, applications with complementary temporal bandwidth demands are grouped in the same virtual network, thus sharing the bandwidth reserved for their VN. Since VNs are completely isolated from one another (in order to provide performance isolation among different groups of applications), bandwidth is shared only inside VNs (a limited form of work-conservation, but enough to minimize resource underutilization in comparison to related work). Second, in Predictor and Packer, the use of SDN/OpenFlow along with their respective allocation and rate enforcement algorithms enable applications to receive minimum bandwidth guarantees while proportionally sharing unused bandwidth according to weights (even if the bandwidth is allocated to an application, it can be used by other applications if the former is not currently using it).

RQ3: How can SDN-based DCNs scalably provide predictable and guaranteed performance?

Answer. In Predictor, we showed how to scalably provide predictable and guaranteed performance in SDN-based DCNs. More specifically, Predictor achieves scalability by proactively installing rules and rate-limiters in forwarding devices (thus eliminating flow setup time) and by managing flows at application-level (thus reducing the number of entries in flow tables). Predictor also employs two novel algorithms to address performance interference.

RQ4: How to enable the detailed specification of temporal requirements of multiple resources for applications, in order to help achieving guaranteed network performance without hurting utilization of other types of resources?

Answer. We developed a novel abstraction for applications in cloud datacenters, called Time-Interleaved Multi-Resource Abstraction (TI-MRA). TI-MRA allows the specification of temporal demands for multiple types of resources (CPU, memory, disk I/O and bandwidth). It uses the same principle of (a) temporal bandwidth requirements in TIVC (XIE et al., 2012), but extends it to all kinds of resources; and (b) communication patterns in TAG (LEE et al., 2014). Furthermore, it also takes into account dependencies other than among tasks (such as between tasks and cloud services), in order to optimize the use of resources.

TI-MRA can be used by all tenants, both with and without a deep knowledge of their application requirements. Users who have a deep understanding of their applications can tune resource demands according to the application requirements, possibly reducing costs without impact on performance. Tenants without knowledge of their applications, in turn, can specify only peak demands for resources (i.e., a constant temporal function for each

type of resource).

RQ5: While providing predictable and guaranteed network performance, can multi-resource utilization in an environment as dynamic as the cloud (i.e., with high rate of application arrival and departure) be maximized (i.e., minimizing multi-resource fragmentation)?
Answer. We addressed this challenge in Packer. More specifically, Packer employs a novel strategy that uses TI-MRA to consider temporal requirements of multiple types of resources at allocation time and a novel allocation strategy to improve utilization and, consequently, to minimize fragmentation. The strategy is based on an algorithm that extends existing heuristics for multi-dimensional bin packing, in order to cope with online arrival of applications, time-varying application requirements and distributed resources (e.g., the set of network links used by communications between tasks).

We consider improving IoNCloud, Predictor and Packer in the following manner. First, since IoNCloud achieves its benefits through grouping of applications in VNs, we plan on developing new grouping methods and thoroughly evaluate them. Second, we intend to implement and evaluate new allocation algorithms based on metrics other than bandwidth (e.g., fault tolerance and energy consumption) for Predictor and Packer. Third, we plan on performing a more extensive evaluation of IoNCloud, Predictor and Packer, considering more metrics, more factors and how these factors influence their respective schemes. For instance, we intend to evaluate how the number and size of groups impact network performance for applications in IoNCloud. Finally, we will evaluate the three schemes on a testbed (such as CloudLab (CloudLab, 2016)).

REFERENCES

ABTS, D.; FELDERMAN, B. A guided tour of data-center networking. **Commun. ACM**, ACM, New York, NY, USA, v. 55, n. 6, p. 44–51, jun. 2012.

ABU-LIBDEH, H.; COSTA, P.; ROWSTRON, A.; O'SHEA, G.; DONNELLY, A. Symbiotic Routing in Future Data Centers. In: ACM SIGCOMM 2010 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2010. (SIGCOMM '10), p. 51–62.

ADAMI, D.; MARTINI, B.; GHARBAOUI, M.; CASTOLDI, P.; ANTICHI, G.; GIORDANO, S. Effective resource control strategies using OpenFlow in cloud data center. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2013. (IM '13), p. 568–574.

AGACHE, A.; DEACONESCU, R.; RAICIU, C. Increasing Datacenter Network Utilisation with GRIN. In: USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Oakland, CA: USENIX Association, 2015. (NSDI '15), p. 29–42.

AGARWAL, S.; KANDULA, S.; BRUNO, N.; WU, M.-C.; STOICA, I.; ZHOU, J. Re-optimizing Data-parallel Computing. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2012. (NSDI'12), p. 21–21.

AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. A Scalable, Commodity Data Center Network Architecture. In: ACM SIGCOMM 2008 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2008. (SIGCOMM '08), p. 63–74.

AL-FARES, M.; RADHAKRISHNAN, S.; RAGHAVAN, B.; HUANG, N.; VAHDAT, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2010. (NSDI'10).

ALIZADEH, M.; EDSALL, T.; DHARMAPURIKAR, S.; VAIDYANATHAN, R.; CHU, K.; FINGERHUT, A.; LAM, V. T.; MATUS, F.; PAN, R.; YADAV, N.; VARGHESE, G. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In: ACM SIGCOMM 2014 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 503–514.

ALIZADEH, M.; GREENBERG, A.; MALTZ, D. A.; PADHYE, J.; PATEL, P.; PRABHAKAR, B.; SENGUPTA, S.; SRIDHARAN, M. Data Center TCP (DCTCP). In: ACM SIGCOMM 2010 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2010. (SIGCOMM '10), p. 63–74.

AMAZON. **Amazon ec2 api ec2-run-instances**. 2013. Available at : <http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference/ApiReference-cmd-RunInstances.html>. Visited on: Apr. 20, 2015.

AMAZON. **Amazon EC2**. 2014. Available at: <http://goo.gl/Fa90nC>. Visited on: Jan. 30, 2015.

ANTHONY, S. **Microsoft now has one million servers – less than Google, but more than Amazon, says Ballmer**. 2013. Available at: https://www.extremetech.com/extreme/161772-microsoft-now-has-one-million-servers-less-than-google-but-more-than-amazon -says-ballmer. Visited on: Oct. 30, 2013.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R. H.; KONWINSKI, A.; LEE, G.; PATTERSON, D. A.; RABKIN, A.; STOICA, I.; ZAHARIA, M. **Above the Clouds: A Berkeley View of Cloud Computing**. Berkeley, CA, USA, 2009. Available at: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>. Visited on: Apr. 21, 2015.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010.

BAI, W.; CHEN, L.; CHEN, K.; HAN, D.; TIAN, C.; WANG, H. Information-agnostic Flow Scheduling for Commodity Data Centers. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2015. (NSDI'15), p. 455–468.

BALLANI, H.; COSTA, P.; KARAGIANNIS, T.; ROWSTRON, A. Towards Predictable Datacenter Networks. In: ACM SIGCOMM 2011 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2011. (SIGCOMM '11), p. 242–253.

BALLANI, H.; JANG, K.; KARAGIANNIS, T.; KIM, C.; GUNAWARDENA, D.; O'SHEA, G. Chatty Tenants and the Cloud Network Sharing Problem. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2013. (NSDI'13), p. 171–184.

BANCILHON, F.; RAMAKRISHNAN, R. An amateur's introduction to recursive query processing strategies. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 15, n. 2, p. 16–52, jun. 1986.

BARI, M.; BOUTABA, R.; ESTEVES, R.; GRANVILLE, L.; PODLESNY, M.; RABBANI, M.; ZHANG, Q.; ZHANI, F. Data Center Network Virtualization: A Survey. **IEEE Communications Surveys & Tutorials**, v. 15, n. 2, p. 909–928, Second 2013.

BENSON, T.; AKELLA, A.; MALTZ, D. A. Network Traffic Characteristics of Data Centers in the Wild. In: ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT. **Proceedings...** New York, NY, USA: ACM, 2010. (IMC '10), p. 267–280.

BENSON, T.; AKELLA, A.; SHAIKH, A.; SAHU, S. CloudNaaS: A Cloud Networking Platform for Enterprise Applications. In: ACM SYMPOSIUM ON CLOUD COMPUTING. **Proceedings...** New York, NY, USA: ACM, 2011. (SOCC '11), p. 8:1–8:13.

BENSON, T.; ANAND, A.; AKELLA, A.; ZHANG, M. MicroTE: Fine Grained Traffic Engineering for Data Centers. In: CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES. **Proceedings...** New York, NY, USA: ACM, 2011. (CoNEXT '11), p. 8:1–8:12.

BERDE, P.; GEROLA, M.; HART, J.; HIGUCHI, Y.; KOBAYASHI, M.; KOIDE, T.; LANTZ, B.; O'CONNOR, B.; RADOSLAVOV, P.; SNOW, W.; PARULKAR, G. ONOS: Towards an Open, Distributed SDN OS. In: WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING. **Proceedings...** New York, NY, USA: ACM, 2014. (HotSDN '14), p. 1–6.

BODÍK, P.; MENACHE, I.; CHOWDHURY, M.; MANI, P.; MALTZ, D. A.; STOICA, I. Surviving Failures in Bandwidth-constrained Datacenters. In: ACM SIGCOMM 2012 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2012. (SIGCOMM '12), p. 431–442.

CHEN, C.; LIU, C.; LIU, P.; LOO, B. T.; DING, L. A Scalable Multi-datacenter Layer-2 Network Architecture. In: **Proceedings...** New York, NY, USA: ACM, 2015. (SOSR '15), p. 8:1–8:12.

CHEN, K.; SINGLAY, A.; SINGHZ, A.; RAMACHANDRANZ, K.; XUZ, L.; ZHANGZ, Y.; WEN, X.; CHEN, Y. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2012. (NSDI'12).

CHEN, L.; CHEN, K.; BAI, W.; ALIZADEH, M. Scheduling Mix-flows in Commodity Datacenters with Karuna. In: ACM SIGCOMM 2016 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 174–187.

CHEN, L.; FENG, Y.; LI, B.; LI, B. Towards Performance-Centric Fairness in Datacenter Networks. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2014. (INFOCOM '14), p. 1599–1607.

CHEN, L.; SHEN, H. Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2014. (INFOCOM '14), p. 1033–1041.

CHOWDHURY, M.; LIU, Z.; GHODSI, A.; STOICA, I. HUG: Multi-Resource Fairness for Correlated and Elastic Demands. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2016. (NSDI'16), p. 407–424.

CHOWDHURY, M.; STOICA, I. Coflow: A Networking Abstraction for Cluster Applications. In: ACM WORKSHOP ON HOT TOPICS IN NETWORKS. **Proceedings...** New York, NY, USA: ACM, 2012. (HotNets-XI), p. 31–36.

CHOWDHURY, M.; STOICA, I. Efficient Coflow Scheduling Without Prior Knowledge. In: ACM SIGCOMM 2015 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2015. (SIGCOMM '15), p. 393–406.

CHOWDHURY, M.; ZHONG, Y.; STOICA, I. Efficient Coflow Scheduling with Varys. In: ACM SIGCOMM 2014 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 443–454.

CHOWDHURY, N.; RAHMAN, M.; BOUTABA, R. Virtual Network Embedding with Coordinated Node and Link Mapping. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2009. (INFOCOM '09), p. 783–791.

CloudLab. 2016. Available at : <https://www.cloudlab.us>. Visited on: July 8, 2015.

COHEN, R.; LEWIN-EYTAN, L.; NAOR, J.; RAZ, D. On the effect of forwarding table size on SDN network utilization. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2014. (INFOCOM '14), p. 1734–1742.

CRONKITE-RATCLIFF, B.; BERGMAN, A.; VARGAFTIK, S.; RAVI, M.; MCKEOWN, N.; ABRAHAM, I.; KESLASSY, I. Virtualized Congestion Control. In: ACM SIGCOMM 2016 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 230–243. Available from Internet: <http://doi.acm.org/10.1145/2934872.2934889>.

CURTIS, A.; CARPENTER, T.; ELSHEIKH, M.; LOPEZ-ORTIZ, A.; KESHAV, S. REWIRE: An optimization-based framework for unstructured data center network design. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2012. (INFOCOM '12), p. 1116–1124.

CURTIS, A.; KIM, W.; YALAGANDULA, P. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2011. (INFOCOM '11), p. 1629–1637.

CURTIS, A. R.; KESHAV, S.; LOPEZ-ORTIZ, A. LEGUP: Using Heterogeneity to Reduce the Cost of Data Center Network Upgrades. In: CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES. **Proceedings...** New York, NY, USA: ACM, 2010. (Co-NEXT '10), p. 14:1–14:12.

CURTIS, A. R.; MOGUL, J. C.; TOURRILHES, J.; YALAGANDULA, P.; SHARMA, P.; BANERJEE, S. DevoFlow: Scaling Flow Management for High-performance Networks. In: ACM SIGCOMM 2011 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2011. (SIGCOMM '11), p. 254–265.

DALLY, W.; TOWLES, B. **Principles and Practices of Interconnection Networks**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

DELIMITROU, C.; SANCHEZ, D.; KOZYRAKIS, C. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In: ACM SYMPOSIUM ON CLOUD COMPUTING. **Proceedings...** New York, NY, USA: ACM, 2015. (SoCC '15), p. 97–110.

DOGAR, F. R.; KARAGIANNIS, T.; BALLANI, H.; ROWSTRON, A. Decentralized Task-aware Scheduling for Data Center Networks. In: ACM SIGCOMM 2014 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 431–442.

DUFFIELD, N. G.; GOYAL, P.; GREENBERG, A.; MISHRA, P.; RAMAKRISHNAN, K. K.; MERIVE, J. E. van der. A Flexible Model for Resource Management in Virtual Private Networks. In: ACM SIGCOMM CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 1999. (SIGCOMM '99), p. 95–108.

DUFFIELD, N. G.; GOYAL, P.; GREENBERG, A.; MISHRA, P.; RAMAKRISHNAN, K. K.; MERIVE, J. E. van der. A flexible model for resource management in virtual private networks. In: ACM SIGCOMM 2015 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 1999. (SIGCOMM '15).

FARRINGTON, N.; RUBOW, E.; VAHDAT, A. Data Center Switch Architecture in the Age of Merchant Silicon. In: IEEE SYMPOSIUM ON HIGH PERFORMANCE INTERCONNECTS. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2009. (HOTI '09), p. 93–102.

FERGUSON, A. D.; GUHA, A.; LIANG, C.; FONSECA, R.; KRISHNAMURTHI, S. Participatory Networking: An API for Application Control of SDNs. In: ACM SIGCOMM 2013 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2013. (SIGCOMM '13), p. 327–338.

GHODSI, A.; ZAHARIA, M.; HINDMAN, B.; KONWINSKI, A.; SHENKER, S.; STOICA, I. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2011. (NSDI'11), p. 323–336.

GILL, P.; JAIN, N.; NAGAPPAN, N. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In: ACM SIGCOMM 2011 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2011. (SIGCOMM '11), p. 350–361.

GRANDL, R.; ANANTHANARAYANAN, G.; KANDULA, S.; RAO, S.; AKELLA, A. Multi-resource Packing for Cluster Schedulers. In: ACM SIGCOMM 2014 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 455–466.

GREENBERG, A.; HAMILTON, J. R.; JAIN, N.; KANDULA, S.; KIM, C.; LAHIRI, P.; MALTZ, D. A.; PATEL, P.; SENGUPTA, S. VL2: A Scalable and Flexible Data Center Network. In: ACM SIGCOMM 2009 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2009. (SIGCOMM '09), p. 51–62.

GREENBERG, A.; HAMILTON, J. R.; JAIN, N.; KANDULA, S.; KIM, C.; LAHIRI, P.; MALTZ, D. A.; PATEL, P.; SENGUPTA, S. VL2: a scalable and flexible data center network. **Commun. ACM**, ACM, New York, NY, USA, v. 54, n. 3, p. 95–104, mar. 2011.

GROSVENOR, M. P.; SCHWARZKOPF, M.; GOG, I.; WATSON, R. N. M.; MOORE, A. W.; HAND, S.; CROWCROFT, J. Queues Don't Matter When You Can JUMP Them! In: USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Oakland, CA: USENIX Association, 2015. (NSDI '15), p. 1–14.

GUO, C.; LU, G.; LI, D.; WU, H.; ZHANG, X.; SHI, Y.; TIAN, C.; ZHANG, Y.; LU, S. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In: ACM SIGCOMM 2009 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2009. (SIGCOMM '09), p. 63–74.

GUO, C.; LU, G.; WANG, H. J.; YANG, S.; KONG, C.; SUN, P.; WU, W.; ZHANG, Y. SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. In: CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES. **Proceedings...** New York, NY, USA: ACM, 2010. (Co-NEXT '10), p. 15:1–15:12.

GUO, C.; WU, H.; TAN, K.; SHI, L.; ZHANG, Y.; LU, S. Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers. In: ACM SIGCOMM 2008 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2008. (SIGCOMM '08), p. 75–86.

GUO, J.; LIU, F.; HUANG, X.; LUI, J.; HU, M.; GAO, Q.; JIN, H. On efficient bandwidth allocation for traffic variability in datacenters. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2014. (INFOCOM '14), p. 1572–1580.

GUO, J.; LIU, F.; TANG, H.; LIAN, Y.; JIN, H.; LUI, J. Falloc: Fair network bandwidth allocation in IaaS datacenters via a bargaining game approach. In: IEEE INTERNATIONAL CONFERENCE ON NETWORK PROTOCOLS. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2013. (ICNP), p. 1–10.

GUO, J.; LIU, F.; ZENG, D.; LUI, J.; JIN, H. A cooperative game based allocation for sharing data center networks. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2013. (INFOCOM '13), p. 2139–2147.

HE, K.; KHALID, J.; GEMBER-JACOBSON, A.; DAS, S.; PRAKASH, C.; AKELLA, A.; LI, L. E.; THOTTAN, M. Measuring Control Plane Latency in SDN-enabled Switches. In: ACM SIGCOMM SYMPOSIUM ON SOFTWARE DEFINED NETWORKING RESEARCH. **Proceedings...** New York, NY, USA: ACM, 2015. (SOSR '15), p. 25:1–25:6.

HE, K.; ROZNER, E.; AGARWAL, K.; GU, Y. J.; FELTER, W.; CARTER, J.; AKELLA, A. AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks. In: ACM SIGCOMM 2016 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 244–257.

HELLER, B.; SHERWOOD, R.; MCKEOWN, N. The Controller Placement Problem. In: WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING. **Proceedings...** New York, NY, USA: ACM, 2012. (HotSDN '12), p. 7–12.

HU, S.; CHEN, K.; WU, H.; BAI, W.; LAN, C.; WANG, H.; ZHAO, H.; GUO, C. Explicit Path Control in Commodity Data Centers: Design and Applications. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2015. (NSDI'15), p. 15–28.

HU, Y.; WENDONG, W.; GONG, X.; QUE, X.; SHIDUAN, C. Reliability-aware controller placement for Software-Defined Networks. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2013. (IM '13), p. 672–675.

JANG, K.; SHERRY, J.; BALLANI, H.; MONCASTER, T. Silo: Predictable Message Latency in the Cloud. In: ACM SIGCOMM 2015 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2015. (SIGCOMM '15).

JARRAYA, Y.; MADI, T.; DEBBABI, M. A Survey and a Layered Taxonomy of Software-Defined Networking. **Communications Surveys Tutorials, IEEE**, v. 16, n. 4, p. 1955–1980, Fourth quarter 2014.

JEYAKUMAR, V.; ALIZADEH, M.; MAZIèRES, D.; PRABHAKAR, B.; KIM, C.; GREENBERG, A. EyeQ: Practical Network Performance Isolation at the Edge. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2013. (NSDI '13), p. 297–312.

JUDD, G. Attaining the promise and avoiding the pitfalls of tcp in the datacenter. In: USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Oakland, CA: USENIX Association, 2015. (NSDI '15), p. 145–157.

JYOTHI, S. A.; DONG, M.; GODFREY, P. B. Towards a Flexible Data Center Fabric with Source Routing. In: ACM SIGCOMM SYMPOSIUM ON SOFTWARE DEFINED NETWORKING RESEARCH. **Proceedings...** New York, NY, USA: ACM, 2015. (SOSR '15), p. 10:1–10:8.

KANDULA, S.; SENGUPTA, S.; GREENBERG, A.; PATEL, P.; CHAIKEN, R. The Nature of Data Center Traffic: Measurements & Analysis. In: ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2009. (IMC '09), p. 202–208.

KELLY, P. F.; MAULLOO, K. A.; TAN, H. D. K. Rate control for communication networks: shadow prices, proportional fairness and stability. **Journal of the Operational Research Society**, v. 49, n. 3, p. 237–252, 1998.

KLEINBERG, J. M. Authoritative sources in a hyperlinked environment. **Journal of the ACM (JACM)**, ACM, v. 46, n. 5, p. 604–632, 1999.

KOPONEN, T.; CASADO, M.; GUDE, N.; STRIBLING, J.; POUTIEVSKI, L.; ZHU, M.; RAMANATHAN, R.; IWATA, Y.; INOUE, H.; HAMA, T.; SHENKER, S. Onix: A Distributed Control Platform for Large-scale Production Networks. In: USENIX CONFERENCE ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2010. (OSDI'10), p. 1–6.

KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-Defined Networking: A Comprehensive Survey. **CoRR**, Lisbon, Portugal, 2014.

LACURTS, K.; DENG, S.; GOYAL, A.; BALAKRISHNAN, H. Choreo: Network-aware Task Placement for Cloud Applications. In: CONFERENCE ON INTERNET MEASUREMENT. **Proceedings...** New York, NY, USA: ACM, 2013. (IMC '13), p. 191–204.

LACURTS, K.; MOGUL, J. C.; BALAKRISHNAN, H.; TURNER, Y. Cicada: Introducing Predictive Guarantees for Cloud Networks. In: USENIX WORKSHOP ON HOT TOPICS IN CLOUD COMPUTING. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2014. (HotCloud '14).

LAM, V. T.; RADHAKRISHNAN, S.; PAN, R.; VAHDAT, A.; VARGHESE, G. Netshare and stochastic netshare: predictable bandwidth allocation for data centers. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 42, n. 3, p. 5–11, 2012.

LANGVILLE, A. N.; MEYER, C. D. **Google's PageRank and beyond: The science of search engine rankings**. Princeton, NJ, USA: Princeton University Press, 2011.

LEE, J.; TURNER, Y.; LEE, M.; POPA, L.; BANERJEE, S.; KANG, J.-M.; SHARMA, P. Application-driven Bandwidth Guarantees in Datacenters. In: ACM SIGCOMM 2014 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 467–478.

LI, Q.; DONG, M.; GODFREY, P. B. Halfback: Running Short Flows Quickly and Safely. In: CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES. **Proceedings...** New York, NY, USA: ACM, 2015. (CoNEXT '15).

LIU, C.; KIND, A.; LIU, T. Summarizing data center network traffic by partitioned conservative update. **IEEE Communications Letters**, v. 17, n. 11, p. 2168–2171, November 2013.

LIU, Y.; MUPPALA, J. Fault-tolerance characteristics of data center network topologies using fault regions. In: IEEE/IFIP INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2013. (DSN '13), p. 1–6.

MANN, V.; KUMAR, A.; DUTTA, P.; KALYANARAMAN, S. VMFlow: Leveraging VM Mobility to Reduce Network Power Costs in Data Centers. In: INTERNATIONAL IFIP TC 6 CONFERENCE ON NETWORKING. **Proceedings...** Berlin, Heidelberg: Springer-Verlag, 2011. (Networking '11), p. 198–211.

MARCON, D. S.; OLIVEIRA, R. R.; GASPARY, L. P.; BARCELLOS, M. P. **Datacenter Networks and Relevant Standards**. John Wiley & Sons, Inc, 2015. 73–104 p. Available from Internet: <http://dx.doi.org/10.1002/9781119042655.ch4>.

MARCON, D. S.; OLIVEIRA, R. R.; NEVES, M. C.; BURIOL, L. S.; GASPARY, L.; BARCELLOS, M. P. Trust-based grouping for cloud datacenters: Improving security in shared infrastructures. In: INTERNATIONAL IFIP TC 6 CONFERENCE ON NETWORKING. **Proceedings...** Berlin, Heidelberg: Springer-Verlag, 2013. (Networking '13), p. 1–9.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008.

MEMCACHED. **Memcached - a distributed memory object caching system**. 2015. Available at : <http://memcached.org/>. Visited on: Jan. 8, 2015.

MENG, X.; PAPPAS, V.; ZHANG, L. Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2010. (INFOCOM '10), p. 1–9.

MITTAL, R.; LAM, V. T.; DUKKIPATI, N.; BLEM, E.; WASSEL, H.; GHOBADI, M.; VAHDAT, A.; WANG, Y.; WETHERALL, D.; ZATS, D. TIMELY: RTT-based Congestion Control for the Datacenter. In: ACM SIGCOMM 2015 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2015. (SIGCOMM '15).

MOENS, H.; HANSSENS, B.; DHOEDT, B.; TURCK, F. D. Hierarchical network-aware placement of service oriented applications in Clouds. In: IEEE NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2014. (NOMS '14), p. 1–8.

MOGUL, J. C.; POPA, L. What We Talk About when We Talk About Cloud Network Performance. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 42, n. 5, p. 44–48, sep. 2012.

MOSHREF, M.; YU, M.; GOVINDAN, R.; VAHDAT, A. Trumpet: Timely and Precise Triggers in Data Centers. In: ACM SIGCOMM 2016 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 129–143.

MULLER, L.; OLIVEIRA, R.; LUIZELLI, M.; GASPARY, L.; BARCELLOS, M. Survivor: An enhanced controller placement strategy for improving SDN survivability. In: **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2014. (GLOBECOM '14), p. 1909–1915.

NAGARAJ, K.; BHARADIA, D.; MAO, H.; CHINCHALI, S.; ALIZADEH, M.; KATTI, S. NUMFabric: Fast and Flexible Bandwidth Allocation in Datacenters. In: ACM SIGCOMM 2016 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 188–201.

NAKAGAWA, Y.; HYOUDOU, K.; LEE, C.; KOBAYASHI, S.; SHIRAKI, O.; SHIMIZU, T. DomainFlow: Practical Flow Management Method Using Multiple Flow Tables in Commodity Switches. In: CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES. **Proceedings...** New York, NY, USA: ACM, 2013. (CoNEXT '13), p. 399–404.

NISHTALA, R.; FUGAL, H.; GRIMM, S.; KWIATKOWSKI, M.; LEE, H.; LI, H. C.; MCELROY, R.; PALECZNY, M.; PEEK, D.; SAAB, P.; STAFFORD, D.; TUNG, T.; VENKATARAMANI, V. Scaling Memcache at Facebook. In: . Lombard, IL: USENIX, 2013. p. 385–398.

NOVIFLOW. **Switching Made Smarter**. 2015. Available at: <http://noviflow.com/products/noviswitch/>. Visited on: Jan. 30, 2016.

OPEN NETWORKING FOUNDATION. **Software-Defined Networking: The New Norm for Networks**. Palo Alto, CA, USA, 2013. ONF white paper. Visited on: Apr. 19, 2015.

OUSTERHOUT, K.; RASTI, R.; RATNASAMY, S.; SHENKER, S.; CHUN, B.-G. Making Sense of Performance in Data Analytics Frameworks. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Oakland, CA: USENIX Association, 2015. (NSDI'15), p. 293–307.

PAGE, L.; BRIN, S.; MOTWANI, R.; WINOGRAD, T. The pagerank citation ranking: bringing order to the web. Stanford InfoLab, 1999.

PANIGRAHY, R.; TALWAR, K.; UYEDA, L.; WIEDER, U. **Heuristics for Vector Bin Packing**. Mountain View, CA, USA, 2011. Available at : <http://research.microsoft.com/apps/pubs/default.aspx?id=147927>. Visited on: Sept. 15, 2015.

PFAFF, B.; PETTIT, J.; KOPONEN, T.; JACKSON, E.; ZHOU, A.; RAJAHALME, J.; GROSS, J.; WANG, A.; STRINGER, J.; SHELAR, P.; AMIDON, K.; CASADO, M. The Design and Implementation of Open vSwitch. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Oakland, CA: USENIX Association, 2015. (NSDI'15), p. 117–130.

POPA, L.; KUMAR, G.; CHOWDHURY, M.; KRISHNAMURTHY, A.; RATNASAMY, S.; STOICA, I. FairCloud: Sharing the Network in Cloud Computing. In: ACM SIGCOMM 2012 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2012. (SIGCOMM '12), p. 187–198.

POPA, L.; YALAGANDULA, P.; BANERJEE, S.; MOGUL, J. C.; TURNER, Y.; SANTOS, J. R. ElasticSwitch: Practical Work-conserving Bandwidth Guarantees for Cloud Computing. In: ACM SIGCOMM 2013 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2013. (SIGCOMM '13), p. 351–362.

RAICIU, C.; PLUNTKE, C.; BARRE, S.; GREENHALGH, A.; WISCHIK, D.; HANDLEY, M. Data center networking with multipath tcp. In: ACM WORKSHOP ON HOT TOPICS IN NETWORKS. **Proceedings...** New York, NY, USA: ACM, 2010. (HotNets-IX), p. 10:1–10:6.

RECIO, R. **The Coming Decade of Data Center Networking Discontinuities**. 2012. ICNC. Keynote speaker.

RODRIGUES, H.; SANTOS, J. R.; TURNER, Y.; SOARES, P.; GUEDES, D. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks. In: CONFERENCE ON I/O VIRTUALIZATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2011. (WIOV'11).

ROS, F. J.; RUIZ, P. M. Five Nines of Southbound Reliability in Software-defined Networks. In: WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING. **Proceedings...** New York, NY, USA: ACM, 2014. (HotSDN '14), p. 31–36.

SCHAD, J.; DITTRICH, J.; QUIANÉ-RUIZ, J.-A. Runtime measurements in the cloud: observing, analyzing, and reducing variance. **Proc. VLDB Endow.**, VLDB Endowment, v. 3, n. 1-2, p. 460–471, sep. 2010.

SHEA, R.; WANG, F.; WANG, H.; LIU, J. A deep investigation into network performance in virtual machine based cloud environments. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2014. (INFOCOM '14), p. 1285–1293.

SHEN, H.; LI, Z. New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2014. (INFOCOM '14), p. 835–843.

SHIEH, A.; KANDULA, S.; GREENBERG, A.; KIM, C. Seawall: Performance Isolation for Cloud Datacenter Networks. In: USENIX CONFERENCE ON HOT TOPICS IN CLOUD COMPUTING. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2010. (HotCloud'10).

SHIEH, A.; KANDULA, S.; GREENBERG, A.; KIM, C.; SAHA, B. Sharing the Data Center Network. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2011. (NSDI'11), p. 309–322.

SINGH, A.; ONG, J.; AGARWAL, A.; ANDERSON, G.; ARMISTEAD, A.; BANNON, R.; BOVING, S.; DESAI, G.; FELDERMAN, B.; GERMANO, P.; KANAGALA, A.; PROVOST, J.; SIMMONS, J.; TANDA, E.; WANDERER, J.; HÖLZLE, U.; STUART, S.; VAHDAT, A. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In: ACM SIGCOMM 2015 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2015. (SIGCOMM '15), p. 183–197.

SINGLA, A.; GODFREY, P. B.; KOLLA, A. High Throughput Data Center Topology Design. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2014. (NSDI'14), p. 29–41.

SINGLA, A.; HONG, C.-Y.; POPA, L.; GODFREY, P. B. Jellyfish: Networking Data Centers Randomly. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION. **Proceedings...** Berkeley, CA, USA: USENIX Association, 2012. (NSDI'12).

SUNG, Y.-W. E.; TIE, X.; WONG, S. H.; ZENG, H. Robotron: Top-down Network Management at Facebook Scale. In: ACM SIGCOMM 2016 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 426–439.

THE APACHE SOFTWARE FOUNDATION. **Hadoop Map Reduce - Fair Scheduler.** 2014. Available at : <http://bit.ly/1p7sJ1I>. Visited on: Jul. 30, 2015.

THE APACHE SOFTWARE FOUNDATION. **Hadoop MapReduce - Capacity Scheduler**. 2014. Available at : <http://bit.ly/1tGpbDN>. Visited on: Jul. 30, 2015.

THE APACHE SOFTWARE FOUNDATION. **Hadoop Scheduler**. 2014. Available at : <http://bit.ly/1tGpbDN>, <http://bit.ly/1tGpbDN>. Visited on: Jul. 30, 2015.

THE APACHE SOFTWARE FOUNDATION. **Hadoop YARN Project**. 2015. Available at : <http://bit.ly/1iS8xvP>. Visited on: Jul. 30, 2015.

WANG, G.; NG, T. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2010. (INFOCOM '10), p. 1–9.

WANG, M.; MENG, X.; ZHANG, L. Consolidating virtual machines with dynamic bandwidth demand in data centers. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2011. (INFOCOM '11), p. 71–75.

WANG, X.; YAO, Y.; WANG, X.; LU, K.; CAO, Q. CARPO: Correlation-aware power optimization in data center networks. In: IEEE INFOCOM. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2012. (INFOCOM '12), p. 1125–1133.

WEHMUTH, K.; ZIVIANI, A.; FLEURY, E. A unifying model for representing time-varying graphs. IEEE Press, Piscataway, NJ, USA, Oct 2015.

WOEGINGER, G. J. There is no asymptotic PTAS for two-dimensional vector packing. **Information Processing Letters**, Elsevier, v. 64, n. 6, p. 293–297, 1997.

XIE, D.; DING, N.; HU, Y. C.; KOMPELLA, R. The Only Constant is Change: Incorporating Time-varying Network Reservations in Data Centers. In: ACM SIGCOMM 2012 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2012. (SIGCOMM '12), p. 199–210.

YEGANEH, S. H.; GANJALI, Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In: WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKS. **Proceedings...** New York, NY, USA: ACM, 2012. (HotSDN '12), p. 19–24.

YU, M.; REXFORD, J.; FREEDMAN, M. J.; WANG, J. Scalable Flow-based Networking with DIFANE. In: ACM SIGCOMM 2010 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2010. (SIGCOMM '10), p. 351–362.

YU, M.; YI, Y.; REXFORD, J.; CHIANG, M. Rethinking virtual network embedding: substrate support for path splitting and migration. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, p. 17–29, March 2008.

ZHANG, H.; CHEN, L.; YI, B.; CHEN, K.; CHOWDHURY, M.; GENG, Y. CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark. In: ACM SIGCOMM 2016 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 160–173.

ZHENG, L.; JOE-WONG, C.; TAN, C. W.; CHIANG, M.; WANG, X. How to Bid the Cloud. In: ACM SIGCOMM 2015 CONFERENCE. **Proceedings...** New York, NY, USA: ACM, 2015. (SIGCOMM '15).

ZHOU, Y.; WILKINSON, D.; SCHREIBER, R.; PAN, R. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. In: **International Conference on Algorithmic Aspects in Information and Management**. Berlin, Heidelberg: Springer-Verlag, 2008. (AAIM '08), p. 337–348.

APPENDIX A RESUMO ESTENDIDO DA TESE

O paradigma de computação em nuvem mudou significativamente o cenário de Tecnologia da Informação (TI), oferecendo provisionamento de recursos sob demanda para os locatários. Nesse modelo, os provedores buscam aumentar a utilização dos recursos, reduzir os custos operacionais e, consequentemente, alcançar economias de escala por meio da implementação de datacenters de nuvem como ambientes compartilhados altamente multiplexados, com diferentes aplicações coexistindo no mesmo conjunto de recursos físicos (ARMBRUST et al., 2009; ARMBRUST et al., 2010). Os locatários, por sua vez, podem escalar suas aplicações de acordo com as demandas, em um modelo de pagamento denominado *pay-as-you-go* (ou seja, os usuários pagam de acordo com o tempo e a quantidade de recursos computacionais consumidos). Isso permite que os locatários possam executar vários tipos de aplicações/serviços na nuvem, tanto as centradas em computação e que necessitam de grande quantidade de largura de banda na rede quanto as que necessitam de baixa latência (XIE et al., 2012; JANG et al., 2015).

Entretanto, os provedores carecem de mecanismos eficientes e confiáveis para oferecer garantias de largura de banda às aplicações (LEE et al., 2014; DOGAR et al., 2014; CHOWDHURY; ZHONG; STOICA, 2014; NAGARAJ et al., 2016). A rede interna da nuvem é tipicamente sobrecarregada (maior quantidade de banda nas bordas que no núcleo) e compartilhada em um modelo de melhor esforço, dependendo do TCP para alcançar alta utilização e escalabilidade. O TCP, no entanto, não fornece isolamento robusto entre diferentes fluxos[1] na rede (GUO et al., 2014; LI; DONG; GODFREY, 2015; CRONKITE-RATCLIFF et al., 2016; HE et al., 2016); na verdade, os fluxos de longa duração com um grande número de pacotes (denominados fluxos elefante) são privilegiados em detrimento dos pequenos (chamados fluxos rato) (ABTS; FELDERMAN, 2012), um problema denominado interferência de desempenho (SHIEH et al., 2011; GROSVENOR et al., 2015; BALLANI et al., 2013).

Estudos recentes (GROSVENOR et al., 2015; JUDD, 2015; SCHAD; DITTRICH; QUIANÉ-RUIZ, 2010; WANG; NG, 2010; BALLANI et al., 2011; JANG et al., 2015; SHEA et al., 2014) concluíram que, devido à interferência de desempenho, o *throughput* na rede alcançado por máquinas virtuais (VMs) pode variar por um fator superior a cinco, resultando em desempenho de rede baixo e imprevisível (BALLANI et al., 2013). Mais especificamente, quando a largura de banda disponível para uma aplicação fica abaixo de um certo limiar, o tempo total de execução da aplicação é alongado (ou seja, o desempenho geral é reduzido) (XIE et al., 2012). Esse comportamento ocorre devido a três razões principais: (i) aplicações realizam computação e comunicação de rede de forma intercalada (CHEN et al., 2014); (ii) aplicações tendem a gerar tráfego em rajadas (JEYAKUMAR et al., 2013); e (iii) a computação muitas vezes depende dos dados recebidos pela rede (se a velocidade de comunicação é reduzida devido à falta de largura de banda disponível, o processamento subsequente é retardado) (GUO et al., 2013b).

A falta de garantias de rede impacta diretamente em locatários e provedores. Os locatários

[1] Fluxos são caracterizados por sequências de pacotes da origem ao destino.

não recebem a alocação de recursos de rede para as suas aplicações (o que pode dificultar tanto aplicações que necessitam de alta taxa de transferência quanto aplicações sensíveis à latência) e, portanto, em certas ocasiões podem executar somente algumas aplicações específicas na nuvem (POPA et al., 2013). Além disso, os custos são imprevisíveis devido à alta variabilidade de banda disponível na rede (aplicações podem demorar mais tempo para executar (XIE et al., 2012)). Os provedores, por sua vez, têm o *throughput* dos seus datacenters reduzido (por causa da interferência desempenho) (JUDD, 2015; HE et al., 2016), o que pode afetar negativamente a receita (BALLANI et al., 2011).

Os trabalhos relacionados propuseram diversas técnicas para lidar com a interferência de desempenho. As propostas são divididas de acordo com o tipo de garantias que elas oferecem: deterministas (por exemplo, Silo (JANG et al., 2015) e Hadrian (BALLANI et al., 2013)) e não-deterministas (por exemplo, PIAS (BAI et al., 2015), QJump (GROSVENOR et al., 2015) e NumFabric (NAGARAJ et al., 2016)). Apesar de melhorar o desempenho da rede, elas apresentam deficiências importantes, incluindo (i) subutilização de recursos (Silo e QJump); (ii) significativa sobrecarga de gerenciamento ao calcular dinamicamente a taxa de banda e aplicar essa taxa para cada fluxo na rede (Hadrian e NumFabric), já que a rede pode ter milhões de fluxos ativos por segundo (BENSON; AKELLA; MALTZ, 2010); e (iii) inanição de fluxos elefante (PIAS). Além disso, nenhuma dessas propostas otimiza a alocação de outros tipos de recursos, o que pode resultar em fragmentação de recursos computacionais (por exemplo, CPU, memória e disco). Portanto, o objetivo é propor esquemas sem essas deficiências, mesmo que isso signifique introduzir alguma complexidade (como exigir que os locatários façam uma especificação detalhada de suas aplicações).

Nesta tese, são apresentadas três propostas para lidar com a interferência de desempenho em redes de datacenters (DCNs): IoNCloud, Predictor e Packer. Inicialmente, utilizou-se a observação fundamental que as demandas temporais de largura de banda das aplicações na nuvem não possuem pico ao mesmo tempo (WANG et al., 2012; CHEN; SHEN, 2014) e, por isso, foi proposta uma estratégia de alocação para a reserva e isolamento de recursos de rede em datacenters. Ela visa minimizar a subutilização de recursos ao mesmo tempo em que provê uma maneira eficiente de compartilhar previsivelmente a largura de banda entre as aplicações, com baixa sobrecarga de gerenciamento. Para mostrar os benefícios da estratégia, foi desenvolvido o IoNCloud, um esquema que implementa a estratégia proposta. O IoNCloud emprega o conceito de atração/repulsão entre aplicações de acordo com os seus requisitos temporais de banda e necessidade de isolamento, e agrupa tais aplicações em redes virtuais (VNs) com garantias de largura de banda. Dessa forma, a estratégia busca explorar o compromisso entre alta utilização da rede (objetivo dos provedores para reduzir os custos operacionais) e garantias estritas de rede (desejadas pelos locatários).

Apesar de atingir desempenho de rede previsível e garantido, o IoNCloud não provê conservação de trabalho entre VNs, o que mantém os recursos reservados ociosos enquanto outras aplicações poderiam se beneficiar ao usá-los. Por isso, foi desenvolvido o Predictor, uma evolução

em relação ao IoNCloud, a fim de fornecer garantias de rede e conservação de trabalho.

O Predictor busca programar e configurar dinamicamente a rede em datacenters baseados em redes definidas por *software* (SDN), visto que SDN é uma tendência crescente em datacenters (SINGH et al., 2015). Nesse contexto, o Predictor foi projetado para ser escalável, considerando o número de regras em tabelas de fluxo e o tempo de instalação das regras para um novo fluxo em DCNs com milhões de fluxos ativos. Para alcançar garantias de rede e escalabilidade tanto em redes com bisseção total de banda quanto em DCNs sobrecarregadas, ele baseia-se em duas observações fundamentais: (i) provedores não precisam controlar cada fluxo individualmente, uma vez que eles cobram os locatários com base na quantidade de recursos consumidos pelas aplicações; e (ii) o controle de congestionamento na rede deve ser proporcional ao pagamento dos locatários (BALLANI et al., 2013; JANG et al., 2015).

Com base nessas observações, o Predictor lida com a interferência desempenho e a escalabilidade de SDN em DCNs da seguinte maneira. A interferência desempenho é abodada com a utilização de SDN e o emprego dois novos algoritmos para melhorar o compartilhamento da rede, o que beneficia tanto locatários quanto provedores. Os locatários podem alcançar desempenho de rede previsível, recebendo garantias de largura de banda para as suas aplicações. Os provedores, por sua vez, mantêm alta taxa de utilização da rede, essencial para alcançar economias de escala.

A escalabilidade de SDN em DCNs é endereçada de duas formas. Primeiro, a ocupação da tabela de fluxos é reduzida por meio do gerenciamento de fluxos em nível de aplicação (considerando as duas observações feitas anteriormente). Essa configuração permite que os provedores possam controlar o tráfego e coletar estatísticas em nível de aplicação para cada enlace e dispositivo na rede. Segundo, o tempo de instalação de regras para um novo fluxo é reduzido por meio da instalação proativa de regras em *switches* SDN no momento da alocação para garantir largura de banda para a comunicação entre VMs da mesma aplicação. As regras para esse tipo de comunicação são proativamente instaladas porque o volume de tráfego entre VMs da mesma aplicação é maior que entre VMs de diferentes aplicações (BALLANI et al., 2013). As regras para comunicação entre VMs de diferentes aplicações, por sua vez, podem ser proativamente instaladas em switches (se os locatários conhecem as outras aplicações com as quais suas aplicações se comunicarão (GROSVENOR et al., 2015) ou se o provedor empregar alguma técnica preditiva (XIE et al., 2012; LACURTS et al., 2013)) ou reativamente instaladas de acordo com as demandas. Note que a instalação proativa de regras tem vantagens e desvantagens: enquanto a latência para novos fluxos é reduzida, algumas regras na tabela de fluxos podem levar mais tempo para expirar (há a possibilidade que elas só sejam removidas quando as suas respectivas aplicações concluírem e forem desalocadas).

Apesar de atingir os seus respectivos objetivos, o IoNCloud e o Predictor (e a maioria das estratégias de alocação presentes na literatura (JANG et al., 2015; BALLANI et al., 2013)) negligenciam a alocação de recursos que não sejam de rede; na verdade, CPU e memória são tipicamente alocadas de acordo com *slots* (GHODSI et al., 2011). A alocação baseada em *slots*,

infelizmente, resulta em superalocação, o que leva ao desperdício (visto que as aplicações não usam todos os seus recursos alocados) e à fragmentação (GRANDL et al., 2014). Em geral, a superalocação resulta em um menor número de aplicações sendo aceitas na infraestrutura e em menor utilização do datacenter.

Para lidar com as limitações acima referidas, utilizam-se duas observações fundamentais: (i) estendendo a observação feita ao IoNCloud, aplicações têm demandas complementares ao longo do tempo para múltiplos tipos de recursos (GRANDL et al., 2014); e (ii) a utilização de diferentes recursos têm picos de demanda em momentos distintos (CHEN; SHEN, 2014). Com base nessas observações, é proposto o Packer. Além de fornecer desempenho de rede garantido com conservação de trabalho (como o Predictor), o Packer visa minimizar a fragmentação de múltiplos tipos de recursos e, consequentemente, aumentar a utilização do datacenter para os principais recursos (rede, CPU, memória e disco), sem considerar *slots*.

O Packer é projetado com base em quatro aspectos: abstração para aplicações, alocação de múltiplos tipos de recursos, compartilhamento de rede e monitoramento de recursos. Primeiro, o Packer utiliza uma nova abstração para as aplicações, chamada *Time-Interleaved Multi-Resource Abstraction* (TI-MRA). Diferentemente das abstrações existentes na literatura (BALLANI et al., 2011; LEE et al., 2014; XIE et al., 2012; BALLANI et al., 2013), o TI-MRA não impõe nenhuma estrutura predefinida para as aplicações e permite a especificação de requisitos para vários tipos de recursos ao longo do tempo. Segundo, o Packer emprega uma nova estratégia de alocação que estende as heurísticas previamente desenvolvidas para o problema de otimização do empacotamento com múltiplas dimensões (*multi-dimensional bin packing*), a fim de reduzir a fragmentação de recursos. Terceiro, o Packer utiliza redes definidas por *software* (SDN) e OpenFlow (JARRAYA; MADI; DEBBABI, 2014) para configurar e assegurar dinamicamente garantias de largura de banda para as aplicações em toda a rede. Quarto, o Packer emprega um mecanismo de monitoramento de recursos para evitar o desperdício e para reagir a eventos inesperados (por exemplo, se uma aplicação atrasa devido a um recurso que está congestionado).

Classe de aplicações para maximizar os benefícios de cada proposta. O IoNCloud, o Predictor e o Packer foram desenvolvidos considerando as classes mais importantes de aplicações para a nuvem (aquelas com grande uso da rede e as sensíveis à latência). Todos os três esquemas fornecem garantias de desempenho de rede. No entanto, cada esquema é mais adequado para certos tipos de aplicações, como segue.

O IoNCloud e o Packer utilizam demandas temporais (ou o pico de demanda, com o custo de uma certa subutilização) para recursos de rede (IoNCloud) e para vários tipos de recursos (Packer). Consequentemente, como o Proteus (XIE et al., 2012) e o CloudMirror (LEE et al., 2014), eles são mais adequados para locatários que executem repetidamente os mesmos tipos de aplicações, com dados de entrada semelhantes entre as diferentes execuções da mesma aplicação. Isso é comum em aplicações que realizam o processamento de dados de forma iterativa (por exemplo, PageRank (LANGVILLE; MEYER, 2011; PAGE et al., 1999), busca de tópicos induzida por hipertexto (KLEINBERG, 1999), consultas relacionais recursivas (BANCILHON;

RAMAKRISHNAN, 1986), análise de redes sociais e análise do tráfego de rede), nas quais a maior parte dos dados permanece inalterada de iteração para iteração (XIE et al., 2012). Nesse caso, o uso de recursos das aplicações pode ser medido periodicamente ou em cada execução. Além disso, outras aplicações também podem tirar proveito do Packer com a especificação do pico de demanda para cada tipo de recurso. Nesse contexto, o mecanismo de monitoramento de recursos do Packer é empregado durante a execução das aplicações, a fim de evitar o desperdício.

O Predictor, por sua vez, não exige demandas temporais de recursos para as aplicações; ele só requer a taxa de largura de banda que deve ser garantida para cada aplicação. No caso das garantias de largura de banda de algumas aplicações serem sub- ou super-provisionadas, o algoritmo de conservação de trabalho do Predictor ajusta a taxa de envio e de recebimento de dados, a fim de evitar o desperdício de banda. Portanto, o Predictor é adequado para a maioria das aplicações na nuvem, incluindo PageRank (LANGVILLE; MEYER, 2011), Memcached (NISHTALA et al., 2013; MEMCACHED, 2015), MapReduce, aprendizado de máquina (ZHOU et al., 2008) e aplicações web voltadas para o usuário (que possuem requisitos estritos de latência).

Contribuições. De modo geral, as principais contribuições dessa tese são:

- O IoNCloud, um esquema para datacenters de nuvem de grande escala. O IoNCloud (i) agrupa aplicações em redes virtuais; (ii) mapeia tais aplicações no substrato físico; e (iii) fornece recursos de rede em cada enlace que a VN foi alocada de acordo com o pico temporal das demandas agregadas das aplicações no mesmo grupo que utilizam o enlace (ou seja, a largura de banda necessária no instante em que a soma das demandas de rede das aplicações pertencentes ao mesmo grupo é a mais elevada). Os resultados da avaliação mostram que o IoNCloud (i) fornece desempenho de rede previsível, com largura de banda garantida aos locatários; e (ii) reduz a subutilização da rede, a quantidade de banda alocada (o que permite mais aplicações serem alocadas na nuvem) e a sobrecarga de gerenciamento;
- O Predictor, um esquema para DCNs baseadas em SDN que emprega dois novos algoritmos para fornecer desempenho de rede previsível. Mais especificamente, o Predictor oferece garantias de largura de banda com conservação de trabalho para locatários e gerenciamento de rede em grão-fino para provedores. Ele também aborda os desafios de escalabilidade de SDN em DCNs por meio do controle de fluxos em nível de aplicação e da instalação proativa de regras em dispositivos de encaminhamento. Os resultados mostram que o Predictor (i) fornece desempenho de rede garantido com conservação de trabalho; (ii) reduz significativamente o número de regras em tabelas de fluxo de switches; e (iii) requer baixa carga no controlador;
- O Packer, um esquema que, além de proporcionar desempenho de rede previsível e garantido, minimiza a fragmentação de múltiplos tipos de recursos. Ele alcança esses objetivos por meio da utilização de SDN e de dois novos algoritmos para (i) alocar aplicações con-

siderando vários recursos; e (ii) definir periodicamente a taxa de banda de cada VM. Os resultados mostram que o Packer fornece garantias mínimas de largura de banda para as aplicações e conservação de trabalho para os provedores, e aumenta a utilização dos recursos do datacenter e a receita do provedor em comparação com os trabalhos relacionados, com o custo de levar mais tempo para alocar aplicações;

- Uma nova abstração para especificar aplicações no Packer, denominada Time-Interleaved Muti-Resource Abstraction (TI-MRA). Diferentemente das abstrações presentes na literatura (BALLANI et al., 2011; XIE et al., 2012; BALLANI et al., 2013; LEE et al., 2014), o TI-MRA permite a especificação de demandas temporais para vários tipos de recursos, sem uma estrutura predefinida para as aplicações.

APPENDIX B PUBLISHED CHAPTER AT BOOK CLOUD SERVICES, NETWORKING AND MANAGEMENT

This chapter presents a in-depth study of datacenter networks, relevant standards and operation. Our goal was three-fold: (i) provide a detailed view of the networking infrastructure connecting the set of servers of the datacenter via high-speed links and commodity off-the-shelf (COTS) switches; (ii) discuss the addressing and routing mechanisms employed in this kind of network; and (iii) show how the nature of traffic may impact DCNs and affect design decisions.

- **Title**: Chapter 4: Datacenter Networks and Relevant Standards
- **Book**: Cloud Services, Networking, and Management
- **ISBN**: 978-1-118-84594-3
- **Editors**: Nelson Fonseca and Raouf Boutaba
- **Date**: April, 2015
- **Publisher**: Wiley-IEEE Press
- **Digital Object Identifier (DOI)**: 10.1002/9781119042655.ch4

PART II

CLOUD NETWORKING AND COMMUNICATIONS

4

DATACENTER NETWORKS AND RELEVANT STANDARDS

Daniel S. Marcon, Rodrigo R. Oliveira, Luciano P. Gaspary, and Marinho P. Barcellos

Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

4.1 OVERVIEW

Datacenters are the core of cloud computing, and their network is an essential component to allow distributed applications to run efficiently and predictably [1]. However, not all datacenters provide cloud computing. In fact, there are two main types of datacenters: production and cloud. Production datacenters are often shared by one tenant or among multiple (possibly competing) groups, services, and applications, but with low rate of arrival and departure. They run data analytics jobs with relatively little variation in demands, and their size varies from hundreds of servers to tens of thousands of servers. Cloud datacenters, in contrast, have high rate of tenant arrival and departure (churn) [2], run both user-facing applications and inward computation, require elasticity (since application demands are highly variable), and consist of tens to hundreds of thousands of physical servers [3]. Moreover, clouds can comprise several datacenters spread around the world. As an example, Google, Microsoft, and Amazon (three of the biggest players in the market) have datacenters in four continents; and each company has over 900,000 servers.

Cloud Services, Networking, and Management, First Edition.
Edited by Nelson L. S. da Fonseca and Raouf Boutaba.

This chapter presents an in-depth study of datacenter networks (DCNs), relevant standards, and operation. Our goal here is three-fold: (i) provide a detailed view of the networking infrastructure connecting the set of servers of the datacenter via high-speed links and commodity off-the-shelf (COTS) switches [4]; (ii) discuss the addressing and routing mechanisms employed in this kind of network; and (iii) show how the nature of traffic may impact DCNs and affect design decisions.

Providers typically have three main goals when designing a DCN [5]: scalability, fault tolerance, and agility. First, the infrastructure must scale to a large number of servers (and preferably allow incremental expansion with commodity equipment and little effort). Second, a DCN should be fault tolerant against failures of both computing and network resources. Third, a DCN ideally needs to be agile enough to assign any virtual machine or, in short, VM (which is part of a service or application) to any server [6]. As a matter of fact, DCNs should ensure that computations are not bottlenecked on communication [7].

Currently, providers attempt to meet these goals by implementing the network as a multi-rooted tree [1], using LAN technology for VM addressing and two main strategies for routing: equal-cost multipath (ECMP) and valiant load balancing (VLB). The shared nature of DCNs among a myriad of applications and tenants and high scalability requirements, however, introduce several challenges for architecture design, protocols and strategies employed inside the network. Furthermore, the type of traffic in DCNs is significantly different from traditional networks [8]. Therefore, we also survey recent proposals in the literature to address the limitations of technologies used in today's DCNs.

We structure this chapter as follows. First, we begin by examining the typical multi-rooted tree topology used in current datacenters and discuss its benefits and drawbacks. Then, we take a look at novel topologies proposed in the literature, and how network expansion can be performed in a cost-efficient way for providers. After addressing the structure of the network, we look into the traffic characteristics of these high-performance, dynamic networks and discuss proposals for traffic management on top of existing topologies. Based on the aspects discussed so far, we present layer-2 and layer-3 routing, its requirements and strategies typically employed to perform such task. We also examine existing mechanisms used for VM addressing in the cloud platform and novel proposals to increase flexibility and isolation for tenants. Finally, we discuss the most relevant open research challenges and close this chapter with a brief summary of DCNs.

4.2 TOPOLOGIES

In this section, we present an overview of datacenter topologies. The topology describes how devices (routers, switches and servers) are interconnected. More formally, this is represented as a graph, in which switches, routers and servers are the nodes, and links are the edges.

4.2.1 Typical Topology

Figure 4.1 shows a canonical three-tiered multi-rooted tree-like physical topology, which is implemented in current datacenters [1, 9]. The three tiers are: (1) the access (edge) layer, comprising the top-of-rack (ToR) switches that connect servers mounted on every rack; (2) the aggregation (distribution) layer, consisting of devices that interconnect ToR switches in the access layer; and (3) the core layer, formed by routers that interconnect switches in the aggregation layer. Furthermore, every ToR switch may be connected to multiple aggregation switches for redundancy (usually 1+1 redundancy) and every aggregation switch is connected to multiple core switches. Typically, a three-tiered network is implemented in datacenters with more than 8000 servers [4]. In smaller datacenters, the core and aggregation layers are collapsed into one tier, resulting in a two-tiered datacenter topology (flat layer-2 topology) [9].

This multitiered topology has a significant amount of oversubscription, where servers attached to ToR switches have significantly more (possibly an order of magnitude) provisioned bandwidth between one another than they do with hosts in other racks [3]. Providers employ this technique in order to reduce costs and improve resource utilization, which are key properties to help them achieve economies of scale.

This topology, however, presents some drawbacks. First, the limited bisection bandwidth[1] constrains server-to-server capacity, and resources eventually get fragmented (limiting agility) [11, 12]. Second, multiple paths are poorly exploited (e.g., only a single path is used within a layer-2 domain by spanning tree protocol), which may potentially cause congestion on some links even though other paths exist in the network and have available capacity. Third, the rigid structure hinders incremental expansion [13].

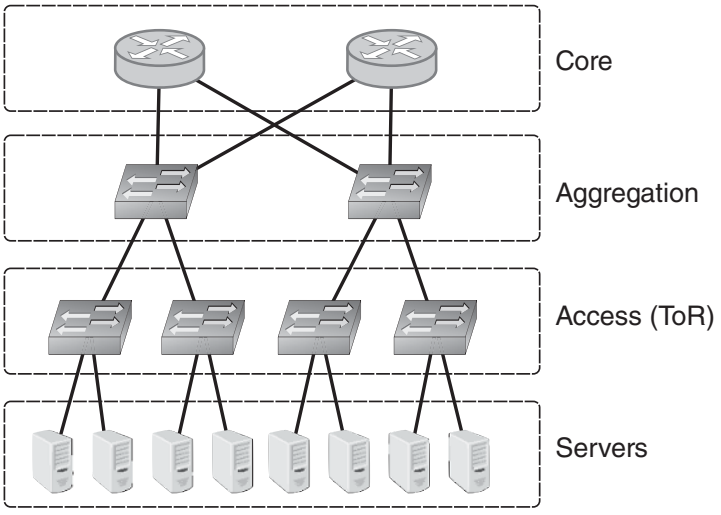


Figure 4.1. A canonical three-tiered tree-like datacenter network topology.

[1]The bisection bandwidth of the network is the worst-case segmentation (i.e., with minimum bandwidth) of the network in two equally-sized partitions [10].

Fourth, the topology is inherently failure-prone due to the use of many links, switches and servers [14]. To address these limitations, novel network architectures have been recently proposed; they can be organized in three classes [15]: switch-oriented, hybrid switch/server and server-only topologies.

4.2.2 Switch-Oriented Topologies

These proposals use commodity switches to perform routing functions, and follow a clos-based design or leverage runtime reconfigurable optical devices. A clos network [16] consists of multiple layers of switches; each switch in a layer is connected to all switches in the previous and next layers, which provides path diversity and graceful bandwidth degradation in case of failures. Two proposals follow the Clos design: VL2 [6] and Fat-Tree [4]. VL2, shown in Figure 4.2a, is an architecture for large-scale datacenters and provides multiple uniform paths between servers and full bisection bandwidth (i.e., it is non-oversubscribed). Fat-Tree, in turn, is a folded Clos topology. The topology, shown in Figure 4.2b, is organized in a non-oversubscribed k-ary tree-like structure, consisting of k-port switches. There are k two-layer pods with $k/2$ switches. Each $k/2$ switch in the lower layer is connected to $k/2$ servers, and the remaining ports are connected to $k/2$ aggregation switches. Each of the $(k/2)^2 k$-port core switches has one port connected to each of k pods. In general, a fat-tree built with k-port switches supports $k^3/4$ hosts. Despite the high capacity offered (agility is guaranteed), these architectures increase wiring costs (because of the number of links).

Optical switching architecture (OSA) [17], in turn, uses runtime reconfigurable optical devices to dynamically change physical topology and one-hop link capacities (within 10 milliseconds). It employs hop-by-hop stitching of multiple optical links to provide all-to-all connectivity for the highly dynamic and variable network demands of cloud applications. This method is shown in the example of Figure 4.3. Suppose that demands change from the left table to the right table in the figure (with a new highlighted entry). The topology must be adapted to the new traffic pattern, otherwise there will be at least one congested link. One possible approach is to increase capacity of link F–G (by reducing capacity of links F–D and G–C), so congestion can be avoided. Despite the flexibility achieved, OSA suffers from scalability issues, since it is designed to connect only a few

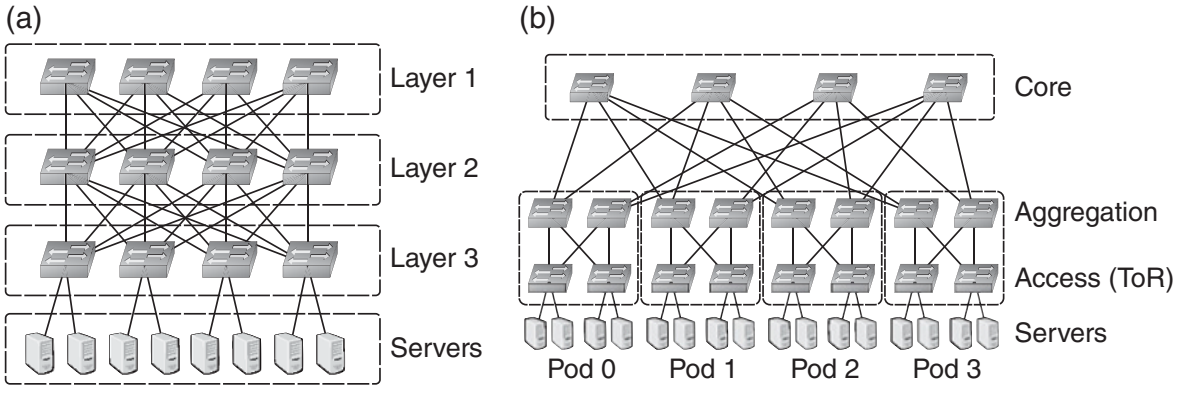


Figure 4.2. Clos-based topologies. (a) VL2 and (b) Fat-tree.

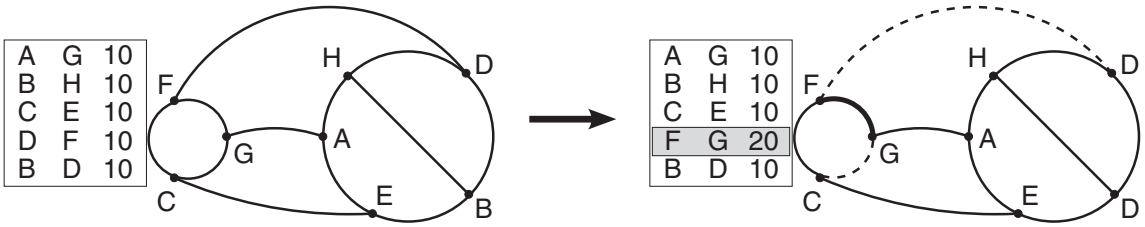


Figure 4.3. OSA adapts according to demands (adapted from Ref. [17]).

thousands of servers in a container, and latency-sensitive flows may be affected by link reconfiguration delays.

4.2.3 Hybrid Switch/Server Topologies

These architectures shift complexity from network devices to servers, i.e., servers perform routing, while low-end mini-switches interconnect a fixed number of hosts. They can also provide higher fault-tolerance, richer connectivity and improve innovation, because hosts are easier to customize than commodity switches. Two example topologies are DCell [5] and BCube [18], which can arguably scale up to millions of servers.

DCell [5] is a recursively built structure that forms a fully connected graph using only commodity switches (as opposed to high-end switches of traditional DCNs). DCell aims to scale out to millions of servers with few recursion levels (it can hold 3.2 million servers with only four levels and six hosts per cell). A DCell network is built as follows. A level-0 DCell (DCell_0) comprises servers connected to a n-port commodity switch. DCell_1 is formed with n + 1 DCell_0; each DCell_0 is connected to all other DCell_0 with one bidirectional link. In general, a level-k DCell is constructed with n + 1 DCell_{k-1} in the same manner as DCell_1. Figure 4.4a shows an example of a two-level DCell topology. In this example, a commodity switch is connected with four servers ($n = 4$) and, therefore, a DCell_1 is constructed with 5 DCell_0. The set of DCell_0 is interconnected in the following way: each server is represented by the tuple (a_1, a_0), where a_1 and a_0 are level 1 and 0 identifiers, respectively; and a link is created between servers identified by the tuples $(i, j-1)$ and (j, i), for every i and every $j > i$.

Similarly to DCell, BCube [18] is a recursively built structure that is easy to design and upgrade. Additionally, BCube provides low latency and graceful degradation of bandwidth upon link and switch failure. In this structure, clusters (a set of servers interconnected by a switch) are interconnected by commodity switches in a hypercube-based topology. More specifically, BCube is constructed as follows: BCube_0 (level-0 BCube) consists of n servers connected by a n-port switch; BCube_1 is constructed from n BCube_0 and n n-port switches; and BCube_k is constructed from n BCube_{k-1} and n^k n-port switches. Each server is represented by the tuple (x_1, x_2), where x_1 is the cluster number and x_2 is the server number inside the cluster. Each switch, in turn, is represented by a tuple (y_1, y_2), where y_1 is the level number and y_2 is the switch number inside the level. Links are created by connecting the level-k port of the i-th server in the j-th BCube_{k-1} to the j-th port of the i-th level-k switch. An example of two-level BCube with $n = 4$ (4-port switches) is shown in Figure 4.4b.

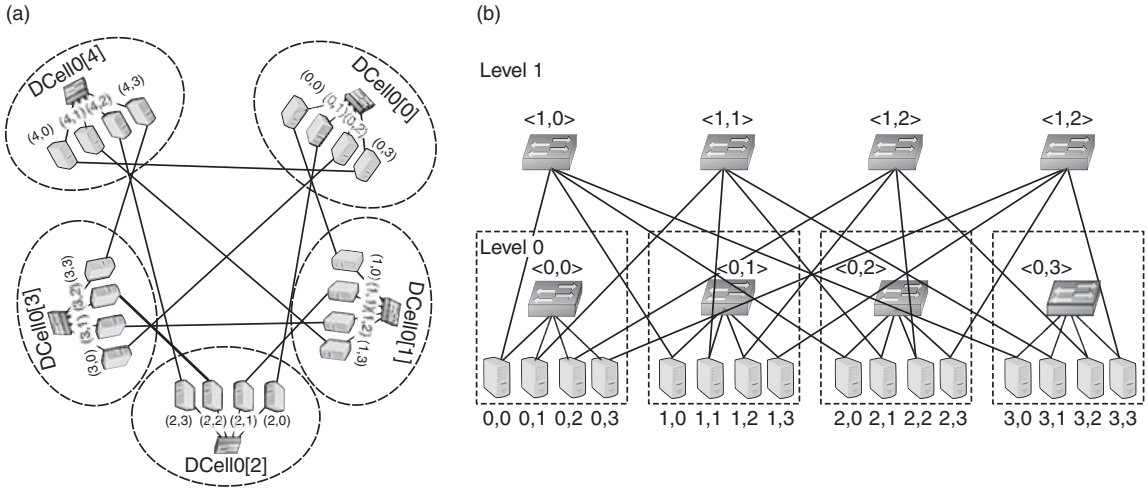


Figure 4.4. Hybrid switch/server topologies. (a) Two-level DCell and (b) two-level BCube.

Despite the benefits, DCell and BCube require a high number of NIC ports at end-hosts — causing some overhead at servers — and increase wiring costs. In particular, DCell results in non-uniform multiple paths between hosts, and level-0 links are typically more utilized than other links (creating bottlenecks). BCube, in turn, provides uniform multiple paths, but uses more switches and links than DCell [18].

4.2.4 Server-Only Topology

In this kind of topology, the network comprises only servers that perform all network functions. An example of architecture is CamCube [19], which is inspired in Content Addressable Network (CAN) [20] overlays and uses a 3D torus (k-ary 3-cube) topology with k servers along each axis. Each server is connected directly to 6 other servers, and the edge servers are wrapped. Figure 4.5 shows a 3-ary CamCube topology, resulting in 27 servers. The three most positive aspects of CamCube are (1) providing robust fault-tolerance guarantees (unlikely to partition even with 50% of server or link failures); (2) improving innovation with key-based server-to-server routing (content is hashed to a location in space defined by a server); and (3) allowing each application to define specific routing techniques. However, it does not hide topology from applications, has higher network diameter $O(\sqrt[3]{N})$ (increasing latency and traffic in the network) and hinders network expansion.

4.2.5 Summary of Topologies

Table 4.1 summarizes the benefits and limitations of these topologies by taking four properties into account: scalability, resiliency, agility and cost. The typical DCN topology has limited scalability (even though it can support hundreds of thousands of servers), as COTS switches have restricted memory size and need to maintain an entry in their Forwarding Information Base (FIB) for each VM. Furthermore, it presents low resiliency, since it provides only 1+1 redundancy, and its oversubscribed nature hinders agility.

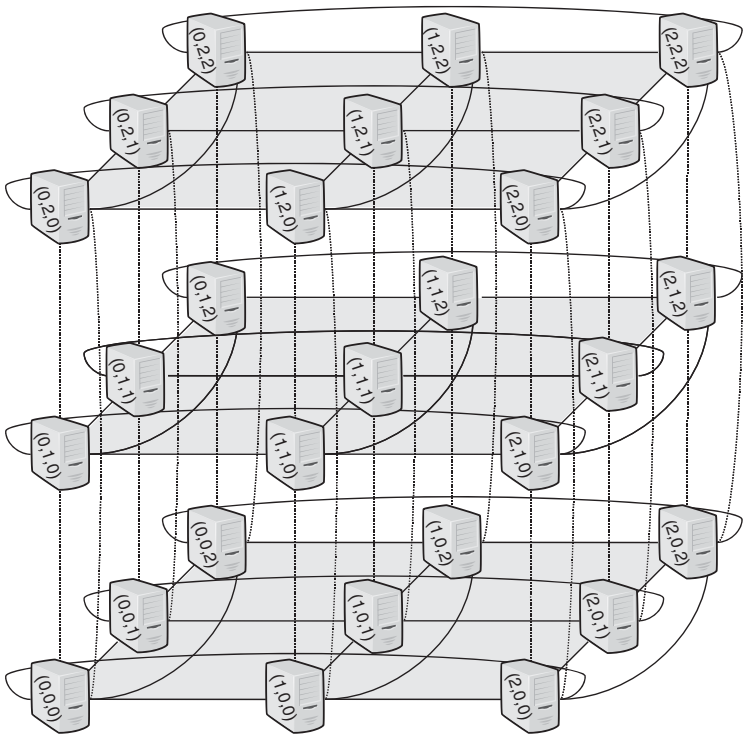


Figure 4.5. Example of 3-ary CamCube topology (adapted from Ref. [21]).

TABLE 4.1. Comparison among datacenter network topologies

Proposal	Properties			
	Scalability	Resiliency	Agility	Cost
Typical DCN	Low	Low	No	Low
Fat-Tree	High	Average	Yes	Average
VL2	High	High	Yes	High
OSA	Low	High	No	High
DCell	Huge	High	No	High
BCube	Huge	High	Yes	High
CamCube	Low	High	No	Average

Despite the drawbacks, it can be implemented with only commodity switches, resulting in lower costs.

Fat-Tree and VL2 are both instances of a Clos topology with high scalability and full bisection bandwidth (guaranteed agility). Fat-Tree achieves average resiliency, as ToR switches are connected only to a subset of aggregation devices, and has average overall costs (mostly because of increased wiring). VL2 scales through packet encapsulation, maintaining forwarding state only for switches in the network, achieves high resiliency by providing multiple shortest paths and by relying on a distributed lookup entity for handling address queries. As a downside, its deployment has increased costs (due to wiring, significant amount of exclusive resources for running the lookup system and the need of switch support for IP-in-IP encapsulation).

OSA was designed taking flexibility into account in order to improve resiliency (i.e., by using runtime reconfigurable optical devices to dynamically change physical topology and one-hop link capacities). However, it has low scalability (up to a few thousands of servers), no agility (as dynamically changing link capacities may result in congested links) and higher costs (devices should support optical reconfiguration).

DCell and BCube aim at scaling to millions of servers while ensuring high resiliency (rich connectivities between end-hosts). In contrast to BCube, DCell does not provide agility, as the set set of non-uniform multiple paths may be bottlenecked by links at level-0. Finally, their deployment costs may be significant, since they require a lot of wiring and more powerful servers in order to efficiently perform routing.

CamCube, in turn, is unlikely to partition even with 50% of server or link failures, thus achieving high resiliency. Its drawback, however, is related to scalability and agility; both properties can be hindered because of high network diameter, which indicates that, on average, more resources are needed for communication between VMs hosted by different servers. CamCube also has average deployment costs, mainly due to wiring and the need of powerful servers (to perform network functions).

As we can see, there is no perfect topology, since each proposal focus on specific aspects. Ultimately, providers are cost-driven: they choose the topology with the lowest costs, even if it cannot achieve all properties desired for a datacenter network running heterogenous applications from many tenants.

4.3 NETWORK EXPANSION

A key challenge concerning datacenter networks is dealing with the harmful effects that their ever-growing demand causes on scalability and performance. Because current DCN topologies are restricted to 1+1 redundancy and suffer from oversubscription, they can become underprovisioned quite fast. The lack of available bandwidth, in turn, may cause resource fragmentation (since it limits VM placement) [11] and reduce server utilization (as computations often depend on the data received from the network) [2]. In consequence, the DCN can loose its ability to accommodate more tenants (or offer elasticity to the current ones); even worse, applications using the network may start performing poorly, as they often rely on strict network guarantees[2].

These fundamental shortcomings have stimulated the development of novel DCN architectures (seen in Section 4.2) that provide large amounts of (or full) bisection bandwidth for up to millions of servers. Despite achieving high bisection bandwidth, their deployment is hindered by the assumption of homogeneous sets of switches (with the same number of ports). For example, consider a Fat-Tree topology, where the entire structure is defined by the number of ports in switches. These homogeneous switches limit the structure in two ways: full bisection bandwidth can only be achieved with

[2]For example, user-facing applications, such as Web services, require low-latency for communication with users, while inward computation (e.g., Map-Reduce) requires reliability and bisection bandwidth in the intra-cloud network.

specific numbers of servers (e.g., 8,192 and 27,648) and incremental upgrade may require replacing every switch in the network [13].

In fact, most physical datacenter designs are unique; hence, expansions and upgrades must be custom-designed and network performance (including bisection bandwidth, end-to-end latency and reliability) must be maximized while minimizing provider costs [11, 12]. Furthermore, organizations need to be able to incrementally expand their networks to meet the growing demands of tenants [13]. These facts have motivated recent studies [7, 11–13] to develop techniques to expand current DCNs to boost bisection bandwidth and reliability with heterogeneous sets of devices (i.e., without replacing every router and switch in the network). They are discussed next.

4.3.1 Legup

Focused on tree-like networks, Legup [12] is a system that aims at maximizing network performance at the design of network upgrades and expansions. It utilizes a linear model that combines three metrics (agility, reliability and flexibility), while being subject to the cloud provider's budget and physical constraints. In an attempt to reduce costs, the authors of Legup develop the *Theory of Heterogeneous Clos Networks* to allow modern and legacy equipment to coexist in the network. Figure 4.6 depicts an overview of the system. Legup assumes an existing set of racks and, therefore, only needs to determine aggregation and core levels of the network (more precisely, the set of devices, how they interconnect, and how they connect to ToR switches). It employs a branch and bound optimization algorithm to explore the solution space only for aggregation switches, as core switches in a heterogeneous Clos network are restricted by aggregation ones. Given a set of aggregation switches in each step of the algorithm, Legup performs three actions. First, it computes the minimum cost for mapping aggregation switches to racks. Second, it finds the minimum cost distribution of core switches to connect to the set of aggregation switches. Third, the candidate solution is bounded to check its optimality and feasibility (by verifying if any constraint is violated, including provider's budget and physical restrictions).

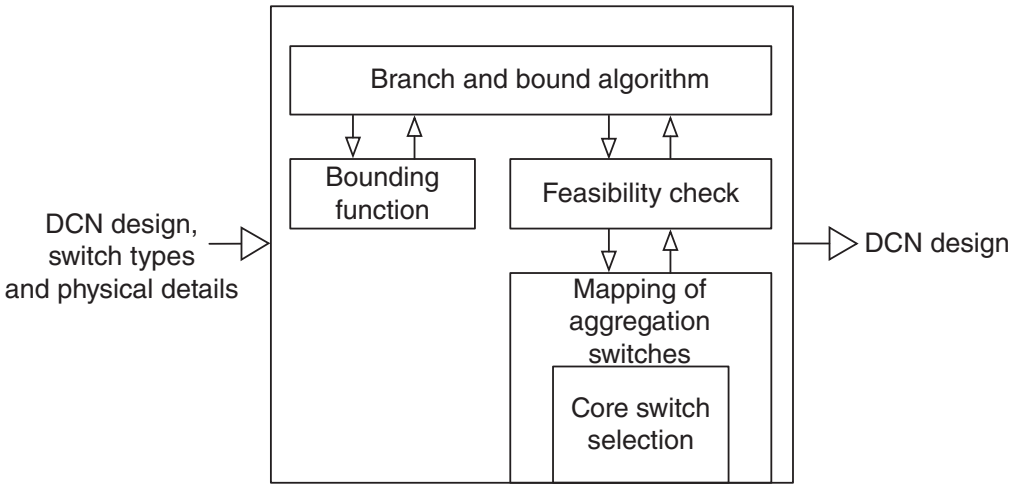


Figure 4.6. Legup's overview (adapted from Ref. [12]).

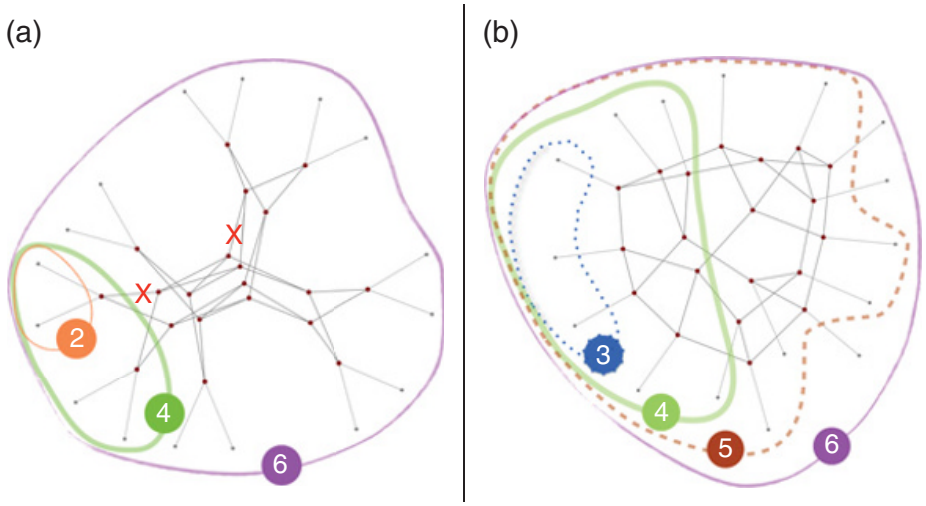


Figure 4.7. Comparison between (a) Fat-Tree and (b) Jellyfish with identical equipment (adapted from Ref. [13]).

4.3.2 Rewire

Recent advancements in routing protocols may allow DCNs to shift from a rigid tree to a generic structure [11, 22–25]. Based on this observation, Rewire [11] is a framework that performs DCN expansion on arbitrary topologies. It has the goal of maximizing network performance (i.e., finding maximum bisection bandwidth and minimum end-to-end latency), while minimizing costs and satisfying user-defined constraints. In particular, Rewire adopts a different definition of latency: while other studies model it by the worst-case hop-count in the network, Rewire also considers the speed of links and the processing time at switches (because unoptimized switches can add an order of magnitude more processing delay). Rewire uses simulated annealing (SA) [26] to search through candidate solutions and implements an approximation algorithm to efficiently compute their bisection bandwidth. The simulated annealing, however, does not take the addition of switches into account; it only optimizes the network for a given set of switches. Moreover, the process assumes uniform queuing delays for all switch ports, which is necessary because Rewire does not possess knowledge of network load.

4.3.3 Jellyfish

End-to-end throughput of a network is quantitatively proved to depend on two factors: (1) the capacity of the network and (2) the average path length (i.e., throughput is inversely proportional to the capacity consumed to deliver each byte) [13]. Furthermore, as noted earlier, rigid DCN structures hinder incremental expansion. Consequently, a degree-bounded[3] random graph topology among ToR switches, called Jellyfish [13], is introduced, with the goal of providing high bandwidth and flexibility. It supports device heterogeneity, different degrees of oversubscription and easy incremental expansion (by naturally allowing the addition of heterogeneous devices). Figure 4.7 shows a comparison

[3]Degree-bounded, in this context, means that the number of connections per node is limited by the number of ports in switches.

of Fat-Tree and Jellyfish with identical equipment and same diameter (i.e., 6). Each ring in the figure contains servers reachable within the number of hops in the labels. We see that Jellyfish can reach more servers in fewer hops, because some links are not useful from a path-length perspective in a Fat-Tree (e.g., links marked with "x"). Despite its benefits, Jellyfish's random design brings up some challenges, such as routing and the physical layout. Routing, in particular, is a critical feature needed, because it allows the use of the topology's high capacity. However, results show that the commonly used ECMP does not utilize the entire capacity of Jellyfish, and the authors propose the use of k-shortest paths and MultiPath TCP [25] to improve throughput and fairness.

4.3.4 Random Graph-Based Topologies

Singla et al. [7] analyze the throughput achieved by random graphs for topologies with both homogeneous and heterogeneous switches, while taking optimization into account. They obtain the following results: random graphs achieve throughput close to the optimal upper-bound under uniform traffic patterns for homogeneous switches, and heterogeneous networks with distinct connectivity arrangements can provide nearly identical high throughput. Then, the acquired knowledge is used as a building block for designing large-scale random networks with heterogeneous switches. In particular, they utilize the VL2 deployed in Microsoft's datacenters as a case study, showing that its throughput can be significantly improved (up to 43%) by only rewiring the same devices.

4.4 TRAFFIC

Proposals of topologies for datacenter networks presented in Sections 4.2 and 4.3 share a common goal: provide high bisection bandwidth for tenants and their applications. It is intuitive that a higher bisection bandwidth will benefit tenants, since the communication between VMs will be less prone to interference. Nonetheless, it is unclear how strong is the impact of the bisection bandwidth. This section addresses this question by surveying several recent measurement studies of DCNs. Then, it reviews proposals for dealing with related limitations. More specifically, it discusses traffic patterns—highlighting their properties and implications for both providers and tenants—and shows how literature is using such information to help designing and managing DCNs.

Traffic can be divided in two broad categories: north/south and east/west communication. North/south traffic (also known as extra-cloud) corresponds to the communication between a source and a destination host where one of the ends is located outside the cloud platform. By contrast, east/west traffic (also known as intra-cloud) is the communication in which both ends are located inside the cloud. These types of traffic usually depend on the kind and mix of applications: user-facing applications (e.g., web services) typically exchange data with users and, thus, generate north/south communication, while inward computation (i.e., MapReduce) requires coordination among its VMs, generating east/west communication. Studies [27] indicate that north/south and east-west traffic correspond to around 25% and 75% of traffic volume, respectively. They also point that

both are increasing in absolute terms, but east/west is growing on a larger scale [27]. Towards understanding traffic characteristics and how it influences the proposal of novel mechanisms, we first discuss traffic properties defined by measurement studies in the literature [9, 28–30] and, then, examine traffic management and its most relevant proposals for large-scale cloud datacenters.

4.4.1 Properties

Traffic in the cloud network is characterized by flows; each flow is identified by sequences of packets from a source to a destination node (i.e., a flow is defined by a set packet header fields, such as source and destination addresses and ports and transport protocol). Typically, a bimodal flow classification scheme is employed, using elephant and mice classes. Elephant flows comprise a large number of packets injected in the network over a short amount of time, are usually long-lived and exhibit bursty behavior. In comparison, mice flows have a small number of packets and are short-lived [3]. Several measurement studies [9, 28–31] were conducted to characterize network traffic and its flows. We summarize their findings as follows:

- *Traffic asymmetry*. Requests from users to cloud services are abundant, but small in most occasions. Cloud services, however, process these requests and typically send responses that are comparatively larger.
- *Nature of traffic*. Network traffic is highly volatile and bursty, with links running close to their capacity at several times during a day. Traffic demands change quickly, with some transient spikes and other longer ones (possibly requiring more than half the full-duplex bisection bandwidth) [32]. Moreover, traffic is unpredictable at long time scales (e.g., 100 seconds or more). However, it can be predictable on shorter timescales (at 1 or 2 seconds). Despite the predictability over small timescales, it is difficult for traditional schemes, such as statistical multiplexing, to make a reliable estimate of bandwidth demands for VMs [33].
- *General traffic location and exchange*. Most traffic generated by servers (on average 80%) stays within racks. Server pairs from the same rack and from different racks exchange data with a probability of only 11% and 0.5%, respectively. Probabilities for intra- and extra-rack communication are as follows: servers either talk with fewer than 25% or to almost all servers of the same rack; and servers communicate with less than 10% or do not communicate with servers located outside its rack.
- *Intra- and inter-application communication*. Most volume of traffic (55%) represents data exchange between different applications. However, the communication matrix between them is sparse; only 2% of application pairs exchange data, with the top 5% of pairs accounting for 99% of inter-application traffic volume. Consequently, communicating applications form several highly connected components, with few applications connected to hundreds of other applications in star-like topologies. In comparison, intra-application communication represents 45% of the total traffic, with 18% of applications generating 99% of this traffic volume.

- *Flow size, duration, and number*. Mice flows represent around 99% of the total number of flows in the network. They usually have less than 10 kilobytes and last only a few hundreds of milliseconds. Elephant flows, in turn, represent only 1% of the number of flows, but account for more than half of the total traffic volume. They may have tens of megabytes and last for several seconds. With respect to flow duration, flows of up to 10 seconds represent 80% of flows, while flows of 200 seconds are less than 0.1% (and contribute to less than 20% of the total traffic volume). Further, flows of 25 seconds or less account for more than 50% of bytes. Finally, it has been estimated that a typical rack has around 10,000 active flows per second, which means that a network comprising 100,000 servers can have over 25,000,000 active flows.
- *Flow arrival patterns*. Arrival patterns can be characterized by heavy-tailed distributions with a positive skew. They best fit a log-normal curve having ON and OFF periods (at both 15 and 100 milliseconds granularities). In particular, inter arrival times at both servers and ToR switches have periodic modes spaced apart by approximately 15 milliseconds, and the tail of these distributions is long (servers may experience flows spaced apart by 10 seconds).
- *Link utilization*. Utilization is, on average, low in all layers but the core; in fact, in the core, a subset of links (up to 25% of all core links) often experience high utilization. In general, link utilization varies according to temporal patterns (time of day, day of week and month of year), but variations can be an order of magnitude higher at core links than at aggregation and access links. Due to these variations and the bursty nature of traffic, highly utilized links can happen quite often; 86% and 15% of links may experience congestion lasting at least 10 and 100 seconds, respectively, while longer periods of congestion tend to be localized to a small set of links.
- *Hot spots*. They are usually located at core links and can appear quite frequently, but the number of hot spots never exceeds 25% of core links.
- *Packet losses*. Losses occur frequently even at underutilized links. Given the bursty nature of traffic, an underutilized network (e.g., with mean load of 10%) can experience lots of packet drops. Measurement studies found that packet losses occur usually at links with low average utilization (but with traffic bursts that go beyond 100% of link capacity); more specifically, such behavior happens at links of the aggregation layer and not at links of the access and core layers. Ideally, topologies with full bisection bandwidth (i.e., a Fat-Tree) should experience no loss, but the employed routing mechanisms cannot utilize the full capacity provided by the set of multiple paths and, consequently, there is some packet loss in such networks as well [28].

4.4.2 Traffic Management

Other set of papers [34–37] demonstrate that available bandwidth for VMs inside the datacenter can vary by a factor of five or more in the worst-case scenario. Such variability results in poor and unpredictable network performance and reduced overall application

performance [1, 38, 39], since VMs usually depend on the data received from the network to execute the subsequent computation.

The lack of bandwidth guarantees is related to two main factors. First, the canonical cloud topology is typically oversubscribed, with more bandwidth available in leaf nodes than in the core. When periods of traffic bursts happen, the lack of bandwidth up the tree (i.e., at aggregation and core layers) results in contention and, therefore, packet discards at congested links (leading to subsequent retransmissions). Since the duration of the timeout period is typically one or two orders of magnitude more than the round-trip time, latency is increased, becoming a significant source of performance variability [3]. Second, TCP congestion control does not provide robust isolation among flows. Consequently, elephant flows can cause contention in congested links shared with mice flows, leading to discarded packets from the smaller flows [2].

Recent proposals address this issue either by employing proportional sharing or by providing bandwidth guarantees. Most of them use the hose model [40] for network virtualization and take advantage of rate-limiting at hypervisors [41], VM placement [42] or virtual network embedding [43] in order to increase their robustness.

Proportional sharing. Seawall [2] and NetShare [44] allocate bandwidth at flow-level based on weights assigned to entities (i.e., VMs or services running inside these VMs) that generate traffic in the network. While both assign weights based on administrator specified policies, NetShare also supports automatic weight assignment. Both schemes are work-conserving (i.e., available bandwidth can be used by any flow that needs more bandwidth), provide max–min fair sharing and achieve high utilization through statistical multiplexing. However, as bandwidth allocation is performed per flow, such methods may introduce substantial management overhead in large datacenter networks (with over 10,000 flows per rack per second [9]). FairCloud [45] takes a different approach and proposes three allocation policies to explore the trade-off among network proportionality, minimum guarantees and high utilization. Unlike Seawall and NetShare, FairCloud does not allocate bandwidth along congested links at flow-level, but in proportion to the number of VMs of each tenant. Despite the benefits, FairCloud requires customized hardware in switches and is designed specifically for tree-like topologies.

Bandwidth guarantees. SecondNet [46], Gatekeeper [47], Oktopus [1], Proteus [48], and Hadrian [49] provide minimum bandwidth guarantees by isolating applications in virtual networks. In particular, SecondNet is a virtualization architecture that distributes the virtual-to-physical mapping, routing and bandwidth reservation state in server hypervisors. Gatekeeper configures each VM virtual NIC with both minimum and maximum bandwidth rates, which allows the network to be shared in a work-conserving manner. Oktopus maps tenants' virtual network requests (with or without oversubscription) onto the physical infrastructure and enforces these mappings in hypervisors. Proteus is built based on the observation that allocating the peak bandwidth requirements for applications leads to underutilization of resources. Hence, it quantifies the temporal bandwidth demands of applications and allocates each one of them in a different virtual network. Hadrian extends previous schemes by also taking inter-tenant communication into account and allocating applications according to a hierarchical hose model (i.e., per VM minimum bandwidth for intra-application communication and per tenant minimum guarantees for inter-tenant traffic). By contrast, a group of related proposals attempt to

provide some level bandwidth sharing among applications of distinct tenants [50–52]. The approach introduced by Marcon et al. [51] groups applications in virtual networks, taking mutually trusting relationships between tenants into account when allocating each application. It provides work-conserving network sharing, but assumes that trust relationships are determined in advance. ElasticSwitch [52] assumes there exists an allocation method in the cloud platform and focuses on providing minimum bandwidth guarantees with a work-conserving sharing mechanism (when there is spare capacity in the network). Nevertheless, it requires two extra management layers for defining the amount of bandwidth for each flow, which may add overhead. Finally, EyeQ [50] leverages high bisection bandwidth of DCNs to support minimum and maximum bandwidth rates for VMs. Therefore, it provides work-conserving sharing, but depends on the core of the network to be congestion-free. None of these approaches can be readily deployed, as they demand modifications in hypervisor source code.

4.5 ROUTING

Datacenter networks often require specially tailored routing protocols, with different requirements from traditional enterprise networks. While the latter presents only a handful of paths between hosts and predictable communication patterns, DCNs require multiple paths to achieve horizontal scaling of hosts with unpredictable traffic matrices [4, 6]. In fact, datacenter topologies (i.e., the ones discussed in Section 4.2) typically present path diversity, in which multiple paths exist between servers (hosts) in the network. Furthermore, many cloud applications (ranging from Web search to MapReduce) require substantial (possibly full bisection) bandwidth [53]. Thus, routing protocols must enable the network to deliver high bandwidth by using all possible paths in the structure. We organize the discussion according to the layer involved, starting with the network layer.

4.5.1 Layer 3

To take advantage of the multiple paths available between a source and its destination, providers usually employ two techniques: ECMP [54] and VLB [6, 55, 56]. Both strategies use distinct paths for different flows. ECMP attempts to load balance traffic in the network and utilize all paths which have the same cost (calculated by the routing protocol) by uniformly spreading traffic among them using flow hashing. VLB randomly selects an intermediate router (occasionally, a L3 switch) to forward the incoming flow to its destination.

Recent studies in the literature [46, 53, 57, 58] propose other routing techniques for DCNs. As a matter of fact, the static flow-to-path mapping performed by ECMP does not take flow size and network utilization into account [59]. This may result in saturating commodity switch L3 buffers and degrading overall network performance [53]. Therefore, a system called Hedera [53] is introduced to allow dynamic flow scheduling for general multi-rooted trees with extensive path diversity. Hedera is designed to maximize

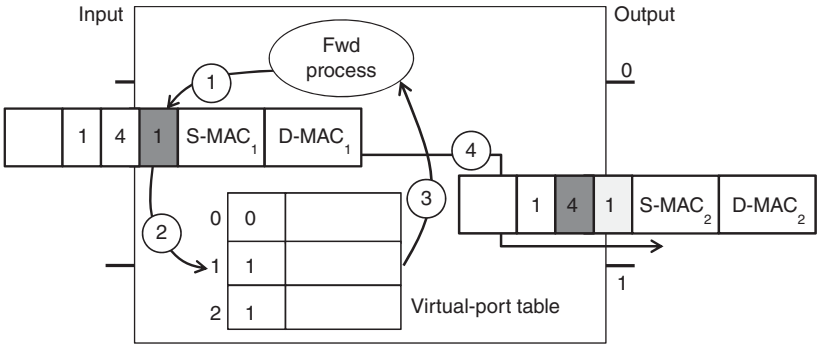


Figure 4.8. PSSR overview (adapted from Ref. [46]).

network utilization with low scheduling overhead of active flows. In general, the system performs the following steps: (1) detects large flows at ToR switches; (2) estimates network demands of these large flows (with a novel algorithm that considers bandwidth consumption according to a max–min fair resource allocation); (3) invokes a placement algorithm to compute paths for them; and (4) installs the set of new paths on switches.

Hedera uses a central OpenFlow controller[4] [60] with a global view of the network to query devices, obtain flow statistics and install new paths on devices after computing their routes. With information collected from switches, Hedera treats the flow-to-path mapping as an optimization problem and uses a simulated annealing metaheuristic to efficiently look for feasible solutions close to the optimal one in the search space. SA reduces the search space by allowing only a single core switch to be used for each destination. Overall, the system delivers close to optimal performance and up to four times more bandwidth than ECMP.

Port-switching based source routing (PSSR) [46] is proposed for the SecondNet architecture with arbitrary topologies and commodity switches. PSSR uses source routing, which requires that every node in the network knows the complete path to reach a destination. It takes advantage of the fact that a datacenter is administered by a single entity (i.e., the intra-cloud topology is known in advance) and represents a path as a sequence of output ports in switches, which is stored in the packet header. More specifically, the hypervisor of the source VM inserts the routing path in the packet header, commodity switches perform the routing process with PSSR and the destination hypervisor removes PSSR information from the packet header and delivers the packet to the destination VM. PSSR also introduces the use of virtual ports, because servers may have multiple neighbors via a single physical port (e.g., in DCell and BCube topologies). The process performed by a switch is shown in Figure 4.8. Switches read the pointer field in the packet header to get the exact next output port number (step 1), verify the next port number in the lookup virtual-port table (step 2), get the physical port number (step 3) and, in step 4, update the pointer field and forward the packet. This routing method introduces some overhead (since routing information must be included in the packet header), but, according to the authors, can be easily implemented on commodity switches using Multi-Protocol Label Switching (MPLS) [61].

[4]We will not focus our discussion in OpenFlow in this chapter. It is discussed in Chapter 6.

Bounded Congestion Multicast Scheduling (BCMS) [57], introduced to efficiently route flows in Fat-trees under the hose traffic model, aims at achieving bounded congestion and high network utilization. By using multicast, it can reduce traffic, thus minimizing performance interference and increasing application throughput [62]. BCMS is an online multicast scheduling algorithm that leverages OpenFlow to (1) collect bandwidth demands of incoming flows; (2) monitor network load; (3) compute routing paths for each flow; and (4) configure switches (i.e., installing appropriate rules to route flows). The algorithm has three main steps, as follows. First, it checks the conditions of uplinks out of source ToR switches (as flows are initially routed towards core switches). Second, it carefully selects a subset of core switches in order to avoid congestion. Third, it further improves traffic load balance by allowing ToR switches to connect to core switches with most residual bandwidth. Despite its advantages, BCMS relies on a centralized controller, which may not scale to large datacenters under highly dynamic traffic patterns such as the cloud.

Like BCMS, Code-Oriented eXplicit multicast (COXcast) [58] also focuses on routing application flows through the use of multicasting techniques (as a means of improving network resource sharing and reducing traffic). COXcast uses source routing, so all information regarding destinations are added to the packet header. More specifically, the forwarding information is encoded into an identifier in the packet header and, at each network device, is resolved into an output port bitmap by a node-specific key. COXcast can support a large number of multicast groups, but it adds some overhead to packets (since all information regarding routing must be stored in the packet).

4.5.2 Layer 2

In the Spanning Tree Protocol (STP) [63], all switches agree on a subset of links to be used among them, which forms a spanning tree and ensures a loop-free network. Despite being typically employed in Ethernet networks, it does not scale, since it cannot use the high-capacity provided by topologies with rich connectivities (i.e., Fat-Trees [24]), limiting application network performance [64]. Therefore, only a single path is used between hosts, creating bottlenecks and reducing overall network utilization.

STP's shortcomings are addressed by other protocols, including Multiple Spanning Tree Protocol (MSTP) [65], Transparent Interconnect of Lots of Links (TRILL) [22] and Link Aggregation Control Protocol (LACP) [66]. MSTP was proposed in an attempt to use the path diversity available in DCNs more efficiently. It is an extension of STP to allow switches to create various spanning trees over a single topology. Therefore, different Virtual LANs (VLANs) [67] can utilize different spanning trees, enabling the use of more links in the network than with a single spanning tree. Despite its objective, implementations only allow up to 16 different spanning trees, which may not be sufficient to fully utilize the high-capacity available in DCNs [68].

TRILL is a link-state routing protocol implemented on top of layer-2 technologies, but bellow layer-3, and is designed specifically to address limitations of STP. It discovers and calculates shortest paths between TRILL devices (called routing bridges or, in short, RBridges), which enables shortest path multihop routing in order to use all available paths

in networks with rich connectivities. RBridges run Intermediate System to Intermediate System (IS-IS) routing protocol (RFC 1195) and handle frames in the following manner: the first RBridge (ingress node) encapsulates the incoming frame with a TRILL header (outer MAC header) that specifies the last TRILL node as the destination (egress node), which will decapsulate the frame.

Link Aggregation Control Protocol (LACP) is another layer-2 protocol used in DCNs. It transparently aggregates multiple physical links into one logical link known as Link Aggregation Group (LAG). LAGs only handle outgoing flows; they have no control over incoming traffic. They provide flow-level load balancing among links in the group by hashing packet header fields. LACP can dynamically add or remove links in LAGs, but requires that both ends of a link run the protocol.

There are also some recent studies that propose novel strategies for routing frames in DCNs, namely Smart Path Assignment in Networks (SPAIN) [24] and Portland [64]. SPAIN [24] focuses on providing efficient multipath forwarding using COTS switches over arbitrary topologies. It has three components: (1) path computation; (2) path setup; and (3) path selection. The first two components run on a centralized controller with global network visibility. The controller first pre-computes a set of paths to exploit the rich connectivities in the DCN topology, in order to use all available capacity of the physical infrastructure and to support fast failover. After the path computation phase, the controller combines these multiple paths into a set of trees, with each tree belonging to a distinct VLAN. Then, these VLANs are installed on switches. The third component (path selection) runs at end-hosts for each new flow; it selects paths for flows with the goals of spreading load across the pre-computed routes (by the path setup component) and minimizing network bottlenecks. With this configuration, end-hosts can select different VLANs for communication (i.e., different flows between the same source and destination can use distinct VLANs for routing). To provide these functionalities, however, SPAIN requires some modification to end-hosts, adding an algorithm to choose among pre-installed paths for each flow.

PortLand [64] is designed and built based on the observation that Ethernet/IP protocols may have some inherent limitations when designing large-scale arbitrary topologies, such as limited support for VM migration, difficult management and inflexible communication. It is a layer-2 routing and forwarding protocol with plug-and-play support for multi-rooted Fat-Tree topologies. PortLand uses a logically centralized controller (called fabric manager) with global visibility and maintains soft state about network configuration. It assigns unique hierarchical Pseudo MAC (PMAC) addresses for each VM to provide efficient, provably loop-free frame forwarding; VMs, however, do not have the knowledge of their PMAC and believe they use their Actual MAC (AMAC). The mapping between PMAC and AMAC and the subsequent frame header rewriting is performed by edge (ToR) switches. PMACs are structured as *pod.position.port.vmid*, where each field respectively corresponds to the pod number of the edge switch, its position inside the pod, the port number in which the physical server is connected to and the identifier of the VM inside the server. With PMACs, PortLand transparently provides location-independent addresses for VMs and requires no modification in commodity switches. However, it has two main shortcomings (1) it requires a Fat-Tree topology (instead of the traditional multi-rooted oversubscribed tree) and (2) at least half of the

ToR switch ports should be connected to servers (which, in fact, is a limitation of Fat-Trees) [69].

4.6 ADDRESSING

Each server (or, more specifically, each VM) must be represented by a unique canonical address that enables the routing protocol to determine paths in the network. Cloud providers typically employ LAN technologies for addressing VMs in datacenters, which means there is a single address space to be sliced among tenants and their applications. Consequently, tenants have neither flexibility in designing their application layer-2 and layer-3 addresses nor network isolation from other applications.

Some isolation is achieved by the use of VLANs, usually one VLAN per tenant. However, VLANs are ill-suited for datacenters for four main reasons [51, 70–72]: (1) they do not provide flexibility for tenants to design their layer-2 and layer-3 address spaces; (2) they use the spanning tree protocol, which cannot utilize the high-capacity available in DCN topologies (as discussed in the previous section); (3) they have poor scalability, since no more than 4094 VLANs can be created, and this is insufficient for large datacenters; and *iv*) they do not provide location-independent addresses for tenants to design their own address spaces (independently of other tenants) and for performing seamless VM migration. Therefore, providers need to use other mechanisms to allow address space flexibility, isolation and location independence for tenants while multiplexing them in the same physical infrastructure. We structure the discussion in three main topics: emerging technologies, separation of name and locator and full address space virtualization.

4.6.1 Emerging Technologies

Some technologies employed in DCNs are: Virtual eXtensible Local Area Network (VXLAN) [73], Amazon Virtual Private Cloud (VPC) [74] and Microsoft Hyper-V [75]. VXLAN [73] is an Internet draft being developed to address scalability and multipath usage in DCNs when providing logical isolation among tenants. VXLAN works by creating overlay (virtual layer-2) networks on top of the actual layer-2 or on top of UDP/IP. In fact, using MAC-in-UDP encapsulation abstracts VM location (VMs can only view the virtual layer-2) and, therefore, enables a VXLAN network to be composed of nodes within distinct domains (DCNs), increasing flexibility for tenants using multi-datacenter cloud platforms. VXLAN adds a 24-bit segment ID field in the packet header (allowing up to 16 million different logical networks), uses ECMP to distribute load along multiple paths and requires Internet Group Management Protocol (IGMP) for forwarding frames to unknown destinations, or multicast and broadcast addresses. Despite the benefits, VXLAN header adds 50 bytes to the frame size, and multicast and network hardware may limit the usable number of overlay networks in some deployments.

Amazon VPC [74] provides full IP address space virtualization, allowing tenants to design layer-3 logical isolated virtual networks. However, it does not virtualize layer-2,

which does not allow tenants to send multicast and broadcast frames [71]. Microsoft Hyper-V [75] is a hypervisor-based system that provides virtual networks for tenants to design their own address spaces; Hyper-V enables IP overlapping in different virtual networks without using VLANs. Furthermore, Hyper-V switches are software-based layer-2 network switches with capabilities to connect VMs among themselves, with other virtual networks and with the physical network. Hyper-V, nonetheless, tends to consume more resources than other hypervisors with the same load [76].

4.6.2 Separation of Name and Locator

VL2 [6] and Crossroads [70] focus on providing location independence for VMs, so that providers can easily grow or shrink allocations and migrate VMs inside or across datacenters. VL2 [6] uses two types of addresses: location-specific addresses (LAs), which are the actual addresses in the network, used for routing; and application-specific addresses (AAs), permanent address assigned to VMs that remain the same even after migrations. VL2 uses a directory system to enforce isolation among applications (through access control policies) and to perform the mapping between names and locators; each server with an AA is associated with the LA from the ToR it is connected to. Figure 4.9 depicts how address translation in VL2 is performed: the source hypervisor encapsulates the AA address with the LA address of the destination ToR for each packet sent; packets are forwarded in the network through shortest paths calculated by the routing protocol, using both ECMP and VLB; when packets arrive at the destination ToR switch, LAs are removed (packets are decapsulated) and original packets are sent to the correct VMs using AAs. To provide location-independent addresses, VL2 requires that hypervisors run a shim layer (VL2 agent) and that switches support IP-over-IP.

Crossroads [70], in turn, is a network fabric developed to provide layer agnostic and seamless VM migration inside and across DCNs. It takes advantage of

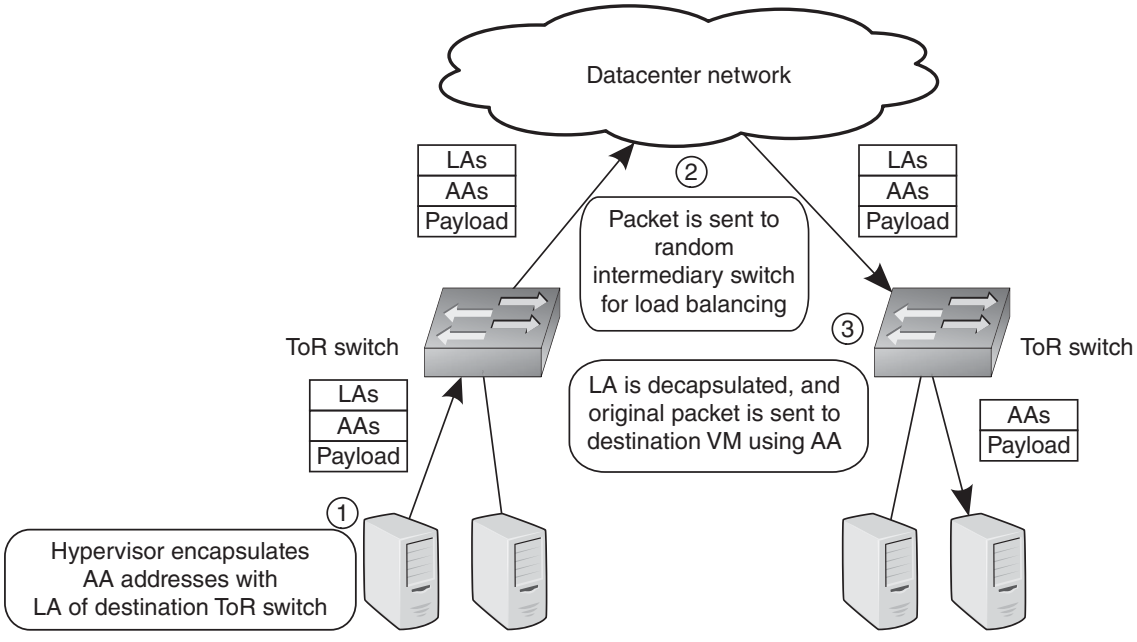


Figure 4.9. Architecture for address translation in VL2.

the Software-Defined Networking (SDN) paradigm [77] and extends an OpenFlow controller to allow VM location-independence without modifications to layer-2 and layer-3 network infrastructure. In Crossroads, each VM possess two addresses: a PMAC and a Pseudo IP (PIP), both with location and topological information embedded in them. The first one ensures that traffic originated from one datacenter and en route to a second datacenter (to which the VM was migrated) can be maintained at layer-2, while the second guarantees that all traffic destined to a migrated VM can be routed across layer-3 domains. Despite its benefits, Crossroads introduces some network overhead, as nodes must be identified by two more addresses (PMAC and PIP) in addition to the existing MAC and IP.

4.6.3 Full Address Space Virtualization

Cloud datacenters typically provide limited support for multi-tenancy, since tenants should be able to design their own address spaces (similar to a private environment) [71]. Consequently, a multi-tenant virtual datacenter architecture to enable specific-tailored layer-2 and layer-3 address spaces for tenants, called NetLord [71], is proposed. At hypervisors, NetLord runs an agent that performs Ethernet+IP (L2+L3) encapsulation over tenants' layer-2 frames and transfers them through the network using SPAIN [24] for multipathing, exploring features of both layers. More specifically, the process of encapsulating/decapsulating is shown in Figure 4.10 and occurs in three steps, as follows: (1) the agent at the source hypervisor creates L2 and L3 headers (with source IP being a tenant-assigned MAC address space identifier, illustrated as MAC_AS_ID) in order to direct frames through the L2 network to the correct edge switch; (2) the edge switch forwards the packet to the correct server based on the IP destination address in the virtualized layer-3 header; (3) the hypervisor at the destination server removes the virtual L2 and L3 headers and uses the IP destination address to deliver the original packet from the source VM to the correct VM. NetLord can be run on commodity switches and scale the network to hundreds of thousands of VMs. However, it requires an agent running on hypervisors (which may add some overhead) and support for IP forwarding on commodity edge (ToR) switches.

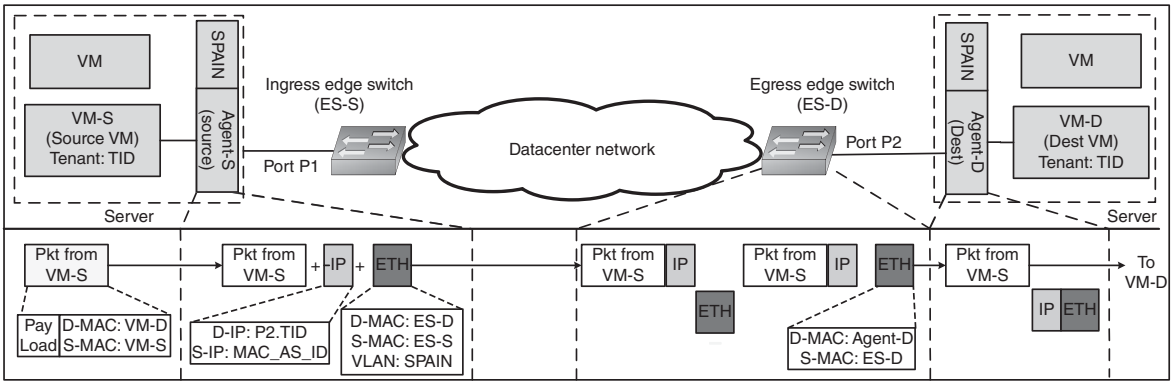


Figure 4.10. NetLord's encapsulation/decapsulation process (adapted from Ref. [71]).

4.7 RESEARCH CHALLENGES

In this section, we analyze and discuss open research challenges and future directions regarding datacenter networks. As previously mentioned, DCNs (i) present some distinct requirements from traditional networks (e.g., high scalability and resiliency); (ii) have significantly different (often more complex) traffic patterns; and (iii) may not be fully utilized, because of limitations in current deployed mechanisms and protocols (for instance, ECMP). Such aspects introduce some challenges, which are discussed next.

4.7.1 Heterogeneous and Optimal DCN Design

Presently, many Internet services and applications rely on large-scale datacenters to provide availability while scaling in and out according to incoming demands. This is essential in order to offer low response time for users, without incurring excessive costs for owners. Therefore, datacenter providers must build infrastructures to support large and dynamic numbers of applications and guarantee quality of service (QoS) for tenants. In this context, the network is an essential component of the whole infrastructure, as it represents a significant fraction of investment and contributes to future revenues by allowing efficient use of datacenter resources [15]. According to Zhang et al. [78], network requirements include (i) scalability, so that a large number of servers can be accommodated (while allowing incremental expansion); (ii) high server-to-server capacity, to enable intensive communication between any pair of servers (i.e., at full speed of their NICs); (iii) agility, so applications can use any available server when they need more resources (and not only servers located near their current VMs); (iv) uniform network utilization to avoid bottlenecks; and (v) fault tolerance to cope with server, switch and link failures. In fact, guaranteeing such requirements is a difficult challenge. Looking at these challenges from the providers viewpoint make them even more difficult to address and overcome, since reducing the cost of building and maintaining the network is seen as a key enabler for maximizing profits [15].

As discussed in Section 4.2, several topologies (e.g., Refs. [4–6, 17, 18]) have been proposed to achieve the desired requirements, with varying costs. Nonetheless, they (i) focus on homogeneous networks (all devices with the same capabilities); and (ii) do not provide theoretical foundations regarding optimality. Singla et al. [7], in turn, take an initial step towards addressing heterogeneity and optimality, as they (i) measure the upper-bound on network throughput for homogeneous topologies with uniform traffic patterns; and (ii) show an initial analysis of possible gains with heterogeneous networks. Despite this fact, a lot remains to be investigated in order to enable the development of more efficient, robust large-scale networks with heterogeneous sets of devices. In summary, very little is known about heterogeneous DCN design, even though current DCNs are typically composed of heterogenous equipment.

4.7.2 Efficient and Incremental Expansion

Providers need to be constantly expanding their datacenter infrastructures to accommodate ever-growing demands. For instance, Facebook has been expanding its datacenters

for some years [79–82]. This expansion is crucial for business, as the increase of demand may negatively impact scalability and performance (e.g., by creating bottlenecks in the network). When the whole infrastructure is upgraded, the network must be expanded accordingly, with a careful design plan, in order to allow efficient utilization of resources and to avoid fragmentation. To address this challenge, some proposals in the literature [7, 11–13] have been introduced to enlarge current DCNs without replacing legacy hardware. They aim at maximizing high bisection bandwidth and reliability. However, they often make strong assumptions (e.g., Legup [12] is designed for tree-like networks, and Jellyfish [13] requires new mechanisms for routing). Given the importance of datacenters nowadays (as home of hundreds of thousands of services and applications), the need for efficient and effective expansion of large-scale networks is a key challenge for improving provider profit, QoS offered to tenant applications and quality of experience (QoE) provided for users of these applications.

4.7.3 Network Sharing and Performance Guarantees

Datacenters host applications with diverse and complex traffic patterns and different performance requirements. Such applications range from user-facing ones (i.e., Web services and online gaming) that require low latency communication to inward computation (e.g., scientific computing) that need high network throughput. To gain better understanding of the environment, studies [1, 9, 30, 49, 83] conducted measurements and concluded that available bandwidth for VMs inside the cloud platform can vary by a factor of five or more during a predefined period of time. They demonstrate that such variability ends up impacting overall application execution time (resulting in poor and unpredictable performance). Several strategies (including Refs. [2, 47, 48, 52, 84]) have been proposed to address this issue. Nonetheless, they have one or more of the following shortcomings: (i) require complex mechanisms, which, in practice, cannot be deployed; (ii) focus on network sharing among VMs (or applications) in a homogeneous infrastructure (which simplifies the problem [85]); (iii) perform static bandwidth reservations (resulting in underutilization of resources); or (iv) provide proportional sharing (no strict guarantees). In fact, there is an inherent trade-off between providing strict guarantees (desired by tenants) and enabling work-conserving sharing in the network (desired by providers to improve utilization), which may be exacerbated in a heterogenous network. We believe this challenge requires further investigation, since such high-performance networks ideally need simple and efficient mechanisms to allow fair bandwidth sharing among running applications in a heterogeneous environment.

4.7.4 Address Flexibility for Tenants

While network performance guarantees require quantitative performance isolation, address flexibility needs qualitative isolation [71]. Cloud DCNs, however, typically provide limited support for multi-tenancy, as they have a single address space divided among applications (according to their needs and number of VMs). Thereby, tenants have no flexibility in choosing layer-2 and layer-3 addresses for applications. Note that,

ideally, tenants should be able to design their own address spaces (i.e., they should have similar flexibility to a private environment), since already developed applications may necessitate a specific set of addresses to correctly operate without source code modification. Some proposals in the literature [6, 70, 71] seek to address this challenge either by identifying end-hosts with two addresses or by fully virtualizing layer-2 and layer-3. Despite adding flexibility for tenants, they introduce some overhead (e.g., hypervisors need a shim layer to manage addresses, or switches must support IP-over-IP) and require resources specifically used for address translation (in the case of VL2). This is an important open challenge, as the lack of address flexibility may hinder the migration of applications to the cloud platform.

4.7.5 Mechanisms for Load Balancing Across Multiple Paths

DCNs usually present path diversity (i.e., multiple paths between servers) to achieve horizontal scaling for unpredictable traffic matrices (generated from a large number of heterogeneous applications) [6]. Their topologies can present two types of multiple paths between hosts: uniform and non-uniform ones. ECMP is the standard technique used for splitting traffic across equal-cost (uniform) paths. Nonetheless, it cannot fully utilize the available capacity in these multiple paths [59]. Non-uniform multiple paths, in turn, complicate the problem, as mechanisms must take more factors into account (i.e., path latency and current load). There are some proposals in the literature [46, 53, 57, 58] to address this issue, but they either cannot achieve the desired response times (e.g., Hedera) [86] or are developed for specific architectures (e.g., PSSR for SecondNet). Chiesa et al. [87] have taken an initial approach towards analyzing ECMP and propose algorithms for improving its performance. Nevertheless, further investigation is required for routing traffic across both uniform and non-uniform parallel paths, considering not only tree-based topologies, but also newer proposals such as random graphs [7, 13]. This investigation should lead to novel mechanisms and protocols that better utilize available capacity in DCNs (e.g., eliminating bottlenecks at level-0 links in DCell).

4.8 SUMMARY

In this chapter, we have presented basic foundations of datacenter networks and relevant standards, as well as recent proposals in the literature that address limitations of current mechanisms. We began by studying network topologies in Section 4.2. First, we examined the typical topology utilized in today's datacenters, which consists of a multi-rooted tree with path diversity. This topology is employed by providers to allow rich connectivity with reduced operational costs. One of its drawbacks, however, is the lack of full bisection bandwidth, which is the main motivation for proposing novel topologies. We used a three-class taxonomy to organize the state-of-the-art datacenter topologies: switch-oriented, hybrid switch/server and server-only topologies. The distinct characteristic is the use of switches and/or servers: switches only (Fat-Tree, VL2 and OSA), switches and servers (DCell and BCube) and only servers (CamCube) to perform packet routing and forwarding in the network.

These topologies, however, usually present rigid structures, which hinders incremental network expansion (a desirable property for the ever-growing cloud datacenters). Therefore, we took a look at network expansion strategies (Legup, Rewire and Jellyfish) in Section 4.3. All of these strategies have the goal of improving bisection bandwidth to increase agility (the ability to assign any VM of any application to any server). Furthermore, the design of novel topologies and expansion strategies must consider the nature of traffic in DCNs. In Section 4.4, we summarized recent measurement studies about traffic and discussed some proposals that deal with traffic management on top of a DCN topology.

Then, we discussed routing and addressing in Sections 4.5 and 4.6, respectively. Routing was divided in two categories: layer-3 and layer-2. While layer-3 routing typically employs ECMP and VLB to utilize the high-capacity available in DCNs through the set of multiple paths, layer-2 routing uses the spanning tree protocol. Despite the benefits, these schemes cannot efficiently take advantage of multiple paths. Consequently, we briefly examined proposals that deal with this issue (Hedera, PSSR, SPAIN and Portland). Addressing, in turn, is performed by using LAN technologies, which does not provide robust isolation and flexibility for tenants. Towards solving these issues, we examined the proposal of a new standard (VXLAN) and commercial solutions developed by Amazon (VPC) and Microsoft (Hyper-V). Furthermore, we discussed proposals in the literature that aim at separating name and locator (VL2 and Crossroads) and at allowing full address space virtualization (NetLord).

Finally, we analyzed open research challenges regarding datacenter networks: (i) the need to design more efficient DCNs with heterogeneous sets of devices, while considering optimality; (ii) strategies for incrementally expanding networks with general topologies; (iii) network schemes with strict guarantees and predictability for tenants, while allowing work-conserving sharing to increase utilization; (iv) address flexibility to make the migration of applications to the cloud easier; and (v) mechanisms for load balancing traffic across different multiple parallel paths (using all available capacity).

Having covered the operation and research challenges of intra-datacenter networks, the next three chapters inside the networking and communications part discuss the following subjects: inter-datacenter networks, an important topic related to cloud platforms composed of several datacenters (e.g., Amazon EC2); the emerging paradigm of SDN, its practical implementation (OpenFlow) and how these can be applied to intra- and inter-datacenter networks to provide fine-grained resource management; and mobile cloud computing, which seeks to enhance capabilities of resource-constrained mobile devices using cloud resources.

REFERENCES

1. Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM*, 2011.
2. Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *USENIX NSDI*, 2011.

3. Dennis Abts and Bob Felderman. A guided tour of data-center networking. *Communication of the ACM*, 55(6):44–51, 2012.
4. Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM*, 2008.
5. Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: A scalable and fault-tolerant network structure for data centers. In *ACM SIGCOMM*, 2008.
6. Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A scalable and flexible data center network. In *ACM SIGCOMM*, 2009.
7. Ankit Singla, P. Brighten Godfrey, and Alexandra Kolla. High throughput data center topology design. In *USENIX NSDI*, 2014.
8. Jian Guo, Fangming Liu, Xiaomeng Huang, John C.S. Lui, Mi Hu, Qiao Gao, and Hai Jin. On efficient bandwidth allocation for traffic variability in datacenters. In *IEEE INFOCOM*, 2014a.
9. Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC*, 2010.
10. Nathan Farrington, Erik Rubow, and Amin Vahdat. Data Center Switch Architecture in the Age of Merchant Silicon. In *IEEE HOTI*, 2009.
11. Andrew R. Curtis, Tommy Carpenter, Mustafa Elsheikh, Alejandro Lopez-Ortiz, and S. Keshav. Rewire: An optimization-based framework for unstructured data center network design. In *IEEE INFOCOM*, 2012.
12. Andrew R. Curtis, S. Keshav, and Alejandro Lopez-Ortiz. Legup: Using heterogeneity to reduce the cost of data center network upgrades. In *ACM Co-NEXT*, 2010.
13. Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *USENIX NSDI*, 2012.
14. Yang Liu and Jogesh Muppala. Fault-tolerance characteristics of data center network topologies using fault regions. In *IEEE/IFIP DSN*, 2013.
15. Lucian Popa, Sylvia Ratnasamy, Gianluca Iannaccone, Arvind Krishnamurthy, and Ion Stoica. A cost comparison of datacenter network architectures. In *ACM Co-NEXT*, 2010.
16. Charles Clos. A Study of non-blocking switching networks. *BellSystem Technical Journal*, 32:406–424, 1953.
17. Kai Chen, Ankit Singlay, Atul Singhz, Kishore Ramachandranz, Lei Xuz, Yueping Zhangz, Xitao Wen, and Yan Chen. Osa: An optical switching architecture for data center networks with unprecedented flexibility. In *USENIX NSDI*, 2012.
18. Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: A high performance, server-centric network architecture for modular data centers. In *ACM SIGCOMM*, 2009.
19. Hussam Abu-Libdeh, Paolo Costa, Antony Rowstron, Greg O'Shea, and Austin Donnelly. Symbiotic routing in future data centers. In *ACM SIGCOMM*, 2010.
20. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, 2001.
21. Paolo Costa, Thomas Zahn, Ant Rowstron, Greg O'Shea, and Simon Schubert. Why should we integrate services, servers, and networking in a data center? In *ACM WREN*, 2009.
22. Transparent Interconnection of Lots of Links (TRILL): RFCs 5556 and 6325, 2013. Available at: http://tools.ietf.org/rfc/index. Accessed November 20, 2014.

23. Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in seattle: A scalable ethernet architecture for large enterprises. In *ACM SIGCOMM*, 2008.
24. Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, and Jeffrey C. Mogul. SPAIN: COTS data-center Ethernet for multipathing over arbitrary topologies. In *USENIX NSDI*, 2010.
25. Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *USENIX NSDI*, 2011.
26. Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
27. Renato Recio. The coming decade of data center networking discontinuities. ICNC, August 2012. keynote speaker.
28. Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. MicroTE: Fine grained traffic engineering for data centers. In *ACM CoNEXT*, 2011.
29. Peter Bodík, Ishai Menache, Mosharaf Chowdhury, Pradeepkumar Mani, David A. Maltz, and Ion Stoica. Surviving failures in bandwidth-constrained datacenters. In *ACM SIGCOMM*, 2012.
30. Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: Measurements & analysis. In *ACM IMC*, 2009.
31. Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *IEEE INFOCOM*, 2010.
32. Chi H. Liu, Andreas Kind, and Tiancheng Liu. Summarizing data center network traffic by partitioned conservative update. *IEEE Communications Letters*, 17(11):2168–2171, 2013.
33. Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *IEEE INFOCOM*, 2011.
34. Alexandrm-Dorin Giurgiu. Network performance in virtual infrastrucures, February 2010. Available at: http://staff.science.uva.nl/~delaat/sne-2009-2010/p29/presentation.pdf. Accessed November 20, 2014.
35. Dave Mangot. Measuring EC2 system performance, May 2009. Availabel at: http://bit.ly/48Wui. Accessed November 20, 2014.
36. Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, 3(1–2):460–471, 2010.
37. Guohui Wang and T. S. Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In *IEEE INFOCOM*, 2010.
38. Haiying Shen and Zhuozhao Li. New bandwidth sharing and pricing policies to achieve A win-win situation for cloud provider and tenants. In *IEEE INFOCOM*. 2014.
39. Eitan Zahavi, Isaac Keslassy, and Avinoam Kolodny. Distributed adaptive routing convergence to non-blocking DCN routing assignments. *IEEE Journal on Selected Areas in Communications*, 32(1):88–101, 2014.
40. Nick G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K. K. Ramakrishnan, and Jacobus E. van der Merive. A flexible model for resource management in virtual private networks. In *ACM SIGCOMM*, 1999.
41. Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex C. Snoeren. Cloud control with distributed rate limiting. In *ACM SIGCOMM*, 2007.

42. Joe W. Jiang, Tian Lan, Sangtae Ha, Minghua Chen, and Mung Chiang. Joint VM placement and routing for data center traffic engineering. In *IEEE INFOCOM*, 2012.
43. Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM Compuer. Communication Review*, 38:17–29, 2008.
44. Vinh The Lam, Sivasankar Radhakrishnan, Rong Pan, Amin Vahdat, and George Varghese. Netshare and stochastic netshare: Predictable bandwidth allocation for data centers. *ACM SIGCOMM CCR*, 42(3), 2012.
45. Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. FairCloud: Sharing the network in cloud computing. In *ACM SIGCOMM*, 2012.
46. Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *ACM CoNEXT*, 2010.
47. Henrique Rodrigues, Jose Renato Santos, Yoshio Turner, Paolo Soares, and Dorgival Guedes. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *USENIX WIOV*, 2011.
48. Di Xie, Ning Ding, Y. Charlie Hu, and Ramana Kompella. The only constant is change: Incorporating time-varying network reservations in data centers. In *ACM SIGCOMM*, 2012.
49. Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gunawardena, and Greg O'Shea. Chatty tenants and the cloud network sharing problem. In *USENIX NSDI*, 2013a.
50. Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Changhoon Kim, and Albert Greenberg. EyeQ: Practical network performance isolation at the edge. In *USENIX NSDI*, 2013.
51. Daniel Stefani Marcon, Rodrigo Ruas Oliveira, Miguel Cardoso Neves, Luciana Salete Buriol, Luciano Paschoal Gaspary, and Marinho Pilla Barcellos. Trust-based Grouping for Cloud Datacenters: Improving security in shared infrastructures. In *IFIP Networking*, 2013.
52. Lucian Popa, Praveen Yalagandula, Sujata Banerjee, Jeffrey C. Mogul, Yoshio Turner, and Jose Renato Santos. ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing. In *ACM SIGCOMM*, 2013.
53. Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: dynamic flow scheduling for data center networks. In *USENIX NSDI*, 2010.
54. C. Hopps. Analysis of an equal-cost multi-path algorithm, 2000. RFC 2992.
55. Albert Greenberg, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Towards a next generation data center architecture: Scalability and commoditization. In *ACM PRESTO*, 2008.
56. Leslie G. Valiant and Gordon J. Brebner. Universal schemes for parallel communication. In *ACM STOC*, 1981.
57. Zhiyang Guo, Jun Duan, and Yuanyuan Yang. On-line multicast scheduling with bounded congestion in Fat-Tree data center networks. *IEEE Journal on Selected Areas in Communications*, 32(1):102–115, 2014b.
58. Wen-Kang Jia. A scalable multicast source routing architecture for data center networks. *IEEE Journal on Selected Areas in Communications*, 32(1):116–123, 2014.

59. Sivasankar Radhakrishnan, Malveeka Tewari, Rishi Kapoor, George Porter, and Amin Vahdat. Dahu: Commodity switches for direct connect data center networks. In *ACM/IEEE ANCS*, 2013.
60. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38:69–74, 2008.
61. E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture, 2001. RFC 3031.
62. Dan Li, Mingwer Xu, Ying Liu, Xia Xie, Yong Cui, Jingyi Wang, and Gihai Chen. Reliable multicast in data center networks., 2014. IEEE Transactions Computers, 63: 2011-2024, 2014.
63. 802.1D - MAC Bridges, 2013. Available at: http://www.ieee802.org/1/pages/802.1D.html. Accessed November 20, 2014.
64. Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. PortLand: A scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM*, 2009.
65. 802.1s - Multiple Spanning Trees, 2013. Available at: http://www.ieee802.org/1/pages/802.1s.html. Accessed November 20, 2014.
66. 802.1ax - Link Aggregation Task Force, 2013. Available at: http://ieee802.org/3/axay/. Accessed November 20, 2014.
67. 802.1Q - Virtual LANs, 2013. Available at: http://www.ieee802.org/1/pages/802.1Q.html. Accessed November 20, 2014.
68. Understanding Multiple Spanning Tree Protocol (802.1s), 2007. Available at: http://www.cisco.com/en/US/tech/tk389/tk621/technologies_white_paper09186a0080094cfc.shtml. Accessed November 20, 2014.
69. Md. Faizal Bari, Raonf Boutaba, Rafael Esteves, Lisandro Z. Granville, Maxim Podlesny, Md. Golam Rabbani, Qi Zhang, and Mohamed F. Zhani. Data center network virtualization: A survey. *IEEE Communications Surveys Tutorials*, 15(2):909–928, 2013.
70. Vijay Mann, Ailkumar Vishnoi, Kalapriya Kannan, and Shivkumar Kalyanaraman. CrossRoads: Seamless VM mobility across data centers through software defined networking. In *IEEE/IFIP NOMS*, 2012.
71. Jayaram Mudigonda, Praveen Yalagandula, Jeff Mogul, Bryan Stiekes, and Yanick Pouffary. NetLord: A scalable multi-tenant network architecture for virtualized datacenters. In *ACM SIGCOMM*, 2011.
72. Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, and John Carter. PAST: Scalable ethernet for data centers. In *ACM CoNEXT*, 2012.
73. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, 2013. Available at: http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-02. Accessed November 20, 2014.
74. Amazon Virtual Private Cloud, 2013. Available at: http://aws.amazon.com/vpc/. Accessed November 20, 2014.
75. Microsoft Hyper-V Server 2012, 2013a. Available at: http://www.microsoft.com/en-us/server-cloud/hyper-v-server/. Accessed November 20, 2014.
76. Hyper-V Architecture and Feature Overview, 2013b. Available at: http://msdn.microsoft.com/en-us/library/dd722833(v=bts.10).aspx. Accessed November 20, 2014.

77. Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *USENIX OSDI*, 2010.
78. Yan Zhang and Nirwan Ansari. On architecture design, congestion notification, tcp incast and power consumption in data centers. *Communications Surveys Tutorials, IEEE*, 15(1):39–64, 2013.
79. Facebook to Expand Prineville Data Center, 2010. Available at: https://www.facebook.com/notes/prineville-data-center/facebook-to-expand-prineville-data-center/411605058132. Accessed November 20, 2014.
80. Tad Andersen. Facebook's Iowa expansion plan goes before council, 2014. Available at: http://www.kcci.com/news/facebook-just-announced-new-expansion-plan-in-iowa/25694956#!0sfWy. Accessed November 20, 2014.
81. David Cohen. Facebook eyes expansion of oregon data center, 2012. Available at: http://allfacebook.com/prineville-oregon-data-center-expansion_b97206. Accessed November 20, 2014.
82. John Rath. Facebook Considering Asian Expansion With Data Center in Korea, 2013. Available at: http://www.datacenterknowledge.com/archives/2013/12/31/asian-expansion-has-facebook-looking-at-korea/. Accessed November 20, 2014.
83. Ryan Shea, Feng Wang, Haiyang Wang, and Jiangchuan Liu. A deep investigation into network performance in virtual machine based cloud environment. In *IEEE INFOCOM*. 2014.
84. Katrina LaCurts, Shuo Deng, Ameesh Goyal, and Hari Balakrishnan. Choreo: Network-aware task placement for cloud applications. In *ACM IMC*, 2013.
85. Fei Xu, Fangming Liu, Hai Jin, and A.V. Vasilakos. Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. *Proceedings of the IEEE*, 102(1):11–31, 2014.
86. Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pp. 254–265, New York, 2011. ACM. Available at: http://doi.acm.org/10.1145/2018436.2018466. Accessed November 20, 2014.
87. Marco Chiesa, Guy Kindler, and Michael Schapira. Traffic engineering with equal-cost-multiPath: an algorithmic perspective. In *IEEE INFOCOM*, 2014.

APPENDIX C PUBLISHED PAPER AT IEEE ICC 2015

In this paper, we introduce a resource allocation strategy that aims at providing an efficient way to predictably share bandwidth among applications and at minimizing resource underutilization while maintaining low management overhead. To demonstrate the benefits of the strategy, we develop IoNCloud, a system that implements the proposed allocation scheme. IoNCloud employs the abstraction of attraction/repulsion among applications according to their temporal bandwidth demands in order to group them in virtual networks. In doing so, we explore the trade-off between high resource utilization (which is desired by providers to achieve economies of scale) and strict network guarantees (necessary for tenants to run jobs predictably). Evaluation results show that IoNCloud can (a) provide predictable network sharing; and (b) reduce allocated bandwidth, resource underutilization and management overhead when compared against state-of-the-art proposals.

- **Title**: IoNCloud: exploring application affinity to improve utilization and predictability in datacenters
- **Conference**: IEEE International Conference on Communications (ICC 2015)
- **Type**: Main track (full-paper)
- **Qualis**: A2
- **URL**: <http://ieeexplore.ieee.org/document/7249198/>
- **Date**: June 8-12, 2015
- **Location**: London, UK
- **Digital Object Identifier (DOI)**: 10.1109/ICC.2015.7249198

IoNCloud: exploring application affinity to improve utilization and predictability in datacenters

Daniel S. Marcon*, Miguel C. Neves*, Rodrigo R. Oliveira*, Leonardo R. Bays*,
Raouf Boutaba†, Luciano P. Gaspary*, Marinho P. Barcellos*
* Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
† David R. Cheriton School of Computer Science, University of Waterloo, Canada
{daniel.stefani, mcneves, ruas.oliveira, lrbays, paschoal, marinho}@inf.ufrgs.br, rboutaba@uwaterloo.ca

***Abstract*—The intra-cloud network is typically shared in a best-effort manner, which causes tenant applications to have no actual bandwidth guarantees. Recent proposals address this issue either by statically reserving a slice of the physical infrastructure for each application or by providing proportional sharing among flows. The former approach results in overprovisioned network resources, while the latter requires substantial management overhead. In this paper, we introduce a resource allocation strategy that aims at providing an efficient way to predictably share bandwidth among applications and at minimizing resource underutilization while maintaining low management overhead. To demonstrate the benefits of the strategy, we develop IoNCloud, a system that implements the proposed allocation scheme. IoNCloud employs the abstraction of attraction/repulsion among applications according to their temporal bandwidth demands in order to group them in virtual networks. In doing so, we explore the trade-off between high resource utilization (which is desired by providers to achieve economies of scale) and strict network guarantees (necessary for tenants to run jobs predictably). Evaluation results show that IoNCloud can (a) provide predictable network sharing; and (b) reduce allocated bandwidth, resource underutilization and management overhead when compared against state-of-the-art proposals.**

I. Introduction

Cloud providers lack practical, efficient and reliable mechanisms to offer bandwidth guarantees for applications [1], [2]. The intra-cloud network is typically oversubscribed and shared in a best-effort manner, relying on TCP to achieve high network utilization and scalability. TCP, nonetheless, does not provide robust isolation among flows in the network [3]; in fact, long-lived flows with a large number of packets are privileged over small ones (which is typically called *performance interference* [4]) [5]. Moreover, recent studies [6], [7] show that bandwidth available for virtual machines (VMs) in the intra-cloud network can vary by a factor of five or more, resulting in poor and unpredictable overall application performance.

The lack of network guarantees directly impacts both tenants and providers. Tenants are unable to enforce the allocation of network resources for their requests (which particularly hinders applications with strict bandwidth requirements) and can only deploy some specific enterprise applications in the cloud [8]. Moreover, costs are unpredictable due to high network variability (in many services, the subsequent computation depends on the data received from the network [9], [10]). Providers, in turn, may lose revenue, because performance interference ends up reducing datacenter throughput [1], [6].

Recent proposals [3], [6], [8], [11], [12] address this issue either by offering minimum guarantees or by providing proportional sharing. The former explicitly reserves a slice of the physical infrastructure for each application, which results in overprovisioned resources for tenants (since the temporal network usage of applications is not constant). The latter, in turn, assigns administrator-specific weights for entities (such as VMs and processes) in order to provide proportional sharing at flow-level in the network. However, it requires substantial management overhead, since bandwidth consumed by each flow is determined according to its weight for each link in the path, and large-scale datacenter networks can have over 10 million flows per second [13].

In this paper, we leverage the key observation that temporal bandwidth demands of cloud applications do not peak at exactly the same time [14], [15] and propose a resource allocation strategy for reserving and isolating network resources in cloud datacenters. It aims at minimizing resource underutilization while providing an efficient way to predictably share bandwidth among applications, with low management overhead. To show the benefits of the strategy, we develop IoNCloud (Isolation of Networks in the Cloud), a system that implements the proposed allocation scheme. IoNCloud employs the abstraction of attraction/repulsion among tenant applications according to their temporal network usage and need of isolation, and groups them into virtual networks (VNs) with bandwidth guarantees. In doing so, we seek to explore the trade-off between high resource utilization (a common goal for providers to reduce operational costs) and strict network guarantees (desired by tenants).

Overall, the major contributions of this paper are threefold. First, we propose a topology-agnostic network-performance-aware resource allocation strategy for cloud datacenters. It improves network predictability by grouping tenant applications into virtual networks according to their temporal bandwidth demands. Second, we develop IoNCloud, a system that implements the proposed strategy for large-scale datacenters. IoNCloud (i) groups applications in VNs; (ii) maps them on the physical substrate; and (iii) provisions network resources at each link the VN was allocated on according to peak aggregate demands of applications in the same group that utilize the link (i.e., the bandwidth required at the period of time when the sum of network demands of applications belonging to the same group is the highest). Third, we evaluate and show that, in comparison with the state-of-the-art [6], IoNCloud provides the same level of network predictability with less bandwidth reserved for applications, reduced resource underutilization and lower management overhead.

The remainder of this paper is organized as follows.

978-1-4673-6432-4/15/$31.00 ©2015 IEEE

Section II examines related work. In Section III, we introduce our resource allocation strategy (and its implementation, IoNCloud), and Section IV presents the evaluation of the proposed strategy. Finally, Section V discusses the generality and limitations of IoNCloud, and Section VI closes the paper.

II. Related Work

Most related approaches attempt to offer bandwidth guarantees by taking advantage of rate-limiting at hypervisors, VM placement and VN embedding in order to increase their robustness. They can be separated in three classes, as discussed below.

Spatial-temporal awareness. Proteus [9], Choreo [16] and the approach developed by Chen and Shen [15] use spatial and temporal demands of applications to map them in the cloud. However, they present some drawbacks. Proteus requires a complex allocation scheme and provides only a rigid network model for each application, defined at its allocation time. Choreo requires its placement algorithm to have detailed knowledge about current network state; such information may not be easily obtained in large-scale datacenter networks. In particular, Proteus and Choreo may add substantial management overhead to achieve their goals. Finally, Chen and Shen only focus on temporal demands of computing resources.

Network guarantees. Oktopus [6] and SecondNet [17] provide strict bandwidth guarantees by isolating each application in a distinct VN. Despite their benefits, these approaches result in underutilization of resources and internal fragmentation of both computing and network resources upon high rate of tenant arrival and departure. EyeQ [12] and Gatekeeper [18], in turn, can offer bandwidth guarantees only when the core of the network is congestion-free. Baraat [1] and Varys [10] achieve high network utilization, but cannot provide strict bandwidth guarantees for tenants. Finally, ElasticSwitch [8] and the Logistic Model [3] are orthogonal to our approach, as they assume there exists an allocation method in the cloud platform (i.e., applications are already allocated).

Proportional sharing. Seawall [4] and Hadrian [11] allow bandwidth sharing at link-level according to weights assigned to VMs. FairCloud [19], in turn, proposes mechanisms that explore the trade-off among network proportionality, minimum guarantees and high utilization. These methods, however, result in substantial management overhead, because bandwidth consumed by each flow at each link is determined according to its weight in comparison to other flows sharing the same link (and the intra-cloud network can have over 10 million flows per second [13]).

In summary, these approaches either result in overprovisioned network resources or require substantial management overhead. Therefore, we introduce a resource allocation strategy that aims at providing network predictability with reduced bandwidth underutilization and low management overhead, and materialize it by developing a system called IoNCloud.

III. IoNCloud

The IoNCloud system implements our novel approach to allocate tenant applications in large-scale cloud platforms. The proposed strategy aims at providing performance predictability in the intra-cloud network while minimizing resource underutilization. To achieve this, unlike previous work [6], [9], [17], [18], IoNCloud groups applications in virtual networks according to their temporal bandwidth demands.

In this strategy, all applications that belong to the same group share the same set of (virtual) network resources (i.e., they have shared bandwidth guarantees). Virtual networks, in turn, are completely isolated from one another, which means that each group has a guaranteed amount of network resources. An abstract view of IoNCloud is shown in Figure 1, which depicts application requests being received and allocated in two steps. The first step is responsible for application demand analysis and grouping, while the second maps VNs (groups composed of sets of applications) onto the physical substrate.

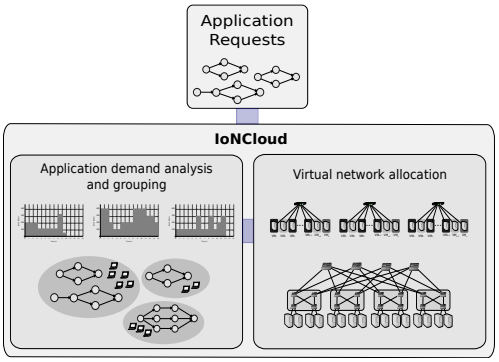


Fig. 1: IoNCloud model overview.

We first discuss how to obtain network profiles of applications (Section III-A) and describe application requests (Section III-B). Then, we present our novel strategy to group complementary applications in VNs (Section III-C) and to embed VNs on the cloud substrate (Section III-D).

A. Network Profile of Applications

IoNCloud assumes that network profiles were previously generated (using techniques proposed in the literature, such as the ones described by Xie et al. [9], LaCurts et al. [16] and Lee et al. [2]) and, thus, uses such information as input for incoming application requests. Like Choreo [16], IoNCloud considers, in each profile, the number of bytes sent by the application rather than the observed rate, as the former is independent of network congestion.

In particular, we highlight the feasibility of obtaining these profiles. Several studies [6], [9], [13], [15], [16] have conducted experiments on VM resource utilizations during both short- and long-term periods of time. Their main findings are summarized as follows: i) application traffic patterns are predictable; ii) VMs of the same application (such as MapReduce) tend to have similar resource consumption; iii) the same application running different datasets has similar patterns of resource usage; and iv) periodical (e.g., daily) patterns of resource utilization can be observed for long-term applications.

These results enabled the proposal of some techniques to profile cloud applications. For instance, Xie et al. [9] and LaCurts et al. [16] use network monitoring tools (sFlow and tcpdump) to collect traffic traces (gathering application communication patterns), while Lee et al. [2] discuss the utilization of application templates (provided as a library for users). Xie et

al. also convert the output of these measurements into coarse-grained pulse functions. Both studies perform these profiling runs during a testing phase or in production environments, which allows them to collect sufficient information to create network profiles before running applications in the cloud. Therefore, such techniques can be used for IoNCloud, so that application profiles are known before allocation.

IoNCloud also considers applications that cannot have their network profiles generated in advance (for instance, because the application requires an elevated amount of resources to run a profiling test or concludes very quickly). In such situations, the time-varying function $B(t)$ (detailed in Section III-B), which indicates the temporal network demands of applications, is represented by a constant function (i.e., the same bandwidth requirement during the entire application lifetime). This constant value is specified by the tenant when submitting the request to the cloud.

B. Application Requests

Tenants request applications using the hose-model (similarly to prior work [8], [9], [19], [20]). In this abstraction, all VMs of an application are connected to a virtual switch through dedicated bidirectional links. Each application request is represented by its resource demand and formally defined by <N, $B(t)$>, with the terms specifying the number of VMs and the temporal bandwidth required by the application. The bandwidth demand is a time-varying function $B(t)$, similarly to Proteus [9]. It represents the bandwidth required by the application at time "t". The amount of bandwidth of each link connecting a VM to the application virtual switch is represented by $\max(B(t))$, which denotes the peak temporal demand of the application's VM. This fine-grained specification allows IoNCloud to capture network requirements of applications in a precise manner.

Without loss of generality, we follow previous work [6], [9] and make two assumptions. First, we abstract away computing resources and assume a homogeneous set of VMs (i.e., equal in terms of CPU, memory and storage consumption). Second, we consider that all VMs of a given application follow the same network model[1].

C. Application Demand Analysis and Grouping

This first step is responsible for analyzing network demands of applications and grouping them in VNs. This way, high resource utilization can be achieved without hurting predictability.

Figure 2 shows an example of how IoNCloud performs this process. In Figure 2(a), bandwidth requirements (dashed lines) of two applications (A and B) are fully guaranteed through a simple static reservation model (where the peak bandwidth is reserved, represented by dotted lines). However, this basic model causes underutilization of resources (shown by arrows in the figure), as unused bandwidth of one application (virtual network) cannot be used by any other application [8], [9]. IoNCloud, in contrast, enables applications with complementary bandwidth requirements to share network resources. This is done by grouping them in the same VN and reserving the peak aggregate demand, represented by the dotted line in Figure 2(b). Therefore, IoNCloud achieves better resource utilization, since periods of low demand from one application can be compensated by periods of high demand from other ones. Note that IoNCloud removes performance interference in the network by reserving the peak aggregate bandwidth of the applications (that belong to the same group) sharing a given link. Furthermore, it significantly reduces management overhead when compared to Proteus [9], since the latter must modify reservations as time passes by.

Algorithm. The key idea is based on minimizing the amount of unused bandwidth for each group created (i.e., reducing wasted bandwidth). Algorithm 1 retrieves one application (app) at a time from the set of applications $Applications$ and verifies three possibilities of grouping: i) creating a new group composed of app and another application from the set $Applications$ (i.e., trying all pairs of possible groupings of app with other incoming applications and selecting the one with least underutilization); ii) inserting app into one of the existing groups; and iii) creating a new group with app only. After verifying these possibilities, the best option is selected. Finally, once the entire bundle of incoming applications has been analyzed and included into groups, the algorithm concludes, returns the set of groups and the process of allocating each group (represented by a VN) on the cloud is started.

Algorithm 1: Network-aware group creation.

Input : Bundle of applications to be allocated in the cloud
Output: List of application groups $GroupList$

```
Create a set Applications with all incoming applications;
Create an empty set GroupList;
foreach app ∈ Applications do
    Evaluate three possibilities of grouping:
        Creating new group containing app and a chosen application from Applications;
        Including app in existing group of the set GroupList;
        Creating new group with single application app;
    Among the three above, select the option with least underutilization;
    Remove grouped applications from Applications;
    if new group was created then
        Include new group in the set GroupList;
return the set GroupList;
```

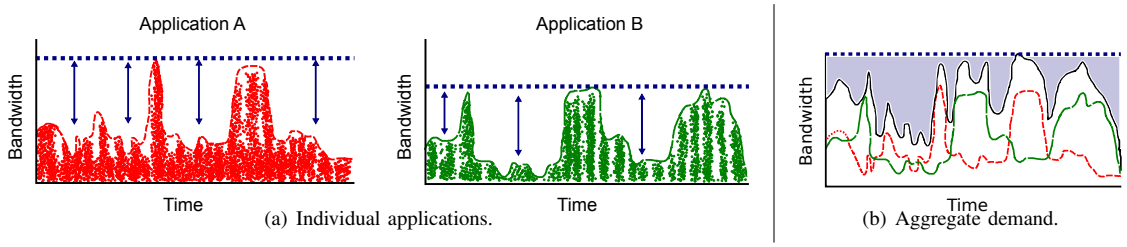
D. Virtual Network Allocation

This step is responsible for allocating each VN (group composed of applications that present complementary temporal bandwidth demands) on the physical infrastructure.

A simplified example is shown in Figure 3, where there are only two groups to be allocated, each one with two applications. For each group, IoNCloud prioritizes clustering VMs of the same application in the same physical server, since good locality reduces the amount of network resources used for intra-application communication[2]. For a single application, VMs located in the same server do not consume network resources when they communicate with each other. VMs allocated in different servers, in turn, need a certain amount of bandwidth to exchange data, which is given by the server with the lowest peak aggregate rate for an application. Consider "Server 1" (S_1) and "Server 3" (S_3) in the figure, which host application 1 (app_1). The bandwidth needed by VMs of this

[1]Many applications, such as MapReduce (which represents an important class of applications running in datacenters), have similar bandwidth demands among their VMs [9].

[2]We follow related work [6], [9], [18] and consider only intra-application communication when allocating applications in the cloud platform, as this type of communication represents most of the traffic in the cloud [11].



(a) Individual applications. (b) Aggregate demand.

Fig. 2: Temporal network usage.

application for communication among themselves is given by $\min(|S_{1,app_1}|, |S_{3,app_1}|) * \max(B(t))$, where $|S_{x,app_1}|$ represents the number of VMs of app_1 placed at the x^{th} server and $\max(B(t))$ denotes the peak bandwidth used by a single VM during its lifetime. In this example, $\min(6, 2) * 15 = 30$ Mbps.

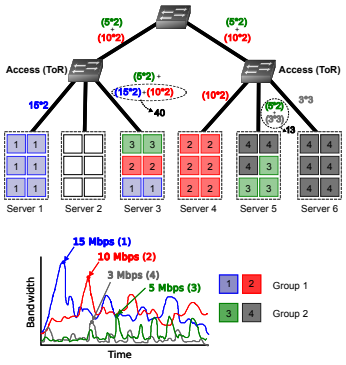


Fig. 3: Shared bandwidth guarantees for applications.

A group of applications, in contrast, requires the peak aggregate bandwidth demand of the group. Therefore, the allocated bandwidth on each physical link l of a VN corresponds to the peak aggregate demand of VMs in the group that use l. This allocation is illustrated in Figure 3 in two situations: i) when more than one application of the same group share a link, the aggregate peak bandwidth of the VMs of these applications is reserved: for instance, in Server 3, VMs of applications 1 and 2 from group 1 share the access link and, if they were isolated, they would require 50 Mbps ($15 \times 2 + 10 \times 2$), but only 40 Mbps is reserved because this is the peak temporal demand of the group for this link (according to the temporal demands of applications shown below the servers in the figure); and ii) when a single application of a group uses a link, the bandwidth reserved corresponds to the amount needed by its VMs alone (other applications of the group do not need to use that link, as we can see in core links). Last but not least, note that VNs do not share bandwidth with one another (i.e., groups are completely isolated).

Algorithm. Algorithm 2 takes advantage of application affinity to allocate VNs on the substrate[3]. Since locality is key to make efficient use of resources, we address it with two granularities: i) VMs of the same application are mapped on the infrastructure according to a VM placement objective (since IoNCloud is agnostic about VM placement, these objectives will be detailed after the overview of the allocation algorithm); and ii) VMs belonging to applications of the same group are allocated close to each other, because their bandwidth demands are complementary and, thus, network underutilization can be reduced (as determined by the grouping algorithm in the previous step). The algorithm allocates one VN at a time, with a coordinated node and link mapping, following insights provided by Chowdhury et al. [21]. The first step is the allocation of nodes (VMs) for each application in the group, according to the VM placement policy defined. After all VMs of a VN are allocated, the algorithm starts the second step of the mapping, which is the allocation of bandwidth for the group according to the example shown in Figure 3. The algorithm returns a success code for each VN that was embedded on the substrate and a failure code otherwise.

[3]Like related work [6], [9], [18], [20], we assume the physical infrastructure topology in cloud datacenters is defined as a multi-rooted tree [5].

Algorithm 2: Virtual network embedding.

Input : Physical infrastructure P, Set of groups $GroupList$
Output: Success/Failure array $allocated$

```
foreach Group g ∈ GroupList do
    // VM allocation
    foreach Application app ∈ g do
        Allocate VMs of app in the cloud infrastructure according to a
        predefined objective (e.g., minimum bandwidth, energy consumption,
        or fault tolerance);
    // Bandwidth allocation
    foreach Level lv from 0 to Height(P) do
        Allocate bandwidth at lv according to demands of VMs at lower
        levels (similarly to Figure 3);
    allocated[g] ← success/failure code for the allocation of group g;
return allocated;
```

VM placement objectives. VM placement is often implemented as a multi-dimensional packing with constraints being defined according to a placement goal. IoNCloud currently supports three different goals, as follows. First, *MinBand* minimizes bandwidth consumption by clustering VMs of the same application and of the same group on the smallest subtree in the physical infrastructure (similarly to Ballani et al. [6]). Second, *MinEnergy* follows insights from Mann et al. [22] and uses a first-fit algorithm to reduce the number of servers turned on, thus minimizing the total amount of power consumed by these servers. Third, *MaxFT* considers fault tolerance by spreading VMs on the cloud platform, so that applications can survive upon link, switch and/or rack failures (similarly to Bodík et al. [23]). The key idea is to increase the number of servers used to allocate VMs in accordance to a given spreading factor (*sf*). In particular, the minimum number of servers is determined considering the servers with available resources, and the number of VMs from the application each one of them can host. The new number of servers that will host these VMs is determined by choosing the minimum value

between (i) the multiplication of the minimum number of servers required to allocate such VMs and *sf* and (ii) the number of VMs of the application: $\texttt{ExpectedNumSrvs} = \min(\texttt{MinNumSrvs(App)} * sf, \texttt{NumVMs(App)})$.

Allocation quality. Algorithm 2 was designed as a constructive heuristic with the focus of providing efficient allocation of resources. It does not consider optimality, because it is computationally expensive to employ optimization strategies for large-scale cloud platforms [6], [20] and the allocation must be performed as quickly as possible (since there are high rates of tenant arrival and departure [4], known as churn). We defer a detailed study of the advantages and drawbacks of employing optimization models for IoNCloud to future work.

IV. Evaluation

In this section, we demonstrate the benefits of IoNCloud for both providers and tenants. Our evaluation focuses primarily on quantifying the advantage of grouping applications in virtual networks in terms of network predictability and resource utilization. Toward this end, we first describe the environment and, then, present the main results.

A. Environment

Datacenter topology. We follow previous work [6], [9], [20] and implement a discrete-event simulator that models a multi-tenant datacenter. We focus on tree-like topologies similar to multi-rooted trees used in current cloud platforms [11]. The physical substrate is defined as a three-level tree topology with 8,000 servers at level 0, each with 4 VM slots (i.e., with a total amount of 32,000 available VMs in the cloud). Every machine is linked to a ToR switch (40 machines form a rack), and every 20 ToRs are connected to an aggregation switch. Finally, all aggregation switches are connected to a core switch. Link capacities are defined as follows: machines are connected to ToR switches with access links of 1 Gbps; links from racks up to aggregation switches are 10 Gbps; and aggregation switches are attached to a core switch with links of 50 Gbps. Thus, the default oversubscription of the network is 4.

Workload. In line with related work [6], [9], [11], we generated the workload according to results obtained by measurement studies [4], [13], [24]. More specifically, the workload is composed of requests of applications to be allocated in the cloud platform. Requests are formed by a heterogeneous set of applications (including MapReduce and Web Services), which is representative of applications running on public cloud platforms [5]. Each application is represented as a tuple <N, $B(t)$>, with N being the number of VMs and $B(t)$ a time-varying function to specify the temporal network demand. The former is exponentially distributed around a mean of 49 VMs (representative of current clouds [4]). The latter was generated following results obtained by Benson et al. [13] and Kandula et al. [24] (we used measurements related to inter-arrival flow time and size at servers to simulate application traffic).

B. Results

We compare IoNCloud, which employs shared bandwidth guarantees, with the approach adopted by most related work [6], [17], [18], which creates one virtual network per application. Ideally, we would have compared IoNCloud with Proteus [9]. Proteus uses as input pulse functions obtained from the temporal network demands of applications. However, the generation of such pulse functions is addressed as a black-box in the paper and, thus, we cannot precisely develop a generator that mimics its behavior.

As previously mentioned (Section III-D), the algorithm used for virtual network allocation is agnostic in terms of VM placement. Hence, three VM placement algorithms are used in experiments: (i) *MinBand*, which minimizes the amount of bandwidth reserved for communication between VMs; (ii) *MinEnergy*, which minimizes energy consumption by reducing the number of used servers; and (iii) *MaxFT*, which maximizes fault-tolerance based on a given parameter (the desired ratio of extra servers used for spreading VMs).

For all experiments, we plot the percentile difference between both approaches given by the following equation: $(\frac{\texttt{IoNCloud}}{\texttt{One VN per App}} - 1) * 100\%$. Hence, negative percentiles mean IoNCloud has achieved a lower value than traditional approaches, while positive percentiles mean IoNCloud has achieved a higher value than traditional approaches. In general lower values are better, with the sole exception being Figure 7.

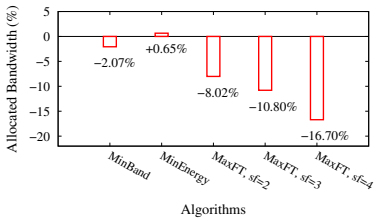


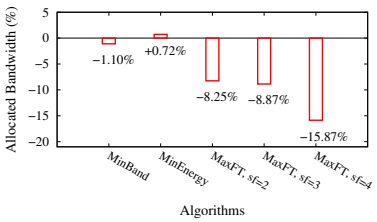
Fig. 4: Amount of reserved network resources.

Amount of reserved network resources. Figure 4 shows the total amount of reserved network resources according to the different placement algorithms. The y-axis represents the percentile difference between both approaches regarding the amount of bandwidth allocated, hence *the lower the value, the better*. We see that for any given approach, the amount of reserved resources increases in accordance with VM spreading. As expected, the shared bandwidth mechanism employed by IoNCloud outperforms the traditional methods when VMs are spread around the network, as it reduces the amount of reserved resources (up to 16.70%). This means that the provider can accept more applications in the cloud, improve resource utilization and, ultimately, increase datacenter throughput.

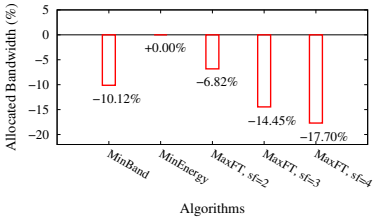
In contrast, IoNCloud is unable to achieve gains (in fact, with 0.65% of overhead in the worst-case) when there is no spreading, that is, when VMs are as packed as possible. This happens because the resource reservation employed by IoNCloud is performed per group, instead of per application (as traditional approaches). Therefore, the bandwidth allocated to each virtual link is only released after all applications in the respective group have finished. This design choice was deliberately chosen; such model can reduce the overhead of calculating the amount of bandwidth to be deallocated for each application that finishes its execution at each virtual link of the group. Moreover, we expect VM spreading to be norm in real cloud networks due to the high churn [4] of applications in these environments.

We further analyze bandwidth allocation by measuring the

amount of reserved resources in access and aggregation levels[4] of the topology for all VM placement algorithms. We see in Figure 5 that IoNCloud allocates less resources at both levels. In particular, note that IoNCloud has better results in the aggregation. This effect also increases the chance of allocating virtual links, since network oversubscription at this level is higher than at the edge, and decreases the probability of packet discards in the network (which usually happens at this level [13]).



(a) At the access.



(b) At the aggregation.

Fig. 5: Per-level analysis of reserved network resources.

Underutilization in the network. Figure 6 depicts the percentile difference of unused bandwidth for different placement algorithms. Underutilization is quantified by measuring the unused bandwidth on each virtual link. *Lower values are better*, since they mean that the cloud infrastructure is making better use of its reserved resources. As expected, IoNCloud achieves lower underutilization than current approaches. In fact, when compared to traditional schemes, IoNCloud is able to reduce waste, saving up to 18% of resources.

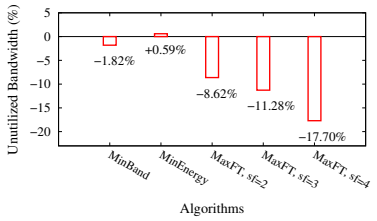


Fig. 6: Overall underutilization of resources.

IoNCloud can reduce resource underutilization, but it still suffers from some underutilization. As mentioned in the previous experiment, this happens because the current implementation of IoNCloud performs bandwidth deallocation at group granularity (as opposed to application granularity).

Ratio of Allocated VMs. This metric shows the proportion of VMs that were allocated in servers. *Higher values are better*, as the revenue of the cloud provider is proportional to the number of VMs it allocates. Figure 7 shows the difference between VM allocation ratios. As observed, IoNCloud performs better for all algorithms. Although the number of slots and VMs is the same, the allocation ratio differs depending on the allocation goal. This is because VMs can only be allocated if there is enough bandwidth for guaranteeing the setup of virtual links. Hence, reducing the amount of allocated bandwidth (as seen in Figure 4) increases the acceptance ratio of VMs in the cloud platform (since bandwidth is the bottleneck resource).

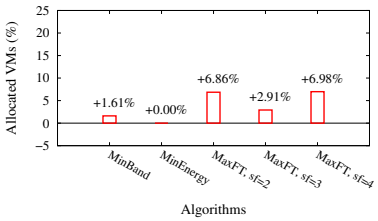


Fig. 7: Ratio of VMs that were placed in physical servers.

To understand the behavior of VM rejection, we perform experiments in a scenario were all datacenter links have unlimited bandwidth. Table I shows a comparison between both scenarios: normal and unlimited bandwidth. As can be observed, the assumption that bandwidth consumption interferes in VM allocation is verified, since all methods achieve 100% allocation with unlimited bandwidth. Note that *MinEnergy* is the only algorithm that achieves 100% VM allocation ratio under normal conditions. This is because VMs are packed together and fragmentation is minimal, thus, the majority of VMs will be closer. When minimizing bandwidth (*MinBand*), VMs may be allocated on free slots that are far from each other, which means that virtual links have a higher probability of reaching a bottlenecked physical link. *MaxFT* worsens this behavior, as it explicitly allocates VMs farther from each other.

TABLE I: VM allocation ratio with normal and unlimited bandwidth capacity on links.

VM placement goal	Bandwidth	
	Normal	Unlimited
MinBand	0.929	1
MinEnergy	1	1
MaxFT, sf=2	0.845	1
MaxFT, sf=3	0.892	1
MaxFT, sf=4	0.888	1

Link Sharing and Management Overhead. We also measure the number of reservations over each link in the datacenter network. Figure 8 shows the percentile difference of the maximum number of virtual links allocated in the network. We find that IoNCloud results in a significantly lower number of reservations to be managed (which can be as high as 22.32% less). In an environment as large and dynamic as a cloud platform, where network devices are limited in terms of the amount of control state and the rate at which these states can be updated, this typically results in a reduced reservation management overhead. Furthermore, during the experiments, we observed relatively small absolute values (an overall value of less than 10,000) for the number of reservations for all

[4]In our experiments, there were no reserved resources at the core.

strategies. This reflects the spatial locality applied by the allocation algorithms and suggests that the bandwidth reservation schemes can be accomplished using technologies already available in current datacenters (e.g., using rate-limiters in off-the-shelf switches or programmability of hypervisors [9]).

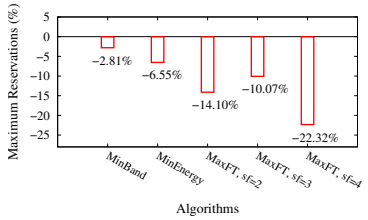


Fig. 8: Maximum number of allocated virtual links.

V. Design Discussion

Datacenter network topology. Current datacenters are typically implemented through a multi-rooted tree topology [11]. Therefore, in this paper, we focus on this kind of topology to show the benefits of IoNCloud. However, IoNCloud can be easily adapted for other topologies, such as random graphs [25]. In particular, it can be applied to multi-path topologies, both where load balancing is uniform across paths and where it is not uniform. For the first case (e.g., Fat-Tree), a single aggregate link can be used as a representation for a set of parallel links for bandwidth reservation [6], [8]. For the latter, IoNCloud would have to use an additional layer at hypervisor-level to control each path and its respective bandwidth for communication between VMs.

Online allocation of applications. IoNCloud allocates groups of applications in order to increase datacenter resource utilization. In this context, there is, at least, two ways of robustly providing online allocation for incoming application requests: i) by allocating an incoming application to an existing group; and ii) by allocating requests according to time slots. The first approach is straightforward, but may introduce some overhead to manage network resources when expanding an existing group. The second one (which we employed in our evaluation) takes advantage of high churn in cloud environments [4]. Thus, for each time slot (i.e., a predefined time period), IoNCloud can allocate the set of incoming requests by grouping them according to their bandwidth demands, without modifying previously allocated groups (less overhead).

Generality of the network model. Currently, IoNCloud adopts a single network model for all VMs of the same application. Nonetheless, it requires no modification when considering VMs of the same application with distinct network profiles. However, it may add some complexity to the resource allocation process. Another option is to extend the system to enforce per-VM traffic models by reserving bandwidth on links according to the VM with the highest demand in each application (at the cost of some underutilization).

VI. Conclusions and Future Work

We have introduced IoNCloud, a system that provides network predictability while minimizing resource underutilization and management overhead. To achieve this, IoNCloud groups applications in VNs according to their temporal bandwidth usage. Evaluation results show the benefits of our strategy, which is able to use available bandwidth more efficiently, reducing allocated bandwidth, network underutilization and management overhead. In future work, we intend to extend IoNCloud in two ways: i) by considering other objectives for application grouping; and ii) by adding VM migration to minimize network traffic.

Acknowledgements

This work has been supported by MCTI/CNPq/Universal (Project Phoenix, 460322/2014-1), MCTI/CNPq/CT-ENERG (Project ProSeG, 33/2013) and Microsoft Azure for Research grant award.

References

[1] F. R. Dogar et al., "Decentralized Task-Aware Scheduling for Data Center Networks," in *ACM SIGCOMM*, 2014.

[2] J. Lee et al., "Application-driven bandwidth guarantees in datacenters," in *ACM SIGCOMM*, 2014.

[3] J. Guo et al., "On Efficient Bandwidth Allocation for Traffic Variability in Datacenters," in *IEEE INFOCOM*, 2014.

[4] A. Shieh et al., "Sharing the data center network," in *USENIX NSDI*, 2011.

[5] D. Abts and B. Felderman, "A guided tour of data-center networking," *Comm. ACM*, vol. 55, no. 6, Jun. 2012.

[6] H. Ballani et al., "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.

[7] R. Shea et al., "A Deep Investigation Into Network Performance in Virtual Machine Based Cloud Environment," in *IEEE INFOCOM*, 2014.

[8] L. Popa et al., "ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing," in *ACM SIGCOMM*, 2013.

[9] D. Xie et al., "The only constant is change: Incorporating time-varying network reservations in data centers," in *ACM SIGCOMM*, 2012.

[10] M. Chowdhury et al., "Efficient Coflow Scheduling with Varys," in *ACM SIGCOMM*, 2014.

[11] H. Ballani et al., "Chatty tenants and the cloud network sharing problem," in *USENIX NSDI*, 2013.

[12] V. Jeyakumar et al., "EyeQ: practical network performance isolation at the edge," in *USENIX NSDI*, 2013.

[13] T. Benson et al., "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010.

[14] X. Wang et al., "Carpo: Correlation-aware power optimization in data center networks," in *IEEE INFOCOM*, 2012.

[15] L. Chen and H. Shen, "Consolidating Complementary VMs with Spatial/Temporal-awareness in Cloud Datacenters," in *IEEE INFOCOM*, 2014.

[16] K. LaCurts et al., "Choreo: Network-aware task placement for cloud applications," in *ACM IMC*, 2013.

[17] C. Guo et al., "SecondNet: a data center network virtualization architecture with bandwidth guarantees," in *ACM CoNEXT*, 2010.

[18] H. Rodrigues et al., "Gatekeeper: supporting bandwidth guarantees for multi-tenant datacenter networks," in *USENIX WIOV*, 2011.

[19] L. Popa et al., "FairCloud: sharing the network in cloud computing," in *ACM SIGCOMM*, 2012.

[20] D. S. Marcon et al., "Trust-based grouping for cloud datacenters: improving security in shared infrastructures," in *IFIP Networking*, 2013.

[21] N. Chowdhury et al., "Virtual Network Embedding with Coordinated Node and Link Mapping," in *IEEE INFOCOM*, 2009.

[22] V. Mann et al., "VMFlow: leveraging VM mobility to reduce network power costs in data centers," in *IFIP Networking*, 2011.

[23] P. Bodík et al., "Surviving failures in bandwidth-constrained datacenters," in *ACM SIGCOMM*, 2012.

[24] S. Kandula et al., "The nature of data center traffic: measurements & analysis," in *ACM IMC*, 2009.

[25] A. Singla et al., "High Throughput Data Center Topology Design," in *USENIX NSDI*, 2014.

APPENDIX D PUBLISHED PAPER AT IFIP/IEEE IM 2015

In this paper, we make two key observations: providers do not need to control each flow individually (e.g., VM-to-VM), since they charge tenants based on the amount of resources consumed by applications; and congestion control in the intra-cloud network is expected to be proportional to the tenant's payment. Based on these insights, we introduce Predictor, a novel system for DCNs that enables fine-grained network management for providers, minimizes flow table size by controlling flows at application-level and reduces flow setup time by proactively installing rules in switches. It also enables tenants to request and receive predictable network performance for both intra- and inter-application communication, with work- conserving bandwidth sharing. Evaluation results show that Predictor provides significant improvements against DevoFlow (reducing flow table size up to 87%) and offers predictable and guaranteed network performance for tenants.

- **Title**: Predictor: providing fine-grained management and predictability in multi-tenant datacenter networks
- **Conference**: IFIP/IEEE Integrated Network Management Symposium (IM 2015)
- **Type**: Main track (full-paper)
- **Qualis**: B1
- **URL**: <http://ieeexplore.ieee.org/document/7140278/>
- **Date**: May 11-15, 2015
- **Location**: Ottawa, CA
- **Digital Object Identifier (DOI)**: 10.1109/INM.2015.7140278

Predictor: providing fine-grained management and predictability in multi-tenant datacenter networks

Daniel S. Marcon, Marinho P. Barcellos
Institute of Informatics – Federal University of Rio Grande do Sul, RS, Brazil
Email: {daniel.stefani, marinho}@inf.ufrgs.br

***Abstract*—Software-Defined Networking (SDN) can simplify traffic management in large-scale datacenter networks (DCNs). On one hand, it provides a robust method to address the challenge of performance interference (bandwidth sharing unfairness) in DCNs. On the other, its pragmatic implementation based on OpenFlow introduces scalability challenges, as it (a) adds latency for new flows (the controller must process hundreds of thousands of requests per second and install appropriate rules in switches); and (b) requires large flow tables in devices (DCNs can have more than 16 million distinct flows per second with different requirements and duration). To employ OpenFlow-based SDN in DCNs, recent work has proposed techniques that require hardware customization to keep up with the high dynamic traffic patterns of these networks. We make two key observations: providers do not need to control each flow individually (e.g., VM-to-VM), since they charge tenants based on the amount of resources consumed by applications; and congestion control in the intra-cloud network is expected to be proportional to the tenant's payment. Based on these insights, we introduce Predictor, a novel system for DCNs that enables fine-grained network management for providers, minimizes flow table size by controlling flows at application-layer and reduces flow setup time by proactively installing rules in switches. It also enables tenants to request and receive predictable network performance for both intra- and inter-application communication, with work-conserving bandwidth sharing. Evaluation results show that Predictor provides significant improvements against DevoFlow (reducing flow table size up to 87%) and offers predictable and guaranteed network performance for tenants.**

I. Introduction

The paradigm of Software-Defined Networking (SDN) has emerged as an efficient method that offers programability to dynamically manage and configure network resources. On one hand, SDN provides a robust method to address the challenge of performance interference [1], [2] in multi-tenant cloud datacenter networks (DCNs). While providers offer guaranteed computing resources, the network is shared in a best-effort manner among all tenants [3]. The lack of network guarantees (for both intra- and inter-application communication) results in unpredictable and poor overall application performance [4]. Several recent papers [1], [5]–[7] have proposed techniques to address this issue. Nonetheless, they do not provide fine-grained control over network resources for providers, as they require complex interactions among hundreds of thousands of hypervisors.

On the other hand, SDN, and pragmatic OpenFlow-based deployments, faces scalability challenges in DCNs (and, in general, in high-performance, dynamic networks) for the following reasons [8], [9]. First, the transition between the data and control planes whenever a new flow arrives at a switch[1] may add some delay (latency for communication between switches and the controller), and the high frequency at which flows arrive and demands change in DCNs hinders controller scalability. Second, the number of entries needed in flow tables of forwarding devices can be significantly higher than the amount of resources available in commodity switches, especially for large-scale DCNs [8], [10] (as such networks can have more than 16 million flows per second [11]).

Scaling the controller has been the main topic of some studies. They address this issue either by devolving control to the data plane [8], [12] or by developing a logically distributed controller [13]. The former approach adds several functionalities to switches and, thus, requires more complex, customized hardware, while the latter does not scale for large DCNs where communications occur between virtual machines (VMs) connected by different top-of-rack (ToR) switches.

In this paper, we make two *key observations*: (i) providers do not need to control each flow individually, since they charge tenants based on the amount of resources consumed by applications; and (ii) congestion control in the intra-cloud network is expected to be proportional to the tenant's payment [1], [14]. Therefore, we adopt a broader definition of flows, considering it at application-layer[2], and introduce Predictor, a novel system for Infrastructure-as-a-Service (IaaS) cloud datacenters. Predictor addresses the two aforementioned challenges (performance interference and scalability of OpenFlow-based deployments in DCNs), offering the following benefits.

First, it enables fine-grained network management for cloud providers by employing the SDN paradigm and allowing control of flows at application-layer. Predictor makes use of wildcards to reduce flow table size at switches and to allow providers to control traffic and gather statistics at application-layer for each link and device in the network.

Second, it enables tenants to request and receive a minimum guaranteed bandwidth for both intra- and inter-application communication, while allowing the use of more bandwidth when there is spare capacity (work-conservation). The controller proactively installs rules in switches at application allocation time to guarantee bandwidth for communication between VMs of the same application, and reactively sends rules for inter-application communication. On one hand, proactively installing rules at switches reduces flow setup time and management traffic in the network (since switches do not

[1]We use the terms "switches" and "forwarding devices" to refer to the same set of SDN-enabled network devices, that is, data plane devices that forward packets based on a set of flow rules.

[2]An application is represented by a set of VMs that consume computing and network resources (see Section III-A for further details).

978-3-901882-76-0 @2015 IFIP

request the controller assistance for each new flow). On the other hand, some flow table entries may take longer to expire (they are removed only when their respective applications conclude and are deallocated). Inter-application forwarding rules, in turn, are reactively installed in switches, because applications may not know all other applications they will communicate with in advance (when they are allocated) and intra-application traffic volume is expected to be higher [1].

Overall, the major contributions of this paper are:

- We propose a topology-agnostic strategy that takes advantage of SDN to provide fine-grained network management for IaaS providers (e.g., by controlling flows at link-level) and predictable performance with guaranteed bandwidth for tenants, without requiring customized hardware in switches.
- We present the design of Predictor, a novel system that implements the proposed strategy for large-scale DCNs.
- We show the benefits of Predictor, comparing it against the state-of-the-art controller for DCNs (DevoFlow [8]). Evaluation results show that Predictor can significantly reduce flow table size at forwarding devices (up to 87%) with small increase of controller load. Furthermore, Predictor offers predictable network performance with guaranteed bandwidth and work-conserving sharing.

The paper is organized as follows. Section II provides the background on the cloud network sharing problem and the pros and cons of using SDN to solve it. Section III describes our strategy (and its implementation, Predictor), while Section IV presents its evaluation. Section V discusses the generality and limitations of our strategy, and Section VI examines related work. Finally, Section VII concludes the paper with final remarks and perspectives for future work.

II. Background

This section first examines the intra-cloud network sharing problem and, then, discusses the benefits and drawbacks of employing SDN on DCNs.

A. Cloud Datacenter Network Sharing

Current cloud providers (such as Amazon EC2) typically offer VMs with guaranteed computing resources. However, the underlying network is shared in a best-effort manner [7]. Measurement studies [4], [5] concluded that the network throughput achieved by VMs can vary by a factor of five or more. Such variability results in poor and unpredictable application performance, and tenants end up spending more money (since their applications take longer to finish) [5].

Popa et al. [14] elaborates on two main requirements for network sharing: i) bandwidth guarantees for both intra- and inter-application communication; and ii) work-conserving sharing to achieve high network utilization for providers. In particular, these two requirements present a trade-off: strict bandwidth guarantees may reduce utilization, since applications have variable network demands over time [6]; and a work-conserving approach means that, if there is residual bandwidth and some applications have demands, they should utilize it (even if the available bandwidth belongs to the guarantees of another application) [15]. Therefore, we employ SDN to enable fine-grained network management in order to develop a robust strategy to explore this trade-off.

B. Software-Defined Networking

SDN [16] seeks to decouple the control and data planes of the network. The pros and cons of using SDN to solve the network sharing problem in DCNs are discussed below.

Benefits. SDN (and OpenFlow) offers programability to dynamically configure and manage the entire network, enabling providers to offer a base level of network performance guarantees for tenants. In particular, it allows administrators to apply a wide-range of policies with little effort (without requiring device by device configuration), including bandwidth guarantees, routing and fault tolerance [8]. SDN also provides near-optimal traffic management, since the controller can request and receive information about network load on a low-level granularity (e.g., by device, link or specific flows traversing a link).

Drawbacks. This paradigm involves the control plane more frequently than traditional networking. The two main issues related to DCNs are flow setup time (the time taken to install a new flow rule in forwarding devices) and flow table size in switches.

Flow setup time is an important factor to be considered, as a new flow in the network is delayed at least two RTTs in forwarding devices (i.e., communication between the ASIC and the management CPU and between that CPU and the controller) [8], so the controller can install the appropriate rules at switches. Furthermore, when inserting high-priority rules at the Ternary Content-Addressable Memory (TCAM), switches must move down the table all other entries with lower priorities (which takes more time as flow table size increases). These delays may be impractical for latency-sensitive flows (adding even 1 ms of latency to these flows is intolerable [17]).

Flow tables, in turn, are a restricted resource in commodity switches, as TCAMs are typically expensive [8], [10]. Such devices usually have a limited number of entries available for OpenFlow rules, which may not be enough when considering that large-scale datacenter networks can have an elevated number of flows per second [11] (and approximately more than 1,500 new flows per second per rack [18]).

III. Predictor

Predictor implements our novel and low-overhead strategy to provide fine-grained management and predictable sharing inside the cloud DCN. It was designed taking four requirements into consideration: scalability, resiliency, predictable and guaranteed network performance and high network utilization. First, scalability is essential for the system to be realistic, as the network sharing strategy must scale to hundreds of thousands of VMs and to the heterogeneous workloads of cloud applications. Second, resiliency to churn at flow- and application-layer, as DCNs have high rate of new flows per second [11] and datacenters can have high rate of application allocation and deallocation [15], respectively. Third, predictable and guaranteed network performance in order to allow applications to maintain a certain level of performance even when the network is congested. Finally, high network utilization, so that all spare bandwidth can be used independently of the bandwidth guarantees assigned to VMs.

An overview of Predictor is shown in Figure 1. The system is composed of five components: Predictor controller, allocation module, application information base (AIB), network information base (NIB) and OpenFlow controller. They are discussed next.

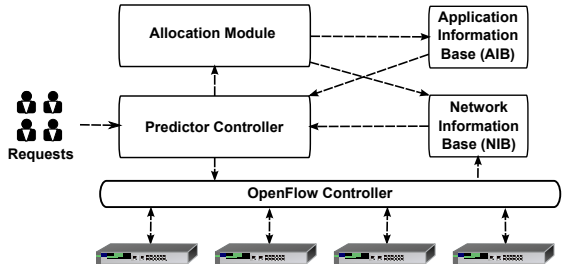


Fig. 1: Predictor system overview.

Predictor Controller. It receives both applications to be allocated (whose resources to be used are determined by the allocation module) and requests for inter-application communication (detailed in Section III-B). In case of an incoming application, it sends the request to the allocation module. Once the allocation is completed (or if the request is for inter-application communication), the Predictor controller generates and sends appropriate flow rules to the OpenFlow module. The OpenFlow module, then, updates the tables (of forwarding devices) that need to be modified.

The controller installs rules to identify flows at application-layer (more details in Section III-E). It can also take advantage of flow management at lower levels (for instance, by matching source and destination MAC and IP fields), since it uses the OpenFlow protocol. Nonetheless, given the amount of resources available in commodity switches and the number of flows that come and go in a small period of time, such low-level rules are expected to be kept to a minimum.

Allocation Module. This component is responsible for allocating incoming applications at the cloud platform, according to available resources. It receives requests from the Predictor controller, determines the set of resources to be allocated for each new request and updates the AIB and NIB. We detail the resource allocation logic in Section III-C.

Application Information Base (AIB). It keeps detailed information regarding each allocated application, including its identifier (ID), VM-to-server mapping, IP addresses, bandwidth guarantees, network weight (for work-conserving sharing), links being used and other applications it communicates with. It provides information for the Predictor controller to compute flow rules that need to be installed in switches.

Network Information Base (NIB). It is composed of a database of resources, including hosts, switches, links and their capabilities (such as rate-limiters, and link capacity and latency). In general, it keeps information about computing and network state, which are received from the OpenFlow controller (current state) and the allocation module (resources used for newly allocated applications). The Predictor controller uses information stored in the NIB to map logical actions (e.g., intra- or inter-application communication) into the physical network. While AIB maintains information at application granularity, NIB keeps information at network layer. The design of NIB was inspired by Onix [16] and PANE [19].

OpenFlow Controller. It is responsible for communication to/from forwarding devices and Open vSwitches at hypervisors, in order to update network state and get information from the network. It receives information from the Predictor controller to modify flow tables in forwarding devices and updates the NIB upon getting information from the network.

We first describe in detail application requests (Section III-A) and how intra- and inter-application communication is handled (Section III-B), and use such information for the allocation of applications in the datacenter (Section III-C). Then, we present how the system provides work-conserving network sharing (Section III-D) and how key functionalities are implemented (Section III-E).

A. Application Requests

Tenants request applications using the hose-model (similarly to prior work [1], [5]–[7]), in order to capture the semantics of the guarantees being offered. In this model, all VMs of an application are connected to a non-blocking virtual switch through dedicated bidirectional links. Each application a is represented by its resource demand and network weight, formally defined as $< N_a, B_a, w_a >$, with the terms being defined as follows: $N_a \in \mathbb{N}^*$ specifies the number of VMs; $B_a \in \mathbb{R}^+$ represents the bandwidth guarantees required by each VM; and $w_a \in [0, 1]$ indicates the network weight. In particular, the network weight enables residual bandwidth (unallocated, or reserved bandwidth for an application and not currently being used) to be proportionally shared among applications with more demands than their guarantees. Therefore, the total amount of bandwidth available for a VM of application a at a given period of time, following the hose model, is denoted by $B_a + spare(s, v_a)$, where $spare(s, v_a)$ identifies the share of spare bandwidth assigned to VM v of application a located at server s:

$$spare(s, v_a) = \frac{w_a}{\sum_{v \uparrow V_s | v \in V_s} w_v} * SpareCapacity \quad (1)$$

where V_s denotes the set of all co-resident VMs (i.e., VMs placed at server s), $v \uparrow V_s \mid v \in V_s$ represents the subset of VMs at server s that need to use more bandwidth than their guarantees and $SpareCapacity$ indicates the residual capacity of the link that connects the physical server to the ToR switch.

Two assumptions are made for the sake of explanation (these are not limitations), as follows. First, we abstract away non-network resources and consider all VMs with the same amount of CPU, memory and storage. Second, we consider that all VMs of a given application receive the same bandwidth guarantees.

B. Bandwidth Guarantees

Predictor provides bandwidth guarantees for both intra- and inter-application communication. We discuss each one next.

Intra-application network guarantees. Typically, this type of communication represents most of the traffic in DCNs [1]. Thus, Predictor allocates and ensures bandwidth guarantees at application allocation time by proactively installing flow rules and rate-limiters in the network through

OpenFlow[3]. Each VM of a given application a is assigned a bidirectional rate of B_a (as detailed in Section III-A). Limiting the communication between VMs located in the same rack is straightforward, since it can be done locally by the Open vSwitch at each hypervisor.

In contrast, for inter-rack communication, bandwidth must be guaranteed throughout the network, along the path used for such communication. Predictor provides guarantees for this traffic by employing the concept of *VM clusters* (set of VMs of the same application located in the same rack). To illustrate this concept, consider a simplified scenario where a given application a has two clusters: $c_{a,1}$ and $c_{a,2}$. Since each VM of a cannot send or receive data at a rate higher than B_a, traffic between the pair of clusters $c_{a,1}$ and $c_{a,2}$ is limited by the smallest cluster: $rate_{c_{a,1},c_{a,2}} = \min(|c_{a,1}|, |c_{a,2}|) * B_a$, where $rate_{c_{a,1},c_{a,2}}$ represents the calculated bandwidth for communication between clusters $c_{a,1}$ and $c_{a,2}$, and $|c_{a,i}|$ denotes the number of VMs in cluster i of application a. In this case, $rate_{c_{a,1},c_{a,2}}$ is guaranteed along the path used for communication between these two clusters by rate-limiters configured in forwarding devices through OpenFlow.

Inter-application communication. Providing static hose guarantees for this type of communication does not scale for DCNs [1], as bandwidth guarantees for each VM would need to be enforced for communication with all other VMs in the network. Furthermore, tenants cannot be expected to know before allocation all other applications and services that their applications will communicate with. Instead, Predictor dynamically sets up guarantees for inter-application communication according to the needs of applications and residual bandwidth in the network. The Predictor controller provides two ways of establishing guarantees for communication between VMs of distinct applications and services, as follows.

Reacting to new flows in the network. When a VM needs to exchange data with one or more VMs of another application, it can simply send packets to those VMs. The hypervisor (through its Open vSwitch) of the server hosting the source VM receives such packets and, since they do not match any rule, sends them to the OpenFlow controller. The Predictor controller will, then, be called to determine the rules needed by the new flows, and the set of rules will be installed along the paths in the network.

Receiving communication requests from applications. Prior to initiating the communication with VMs belonging to other applications, the source VM can send a request to the Predictor controller for communication with VMs from other application(s). This request is composed of the set of destination VMs, the bandwidth needed and the expected amount of time the communication will last. Upon receiving the request, the Predictor controller verifies residual resources in the network, sends a reply, and, in case there are enough available resources, generates and installs the appropriate set of rules for this communication. This approach is similar to providing an API for applications to request network resources, which is employed by PANE [19].

[3]While Predictor may overprovision bandwidth at the moment applications are allocated, it does not waste bandwidth because of its work-conserving strategy (explained in Section III-D). Without overprovisioning bandwidth at first, it would not be feasible to provide bandwidth guarantees for applications (as DCNs are typically oversubscribed).

C. Resource Allocation

The allocation process is responsible for performing admission control and mapping application requests in the datacenter infrastructure. A valid allocation must satisfy two requirements: computing and network resource availability [20]. For simplicity, we discuss Predictor and its allocation component in the context of traditional tree-based topologies implemented in current datacenters [7].

We design a location-aware heuristic to efficiently allocate tenant applications in cloud platforms. The key principle is minimizing bandwidth for intra-application communication (thus allocating VMs of the same application as close as possible to each other), since this type of communication generates most of the traffic in DCNs (as discussed before).

Algorithm 1 allocates one application at a time, as requests are received. First, it searches for the best placement in the infrastructure for the incoming application via dynamic programming. To this end, three data structures are defined and dynamically initialized for each request: i) set R_a stores subgraphs with enough computing resources for application a; ii) V_s^a stores the total number of VMs of application a the s-rooted subgraph can hold; and iii) C_s^a stores the number of VM clusters that can be formed in subgraph s. The algorithm traverses the topology starting at rack level, and up to the core, and determines subgraphs with enough available resources to allocate the incoming request (lines 2 - 12).

Algorithm 1: Location-aware algorithm.

Input : Physical infrastructure P, Application a
Output: Success/Failure code $allocated$

```
1  R_a ← ∅;
2  foreach level l of P do
3      if l == 1 then // Top-of-Rack switches
4          foreach ToR r do
5              V_r^a ← number of available VMs in the rack;
6              C_r^a ← 1;
7              if V_r^a ≥ N_a then R_a ← R_a ∪ {r};
8      else // Aggregation and core switches
9          foreach Switch s at level l do
10             V_s^a ← Σ_{w ∈ {set of directly connected switches at level l−1}} V_w^a;
11             C_s^a ← Σ_{w ∈ {set of directly connected switches at level l−1}} C_w^a;
12             if V_s^a ≥ N_a then R_a ← R_a ∪ {s};
13 allocated ← failure code;
14 while Application a not allocated and R_a not empty do
15     r_a ← Select subgraph from R_a;
16     allocated ← Allocation of VMs and bandwidth for application a at r_a;
17 return allocated;
```

After verifying the physical infrastructure and determining possible placements, the algorithm selects one subgraph r_a at a time to allocate the application (lines 13 - 16). The selection of a candidate subgraph takes into account the number of VM clusters. Therefore, the selected subgraph is the one with the minimum number of VM clusters, so that VMs of the same application are allocated close to each other, reducing the amount of bandwidth needed for intra-application communication (recall that the network often represents the bottleneck when compared to computing resources). When a subgraph is selected, the algorithm allocates the application, reserving bandwidth for communication between its VMs as presented in Section III-B.

Finally, the algorithm returns a success code if application a was allocated or a failure code otherwise (line 17). Applications that could not be allocated upon arrival are discarded, similarly to Amazon EC2 admission control.

D. Work-Conserving Rate Enforcement

Predictor provides bandwidth guarantees with work-conserving sharing. This is because only enforcing guarantees through static provisioning results in underutilization and fragmentation [7], while only work-conserving sharing does not provide strict guarantees for tenants [1]. Therefore, in addition to ensuring a base level of guaranteed rate, Predictor can proportionally share available bandwidth among applications with more demands than their guarantees, as defined in Equation 1.

We design an algorithm to periodically set the allowed rate for each co-resident VM on a server. In order to provide smooth interaction with TCP, we follow ElasticSwitch [7] and execute the work-conserving algorithm between periods of time one order of magnitude larger than the network round-trip time (RTT), i.e., 10 ms instead of 1 ms.

Algorithm 2 aims at enabling smooth response to bursty traffic (since traffic in DCNs may be highly variable over short periods of time [21]). It receives as input the list of VMs (V_s) hosted on the server (s) and their current demands (which are determined by monitoring VM socket buffers, similarly to Mahout [22]). First, the rate of each VM with demands less or equal than its bandwidth guarantees (represented by $v \downarrow V_s \mid v \in V_s$) is calculated and enforced (lines 1 - 2). Then, the algorithm calculates the residual bandwidth of the link connecting the server to the ToR switch (line 3). The residual bandwidth is calculated by subtracting from the link capacity the guarantees of VMs with higher demands than their guarantees (represented by $v \uparrow V_s \mid v \in V_s$) and the rate of VMs with less or equal demands than their guarantees.

Algorithm 2: Work-conserving rate allocation.

Input : VM set V_s, Current demand of VMs $demand$
Output: Rate $nRate$ for each co-resident VM

1 **foreach** $v \downarrow V_s \mid v \in V_s$ **do**
2 $\quad nRate[v] \leftarrow \min(B_v, demand[v])$;
3 $residual \leftarrow LinkCapacity - \left(\sum_{v \uparrow V_s} B_v + \sum_{v \downarrow V_s} nRate[v]\right)$;
4 **foreach** $v \uparrow V_s \mid v \in V_s$ **do**
5 $\quad nRate[v] \leftarrow B_v + \min\left(demand[v] - B_v, \left(\frac{w_v}{\sum_{v \uparrow V_s} w_v} \times residual\right)\right)$;
6 **return** $nRate$;

The last step establishes the bandwidth for VMs with higher demands than their guarantees (line 4 - 5). The rate is determined by adding the guarantees (B_v) and the minimum bandwidth between i) the difference of the current demand ($demand[v]$) and the guarantees (B_v); and ii) the proportional share of residual bandwidth the VM would be able to receive according to its weight w_v. By calculating the minimum bandwidth between these two values, Predictor guarantees that VMs will not receive more bandwidth than they need (which would waste network resources). With this approach, the algorithm can adapt to the significant variable communication demands of applications.

In summary, if the demand of a VM exceeds its guaranteed rate, data can be sent and received at least at that guaranteed rate. Otherwise, if it doesn't, the unutilized bandwidth will be shared among co-resident VMs whose traffic demands exceed their guarantees.

E. Key Implementation Aspects

Predictor relies on two key aspects: i) identifying flows at application-layer; and ii) providing network guarantees and dynamically enforcing rates for VMs. Figure 2 shows how our proof-of-concept implementation of Predictor handles these aspects, which are discussed next.

First, to perform application-layer flow identification, Predictor utilizes Multiprotocol Label Switching (MPLS). More specifically, applications are identified in OpenFlow rules through the label field in the MPLS header. The MPLS label is composed of 20 bits, which allows Predictor to identify 1,048,576 different applications. The complete operation of identifying and routing packets at application-layer works as follows. For each packet received from the source VM, the Open vSwitch (controlled via the OpenFlow protocol) in the hypervisor pushes a MPLS header (four bytes) with the application ID of the source VM in the label field. Subsequent switches in the network use both MPLS label and IP destination address (which may be wildcarded, depending on the possibilities of routing) matching fields to choose the correct output port to forward incoming packets. When packets arrive at the destination hypervisor, the Open vSwitch pops the MPLS header and forwards the packet to the correct VM.

Second, Predictor runs a local controller at hypervisor-level of each server in order to rate-limit VMs. More precisely, the local controller installs the appropriate rules at the Open vSwitch to dynamically set the allowed rate for each hosted VM. Such rate is calculated by Algorithm 2, discussed in Section III-D. Note that Predictor also reduces rate-limiting overhead when compared to previous schemes (e.g., Hadrian [1], CloudMirror [2] and ElasticSwitch [7]), as it only rate-limits the source VM while other schemes rate-limit each pair of source-destination VMs.

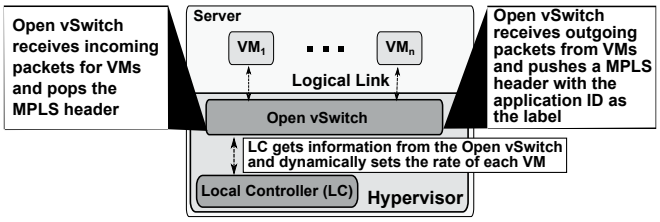


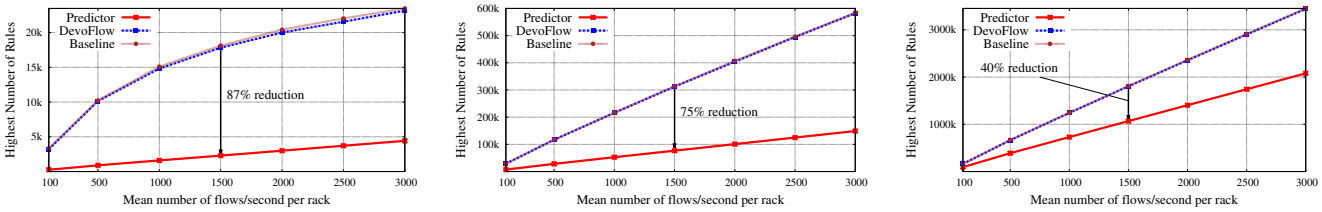
Fig. 2: Server-level implementation of Predictor features.

IV. Evaluation

We focus on showing that Predictor (i) can scale to large DCNs; (ii) provides both predictable network performance (with bandwidth guarantees) and work-conserving sharing; and (iii) outperforms existing schemes for DCNs (the baseline SDN/OpenFlow controller and DevoFlow [8]).

A. Setup

Environment. We have implemented a simulator that models an IaaS multi-tenant datacenter. The network is defined as a tree-like topology, similar to current DCNs. It is composed of a three-tier topology with 16,000 servers at level 0; each server has 4 VM slots, resulting in a total amount of 64,000 available VMs. Every 40 machines form a rack, and every 20 ToRs are connected to an aggregation switch. Finally, all



(a) At hypervisors. (b) At ToRs. (c) At aggregation switches.

Fig. 3: Highest number of rules at forwarding devices.

aggregation switches are connected to a core switch. The capacity of each link is defined as follows: 1 Gbps for server-ToR links, 10 Gbps for ToR-aggregation links and 50 Gbps for aggregation-core links. Therefore, the default oversubscription of the network is 4.

Workload. The workload is composed of incoming application requests (to be allocated in the cloud platform) arriving over time. In particular, we consider an heterogeneous set of applications, including MapReduce and Web Services. As defined in Section III-A, each application a is represented as a tuple $< N_a, B_a, w_a >$. Given the lack of proper traces for DCNs, the workload was generated in line with related work [1], [6]. N_a is exponentially distributed around a mean of 49 VMs (following measurements from prior work [15]). B_a was generated by reverse engineering the traces used by Benson et al. [11] and Kandula et al. [18]. More specifically, we used their measurements related to inter-arrival flow-time and flow-size at servers to generate and simulate network demands of applications. Of all traffic, 20% of flows are destined to other applications [1]. We pick the destination of each flow by first determining whether it is an intra- or inter-application flow, and then uniformly select a destination. The weight w_a is uniformly distributed in the interval $[0, 1]$.

B. Results

We begin by examining the scalability of employing Predictor on large DCNs. To this end, we verify controller load as well as the number of rules in flow tables, as these are typically the factors that restrict scalability the most [7], [9]. We compare Predictor against the baseline SDN/OpenFlow controller and the state-of-the-art controller for DCNs (DevoFlow [8]). In the baseline scheme, switches forward to the controller packets that do not match any rule in the flow table (we consider the default behavior of OpenFlow versions 1.3 and 1.4 upon a table-miss event). The controller, then, responds with the appropriate set of rules specifically designed to handle the new flow. DevoFlow, in turn, considers flows at the same granularity than the baseline. However, forwarding devices, through the use of more powerful hardware and template rules, generate rules for small flows (without involving the controller) and forward only packets of large flows to the controller. Therefore, switches have similar number of rules when compared to the baseline, but controller load is significantly reduced.

Reduced number of flow table entries. Figure 3 shows how the offered load (in new flows per second per rack) affects the occupancy of flow tables. More precisely, the plots show in Figures 3(a), 3(b) and 3(c), respectively, the largest number of entries required at any hypervisor, ToR and aggregation switch for a given mean rate of new flows at each rack (in core devices, results are not shown, as Predictor provides negligible gains).

In all three plots, we see that the mean number of arriving flows during an experiment affects directly the number of rules needed in devices. Results in Figure 3(a) present a logarithmic behavior according to the mean number of new flows, while in Figures 3(b) and 3(c), a linear one. These results are explained as follows: (i) the number of different flows that pass through ToR and aggregation switches is large and may quickly increase due to the elevated number of end-hosts (VMs) and arriving flows in the network; and (ii) in hypervisors, the number of rules (and distinct flows) is smaller (with a reduced growth when increasing the number of new flows in the network) because of the limited number of VMs hosted at each physical server.

Overall, the increase of the total number of flows requires more rules for the correct operation of the network (according to the needs of tenants) and enables richer communication patterns (representative of cloud datacenters [1]). Additionally, the number of rules for the baseline and DevoFlow is similar because (i) they consider flows at the same granularity; and (ii) the same default timeout for rules was adopted.

Because Predictor manages flows at application-layer and also wildcards the destination address in rules when possible (as explained in Section III-E, when the routing process was described), it significantly reduces the number of rules needed in forwarding devices, especially for realistic numbers of flows in large-scale DCNs (i.e., higher than 1500 new flows per second per rack [7]). In fact, Predictor reduces the number of rules up to 87% at hypervisors, 75% at ToRs and 40% at aggregation devices. In core devices, the reduction is negligible (around 1%), because (a) a high number of flows does not need to traverse core links to reach their destinations, thus baseline and DevoFlow (which handle flows at VM-to-VM granularity) do not install many rules at core devices, while Predictor installs application-layer rules at these devices; and (b) Predictor proactively installs rules for intra-application traffic (while other schemes install rules reactively).

In summary, Predictor reduces the number of rules required in forwarding devices, which can (i) minimize the amount of resources occupied by rules in TCAMs of forwarding devices (TCAM is a very expensive resource [10]); (ii) improve hypervisor performance (as measured by LaCurts et al. [23]); and (iii) minimize the time needed to install high-priority rules in TCAMs, since all lower-priority rules in the table must be moved down to perform this operation.

Small controller load. As DCNs typically have a high

number of flows per second, it is important that the controller is able to efficiently handle flow setups. Figure 4 shows the flow request rate the controller needs to be able to process in each case. As expected, the number of messages sent to the controller increases according to the mean number of new flows per rack, because the controller must set up network paths and allocate resources for these new flows (i.e., flows without matching rules in forwarding devices). The baseline imposes a higher load to its controller than DevoFlow (recall that DevoFlow only requires controller intervention to install rules for large flows, at the cost of more powerful hardware at forwarding devices). Predictor, in turn, proactively installs application-layer rules for intra-application communication at allocation time and reactively sends rules for inter-application traffic upon receiving communication requests, significantly reducing the number of flow requests when compared to the baseline (around 77%).

In contrast, Predictor requires more flow request messages than DevoFlow (around 22% more, considering the total number of flows in the network), most of which are from inter-application traffic. Nonetheless, Predictor controller load can be reduced by allowing tenants to specify at allocation time some (or all) other applications and services that their applications will communicate with (at the cost of some burden, as tenants will need more knowledge of their applications).

In general, the Predictor controller is aware of all traffic in the network, while the DevoFlow controller has knowledge of only large flows (approximately 50% of the total traffic [11]), to perform fine-grained management in the network.

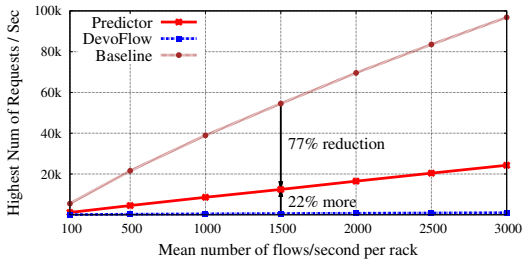
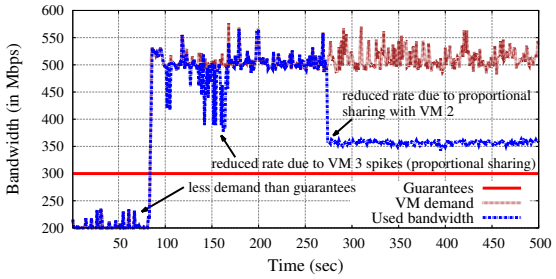


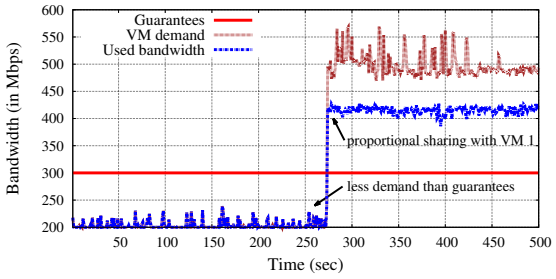
Fig. 4: Controller load.

After verifying the feasibility of employing Predictor on DCNs, we turn our focus to the challenge of bandwidth sharing unfairness. In particular, we show that Predictor can provide minimum bandwidth guarantees with work-conserving resource sharing under worst-case scenarios, achieving both predictability for tenants and high utilization for providers.

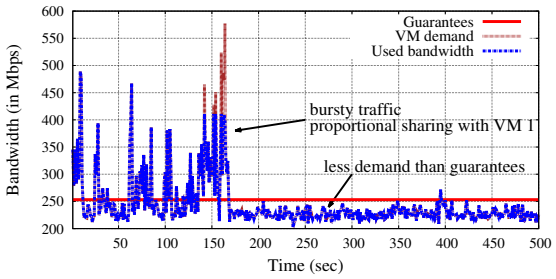
Minimum bandwidth guarantees for VMs. We define it as follows: the VM rate should be (a) at least the guaranteed rate if the demand is equal or higher than the guarantees; or (b) equal to the demand if it is lower than the guarantees. To illustrate this point, we show, in Figure 5, the set of VMs (in this case, three VMs from different applications) allocated on a given server during a predefined time period of an experiment. Note that VM1 [Figure 5(a)] and VM2 [Figure 5(b)] have similar guarantees, but receive different rates ("used bandwidth"). This happens because they have different network weights, and the rate is calculated according to their demands, bandwidth guarantees, network weight and residual bandwidth. In summary, we see that Predictor provides minimum bandwidth guarantees for VMs, since the actual rate of each VM is always equal or higher than the minimum between the application demand and the guarantees. Therefore, applications have minimum bandwidth guarantees and, thus, can achieve predictable network performance.



(a) VM 1.



(b) VM 2.



(c) VM 3.

Fig. 5: Bandwidth allocation for VMs on a given server during a predefined period of time.

Work-conserving network sharing on servers. It means that bandwidth which is not allocated, or allocated but not currently used, should be proportionally shared among other VMs with more demands than their guarantees (according to the weights of each application). Figure 6 shows the aggregate bandwidth[4] on the server holding the set of VMs in Figure 5. In these two figures, we verify that Predictor provides work-conserving sharing in the network, as VMs can receive more bandwidth (if their demands are higher than their guarantees) when there is spare bandwidth. Thus, providers can achieve high network utilization.

In general, Predictor provides significant improvements

[4]Note that Predictor considers bandwidth guarantees when allocating VMs (i.e., it does not take into account temporal demands). Therefore, even though the sum of temporal demands of all VMs allocated on a given server may exceed the server link capacity, the sum of bandwidth guarantees of these VMs will not exceed the link capacity.

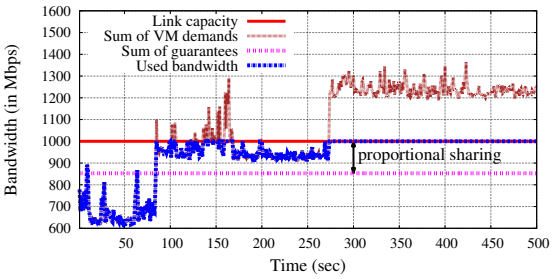


Fig. 6: Work-conserving sharing on a given server.

over DevoFlow, as it allows high utilization and fine-grained management in the network for providers and network predictability with guarantees for tenants and their applications. As a side effect, Predictor has higher controller load than DevoFlow (the cost of providing fine-grained management in the network).

V. Discussion

Application-layer flow identification. In our proof-of-concept implementation, Predictor identifies flows at application-layer through an MPLS label (application ID with 20 bits). Therefore, it needs an MPLS header in each packet (adding four bytes of overhead). In practice, when considering the matching fields defined by OpenFlow, application-layer flows could also be identified by utilizing IEEE standard 802.1ad (Q-in-Q) with double VLAN tagging. The advantage of double tagging is higher number of IDs available (24 bits), while the drawback is an overhead of eight bytes (two VLAN headers) per packet.

Dynamic rate allocation with feedback from the network. The designed work-conserving algorithm does not take into account the network feedback provided by the OpenFlow module. This design choice was deliberately made; we aim at reducing the amount of management traffic in the network, since DCNs are typically oversubscribed networks with scarce resources [6]. Nonetheless, the algorithm can be easily extended to consider feedback, which can further help controlling the bandwidth used by flows traversing congested links.

Application ID management. Predictor controller assigns IDs for applications (in order to identify flows at application-layer) upon allocation and releases IDs upon deallocation. Therefore, ID management is straightforward, as Predictor has full control over which IDs are in use at each period of time.

VM migration. Ideally, VM migration should be transparent to tenants. Predictor can maintain IPs and guarantee connectivity of VMs upon migration by readily installing new flow rules (and, if needed, removing old rules) in the appropriate forwarding devices.

Failures. Link and forwarding device failure impacts the traffic using the failed resource. By providing a logically centralized control (with real-time network state gathered by the OpenFlow module), Predictor can quickly detect failures and react accordingly, for instance, by rerouting the affected traffic (i.e., installing alternative paths).

VI. Related Work

Researchers have proposed several schemes to enable the use of the SDN paradigm in DCNs and to improve management and network sharing among tenants. Proposals related to Predictor can be divided into three classes, as follows.

OpenFlow controllers. DevoFlow [8] and DIFANE [12] propose to devolve control to the data plane. The first one introduces new mechanisms to detect elephant flows and make routing decisions at forwarding devices, while the second keeps all packets in the data plane. These schemes, however, require more complex, customized hardware at forwarding devices. Kandoo [13] provides a logically distributed control plane for large-scale networks. Nonetheless, it presents scalability issues when communication occurs between VMs located at different racks (a common case in DCNs).

Distributed rate-limiting. CloudMirror [2], Oktopus [5], Proteus [6] and Choreo [23] provide strict bandwidth guarantees for tenants by isolating applications in virtual networks. Despite their benefits, these approaches are not work-conserving and address only intra-application communication. ElasticSwitch [7] and EyeQ [24] attempt to provide strict bandwidth guarantees with work-conservation. However, they focus only on intra-application communication, and EyeQ cannot provide guarantees upon core-link congestion [25]. Finally, Hadrian [1] introduces a strategy that considers inter-application communication, but (i) it requires a larger, custom packet header (hindering its deployment); and (ii) its switches must dynamically perform rate calculation (and enforce such rate) for each flow in the network.

Flow table management. FasTrak [26] seeks to reduce hypervisor overhead by managing ToR devices and hypervisors as a unified set, moving rules back and forth. Despite the benefits, it (i) requires generic routing encapsulation (GRE) between the source and destination ToRs (20 bytes of data overhead per packet); (ii) may add latency for flows when moving their respective rules between hypervisors and ToRs; and (iii) cannot precisely rate-limit VMs (as their rate must be dynamically calculated by both ToRs and hypervisors according to the rules at each one of them).

VII. Final Remarks

We have introduced Predictor, a system that takes advantage of the SDN paradigm to provide fine-grained network management and predictable bandwidth sharing among applications in DCNs. To achieve this goal, Predictor addresses the scalability challenges of OpenFlow: i) it minimizes flow setup time by proactively installing rules for intra-application communication at allocation time (since this type of communication represents most of the traffic in DCNs); and ii) it reduces the number of flow rules in forwarding devices by managing flows at application-layer. Evaluation results show the benefits of Predictor, which can scale to large networks and provide predictable network performance. In future work, we will address Predictor's placement in DCNs, as controller placement is an important challenge of SDNs [27].

Acknowledgements

This work has been supported by MCTI/CNPq/Universal (Project Phoenix, 460322/2014-1).

REFERENCES

[1] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *USENIX NSDI*, 2013.

[2] J. Lee, Y. Turner, M. Lee, L. Popa, J.-M. Kang, S. Banerjee, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *ACM SIGCOMM*, 2014.

[3] J. Guo, F. Liu, X. Huang, J. C. Lui, M. Hu, Q. Gao, and H. Jin, "On Efficient Bandwidth Allocation for Traffic Variability in Datacenters," in *IEEE INFOCOM*, 2014.

[4] R. Shea, F. Wang, H. Wang, and J. Liu, "A Deep Investigation Into Network Performance in Virtual Machine Based Cloud Environment," in *IEEE INFOCOM*, 2014.

[5] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.

[6] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers," in *ACM SIGCOMM*, 2012.

[7] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing," in *ACM SIGCOMM*, 2013.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *ACM SIGCOMM*, 2011.

[9] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," vol. PP, no. 99, 2014, pp. 1–29.

[10] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on SDN network utilization," in *IEEE INFOCOM*, 2014.

[11] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010.

[12] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *ACM SIGCOMM*, 2010.

[13] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *ACM HotSDN*, 2012.

[14] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: sharing the network in cloud computing," in *ACM SIGCOMM*, 2012.

[15] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *USENIX NSDI*, 2011.

[16] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in *USENIX OSDI*, 2010.

[17] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *ACM SIGCOMM*, 2010.

[18] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM IMC*, 2009.

[19] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory Networking: An API for Application Control of SDNs," in *ACM SIGCOMM*, 2013.

[20] H. Moens, B. Hanssens, B. Dhoedt, and F. De Turck, "Hierarchical network-aware placement of service oriented applications in clouds," in *IEEE/IFIP NOMS*, 2014.

[21] D. Abts and B. Felderman, "A guided tour of data-center networking," *Commun. ACM*, vol. 55, no. 6, pp. 44–51, Jun. 2012.

[22] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *IEEE INFOCOM*, 2011.

[23] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan, "Choreo: Network-aware task placement for cloud applications," in *ACM IMC*, 2013.

[24] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: practical network performance isolation at the edge," in *USENIX NSDI*, 2013.

[25] J. Guo, F. Liu, D. Zeng, J. Lui, and H. Jin, "A cooperative game based allocation for sharing data center networks," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 2139–2147.

[26] R. Niranjan Mysore, G. Porter, and A. Vahdat, "FasTrak: Enabling Express Lanes in Multi-tenant Data Centers," in *ACM CoNEXT*, 2013.

[27] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *IFIP/IEEE IM*, 2013.

APPENDIX E SUBMITTED PAPER TO IFIP NETWORKING 2017

In this paper, we introduce Packer, a scheme that aims at minimizing multi-resource fragmentation and providing predictable and guaranteed network performance with work-conservation. Packer employs a novel allocation strategy that (a) extends previous heuristics developed for multi-dimensional bin packing; and (b) uses as input a new abstraction, called Time-Interleaved Multi-Resource Abstraction (TI-MRA), for specifying temporal multi-resource requirements of applications. It also leverages Software-Defined Networking to dynamically enforce bandwidth guarantees and to provide work-conserving sharing. Since Packer brings more benefits if temporal requirements are specified, it is better suited for applications that present a predefined behavior, repeatedly running the same type of tasks with similar input sizes and data sets (such as PageRank and traffic analysis). Results show that, in comparison to the state-of-the-art, acceptance ratio is increased, datacenter utilization is improved (i.e., fragmentation is minimized), provider revenue is augmented and applications achieve predictable and guaranteed network performance with work-conservation, with the cost of taking more (yet acceptable) time to allocate applications.

- **Title**: Packer: Minimizing Multi-Resource Fragmentation and Performance Interference in Datacenters
- **Conference**: IFIP Networking 2017
- **Type**: Main track (full-paper)
- **Qualis**: B1
- **Date**: June 13-15, 2017
- **Location**: Stockholm, Sweden

Packer: Minimizing Multi-Resource Fragmentation and Performance Interference in Datacenters

Daniel S. Marcon, Marinho P. Barcellos
Institute of Informatics – Federal University of Rio Grande do Sul, RS, Brazil
Email: {daniel.stefani, marinho}@inf.ufrgs.br

***Abstract*—Cloud applications perform rich and complex tasks, with time-varying demands for multiple types of resources (CPU, memory, disk I/O and network). However, multi-resource allocation is APX-Hard and, consequently, providers simplify it by (i) allocating computing resources according to slots (which leads to fragmentation); and (ii) allowing the network to be shared in a best-effort manner (which leads to performance interference among applications). Recent efforts cannot minimize multi-resource fragmentation and, at the same time, provide guaranteed network performance. In this paper, we introduce Packer, a scheme that aims at minimizing multi-resource fragmentation and providing predictable and guaranteed network performance with work-conservation. Packer employs a novel allocation strategy that (a) extends previous heuristics developed for multi-dimensional bin packing; and (b) uses as input a new abstraction, called Time-Interleaved Multi-Resource Abstraction (TI-MRA), for specifying temporal multi-resource requirements of applications. It also leverages Software-Defined Networking to dynamically enforce bandwidth guarantees and to provide work-conserving sharing. Since Packer brings more benefits if temporal requirements are specified, it is better suited for applications that present a predefined behavior, repeatedly running the same type of tasks with similar input sizes and data sets (such as PageRank and traffic analysis). Results show that, in comparison to the state-of-the-art, acceptance ratio is increased, datacenter utilization is improved (i.e., fragmentation is minimized), provider revenue is augmented and applications achieve predictable and guaranteed network performance with work-conservation, with the cost of taking more (yet acceptable) time to allocate applications.**

I. Introduction

Cloud applications often perform rich and complex tasks, with different temporal demands for multiple types of resources (CPU, memory, disk I/O and network) [1]. They typically have multiple stages, where a subsequent stage can only start after the previous stage finishes [2]. Consequently, any resource that becomes a bottleneck and delays a stage may slow down the entire application.

In this context, multi-resource allocation is a key building block of cloud datacenters. Because the allocation problem is APX-Hard [3], state-of-the-art proposals typically simplify it by (i) allocating computing resources according to slots [1]; and (ii) allowing the network to be shared in a best-effort manner, which leads to performance interference among applications [4], [5]. The former (slot-based allocation) usually causes over-allocation, leading to wastage (as applications do not use all of their allocated resources) and fragmentation [6]. In particular, fragmentation results in less applications being accepted in the infrastructure and in lower datacenter utilization. The latter (performance interference) results in poor and unpredictable network performance for applications.

Allocating resources to applications in datacenters has been the main focus of several recent efforts [1], [4], [6]. Tetris [6] considers multiple resources at allocation time, but does not provide bandwidth guarantees along the entire network. Dominant Resource Fairness (DRF) [1] also considers multiple resources, but focuses on fairness (which may result in fragmentation [6]). Silo [4] offers predictable network performance, but allocates computing resources according to slots. In general, these approaches cannot minimize multi-resource fragmentation and, at the same time, provide guaranteed network performance.

In this paper, we propose Packer, a scheme for large-scale Infrastructure-as-a-Service cloud datacenters. Its design is based on two observations: (i) applications have complementary demands across time for multiple resources [6]; and (ii) utilization of different resources peaks at different times [7]. Packer has two objectives: minimizing multi-resource fragmentation (consequently, increasing datacenter utilization); and providing predictable and guaranteed network performance with work-conserving sharing. To achieve these goals, it takes into account multiple types of resources to admit (allocate) applications in the datacenter without considering slots, so that tenants can request and receive the necessary amount of resources that their applications need to correctly execute and finish without delay and providers can avoid over-allocation.

Packer is designed with four aspects in mind: application abstraction, multi-resource allocation, network sharing and resource monitoring. First, Packer utilizes a novel abstraction for applications, called Time-Interleaved Multi-Resource Abstraction (TI-MRA). Unlike previous abstractions [2], [8]–[10], TI-MRA imposes no predefined structure for applications and allows the specification of requirements for multiple resources across time. Second, Packer employs a new allocation strategy that extends previous heuristics developed for multi-dimensional bin packing, in order to reduce multi-resource fragmentation. Third, Packer leverages Software-Defined Networking (SDN) and OpenFlow [11] to dynamically configure and enforce bandwidth guarantees for applications throughout the entire network. Fourth, Packer employs a monitoring mechanism to avoid resource wastage and to provide fast and up-to-date information upon unexpected events (e.g., if an application gets delayed due to a resource being congested).

Like Proteus [2] and the strategy in [7], Packer uses temporal resource demands to achieve maximum benefits. Consequently, it is better suited for applications that present a predefined behavior, repeatedly running the same type of tasks with similar input and data sets. This is common in iterative data processing (e.g., PageRank, hypertext-induced topic search, recursive relational queries, social network analysis and network traffic analysis), where much of the data stays unchanged from iteration to iteration [2]. In this case, applications are profiled periodically or on each run. Furthermore, other applications can also take advantage of Packer by specifying only peak demands for multiple resources and, at runtime, employing its monitoring mechanism not to waste

resources.

Overall, the major contributions of this paper are:

- A novel abstraction for applications, called Time-Interleaved Muti-Resource Abstraction (TI-MRA). In contrast to previous abstractions [2], [8]–[10], TI-MRA allows the specification of demands for multiple resources without a predefined structure for applications.
- A novel admission control algorithm that, by extending existing heuristics for multi-dimensional bin packing, minimizes resource fragmentation. The algorithm uses TI-MRA as input to coordinate requirements of applications in different resource dimensions across time.
- Packer, a scheme that combines TI-MRA and the novel admission control algorithm to provide predictable and guaranteed network performance with work-conserving sharing. Evaluation results show that it provides network performance guarantees, and improves datacenter utilization and provider revenue in comparison to related work, with the cost of taking more (yet acceptable) time to allocate applications.

This paper is organized as follows. Section II introduces Packer, while Section III presents its evaluation. Section IV discusses the generality and limitations of Packer, and Section V examines related work. Finally, Section VI concludes the paper with final remarks and perspectives for future work.

II. PACKER

Packer implements a novel strategy for minimizing multi-resource fragmentation and for providing predictable and guaranteed network performance in large-scale cloud datacenters. Figure 1 shows an overview of Packer. The scheme is composed of three components: datacenter resource manager (DRM), OpenFlow-enabled switches and one local controller per server. They are discussed next.

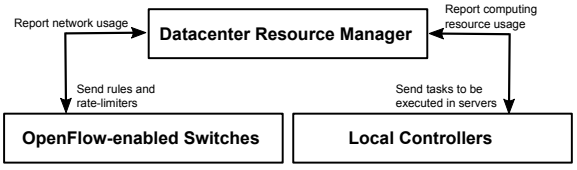


Fig. 1: Packer overview.

Datacenter resource manager (DRM). It is responsible for (i) allocating applications in the datacenter; and (ii) handling global events (e.g., bandwidth enforcement throughout the entire network for applications). More specifically, it receives an application request (in the form of a TI-MRA specification) and employs a novel resource allocation strategy (described in § II-B) to determine the set of resources to be used by the application. Then, it sends the application tasks to the servers that will execute them (as determined by the allocation strategy) and, via OpenFlow, configures switches (with rules and rate-limiters) to provide connectivity and network performance guarantees for the application. Furthermore, the DRM periodically receives up-to-date information regarding resource usage from local controllers at servers and from OpenFlow switches.

TABLE I: Notations adopted throughout the paper.

Symbol	Description
A	Set of application requests
G^a_{TI-MRA}	TI-MRA graph of application $a \in A$
V^a	Set of nodes of application $a \in A$ ($V^a = K^a \cup C^a$)
K^a	Set of tasks of application $a \in A$ ($K^a \subseteq V^a$)
C^a	Set of cloud services used by app $a \in A$ ($C^a \subset V^a$)
E^a	Set of edges (dependencies between nodes) of app $a \in A$
T^a	Discrete time instants of application $a \in A$ ($T^a \subseteq \mathcal{T}$)
w^a	Weight of application $a \in A$
N	Set of all infrastructure nodes ($N = S \cup \mathcal{J}$)
S	Set of servers in the datacenter infrastructure ($S \subseteq N$)
$\mathcal{J}$	Set of services available in the datacenter ($\mathcal{J} \subset N$)
$\mathcal{L}$	Set of links in the datacenter network
$\mathcal{T}$	Discrete time instants of the infrastructure
$\mathcal{P}$	Set of all paths available in the network
$\mathcal{P}(n_1, n_2)$	Set of paths from src node $n_1 \in N$ to dest node $n_2 \in N$
$w^{\mathtt{r}}$	Weight of $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}, \mathtt{BAND}\}$
$\delta(v, \mathtt{r}, t)$	Amount of resource $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$ required by node $v \in V^a$ at time $t \in T^a$ for app $a \in A$
$\sigma(e = (u, v), t)$	Bandwidth for communication between nodes $u, v \in V^a$ | $e = (u, v) \in E^a$ at time $t \in T^a$ for application $a \in A$
$\mathcal{M}_n(v)$	Node $n \in N$ that holds the node $v \in V^a$ of app $a \in A$
$\mathcal{M}_e(u, v)$	Infrastructure path ($\mathcal{P}(\mathcal{M}_n(u), \mathcal{M}_n(v))$) used for communication between nodes $u, v \in V^a$ of app $a \in A$
$\mathcal{A}(n)$	Tasks running at infrastructure node $n \in N$
$\mathcal{B}(l)$	Total capacity (bandwidth) of link $l \in \mathcal{L}$
$\mathcal{C}(l)$	Communications (edges in the TI-MRA) using link $l \in \mathcal{L}$
$\mathcal{D}(u)$	Application that task u belongs to
$\mathcal{E}(v)$	Nodes that node $v \in V^a \mid a \in A$ depends on
$\mathcal{Q}(n, \mathtt{r}, t)$	Amount of available resource $\mathtt{r}$ on $n \in N$ at time $t \in \mathcal{T}$
$\mathcal{R}(n, \mathtt{r})$	Capacity of infrastructure node $n \in N$ for resource type $\mathtt{r}$

OpenFlow switches. These devices are responsible for forwarding traffic according to the instructions received from the DRM. They receive rules and rate-limiters to correctly handle traffic and enforce bandwidth for applications. Moreover, they periodically report resource usage statistics to the DRM.

Local controllers (LCs) at servers. They are part of the resource monitoring mechanism utilized in Packer (described in § II-D). LCs are responsible for (i) monitoring multi-resource usage at servers and reporting it to the DRM; (ii) enforcing allocations; and (iii) reacting to local events (e.g., dealing with congested resources inside their respective server).

We detail Packer in the following manner. We first present the novel abstraction (called TI-MRA) used for applications in § II-A. Then, we utilize TI-MRA as input for the new allocation algorithm (§ II-B) and describe the strategy used for providing predictable and guaranteed network performance (§ II-C). Finally, we detail the resource monitoring mechanism in § II-D. The notations used throughout the paper are presented in Table I.

A. Time-Interleaved Multi-Resource Abstraction (TI-MRA)

Prior work has designed abstractions expressed as physical network models (i.e., the hose model) [2], [4], [8], two-level trees (hierarchical hose) [8], [10] or based on communication patterns (TAG) [9]. However, they focus on the network and neglect other resources. In particular, the hose model (used by most related work) does not accurately capture the network requirements of applications with complex traffic interactions [9].

An effective abstraction is expected to consider two purposes. The first is to allow tenants to specify their application requirements in a simple and accurate manner. The second is to

allow providers to minimize over-allocation (i.e., allocating the correct amount of resources required by applications), which may increase the percentage of allocated applications and, consequently, may improve datacenter throughput.

Based on these purposes and the limitations of prior work, we propose a novel abstraction for applications, called Time-Interleaved Multi-Resource Abstraction (TI-MRA). TI-MRA allows the specification of not only network demands but also other types of resources. TI-MRA leverages tenants' knowledge of their applications to yield a flexible representation of the applications' resource consumption. It uses the same principle of (a) temporal bandwidth requirements in TIVC [2], but extends it to all kinds of resources; and (b) communication patterns in TAG [9]. Furthermore, it also takes into account dependencies other than among tasks (such as between tasks and cloud services), in order to optimize the use of resources. The intuition is that TI-MRA allows a flexible representation of application requirements rather than imposing a predefined abstraction (e.g., the hose model) for applications to map their requirements to.

TI-MRA extends the concept of time-varying graphs (TVGs) [12] to represent temporal demands of multiple resources. A TI-MRA graph of application $a \in A$ is represented as $G^a_{\texttt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$, with the terms being defined as follows: $V^a = K^a \cup C^a$ is the set of application nodes, composed of tasks (K^a) and cloud services required by the application (C^a); E^a is the set of edges, representing the dependencies between nodes; T^a is the set of discrete time instants, from the time the first node of application a begins its computation to the time the last node finishes; $w^a \in [0,1]$ indicates the weight of application a, so that residual resources (unallocated, or reserved resources for an application and not currently being used) can be proportionally shared among applications with more demands than their guarantees (work-conservation); and $\delta(v, \mathtt{r}, t) \in \mathbb{R}^+$ returns the demand of node $v \in V^a$ at time $t \in T^a$ for resource $\mathtt{r} \in \{\texttt{CPU}, \texttt{MEM}, \texttt{IO\_WRITE}, \texttt{IO\_READ}\}$. The last function, $\sigma(e = (u,v), t) \in \mathbb{R}^+$, denotes the bandwidth necessary for communication between nodes $u \in V^a$ and $v \in V^a \mid u \neq v$ at time $t \in T^a$, for $e = (u,v) \in E^a$. Note that we do not consider moving nodes and edges across time. This does not impact the generality of TI-MRA because when a node or edge has no demand for a given resource, the call for the respective function ($\delta(v, \mathtt{r}, t)$ or $\sigma(e = (u,v), t)$) returns zero.

An example of TI-MRA is shown in Figure 2. The figure depicts a simple application composed of five tasks and temporal resource requirements for CPU, memory, disk I/O write, disk I/O read and bandwidth. In this example, tasks T_1 and T_2 get their input data from storage service STS_1; T_3 depends on tasks T_1 and T_2 and on data sent from cloud service CS_1; T_4 reads data from storage service STS_2 to perform its computation; and T_5 depends on tasks T_3 and T_4 and stores the final result in STS_3. Moreover, note that edges (links representing the exchange of data) are unidirectional (different amounts of bandwidth for sending and receiving data). Having two links instead of a single bidirectional link avoids over-allocation and bandwidth wastage.

Producing TI-MRA models. TI-MRA can be used not only by tenants who have a deep understanding of their application demands, but also by users who do not know it in advance. The former can tune resource demands according to the application requirements, possibly reducing costs (avoiding over-allocation) without impact on performance. The latter, in turn, can specify only peak demands for resources (i.e., a constant temporal function). This would be similar to the hose model specification. Alternatively, the same strategy employed in CloudMirror [9] for generating TAG models could be used here: application templates for TI-MRA could be provided as a library for users through the extension of cloud orchestration systems like OpenStack Heat and AWS CloudFormation.

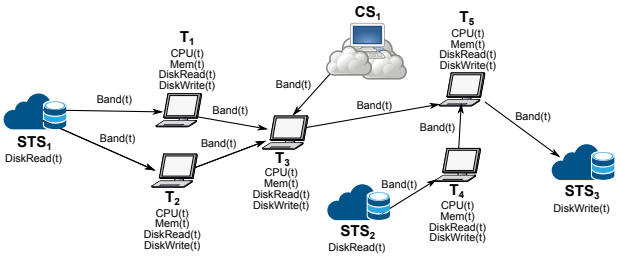


Fig. 2: TI-MRA of a simple application composed of five tasks (T_1 to T_5), where tasks read and write data from/to storage services (STS_1, STS_2 and STS_3) and use cloud service CS_1.

B. Allocation Strategy

The problem of allocating applications considering multiple types of resources in datacenters can be reduced to multi-dimensional bin packing (also called vector bin packing – VBP) [6], which is NP-Hard for every dimension d and APX-Hard for $d \geq 2$ [3]. Given balls (application tasks) and bins (servers) with sizes for each property (resource) considered, VBP assigns the balls to bins according to an optimization objective. In our case, the goal is to reduce fragmentation and over-provisioning and, consequently, improve the percentage of allocated applications and their tasks.

In case applications are constrained by a single resource (e.g., network), the problem becomes, in essence, a one dimensional bin packing [3]. However, cloud applications are typically constrained by multiple resources, including network [4] and CPU [13]. Moreover, the network is a distributed resource (composed of several links); therefore, the amount of resources consumed depends on bandwidth demands of tasks as well as their location in the infrastructure (the whole path used for communication must be taken into account), which increases the difficulty in efficiently optimizing the use of resources.

Problem definition. The process of multi-resource allocation is formally defined as follows. The TI-MRA specification of application $a \in A$ is given by $G^a_{TI-MRA} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$. A node $v \in V^a$ can be either a task or a cloud service and is mapped to an infrastructure node $n \in N$. Each task $k \in K^a \mid K^a \subseteq V^a$ is assigned to an infrastructure server ($s \in S \mid S \subseteq N$) by mapping $\mathcal{M}_n : K^a \rightarrow S,\ \forall a \in A$ (Equation 1). Each cloud service $c \in C^a \mid C^a \subset V^a$ required by application $a \in A$, in turn, is part of the set of services ($\mathcal{J} \mid \mathcal{J} \subset N$) available in the platform (offered by either the provider or a tenant) and is assigned to a node $j \in \mathcal{J}$ that runs the requested service by mapping $\mathcal{M}_n : C^a \rightarrow \mathcal{J},\ \forall a \in A$ (Equation 2).

$$\mathcal{M}_n(k) \in S \mid k \in K^a \text{ or} \tag{1}$$

$$\mathcal{M}_n(c) \in \mathcal{J} \mid c \in C^a \tag{2}$$

Each dependency between nodes (i.e., edges in the TI-MRA graph, specified in set E^a) is mapped to a single path between the corresponding infrastructure nodes (servers for tasks and cloud services for services needed by the application). The assignment is defined by mapping $\mathcal{M}_e : E^a \rightarrow \mathcal{P},\ \forall a \in A$, where $\mathcal{P}$ denotes the set of all available paths in the network, such that for all $e = (u, v) \in E^a,\ \forall a \in A$:

$$\mathcal{M}_e(u, v) \in \mathcal{P}\left(\mathcal{M}_n(u), \mathcal{M}_n(v)\right) \tag{3}$$

The nodes in V^a run for at most T^a discrete time units to perform the computation required by application $a \in A$ and communicate among themselves using links $e \in E^a$. Each node $v \in V^a$ presents temporal demands for computing resource $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$ ($\delta(v, \mathtt{r}, t)$), $\forall t \in T^a$. Furthermore, each edge $e = (u, v) \in E^a$ between communicating nodes u and v (such that $u \neq v$) presents temporal bandwidth demands: $\sigma(e = (u, v), t), \forall t \in T^a$.

A valid allocation is constrained by the amount of available resources. More specifically, a cloud service $j \in \mathcal{J}$ will only perform the task required by node $c \in C^a \mid a \in A$ if it has enough available resources to satisfy the demands. Similarly, a task can only be allocated in a server if the server has enough available capacity for each type of resource being considered. Given $\mathcal{A}(n)$ a function that returns all tasks running at infrastructure node $n \in N$, $\mathcal{R}(n, \mathtt{r})$ a function that returns the capacity of infrastructure node $n \in N$ for resource $\mathtt{r}$ and $\mathcal{T}$ the discrete time instants of the infrastructure, the constraint for computing resources is defined as follows:

$$\sum_{v \in \mathcal{A}(n)} \delta(v, \mathtt{r}, t) \leq \mathcal{R}(n, \mathtt{r}) \quad \forall n \in N,\ \forall \mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\},\ \forall t \in \mathcal{T} \tag{4}$$

The network is a special case, since it is a distributed resource. Specifically, bandwidth must be taken into account at the entire path used for each communication between application nodes and the amount of allocated bandwidth at a link l must not exceed the total capacity of l (Equation 5).

$$\sum_{e=(u,v) \in \mathcal{C}(l)} \sigma(e, t) \leq \mathcal{B}(l) \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T} \tag{5}$$

where $\mathcal{L}$ represents the set of links in the datacenter network, $\mathcal{C}(l)$ returns the set of communications (edges in TI-MRA graphs of applications) using link l and $\mathcal{B}(l)$ returns the total capacity (bandwidth) of link l.

Note that the above constraints are non-linear, in particular functions $\mathcal{A}$ and $\mathcal{C}$ (since they depend on the allocation of application nodes in infrastructure nodes). Efficient solvers are known only for some non-linear problems, such as the quadratic assignment problem. However, even when placement considerations are eliminated, the problem of VBP is APX-Hard [14] and re-solving it whenever new applications arrive worsens the process. Consequently, finding the optimal solution is expensive and requires a lot of time. Unfortunately, large-scale cloud datacenters require the allocation process to be performed as fast as possible, since they typically have high rate of application arrival and departure.

Algorithm. VBP has several proposed heuristics [3]. However, those heuristics cannot be used in datacenters without substantial modification, as they (*i*) assume that all input (i.e., applications) is known a priori, whereas we need to cope with online arrival of applications; (*ii*) consider "balls" of a fixed size, while applications have time-varying demands; and (*iii*) do not consider a distributed resource such as the network (with paths composed of multiple links). Therefore, we design a novel algorithm to efficiently allocate applications in cloud datacenters. The key principle is minimizing multi-resource fragmentation, thus improving the ratio of allocated applications and, consequently, maximizing datacenter utilization and provider revenue.

Algorithm 1 allocates one application at a time, as requests arrive. It receives as input the datacenter infrastructure ($\langle N, \mathcal{L}, \mathcal{T}, \mathcal{P} \rangle$) and the TI-MRA specification of an application $a \in A$ ($G^a_{\mathtt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$). First, it calculates available resources (CPU, memory, disk I/O write and disk I/O read) in servers (lines 1 – 2) and available bandwidth in links (lines 3 – 4). Since each node of application a may have a different duration, available resources in the infrastructure are calculated for T^a time units (the duration of a).

After that, nodes in V^a are sorted sequentially according to their initial execution time and the nodes they depend on (line 5). Based on the sorted list of nodes (`sNodes`), the algorithm gets one node v at a time (line 6), initializes as empty the list of infrastructure nodes with enough available resources to hold node v (line 7) and calculates, based on our novel metric shown in Equation 6, the score of v on infrastructure nodes (lines 8 – 11). The metric extends the ones in Panigrahy et al. [3] and works as follows. Equation 6 seeks to maximize the score achieved by both computing and network resources according to their current utilization level (calculated in Equations 7 and 8, respectively) in case there are enough available resources (i.e., node $n \in N$ and link $l \in \mathcal{L}$, which v would use for communication with the application nodes it depends on if it were allocated on n, have enough available resources at all times $t \in T^a$). Otherwise, it returns $-\infty$. For each computing resource $R = \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$ (Equation 7) and for bandwidth `BAND` (Equation 8), the metric subtracts the resource demand from the respective available resource, raises the resulting value by the power of 3, multiplies it by the amount of the respective available resource and multiplies it again by the weight associated to the resource ($w^{\mathtt{r}}$). In particular, $w^{\mathtt{r}}$ is dynamically defined as being inversely proportional to the current utilization level of $\mathtt{r}$ and is calculated as $w^{\mathtt{r}} = 1 - (\frac{\mathtt{util(r)}}{\sum_{s \in R \cup \{\mathtt{BAND}\}} \mathtt{util(s)}}), \forall \mathtt{r} \in R \cup \{\mathtt{BAND}\}$. That is, the higher the current utilization of $\mathtt{r}$, the lower its weight. This way, the metric prioritizes resources with lower utilization.

$$\mathtt{S}[n, v] = \begin{cases} \mathtt{S_C}[n, v] + \mathtt{S_B}[n, v] & \delta(v, \mathtt{r}, t) \leq \mathcal{Q}(n, \mathtt{r}, t) \text{ and } \sigma((u, v), t) \leq \mathcal{Q}(l, \mathtt{BAND}, t), \\ & \forall \mathtt{r} \in R, \forall u \in \mathcal{E}(v), \forall t \in T^a, \\ & \forall l \in \mathcal{M}_e(u, v); \\ -\infty & \text{otherwise.} \end{cases} \tag{6}$$

$$\mathtt{S_C}[n, v] = \sum_{t \in T^a} \sum_{\mathtt{r} \in R} w^{\mathtt{r}} * \left(\mathcal{Q}(n, \mathtt{r}, t) - \delta(v, \mathtt{r}, t)\right)^3 * \mathcal{Q}(n, \mathtt{r}, t) \tag{7}$$

$$\mathtt{S_B}[n, v] = \sum_{t \in T^a} \sum_{u \in \mathcal{E}(v)} \sum_{l \in \mathcal{M}_e(u, v)} w^{\mathtt{BAND}} * \left(\mathcal{Q}(l, \mathtt{BAND}, t) - \sigma((u, v), t)\right)^3 * \mathcal{Q}(l, \mathtt{BAND}, t) \tag{8}$$

Note that the metric described here uses normalized values (for resource requirements of applications as well as residual resources in the datacenter infrastructure) by the capacity of the node/link being considered.

The next step is the allocation of v and its communication dependencies (edges with v as the destination node in the

TI-MRA graph) in lines 12 – 17, according to Equations 1 – 5. Function `GetNodeWithBestScore` returns the infrastructure node with the best (maximum) score in the list `FeasibleNodes` for holding node v (line 12). In case no infrastructure node has enough available resources to hold v, the algorithm returns a failure code and application a is discarded (line 13). Otherwise, node v is allocated at infrastructure node n (line 15) and, since nodes that v depends on are already allocated (i.e., dependencies are allocated first, according to the sorted list of nodes), bandwidth for communication between v and its dependencies is also allocated (lines 16 – 17). When all nodes and edges in the TI-MRA graph of application a are successfully allocated, the algorithm returns a success code and finishes (line 18).

Algorithm 1: Multi-Resource Allocation Algorithm.

Input : Datacenter infrastructure $\langle N, \mathcal{L}, \mathcal{T}, \mathcal{P} \rangle$, Application a represented by $G^a_{\texttt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$
Output: Success/Failure code

1 **foreach** *Infrastructure node* $n \in N$ **do**
2 $\quad \mathcal{Q}[n, \mathtt{r}, t] \leftarrow \mathcal{R}(n, \mathtt{r}) - \sum_{v \in \mathcal{A}(n)} \delta(v, \mathtt{r}, t),\ \forall \mathtt{r} \in \{\texttt{CPU}, \texttt{MEM}, \texttt{IO\_WRITE}, \texttt{IO\_READ}\},\ \forall t \in T^a \mid T^a \subseteq \mathcal{T};$
3 **foreach** *Infrastructure link* $l \in \mathcal{L}$ **do**
4 $\quad \mathcal{Q}[l, \texttt{BAND}, t] \leftarrow \mathcal{B}(l) - \sum_{e=(u,v) \in \mathcal{C}(l)} \sigma(e, t),\ \forall t \in T^a \mid T^a \subseteq \mathcal{T};$
5 $\texttt{sNodes} \leftarrow \texttt{SortNodes}\ (V^a);$
6 **foreach** $v \in \texttt{sNodes}$ **do**
7 $\quad \texttt{FeasibleNodes} \leftarrow \emptyset;$
8 $\quad$ **foreach** $n \in N$ **do**
9 $\quad\quad \texttt{S}[n, v] \leftarrow$ calculate score according to Equations 6, 7 and 8;
10 $\quad\quad$ **if** $\texttt{Score}[n, v] \neq -\infty$ **then**
11 $\quad\quad\quad \texttt{FeasibleNodes} \leftarrow \texttt{FeasibleNodes} \cup \{n\};$
12 $\quad n \leftarrow \texttt{GetNodeWithBestScore}(\texttt{FeasibleNodes});$
13 $\quad$ **if** n *is null* **then return** failure code;
14 $\quad$ **else**
15 $\quad\quad \mathcal{M}_n(v) \leftarrow n;$
16 $\quad\quad$ **foreach** $u \in \mathcal{E}(v)$ **do**
17 $\quad\quad\quad \mathcal{M}_e(u, v) \leftarrow p \mid p \in \mathcal{P}\,(\mathcal{M}_n(u), \mathcal{M}_n(v));$
18 **return** success code;

C. Network Sharing Strategy

The network sharing strategy has two objectives: (i) providing predictable and guaranteed network performance for applications, in order to avoid performance interference [4]; and (ii) achieving work-conserving sharing, so that applications have the possibility of using more bandwidth than their guarantees when needed and providers can achieve high network utilization.

To achieve these goals, we leverage the paradigm of SDN to dynamically configure the network, in order to enforce bandwidth guarantees and to provide work-conserving sharing. The strategy works as follows. According to the output of Algorithm 1 for application $a \in A$, the DRM performs two actions. First, it sends each task of a to the local controller of its selected server (as LCs manage and enforce resource allocation at servers). Second, it installs rules and rate-limiters in forwarding devices to guarantee connectivity and bandwidth for communication between these nodes.

In addition to ensuring a base level of guaranteed rate for applications, the strategy can proportionally share available bandwidth among applications with more demands than their guarantees. Towards this end, local controllers run an algorithm to periodically set the allowed rate for each allocated application node. Algorithm 2 aims at enabling smooth response to bursty traffic (since traffic in DCNs may be highly variable over short periods of time [15]). It receives as input the infrastructure node $n \in N$ that it belongs to, the current time $t \in \mathcal{T}$, current bandwidth demands of application nodes allocated at n (which are determined by monitoring socket buffers, similarly to Mahout [16]) and temporal bandwidth requirements of these nodes (specified in the request). First, the algorithm initializes as empty the list of application nodes with more bandwidth demands than the value specified in the request (line 1). Then, for each application node v allocated at n (line 2), the minimum rate between (i) the specified demand at time t ($\sigma(\sum(v, *), t)$, which represents the sum of bandwidth required by node v for communication with all nodes that depend on v at time t) and (ii) the current demand of v ($d[v]$) is assigned to `nRate` (line 3). If the current demand is higher than the specified demand, the node is added to the list of nodes with more demands than their guarantees (called `hungryNodes`, in line 4).

Algorithm 2: Work-conserving algorithm.

Input : Infrastructure node n, Time $t \in \mathcal{T}$, Current bandwidth demands of applications nodes d, Temporal bandwidth requirements of application nodes σ
Output: Rate `nRate` for each application node

1 $\texttt{hungryNodes} \leftarrow \emptyset;$
2 **foreach** $v \in \mathcal{A}(n)$ **do**
3 $\quad \texttt{nRate[v]} \leftarrow \min(\sigma(\sum(v, *), t), d[v]);$
4 $\quad$ **if** $\sigma(\sum(v, *), t) < d[v]$ **then** $\texttt{hungryNodes} \leftarrow \texttt{hungryNodes} \cup v;$
5 $\mathcal{Q}(n, \texttt{BAND}, t) \leftarrow \mathcal{B}(link) - \sum_{v \in \mathcal{A}(n)} \texttt{nRate[v]}$, at time t;
6 **while** $\mathcal{Q}(n, \texttt{BAND}, t) > 0$ *and* `hungryNodes` *not empty* **do**
7 $\quad$ **foreach** $v \in \texttt{hungryNodes}$ **do**
8 $\quad\quad \texttt{value} \leftarrow \min\left(d[v] - \texttt{nRate[v]}, \left(\frac{w^{\mathcal{D}(v)}}{\sum_{u \in \texttt{hungryNodes}} w^{\mathcal{D}(u)}} \times \mathcal{Q}(n, \texttt{BAND}, t)\right)\right);$
9 $\quad\quad \texttt{nRate[v]} \leftarrow \texttt{nRate[v]} + \texttt{value};$
10 $\quad\quad \mathcal{Q}(n, \texttt{BAND}, t) \leftarrow \mathcal{Q}(n, \texttt{BAND}, t) - \texttt{value};$
11 $\quad\quad$ **if** $\texttt{nRate[v]} == d[v]$ **then** $\texttt{hungryNodes} \leftarrow \texttt{hungryNodes} \setminus \{v\};$
12 **return** `nRate`;

Then, the algorithm calculates the residual bandwidth ($\mathcal{Q}(n, \texttt{BAND}, t)$) of the link connecting server n to its top-of-rack (ToR) switch at time t (line 5). The residual bandwidth is calculated by subtracting from the link capacity the rate assigned to the application nodes (in line 3). The last step establishes the bandwidth rate for application nodes with more demands than their guarantees, if there is available bandwidth (lines 6 – 11). The rate of each node $v \in$ `hungryNodes` (in line 9) is determined by adding `nRate[v]` (initialized in line 3) and the minimum bandwidth between (i) the difference of the current demand ($d[v]$) and the rate (`nRate[v]`); and (ii) the proportional share of residual bandwidth the application node can receive according to its weight $w^{\mathcal{D}(v)}$ (calculated in line 8), where $\mathcal{D}(v)$ indicates the application that node v belongs to. The residual bandwidth is updated in line 10 and, in case the demands of node v were satisfied, it is removed from the list `hungryNodes` (line 11). Note that there is a "while" loop (lines 6 – 11) to guarantee that all residual bandwidth is used or all demands are satisfied. If this loop were not used, there could be occasions when there would be unsatisfied demands even though some bandwidth would be available.

In summary, if the demand of an application node exceeds its guaranteed rate (the rate specified in the request – σ), data can be sent and received at least at the guaranteed rate. Otherwise, if it does not, the unutilized bandwidth will be shared among co-resident application nodes whose traffic demands exceed their guarantees (work-conservation).

Finally, note that SDN has scalability challenges on DCNs [11]: (i) elevated flow setup time, as forwarding devices ask the controller for appropriate rules when they receive the first packet of a new flow; and (ii) large flow tables in switches, since DCNs may have millions of flows per second [17] and, thus, the number of entries needed in TCAMs may be significantly higher than the amount of resources available in commodity switches. We adopt the strategy proposed in Marcon et al. [18] to address these challenges. The interested reader may refer to [18] for more details.

D. Resource Monitoring Mechanism

Packer is designed with scalability and high multi-resource utilization (i.e., minimizing fragmentation of multiple resources) in mind. This implies that the resource monitoring mechanism (i) should not incur significant overhead (especially to scarce resources such as the network [2]); and (ii) needs to be able to acquire real-time information about resource usage, so that idle resources can be allocated to applications that need them. Moreover, the mechanism is expected to provide fast and up-to-date information upon unexpected events (e.g., in case an application gets delayed due to a resource being congested).

We designed a two-level strategy for resource monitoring, composed of (i) the DRM and (ii) local controllers at servers and OpenFlow switches. First, a local controller runs at each server and coordinates the allocation of the server's resources to application tasks. LCs have two objectives, described as follows. The first objective is to observe aggregate resource usage and periodically report it to the DRM (so that the DRM gets updated information about the infrastructure utilization). The second objective is related to handling local events: since LCs have no interconnection among themselves (in order to reduce management traffic in the network) and no knowledge of infrastructure-wide state, they are allowed to handle only local events (e.g., dealing with local congested resources and enforcing allocations to tasks). This is important for relieving the load on the DRM and for reducing the amount of bandwidth used for communication between LCs and the DRM.

Second, the DRM maintains infrastructure-wide state, as it periodically receives resource usage statistics from local controllers at servers and from switches (via the OpenFlow protocol). With the information received from servers and switches, it reacts to global events such as the allocation of applications and bandwidth enforcement throughout the entire network for applications.

III. Evaluation

In this section, we focus on showing that Packer (i) minimizes multi-resource fragmentation; (ii) improves provider revenue; (iii) incurs acceptable overhead; (iv) provides predictable and guaranteed network performance with work-conserving sharing; and (v) outperforms existing state-of-the-art schemes (Tetris [6] and slot-based allocation [19], [20]).

A. Setup

Environment. We have implemented a simulator that models computing and network resources of a multi-tenant datacenter. For computing resources, we follow Tetris [6] and use a similar server configuration: 16 CPU cores, 32 GB of memory, 4 disks operating at 50 MBps each for read and write operations and a 1 Gbps NIC. For the slot-based scheme, we follow related work [8] and divide each server into four equal slots. The network, in turn, is defined as a tree-like topology, similar to current DCNs and related work [4]. It is composed of a three-tier topology with 1,200 servers at level 0. Every 40 machines form a rack, and every 10 ToRs are connected to an aggregation switch. Finally, all aggregation switches are connected to a core switch. Unless otherwise specified, the capacity of each link is defined as follows: 1 Gbps for server-ToR links, 10 Gbps for ToR-aggregation links and 100 Gbps for aggregation-core links.

Workload. We built a workload suite composed of incoming application requests (to be allocated in the datacenter) arriving over time. We consider a heterogeneous set of applications, including MapReduce and Web Services. As defined in § II-A, each application a is represented by a TI-MRA graph $G^a_{\texttt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$. Given the lack of publicly available traces for DCNs, the workload was generated in line with related work [2], [6], [17], [21], [22]. First, like Tetris [6], computing resources of tasks were picked uniformly at random between 0 and the maximum value of a slot. Note that we limit the demand for computing resources of each task from each application to the maximum size of a slot in order to provide a fair and accurate comparison (otherwise, since we use the same workload for all schemes, some tasks would never be allocated with the slot-based approach). Second, bandwidth demands were generated based on the measurements from Benson et al. [17] and Kandula et al. [22]. Finally, the weight w^a of each application a is uniformly distributed in the interval $[0, 1]$.

B. Results

We compare Packer with Tetris [6] and the slot-based allocation [19], [20]. For all experiments comparing different strategies, we plot the percentage improvement (or reduction) between Packer and the related work being compared as $\frac{\texttt{Packer}-\texttt{related\_work}}{\texttt{Packer}} * 100\%$. Hence, positive values mean Packer has achieved a higher value than the approach being compared, while negative values mean Packer has achieved a lower value. In general, higher values are better, with the sole exception being the overhead of the allocation algorithm (Figure 6).

Increased acceptance ratio. Figure 3 shows the proportion of application tasks that were allocated between Packer and Tetris and Packer and slot-based according to the time. *Higher values are better*, as they mean that Packer allocates more tasks than the respective proposal being compared. At first, the gains of Packer in comparison to the other proposals have high variability because there are ample resources and, therefore, most incoming applications are allocated. As time passes and the cloud-load increases (less available resources), the gains tend to stabilize (around time 1,000), because new applications are allocated only when already allocated applications conclude their execution and are deallocated (which releases resources). In general, we observe that Packer consistently outperforms Tetris (≈30%) and slot-based allocation (≈67%). Although the amount of available resources in the infrastructure is the same, the allocation ratio differs for each approach. This happens because each scheme uses a different allocation strategy. More specifically, Tetris seeks to minimize computing resource fragmentation, while only penalizing the bandwidth used. This may not result in good choices for allocation because of network fragmentation (as the network is an important bottleneck in datacenters [2]). The slot-based

allocation, in turn, is constrained by the static number and size of slots in the servers, which limit the feasible choices for allocating tasks to servers. In contrast to both proposals, Packer employs our novel algorithm described in § II-B and better explores the trade-off between using local computing resources (CPU, memory and disk I/O) and remote distributed resources (the network).

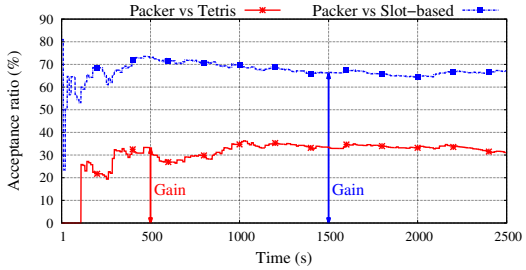


Fig. 3: Acceptance ratio of application tasks.

Maximized resource utilization. Figure 4 depicts the percentage difference between Packer and Tetris and Packer and slot-based allocation for the utilization of different types of resources. Positive values indicate that Packer achieves better utilization than the respective proposal being compared (*the higher the value, the better*), while negative values denote that the proposal being compared achieves better results than Packer. Figures 4(a), 4(b), 4(c) and 4(d) show results for CPU, memory, disk I/O write and disk I/O read, respectively. We see that, during most of the time, Packer allows better use of available resources, improving utilization by a significant percentage. Nonetheless, Packer has lower utilization of some types of resources at some periods of time (negative values in the plots). This happens because, with different allocation strategies, different applications are accepted and rejected in each scheme. Therefore, despite allocating significantly more application tasks than Tetris and slot-based (Figure 3), there are some small periods of time when the tasks allocated in Packer consume less resources.

Figure 4(e), in turn, shows the percentage difference of bandwidth utilization. We verify that Packer can maintain significantly higher utilization of the network than Tetris (≈76% more) and slot-based (≈87% more). Moreover, during the experiments, Packer never achieved lower network utilization than the proposals being compared. In general, Figures 3 and 4 show that Packer accepts more applications and, consequently, maximizes utilization, which indicates that fragmentation is minimized.

Increased provider revenue. We follow related work [2], [8] and adopt a simple pricing model to quantify provider revenue for Packer, Tetris and slot-based allocation, which effectively charges both computation and networking. A tenant running application a pays: $\sum_{t \in T^a} \sum_{v \in V^a} (\sum_{\mathtt{r}} \delta(v, \mathtt{r}, t) * k_v + \sum_{u \in \mathcal{E}(v)} \sigma((u,v),t) * k_b)$, where $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$, k_v is the unit-time computing resource cost and k_b is the unit-volume bandwidth cost. Figure 5 depicts the revenue of Packer in comparison to Tetris and slot-based allocation (error bars show 95% confidence interval). *Higher values are better*, as they mean that Packer provides more revenue than the respective proposal being compared. We see that, by improving the allocation ratio of application tasks (Figure 3) and resource utilization (Figure 4), Packer can significantly increase provider revenue (≈29% and ≈60% in comparison to Tetris and slot-based, respectively).

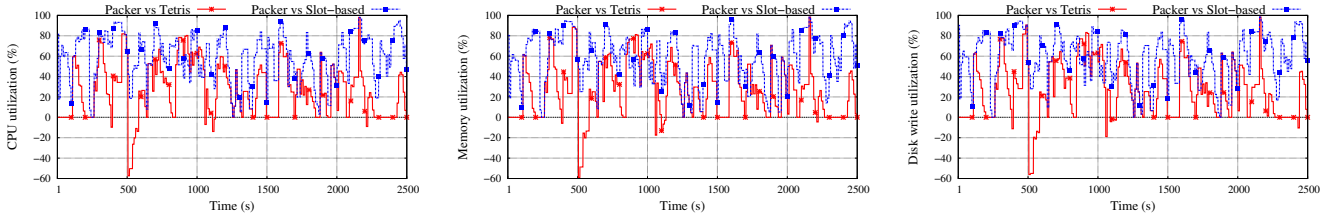
Acceptable overhead. Figure 6 quantifies the overhead introduced by Packer in comparison to Tetris and slot-based (error bars show 95% confidence interval). The overhead is given by the mean time taken to allocate an incoming application in the infrastructure. Here, positive values indicate that Packer takes more time to allocate applications than the respective proposal being compared, while negative values would indicate that Packer takes less time (i.e., *lower values are better*). We see that Packer takes more time to allocate applications than the other two proposals (≈53% more time than Tetris and ≈81% more than slot-based). This is justified by three factors (i) the complexity of the allocation metric (§ II-B); (ii) the fact that Packer considers the whole network (as opposed to Tetris that only penalizes network use); and (iii) the fact that Packer verifies each computing resource (CPU, memory and disk I/O) according to the applications' requirements (as opposed to slot-based that statically divides computing resources into slots). Nonetheless, while the percentage is high, the median time taken to allocate applications (observed in our experiments) is small for all three proposals: ≈15.4s in Packer, ≈3.5s in Tetris and ≈0.3s in slot-based allocation. Thus, considering the benefits provided by Packer (shown in Figures 3, 4 and 5), it is acceptable to take some additional seconds to allocate applications.

Now, we turn our focus to the challenge of performance interference. In particular, we show that Packer provides (i) minimum bandwidth guarantees for applications; and (ii) work-conserving sharing, achieving both predictability for tenants and high utilization for providers. To show the results in a clear way, here we consider the requested temporal bandwidth guarantees of application tasks (σ) as a constant function (while the actual requirements vary over time).

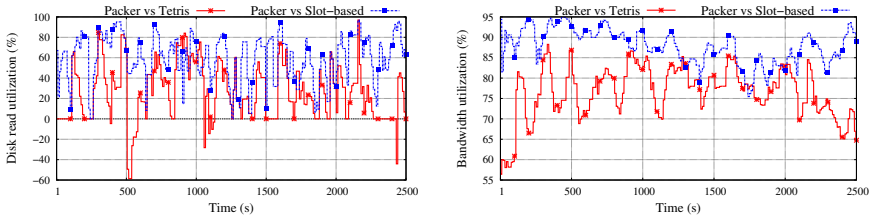
Minimum bandwidth guarantees for applications. Packer adopts the following definition of minimum bandwidth guarantees: the task rate should be (a) at least the guaranteed rate if the demands are equal or higher than the guarantees; or (b) equal to the demands if they are lower than the guarantees. To illustrate it, we show, in Figure 7, a task allocated on a given server during a predefined time period of an experiment. We see that the task may not get the desired rate to satisfy all of its demands instantaneously (when its demands exceed its guarantees) because (i) the link capacity is limited; and (ii) available bandwidth is proportionally shared among tasks. In summary, we verify that Packer provides minimum bandwidth guarantees for tasks, since the actual rate is always equal or higher than the minimum between the demands and the guarantees. Therefore, applications have minimum bandwidth guarantees and, thus, can achieve predictable network performance.

Work-conserving sharing. Work-conservation is the ability to use more bandwidth if the task has higher demands than its guarantees and there is available bandwidth in the network. In other words, bandwidth which is not allocated, or allocated but not currently used, should be proportionally shared among tasks with more demands than their guarantees (according to the weights of each application – w^a, using Algorithm 2). Figure 8 shows the aggregate bandwidth[1] on the server holding

[1]Note that Packer considers the temporal bandwidth guarantees requested (σ) when allocating tasks. Therefore, although the sum of the actual demands of all tasks allocated on a given server may exceed the server link capacity, the sum of bandwidth guarantees of these tasks will not exceed the link capacity.



(a) CPU. (b) Memory. (c) Disk I/O write.



(d) Disk I/O read. (e) Bandwidth.

Fig. 4: Resource utilization (positive values in the y-axis indicate that Packer achieves better utilization than the respective proposal being compared, while negative values denote that the proposal being compared achieves better utilization than Packer).

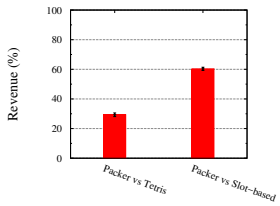


Fig. 5: Revenue.

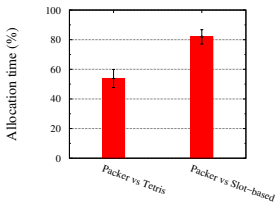


Fig. 6: Allocation time.

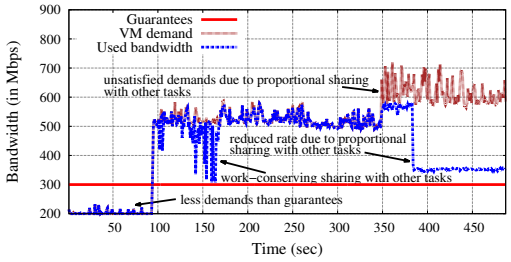


Fig. 7: Bandwidth allocation for a task on a given server.

the task in Figure 7. In these two figures, we verify that Packer provides work-conserving sharing in the network, as tasks can receive more bandwidth (if their demands are higher than their guarantees) when there is spare bandwidth. Thus, providers can achieve high utilization.

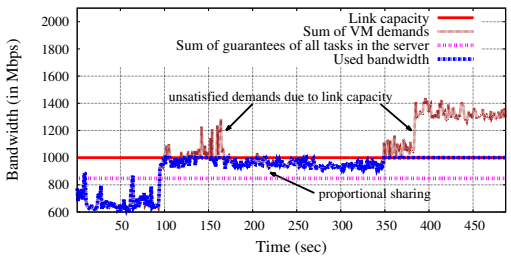


Fig. 8: Work-conserving sharing on a given server.

IV. Discussion

We discuss here some questions that have arisen during the design of Packer.

TI-MRA pros and cons. This abstraction considers temporal demands of resources. There are advantages and drawbacks of adopting this approach. The main advantage is to minimize multi-resource underutilization, which significantly improves datacenter utilization (i.e., reduces wastage). The main drawback is the need for fine-grained specification of applications, which may be a burden on some tenants (the ones with less knowledge of their applications). Hence, TI-MRA also allows the specification of only peak demands for multiple resources, minimizing the burden of application specification. While this may reduce infrastructure utilization, it does not impact provider revenue, as tenants are allocating such resources and paying for them.

Profiling application demands for specifying TI-MRA. Datacenter application demands are often known [23] or can be obtained from tenants [4], [8]. Alternatively, we employ the techniques described by Grandl et al. [6], Chen and Shen [7] and Lee et al. [9]. First, according to Chen and Shen [7], the same task (i.e., the same program with the same options) running on different servers tends to have similar resource utilization patterns. In this context, recurring applications are common in datacenters [24]; for instance, analytic applications repeat hourly or daily to perform the same computation on new data [6]. Therefore, Packer can use statistics measured in prior runs of the same application. Second, according to Lee et al. [9], orchestration systems like OpenStack Heat and AWS CloudFormation could be used to generate abstract models. They use templates (provided as a library for tenants) that explicitly describe the structure of applications and their resource demands. In this sense, these systems could be extended with temporal multi-resource requirements to generate TI-MRAs. Third, Packer can use the pattern detection algorithm for resource demands developed by Chen and Shen [7]. The algorithm utilizes logs of resource usage recorded by the cloud datacenter from previous runs of the same application and, thereby, can estimate utilization patterns for the requested application. Fourth, in case none of the previous methods can

be used, we follow Grandl et al. [6] and over-estimate resource demands (by considering a constant temporal function). Note that over-estimation is better than under-estimation, as the former does not slow down applications. Furthermore, Packer's resource monitoring mechanism verifies idle resources and reports them to the DRM, so that they can be allocated to applications.

Employing existing abstractions in the literature for Packer. Packer could use existing abstractions for application specification, with the constraint that those abstractions take into account multiple types of resources. Adapting other abstractions for Packer could be performed with the development of a module that reads the specification and converts it to a TI-MRA, so that the use of other abstractions would be seamless to Packer's allocation process.

V. Related Work

Proposals related to Packer are divided in two categories, as follows.

Multi-resource allocation. On one hand, schemes such as [19], [20] allocate computing resources based on slots. However, this leads to wastage and fragmentation, as the amount of resources in each slot is statically defined. On the other hand, Tetris [6], Dominant Resource Fairness (DRF) [1] and the spatial/temporal strategy [7] propose to dynamically adjust the allocation according to resource demands. Nonetheless, they present the following drawbacks: Tetris may result in starvation (depending on the workload) and may not provide guarantees in oversubscribed networks; DRF may result in fragmentation [6], as it focuses solely on fairness; and the strategy in [7] considers only temporal demands of CPU and memory, but neglects the network. Packer, in contrast, minimizes fragmentation (unlike slot-based schemes and DRF) and provides bandwidth guarantees even in oversubscribed networks (unlike Tetris and the strategy in [7]).

Network performance in DCNs. There is an extensive body of literature that addresses network performance in DCNs. We focus on the most important proposals related to Packer. Oktopus [8], CloudMirror [9] and Silo [4] provide network guarantees for applications. However, they focus only on network resources and may result in underutilization, as they statically reserve resources for applications based on their peak bandwidth demands. Proteus, in turn, allocates applications according to their temporal network demands. Despite reducing network underutilization, it neither considers other types of resources nor provides work-conserving sharing among applications (i.e., it uses rigid network models for each allocated application). Unlike Oktopus, CloudMirror, Silo and Proteus, Packer considers multiple types of resources and provides work-conserving sharing.

VI. Final Remarks

In this paper, we introduced Packer, a scheme that addresses the challenges of multi-resource allocation and performance interference in the network. It employs a novel abstraction called Time-Interleaved Multi-Resource Abstraction (TI-MRA) and a new algorithm for allocating multiple types of resources with reduced fragmentation. Furthermore, Packer uses (a) SDN to dynamically configure and manage the network according to available resources and requirements of applications; and (b) a monitoring mechanism to avoid wastage and congested resources. Evaluation results show that (i) acceptance ratio of applications is increased; (ii) datacenter utilization is maximized (i.e., fragmentation is minimized); (iii) provider revenue is augmented; and (iv) applications achieve predictable and guaranteed network performance with work-conserving sharing. In future work, we intend to develop a prototype and evaluate Packer in a testbed (e.g., CloudLab [25]).

References

[1] A. Ghodsi et al., "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in *USENIX NSDI*, 2011.

[2] D. Xie et al., "The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers," in *ACM SIGCOMM*, 2012.

[3] R. Panigrahy et al., "Heuristics for Vector Bin Packing," Tech. Rep., 2011, available at : http://goo.gl/9BDXIL.

[4] K. Jang et al., "Silo: Predictable Message Latency in the Cloud," in *ACM SIGCOMM*, 2015.

[5] M. Chowdhury et al., "HUG: Multi-Resource Fairness for Correlated and Elastic Demands," in *USENIX NSDI*, 2016.

[6] R. Grandl et al., "Multi-resource Packing for Cluster Schedulers," in *ACM SIGCOMM*, 2014.

[7] L. Chen and H. Shen, "Consolidating Complementary VMs with Spatial/Temporal-awareness in Cloud Datacenters," in *IEEE INFOCOM*, 2014.

[8] H. Ballani et al., "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.

[9] J. Lee et al., "Application-driven bandwidth guarantees in datacenters," in *ACM SIGCOMM*, 2014.

[10] H. Ballani et al., "Chatty tenants and the cloud network sharing problem," in *USENIX NSDI*, 2013.

[11] Y. Jarraya et al., "A Survey and a Layered Taxonomy of Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. PP, no. 99, pp. 1–29, 2014.

[12] K. Wehmuth et al., "A unifying model for representing time-varying graphs," 2015.

[13] K. Ousterhout et al., "Making Sense of Performance in Data Analytics Frameworks," in *USENIX NSDI*, 2015.

[14] G. J. Woeginger, "There is no asymptotic PTAS for two-dimensional vector packing," *Information Processing Letters*, vol. 64, no. 6, pp. 293–297, 1997.

[15] D. Abts and B. Felderman, "A guided tour of data-center networking," *Commun. ACM*, vol. 55, no. 6, pp. 44–51, Jun. 2012.

[16] A. R. Curtis et al., "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *IEEE INFOCOM*, 2011.

[17] T. Benson et al., "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010.

[18] D. S. Marcon and M. P. Barcellos, "Predictor: providing fine-grained management and predictability in multi-tenant datacenter networks," in *IFIP/IEEE IM*, 2015.

[19] "Hadoop Scheduler," The Apache Software Foundation, 2014, available at : http://bit.ly/1tGpbDN, http://bit.ly/1tGpbDN.

[20] "Hadoop YARN Project," The Apache Software Foundation, 2015, available at : http://bit.ly/1iS8xvP.

[21] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *USENIX NSDI*, 2011.

[22] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM IMC*. New York, NY, USA: ACM, 2009.

[23] M. P. Grosvenor et al., "Queues Don't Matter When You Can JUMP Them!" in *USENIX NSDI*, 2015.

[24] S. Agarwal et al., "Re-optimizing Data-parallel Computing," in *USENIX NSDI*, 2012.

[25] "CloudLab," 2016, available at : https://www.cloudlab.us.

APPENDIX F SUBMITTED PAPER TO ELSEVIER COMPUTER NETWORKS

Performance interference has been a well-known problem in datacenters and one that remains a constant topic of discussion in the literature. Software-Defined Networking (SDN) may enable the development of a robust solution for interference, as it allows dynamic control over resources through programmable interfaces and flow-based management. However, to date, the scalability of existing SDN-based approaches is limited, because of the number of entries required in flow tables and delays introduced. In this paper, we propose Predictor, a scheme to scalably address performance interference in SDN-based datacenter networks (DCNs), providing minimum bandwidth guarantees for applications and work-conservation for providers. Two novel SDN-based algorithms are proposed to address performance interference. Scalability is improved in Predictor as follows: first, it minimizes flow table size by controlling flows at *application-level*; second, it reduces flow setup time by proactively installing rules in switches. We conducted an extensive evaluation, in which we verify that Predictor provides (i) guaranteed and predictable network performance for applications and their tenants; (ii) work-conserving sharing for providers; and (iii) significant improvements over DevoFlow (the state-of-the-art SDN-based proposal for DCNs), reducing flow table size (up to 94%) and having similar controller load and flow setup time.

- **Title**: Achieving Minimum Bandwidth Guarantees and Work-Conservation in Large-Scale, SDN-Based Datacenter Networks
- **Journal**: Elsevier Computer Networks
- **Qualis**: A1
- **Publisher**: Elsevier Science Publishers B. V.
- **Address**: Amsterdam, The Netherlands

Achieving Minimum Bandwidth Guarantees and Work-Conservation in Large-Scale, SDN-Based Datacenter Networks

Daniel S. Marcon, Fabrício M. Mazzola, Marinho P. Barcellos

Institute of Informatics – Federal University of Rio Grande do Sul
Av. Bento Gonçalves, 9500 – 91.501-970 – Porto Alegre, RS, Brazil – Postal Code 15 064
Email: {daniel.stefani, fmmazzola, marinho}@inf.ufrgs.br

Abstract

Performance interference has been a well-known problem in datacenters and one that remains a constant topic of discussion in the literature. Software-Defined Networking (SDN) may enable the development of a robust solution for interference, as it allows dynamic control over resources through programmable interfaces and flow-based management. However, to date, the scalability of existing SDN-based approaches is limited, because of the number of entries required in flow tables and delays introduced. In this paper, we propose Predictor, a scheme to scalably address performance interference in SDN-based datacenter networks (DCNs), providing minimum bandwidth guarantees for applications and work-conservation for providers. Two novel SDN-based algorithms are proposed to address performance interference. Scalability is improved in Predictor as follows: first, it minimizes flow table size by controlling flows at *application-level*; second, it reduces flow setup time by proactively installing rules in switches. We conducted an extensive evaluation, in which we verify that Predictor provides (*i*) guaranteed and predictable network performance for applications and their tenants; (*ii*) work-conserving sharing for providers; and (*iii*) significant improvements over DevoFlow (the state-of-the-art SDN-based proposal for DCNs), reducing flow table size (up to 94%) and having similar controller load and flow setup time.

Keywords:
Datacenter networks, Software-Defined Networking, Network sharing, Performance interference, Bandwidth guarantees, Work-conservation

1. Introduction

We study how to address the challenge of performance interference [1, 2] in large-scale, SDN-based datacenter networks (DCNs). Specifically, interference remains a constant topic of discussion in the literature [3, 4, 5, 6, 7, 8, 9, 10]. In this context, we aim at achieving minimum bandwidth guarantees for applications and their tenants while maintaining high utilization (i.e., providing work-conserving capabilities) in large DCNs.

Software-Defined Networking (SDN) [11] may enable the development of a robust solution to deal with performance interference, as it allows dynamic control over resources through programmable interfaces and flow-based management [12]. However, to date, the scalability of existing SDN-based approaches is limited, because of the number of entries required in flow tables and delays introduced (mostly related to flow setup time) [12, 13, 14]. The number of entries required in flow tables can be significantly higher than the amount of resources available in commodity switches used in DCNs [13, 15], as such networks typically have very large flow rates (e.g., over 16 million/s [16]). Flow setup time, in turn, is associated with the transition between the data and control planes whenever a new flow arrives at a switch[1] (latency for communication between switches and the controller), and the high frequency at which flows arrive and demands change in DCNs restricts controller scalability [17]. As a result, the lack of scalability hinders the use of SDN to address interference in large DCNs.

The scalability of SDN-based datacenters could be improved by devolving the control to the data plane, such as proposed by DevoFlow [13] and Difane [18], but deployability is limited since they require switches with customized hardware. Another approach would be using a logically distributed controller, such as proposed by Kandoo [19]. However, it does not scale for large DCNs where communications occur between virtual machines (VMs) connected by different top-of-rack (ToR) switches. This happens because the distributed set of controllers needs to maintain synchronized information (strong consistency) for the whole network. This is necessary in order to route traffic through less congested paths and to reserve resources for applications.

We rely on two key observations to address performance interference and scalability of SDN in DCNs: (*i*) providers do not need to control each flow individually, since they charge tenants based on the amount of resources consumed by applica-

[1] We use the terms "switches" and "forwarding devices" to refer to the same set of SDN-enabled network devices, that is, data plane devices that forward packets based on a set of flow rules.

tions[2]; and (*ii*) congestion control in the network is expected to be proportional to the tenant's payment [3, 20]. Therefore, we adopt a broader definition of flow, considering it at application-level[3], and introduce Predictor, a scheme for large-scale datacenters. Predictor deals with the two aforementioned challenges (namely, performance interference and scalability of SDN/OpenFlow in DCNs) in the following manner.

Performance interference is addressed by employing two SDN-based algorithms (described in Section 5) to dynamically program the network, improving resource sharing. By doing so, both tenants and providers have benefits. Tenants achieve predictable network performance by receiving minimum bandwidth guarantees for their applications (using Algorithm 1). Providers, in turn, maintain high network utilization (due to work-conservation provided by Algorithm 2), essential to achieve economies of scale.

Scalability is improved in two ways. First, as we show through measurements (Section 2), reducing flow table size also decreases the time taken to install rules in flow tables (stored in Ternary Content-Addressable Memory – TCAM) of switches. In the proposed approach, flow table size is minimized by managing flows at application-level and by using wildcards (when possible). This setting allows providers to control traffic and gather statistics at application-level for each link and device in the network.

Second, we propose to proactively install rules for intra-application communication, guaranteeing bandwidth between VMs of the same application. By installing rules at application allocation time, flow setup time is reduced. Inter-application rules, in turn, may be either proactively installed in switches (if tenants know other applications that their applications will communicate with [5] or if the provider employs some predictive technique [21, 22]) or reactively installed according to demands. Proactively installing rules has both benefits and drawbacks: while flow setup time is eliminated, some flow table entries may take longer to expire (they might be removed only when their respective applications conclude and are deallocated). Our decision is motivated by the fact that intra-application traffic volume is expected to be the highest type of traffic [20].

Contributions. In comparison to our previous work [23], in this paper we present a substantially improved version of Predictor, in terms of both efficiency and resource usage. We highlight five main contributions. First, we run experiments to motivate Predictor and show that the operation of inserting rules at the TCAM takes more time and is more variable according to flow table occupancy. Thereby, the lower the number of rules in TCAMs, the better. Second, we extend Predictor to proactively provide inter-application communication guarantees (rather than only reactively providing it), which can further reduce flow setup time. Third, we develop improved versions of the allocation and work-conserving rate enforcement algorithms to provide better utilization of available resources (without adding significant complexity to the algorithms). Fourth, we address the design of the control plane for Predictor, as it is an essential part of a software-defined network to provide efficient and dynamic control of resources. Fifth, we conduct a more extensive evaluation, comparing Predictor with different modes of operation of DevoFlow [13] and considering several factors to analyze its benefits, overheads and technical feasibility. Predictor reduces flow table size up to 94%, offers low average flow setup time and presents low controller load, while providing minimum bandwidth guarantees for tenants and work-conserving sharing for providers.

The remainder of this paper is organized as follows. Section 2 examines the challenges of performance interference and scalability of SDN in DCNs. Section 3 provides an overview of Predictor, while Section 4 details application requests. Sections 5 and 6 describe, respectively, the mechanisms employed for resource sharing and the control plane design. Section 7 presents the evaluation, and Section 8 discusses generality and limitations. Finally, Section 9 examines related work and Section 10 concludes the paper with final remarks and perspectives for future work.

2. Motivation and Research Challenges

In this section, we review performance interference (Section 2.1) and discuss the challenges of using SDN in large-scale DCNs to build a solution for interference (Section 2.2). In the context of SDN, we adopt an OpenFlow [24] view, since it is the most accepted SDN implementation by Academia and Industry. Through OpenFlow switch measurements, we quantify flow setup time and show it can hinder scalability of SDN as a solution to the performance interference problem.

2.1. Datacenter Network Sharing

Several recent measurement studies [2, 3, 5, 25, 26, 27, 28] concluded that, due to perfomance interference, the network throughput achieved by VMs can vary by a factor of five or more. For instance, Grosvenor et al. [5] show that variability can worsen tail performance by 50× for clock synchronization (PTPd) and 85× for key-value stores (Memcached). As the computation typically depends on the data received from the network [21] and the network is agnostic to application-level requirements [9], such variability often results in poor and unpredictable application performance [29]. In this situation, tenants end up spending more money.

Performance variability is usually associated with two factors: type of traffic and congestion control. The type of traffic in DCNs is remarkably different from other networks [30]. Furthermore, the heterogeneous set of applications generates flows that are sensitive to either latency or throughput [31]; throughput-intensive flows are larger, creating contention in some links, which results in packets from latency-sensitive flows being discarded (adding significant latency) [32, 33]. TCP congestion control (used in such networks), in turn, cannot ensure performance isolation among applications [34]; it only guarantees fairness among flows. Judd and Stanley [25] show

[2] Without loss of generality, we assume one application per tenant.

[3] An application is represented by a set of VMs that consume computing and network resources (see Section 4 for more details).

through measurements that many TCP design assumptions do not hold in datacenter networks, leading to inadequate performance. While TCP can provide high utilization, it does so very inefficiently. They conclude that the overall median throughput of the network is low and that there is a large variation among flow throughput.

Popa et al. [35] examines two main requirements for network sharing: (*i*) bandwidth guarantees for tenants and their applications; and (*ii*) work-conserving sharing to achieve high network utilization for providers. In particular, these two requirements present a trade-off: strict bandwidth guarantees may reduce utilization, since applications have variable network demands over time [21]; and a work-conserving approach means that, if there is residual bandwidth, applications should use it as needed (even if the available bandwidth belongs to the guarantees of another application) [1].

In this context, SDN/OpenFlow can enable dynamic, fine-grained network management in order to develop a robust strategy to explore this trade-off and achieve predictable network performance with bandwidth guarantees and work-conserving sharing.

2.2. *Scalability Challenges of SDN/OpenFlow in DCNs*

SDN-based networks involve the control plane more frequently than traditional networking [13]. In the context of large-scale DCNs, this aspect leads to two scalability issues: flow setup time (the time taken to install new flow rules in forwarding devices) and flow table size in switches.

Flow setup time. It may add impractical delay for flows, especially for latency-sensitive ones [3] (as adding even 1 ms of latency to these flows is intolerable [36]). As SDN relies on the communication between network devices (data plane) and a logically centralized controller (control plane), it increases (*i*) control plane load and (*ii*) latency (sources for augmented delay). Control plane load is increased because a typical ToR switch will have to request rules to the controller for approximately more than 1,500 new flows per second [37] and the controller is expected to process and reply to all requests in, at most, a few milliseconds. Consequently, this may end up making both the communication with the controller and the controller itself bottlenecks. Latency is augmented because new flows are delayed at least two RTTs (i.e., communication between the ASIC and the management CPU and between that CPU and the controller) [13], so that the controller can install the appropriate rules at forwarding devices.

Experiments to measure flow setup time. We evaluated the time taken to perform the operation of inserting rules at a switch's TCAM. Our measurement setup (shown in Figure 1) consists of one host with three 1 Gbps interfaces connected to an OpenFlow switch (Centec v350): eth0 interface is connected to the control port and eth1 and eth2 are connected to data ports on the switch. The switch uses OpenFlow 1.3 and has a TCAM that stores at most 2,000 rules. The host runs OpenFlow controller Ryu, which listens for control packets on eth0.

The experiment works as follows. The switch begins with a given number of rules installed in the TCAM (which represents its table occupancy). The host runs a packet generator to

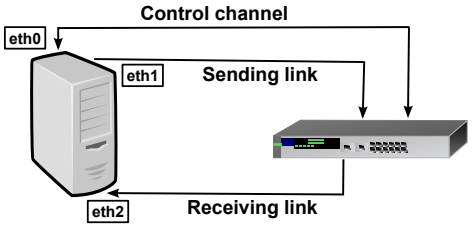


Figure 1: Measurement setup.

send a single UDP flow on eth1. This flow generates a table-miss event in the switch (i.e., the switch does not have an appropriate rule to handle the flow sent by the packet generator). Consequently, the switch sends a packet_in message to the controller. Upon receiving the packet_in, the controller processes the request and sends back a flow_mod message with the appropriate rule to be installed in the switch TCAM to handle the flow. Once the switch installs the rule, it forwards the matching packets to the link connected on the host eth2 interface. Like He et al. [38], the latency of the operation is calculated as follows: (*i*) timestamp1 is recorded when the controller sends the flow_mod to the switch on eth0; and (*ii*) timestamp2 is recorded when the first packet of the flow arrives on the host eth2 interface. Since the round-trip time (RTT) between the switch and host is negligible in our experiments[4], the latency is calculated by subtracting timestamp1 from timestamp2.

Figure 2 shows the latency of inserting new rules at the TCAM (y-axis) according to the number of rules already installed in the table (x-axis). The experiment was repeated 10 times; each point in the plot represents one measured value of one repetition and the line depicts the median value. Results show that median latency and variability increase according to flow table occupancy. These results are in line with previous measurements in the literature, such as He et al. [38]. Since adding even 1ms may be intolerable for some applications (e.g., latency-sensitive ones) [36], reduced flow table occupancy is highly desirable in DCNs because of flow setup time.

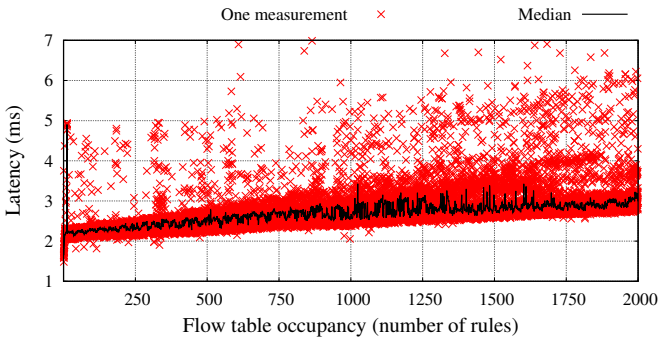


Figure 2: Latency of inserting new rules according to flow table occupancy.

[4]While the RTT is negligible in our experiments (as the switch is directly connected to the controller), it may not be the case in large-scale datacenters with hundreds of switches.

Flow table size. Flow tables are a restricted resource in commodity switches, as TCAMs are typically expensive and power-hungry [14, 15, 39]. Such devices usually have a limited number of entries available for OpenFlow rules, which may not be enough when considering that large-scale datacenter networks have an elevated number of active flows per second [16].

Therefore, the design of Predictor takes both flow setup time and flow table size in switches into account. More specifically, Predictor proactively installs rules for intra-application communication at allocation time (thereby eliminating the latency of flow setup for most traffic in DCNs) and considers flows at *application-level* (reducing the number of flow table entries and, consequently, the time taken to install new rules in forwarding devices). We detail Predictor and justify the decisions in the next sections.

3. Predictor Overview

We first present an overview of Predictor, including its components and the interactions between them in order to address the challenge of performance interference in large-scale, SDN-based DCNs.

Predictor is designed taking four requirements into consideration: (*i*) scalability, (*ii*) resiliency, (*iii*) predictable and guaranteed network performance, and (*iv*) high network utilization. First, any design for network sharing must scale to hundreds of thousands of VMs and deal with heterogeneous workloads of applications (typically with bursty traffic). Second, it needs to be resilient to churn both at flow-level (because of the rate of new flows/s [16]) and at application-level (given the rates of application allocation/deallocation observed in datacenters [1]). Third, it needs to provide predictable and guaranteed network performance, allowing applications to maintain a base-level of performance even when the network is congested. Finally, any design should achieve high network utilization, so that spare bandwidth can be used by applications with more demands than their guarantees.

Predictor is designed to fulfill the above requirements. While providers can reduce operational costs and achieve economies of scale, tenants can run their applications predictably (possibly faster, reducing costs). Figure 3 shows an overview of Predictor, which is composed of five components: Predictor controller, allocation module, application information base (AIB), network information base (NIB) and OpenFlow controller. They are discussed next.

Predictor Controller. It receives requests from tenants. A request can be either an application to be allocated (whose resources to be used are determined by the allocation module) or a solicitation for inter-application bandwidth guarantees (detailed in Sections 4 and 5). In case of an incoming application, it sends the request to the allocation module. Once the allocation is completed (or if the request is for inter-application communication), the Predictor controller generates and sends appropriate flow rules to the OpenFlow controller. The OpenFlow controller, then, updates the tables (of forwarding devices) that need to be modified.

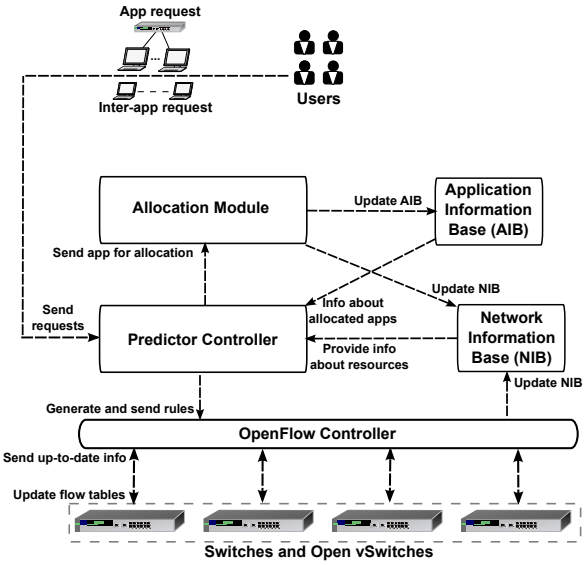


Figure 3: Predictor overview.

Note that the controller installs rules to identify flows at application-level[5] (more details in Sections 7 and 8). Predictor can also take advantage of flow management at lower levels (for instance, by matching source and destination MAC and IP fields), since it uses the OpenFlow protocol. Nonetheless, given the amount of resources available in commodity switches and the number of flows that come and go in a small period of time, low-level rules are kept to a minimum.

Allocation Module. This component is responsible for allocating incoming applications in the datacenter infrastructure, according to available resources. It receives requests from the Predictor controller, determines the set of resources to be allocated for each new request and updates the AIB and NIB. We detail the allocation logic in Section 5.2.

Application Information Base (AIB). It keeps detailed information regarding each allocated application, including its identifier (ID), VM-to-server mapping, IP addresses, bandwidth guarantees, network weight (for work-conserving sharing), links being used and other applications it communicates with. It provides information for the Predictor controller to compute flow rules that need to be installed in switches.

Network Information Base (NIB). It is composed of a database of resources, including hosts, switches, links and their capabilities (such as link capacity and latency). In general, it keeps information about computing and network state, received from the OpenFlow controller (current state) and the allocation module (resources used for newly allocated applications). The Predictor controller uses information stored in the NIB to map logical actions (e.g., intra- or inter-application communication) into the physical network. While the AIB maintains informa-

[5]The granularity of rules (at application-level) hinders neither network controllability for providers nor network sharing among tenants and their applications because, as discussed in Section 1, (*a*) providers charge tenants based on the amount of resources consumed by applications; and (*b*) congestion control in the network is expected to be performed at application-level [20].

tion at application granularity, the NIB keeps information at network layer. The design of the NIB was inspired by Onix [11] and PANE [40].

OpenFlow Controller. It is responsible for communication to/from forwarding devices and Open vSwitches [41] in hypervisors, in order to update network state and get information from the network (e.g., congested links and failed resources). It receives information from the Predictor controller to modify flow tables in forwarding devices and updates the NIB upon getting information from the network.

We explain Predictor in detail by describing application requests (Section 4), the mechanisms employed for resource sharing (Section 5) and the control plane design (Section 6).

4. Application Requests

Tenants request applications using the hose model (similarly to past proposals [2, 3, 17, 20, 21]) to capture the semantics of guarantees being offered, as shown in Figure 4. In this model, all VMs of an application are connected to a non-blocking virtual switch through dedicated bidirectional links. Each application a is represented by its resource demands and network weight, or more formally, $\langle N_a, B_a, w_a, comm_a^{inter} \rangle$. Its terms are: $N_a \in \mathbb{N}^*$ specifies the number of VMs; $B_a \in \mathbb{R}^+$ represents the bandwidth guarantees required by each VM; $w_a \in [0, 1]$ indicates the network weight; and $comm_a^{inter}$ is an optional field that contains information about inter-application communication for application a.

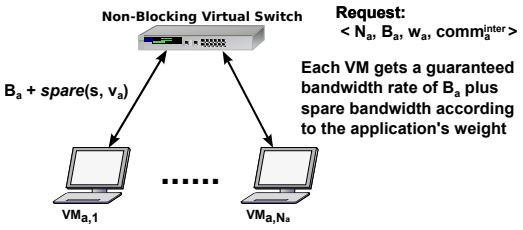


Figure 4: Virtual network topology of a given application.

The network weight enables residual bandwidth (unallocated or reserved bandwidth for an application and not currently being used) to be proportionally shared among applications with more demands than their guarantees (work-conservation). Therefore, the total amount of bandwidth available for each VM of application a at a given period of time, following the hose model, is denoted by $B_a + spare(s, v_a)$, where $spare(s, v_a)$ identifies the share of spare bandwidth assigned to VM v of application a located at server s:

$$\texttt{spare}(s, v_a) = \frac{w_a}{\sum_{v \uparrow V_s | v \in V_s} w_v} * \texttt{SpareCapacity} \qquad (1)$$

where V_s denotes the set of all co-resident VMs (i.e., VMs placed at server s), $v \uparrow V_s \mid v \in V_s$ represents the subset of VMs at server s that need to use more bandwidth than their guarantees and `SpareCapacity` indicates the residual capacity of the link that connects server s to the ToR switch.

The term $comm_a^{inter}$ is optional and allows tenants to proactively request guarantees for inter-application communication (since it would be infeasible to provide all-to-all communication between VMs in the datacenter [20]). It is a set composed of elements in the form of $\langle \texttt{srcVM}_a, \texttt{dstVMs}, \texttt{begTime}, \texttt{endTime}, \texttt{reqRate} \rangle$, where: $\texttt{srcVM}_a$ denotes the source VM of application a; `dstVMs` is the set of destination VMs (i.e., unicast or multicast communication from the source VM to each destination VM); `begTime` and `endTime` represent, respectively, the time that the communication starts and ends; and `reqRate` indicates the total amount of bandwidth per second needed by flows belonging to traffic from this (these) communication(s).

By providing optional specification of inter-application communication, Predictor allows requests from tenants with and without knowledge of application communication patterns and desired resources. Application traffic patterns are often known [5, 9] or can be estimated by employing the techniques described by Lee et al. [7], Xie et al. [21] and LaCurts et al. [22]. Note that, even if tenants do not proactively request resources for communication with other applications/services (i.e., they do not use $comm_a^{inter}$), their applications will still be allowed to reactively receive guarantees for communication with others (as detailed in Section 5.1).

In line with past proposals [2, 3, 20, 21], two assumptions are made. First, we abstract away non-network resources and consider all VMs with the same amount of CPU, memory and storage. Second, we consider that all VMs of a given application receive the same bandwidth guarantees (B_a).

5. Resource Sharing

In this section, we discuss how resources are shared among applications. In particular, we first examine how bandwidth guarantees are provided. Then, we take a look at the process of resource allocation and, finally, we present the logic behind the work-conserving mechanism employed by Predictor.

5.1. Bandwidth Guarantees

Predictor provides bandwidth guarantees for both intra- and inter-application communication. We discuss each one next.

Intra-application network guarantees. Typically, this type of communication represents most of the traffic in DCNs [20]. Thus, Predictor allocates and ensures bandwidth guarantees at application allocation time[6] by proactively installing flow rules and rate-limiters in the network through OpenFlow.

Each VM of a given application a is assigned a bidirectional rate of B_a (as detailed in Section 4). Limiting the communication between VMs located in the same server or in the same rack is straightforward, since it can be done locally by the Open vSwitch at each hypervisor.

[6]While Predictor may overprovision bandwidth at the moment applications are allocated, it does not waste bandwidth because of its work-conserving strategy (explained in Section 5.3). Without overprovisioning bandwidth at first, it would not be feasible to provide bandwidth guarantees for applications (as DCNs are typically oversubscribed).

In contrast, for inter-rack communication, bandwidth must be guaranteed throughout the network, along the path used for such communication. Predictor provides guarantees for this traffic by employing the concept of *VM clusters*[7]. To illustrate this concept, Figure 5 shows a simplified scenario where a given application a has four clusters: $c_{a,1}$, $c_{a,2}$, $c_{a,3}$ and $c_{a,4}$. Since each VM of a cannot send or receive data at a rate higher than B_a, traffic between a pair of clusters $c_{a,x}$ and $c_{a,y}$ is limited by the smallest cluster: $rate_{c_{a,x},c_{a,y}} = \min(|c_{a,x}|, |c_{a,y}|) * B_a$, where $rate_{c_{a,x},c_{a,y}}$ represents the calculated bandwidth for communication between clusters $c_{a,x}$ and $c_{a,y}$ (for $x, y \in \{1, 2, 3, 4\}$ and $x \neq y$), and $|c_{a,i}|$ denotes the number of VMs in cluster i of application a. In this case, $rate_{c_{a,x},c_{a,y}}$ is guaranteed along the path used for communication between these two clusters by rules and rate-limiters configured in forwarding devices through OpenFlow.

We apply this strategy at each level up the topology (reserving the minimum rate required for the communication among clusters). In general, the bandwidth required by one VM cluster to communicate with all other clusters of the same application is given by the following expression:

$$rate_{c_{a,x}} = \min\left(|c_{a,x}| * B_a, \sum_{c \in C_a, c \neq c_{a,x}} |c| * B_a\right) \qquad \forall c_{a,x} \in C_a \quad (2)$$

where $rate_{c_{a,x}}$ denotes the bandwidth required by cluster x to communicate with other clusters associated with application a and C_a indicates the set of all clusters of application a.

Inter-application communication. Applications in datacenters may exhibit complex communication patterns. However, providing them with static hose guarantees does not scale for DCNs [20], since bandwidth guarantees would have to be enforced between all pairs of VMs. Furthermore, tenants may not know in advance all applications/services that their applications will communicate with.

Predictor can dynamically set up guarantees for inter-application communication according to the needs of applications and residual bandwidth in the network. In case guarantees were not requested using the field $comm_a^{inter}$ (as described in Section 4), the Predictor controller provides two ways of establishing guarantees for communication between VMs of distinct applications and services, as follows.

Reacting to new flows in the network. When a VM needs to exchange data with one or more VMs of another application, it can simply send packets to those VMs. The hypervisor (through its Open vSwitch) of the server hosting the source VM receives such packets and, since they do not match any rule, sends them to the controller. The Predictor controller, then, determines the rules needed by the new flows and installs the set of rules along the appropriate path(s) in the network.

Receiving communication requests from applications. Prior to initiating the communication with VMs belonging to other applications, the source VM can send a request to the Predictor controller for communication with VMs from other application(s). This request is composed of the set of destination VMs, the bandwidth needed and the expected amount of time the communication will last. Upon receiving the request, the Predictor controller verifies residual resources in the network, sends a reply and, in case there are enough available resources, generates and installs the appropriate set of rules and rate-limiters for this communication. This approach is similar to providing an API for applications to request network resources, like PANE [40].

[7] A VM cluster is a set of VMs of the same application located in the same rack.

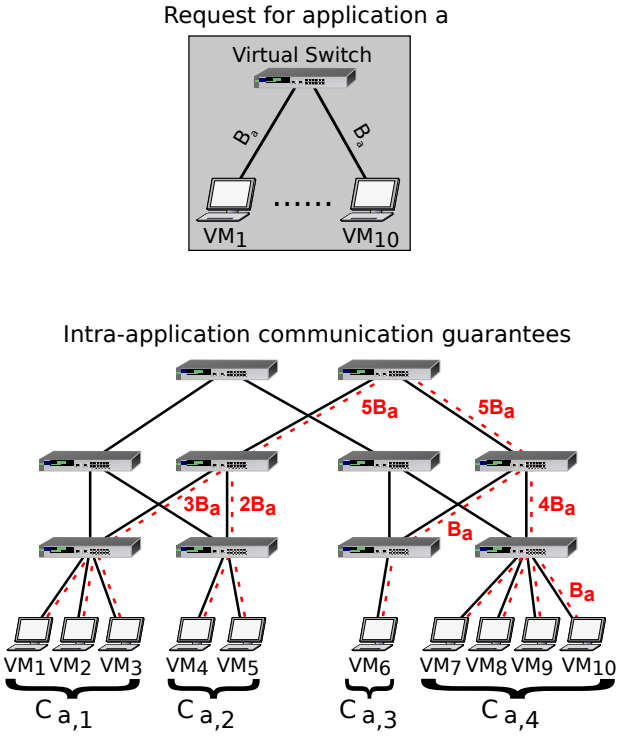


Figure 5: Example of intra-application bandwidth guarantees.

5.2. Resource Allocation

The allocation process is responsible for performing admission control and mapping application requests in the datacenter infrastructure. An allocation can only be made if there are enough computing and network resources available [42]. That is, VMs must only be mapped to servers with available resources, and there must be enough residual bandwidth for communication between VMs (as specified in the request). For simplicity, we follow related work [2, 3, 21] and discuss Predictor and its allocation component in the context of traditional tree-based topologies implemented in current datacenters.

We design a location-aware heuristic to efficiently allocate tenant applications in the infrastructure. The key principle is minimizing bandwidth for intra-application communication (thus allocating VMs of the same application as close as possible to each other), since this type of communication generates most of the traffic in the network (as discussed before) and DCNs typically have scarce resources [21].

Algorithm 1 allocates one application at a time, as requests are received. It receives as input the physical infrastructure P (composed of servers, racks, switches and links) and the incoming application request a (formally defined in Section 4 as $\langle N_a, B_a, w_a, comm_a^{inter}\rangle$), and works as follows. First, it searches for the best placement in the infrastructure for the incoming application via dynamic programming (lines 1 – 12). To this end, $N_s^P(l-1)$ represents the set of neighbors (directly connected switches) of switch s at level $l-1$. Furthermore, three data structures are defined and dynamically initialized for each request: (*i*) set R^a stores subgraphs with enough computing resources for application a; (*ii*) V_s^a stores the total number of VMs of application a the s-rooted subgraph can hold; and (*iii*) C_s^a stores the number of VM clusters that can be formed in subgraph s. The algorithm traverses the topology starting at rack level (level 1), up to the core, and determines subgraphs with enough available resources to allocate the incoming request.

Algorithm 1: Location-aware algorithm.

Input : Physical infrastructure P (composed of servers, racks, switches and links), Application a=$\langle N_a, B_a, w_a, comm_a^{inter}\rangle$
Output: Success/Failure code *allocated*

```
   // Search for the best placement in the infrastructure
1  R^a ← ∅;
2  foreach level l of P do
3      if l == 1 then // Top-of-Rack switches
4          foreach ToR r do
5              V_r^a ← num. available VMs in the rack;
6              C_r^a ← 1;
7              if V_r^a ≥ N^a then R^a ← R^a ∪ {r} ;
8      else // Aggregation and core switches
9          foreach Switch s at level l do
10             V_s^a ← Σ_{w∈{N_s^P(l−1)}} V_w^a;
11             C_s^a ← Σ_{w∈{N_s^P(l−1)}} C_w^a;
12             if V_s^a ≥ N^a then R^a ← R^a ∪ {s} ;
   // Proceed to the allocation
13 allocated ← failure code;
14 while Application a not allocated and R_a not empty do
15     r ← Select subgraph from R^a;
16     R^a ← R^a \ {r};
       // VM placement
17     Allocate VMs of application a at r;
       // Bandwidth allocation
18     foreach Level l from 0 to Height(r) do
19         Allocate bandwidth at l according to Section 5.1 and Equation 2;
20     foreach Inter-application communication c ∈ comm_a^inter do
21         Allocate bandwidth for inter-application communication c specified at
           allocation time (as defined in Section 4);
22     if Application was successfully allocated at r then
23         allocated ← success code
24     else allocated ← failure code;
25 return allocated;
```

After verifying the physical infrastructure and determining possible placements, the algorithm starts the allocation phase (lines 13 - 24). First, it selects one subgraph r at a time from the set R^a to allocate the application (line 15). The selection of a candidate subgraph takes into account the number of VM clusters. Therefore, the selected subgraph is the one with the minimum number of VM clusters, so that VMs of the same application are allocated close to each other, reducing the amount of bandwidth needed for communication between them (as the network often represents the bottleneck when compared to computing resources [43]).

When a subgraph is selected, the algorithm allocates the application with a coordinated node (VM-to-server, in line 17) and link (bandwidth reservation, in lines 18 – 21) mapping, similarly to the virtual network embedding problem [44]. In particular, bandwidth for intra-application communication (lines 18 – 19) is allocated through a bottom-up strategy, as follows. First, it is reserved at servers (level 0). Then, it is reserved, in order, for each subsequent level of the topology, according to the bandwidth needed by communication between VMs from distinct racks that belong to the same application (as explained in Section 5.1 and in Equation 2, and exemplified in Figure 5). After that, bandwidth for inter-application communication (that was specified at allocation time in field $comm_a^{inter}$) is allocated in lines 20 – 21 (recall that $comm_a^{inter}$ was defined in Section 4).

Finally, the algorithm returns a success code if application a was allocated or a failure code otherwise (line 25). Applications that could not be allocated upon arrival are discarded, similarly to Amazon EC2 [45].

5.3. Work-Conserving Rate Enforcement

Predictor provides bandwidth guarantees with work-conserving sharing. This is because only enforcing guarantees through static provisioning leads to underutilization and fragmentation [17], while offering work-conserving sharing only does not provide strict guarantees for tenants [20]. Therefore, in addition to ensuring a base-level of guaranteed rate, Predictor proportionally shares available bandwidth among applications with more demands than their guarantees, as defined in Equation 1.

We design an algorithm to periodically set the allowed rate for each co-resident VM on a server. In order to provide smooth interaction with TCP, we follow ElasticSwitch [17] and execute the work-conserving algorithm between periods of time one order of magnitude larger than the network round-trip time (RTT), e.g., 10 ms instead of 1 ms.

Algorithm 2 aims at enabling smooth response to bursty traffic (since traffic in DCNs may be highly variable over short periods of time [33, 46]). It receives as input the list of VMs (V_s) hosted on server s, their current demands (which are determined by monitoring VM socket buffers, similarly to Mahout [31]), their bandwidth guarantees and their network weight (specified in the application request and defined in Section 4).

First, the rate for each VM is calculated based on their demands and the guaranteed bandwidth $B[v]$ (lines 1 – 3). In case the demand of a VM is equal or lower than its bandwidth guarantees (represented by $v \downarrow V_s \mid v \in V_s$), the rate is assigned and enforced (line 2), so that the exact amount of bandwidth needed for communication is used (wasting no network resources). In contrast, the guarantee $B[v]$ is initially assigned to $nRate[v]$ for each VM $v \in V_s$ with higher demands than its guarantees (represented by $v \uparrow V_s \mid v \in V_s$), in line 3. Then, the algorithm calculates the residual bandwidth of the link connecting the server to the ToR switch (line 4). The residual bandwidth is calculated by subtracting from the link capacity the guarantees of VMs with higher demands than their guarantees and the rate of VMs with equal or lower demands than their guarantees.

The last step establishes the bandwidth for VMs with higher demands than their guarantees (line 5 - 10). The rate (line 8)

Algorithm 2: Work-conserving rate allocation.

Input : Set of VMs V_s allocated on server s, Current demands of VMs *demand*, Bandwidth guarantees B for each VM, Network weight w for each VM
Output: Rate *nRate* for each co-resident VM

1 **foreach** $v \in V_s$ **do**
2 **if** $v \downarrow V_s$ **then** $nRate[v] \leftarrow demand[v]$;
3 **else** $nRate[v] \leftarrow B[v]$;
4 $residual \leftarrow LinkCapacity - \left(\sum_{v \uparrow V_s} B[v] + \sum_{v \downarrow V_s} demand[v]\right)$;
5 hungryVMs $\leftarrow v \uparrow V_s \mid v \in V_s$;
6 **while** $residual > 0$ *and* hungryVMs *not empty* **do**
7 **foreach** $v \in$ hungryVMs **do**
8 $nRate[v] \leftarrow nRate[v] + \min\left(demand[v] - nRate[v], \left(\frac{w[v]}{\sum_{u \uparrow V_s} w[u]} \times residual\right)\right)$;
9 **if** $nRate[v] == demand[v]$ **then**
10 hungryVMs $\leftarrow$ hungryVMs $\setminus \{v\}$;
11 **return** *nRate*;

is determined by adding $nRate[v]$ (initialized in line 3) and the minimum bandwidth between (*i*) the difference of the current demand ($demand[v]$) and the rate ($nRate[v]$); and (*ii*) the proportional share of residual bandwidth the VM would be able to receive according to its weight $w[v]$. Note that there is a "while" loop (lines 6 – 10) to guarantee that all residual bandwidth is used or all demands are satisfied. If this loop were not used, there could be occasions when there would be unsatisfied demands even though some bandwidth would be available.

With this algorithm, Predictor guarantees that VMs will not receive more bandwidth than they need (which would waste network resources) and bandwidth will be fully utilized if there are demands (work-conservation). Moreover, the algorithm has fast convergence on bandwidth allocation and can adapt to the significant variable communication demands of cloud applications. Therefore, if there is available bandwidth, VMs can send traffic bursts at a higher rate (unlike Silo [3], Predictor allows traffic bursts with complete work-conservation).

In summary, if the demand of a VM exceeds its guaranteed rate, data can be sent and received at least at the guaranteed rate. Otherwise, if it does not, the unutilized bandwidth will be shared among co-resident VMs whose traffic demands exceed their guarantees. We provide an extensive evaluation in Section 7 to verify the benefits of the algorithm.

6. Control Plane Design

The control plane design of the network is an essential part of software-defined networks, as disconnection between the control and data planes may lead to severe packet loss and performance degradation (forwarding devices can only operate correctly while connected to a controller) [47, 48]. Berde et al. [49] define four requirements for the control plane: (*i*) high availability (usually five nines [50]); (*ii*) global network state, as the control plane must be aware of the entire network state to provide guarantees for tenants and their applications; (*iii*) high throughput, to guarantee performance in terms of satisfying requests even at periods of high demands; and (*iv*) low latency, so that end-to-end latency for control plane communication (i.e., updating network state in response to events) is small.

Based on these requirements, Figure 6 shows the control plane design for Predictor. In this figure, we show, as a basic example, a typical 2-layer tree-like topology with decoupled control and data planes. We can see two major aspects: (*i*) the placement of controller instances (control plane logic) as a cluster in one location of the network (connected to all core switches); and (*ii*) the separation between resources for both planes (represented by different line styles and colors for link bandwidth), indicating out-of-band control plane communication. We discuss them next.

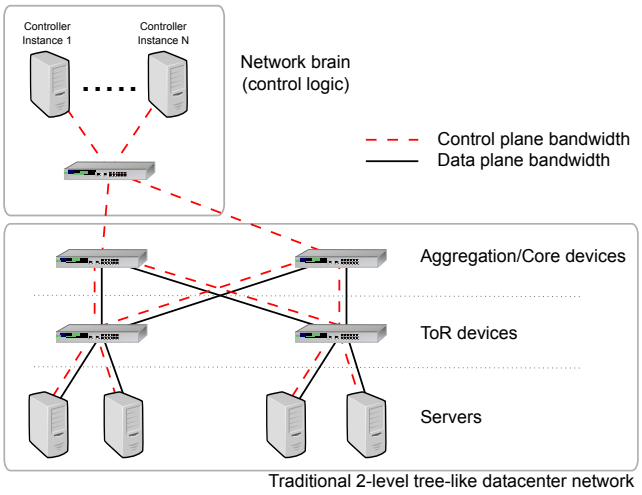


Figure 6: Design of Predictor's control plane.

Cluster of controller instances. Following WL2 [51], the control plane logic is composed of a cluster of controller instances. There are two reasons for this. First and most important, Predictor needs strong consistency among the state of its controllers to provide network guarantees for tenants and their applications. If instances were placed at different locations of the topology, the amount of synchronization traffic would be unaffordable, since DCNs typically have highly dynamic traffic patterns with variable demands [30, 33, 46]. Moreover, DCNs are typically oversubscribed with scarce bandwidth [21].

Second, the control plane is expected to scale-out (periodically grow or shrink the number of active controller instances) according to its load, needed for high availability and throughput. Since DCNs usually count with multiple paths [32], one controller location is often sufficient to meet existing requirements [52]. Furthermore, if controllers were placed at several locations, a controller placement algorithm (e.g., Survivor [47]) would have to be executed each time the number of instances were adjusted, which would delay the response to data plane requests (as this is a NP-Hard problem [52]).

Out-of-band control. Predictor uses out-of-band control to manage the network. As the network load may change significantly over small periods of time [16] and some links may get congested [33] (due to the high oversubscription factor [53]), the control and data planes must be kept isolated from one an-

other, so that traffic from one plane does not interfere[8] with the other. In other words, control plane traffic should not be impacted by rapid changes in data plane traffic patterns (e.g., bursty traffic). Using out-of-band control, some bandwidth of each link shared with the data plane (or all bandwidth from links dedicated to control functions) is reserved for the control plane (represented in Figure 6 as red dotted lines). In the next section, we show how the amount of bandwidth reserved for the control plane affects efficiency of Predictor.

7. Evaluation

Below, we evaluate the benefits and overheads of Predictor. We focus on showing that Predictor (*i*) can scale to large SDN-based DCNs; (*ii*) provides both predictable network performance (with bandwidth guarantees) and work-conserving sharing; and (*iii*) outperforms existing schemes for DCNs (the baseline SDN/OpenFlow controller and DevoFlow [13]). Towards this end, we first describe the environment and workload used (in Section 7.1). Then, we examine the main aspects of the implementation of Predictor (in Section 7.2). Finally, we present the results in Section 7.3.

7.1. Setup

Environment. We have implemented a simulator that models an IaaS multi-tenant, SDN-based datacenter. The network is defined as a tree-like topology, similar to current DCNs and related work [3, 20, 21]. It is composed of a three-tier topology with 16,000 servers at level 0. We follow current schedulers and related work [54] and divide computing resources of servers (corresponding to some amount of CPU, memory and storage) into slots for hosting VMs; each server is divided into 4 slots, resulting in a total amount of 64,000 available VMs in the datacenter. Every 40 machines form a rack, and every 10 ToRs are connected to an aggregation switch. Finally, all aggregation switches are connected to a core switch. The capacity of each link is defined as follows: 1 Gbps for server-ToR links, 10 Gbps for ToR-aggregation links and 50 Gbps for aggregation-core links.

Workload. The workload is composed of incoming application requests (to be allocated in the datacenter) arriving over time. In particular, we consider a heterogeneous set of applications, including MapReduce and Web Services. As defined in Section 4, each application a is represented as a tuple $\langle N_a, B_a, w_a, comm_a^{inter} \rangle$. Given the lack of publicly available traces for DCNs, the workload was generated in line with related work [20, 21]. N_a is exponentially distributed around a mean of 49 VMs (following measurements from prior work [1]). B_a was generated by reverse engineering the traces used by Benson et al. [16] and Kandula et al. [37]. More specifically, we used their measurements related to inter-arrival flow-time and flow-size at servers to generate and simulate network demands of applications. Unless otherwise specified, of all traffic, 20% of flows are destined to other applications [20] and 1% is classified as large flows [33]. We pick the destination of each flow by first determining whether it is an intra- or inter-application flow and then uniformly selecting a destination. The weight w_a is uniformly distributed in the interval $[0, 1]$.

[8] In oversubscribed networks, such as DCNs, where traffic may exceed link capacities in some occasions, in-band control may result in network inconsistencies, as control packets may not (or take a long time to) reach the destination.

7.2. Implementation Aspects of Predictor

Figure 7 shows the architecture of the server-level implementation of Predictor. As described by Pfaff et al. [41], the virtual machines allocated on the server send and receive packets to/from the network through the hypervisor, using an Open vSwitch. We implemented a local controller which directly communicates with the Open vSwitch. Together, the Open vSwitch and the local controller are responsible for handling all traffic to/from local virtual machines.

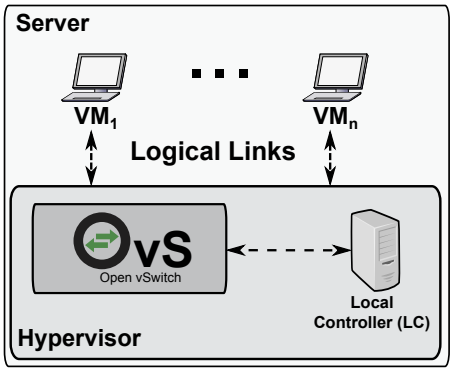


Figure 7: Architecture of server-level implementation of Predictor.

This architecture leverages the relatively large amount of processing power at end-hosts [55] in the datacenter to implement two key aspects of Predictor (following the description presented in the previous sections): (*i*) identifying flows at application-level; and (*ii*) providing network guarantees and dynamically enforcing rates for VMs. Both aspects are discussed next.

First, to perform application-level flow identification, Predictor utilizes Multiprotocol Label Switching (MPLS). More specifically, applications are identified in OpenFlow rules (at the Open vSwitch) through the label field in the MPLS header. The MPLS label is composed of 20 bits, which allows Predictor to identify 1,048,576 different applications. The complete operation of identifying and routing packets at application-level works as follows. For each packet received from the source VM, the Open vSwitch (controlled via the OpenFlow protocol) in the source hypervisor pushes a MPLS header (four bytes) with an ID in the label field (the application ID of the source VM for intra-application communication or a composite ID for inter-application communication). Subsequent switches in the network use MPLS label and IP source and destination addresses (which may be wildcarded, depending on the possibilities of routing) matching fields to choose the correct output port to forward incoming packets. When packets arrive at the des-

tination hypervisor, the Open vSwitch pops the MPLS header and forwards the packet to the correct VM.

Second, the local controller at each server performs rate-limiting of VMs. More precisely, the local controller dynamically sets the allowed rate for each hosted VM by installing the appropriate rules and rate-limiters at the Open vSwitch. The rate is calculated by Algorithm 2, discussed in Section 5.3. Note that Predictor also reduces rate-limiting overhead when compared to previous schemes (e.g., Silo [3], Hadrian [20], CloudMirror [7] and ElasticSwitch [17]), for it only rate-limits the source VM while other schemes rate-limit each pair of source-destination VMs.

7.3. Results

Next, we explain the behavior of the three schemes we are comparing against each other (Predictor, DevoFlow and the baseline). Then, we show the results of the evaluation: (*i*) we examine the scalability of employing Predictor on large SDN-based DCNs; and (*ii*) we verify bandwidth guarantees and predictability.

Comparison. We compare Predictor with the baseline SDN/OpenFlow controller and the state-of-the-art controller for DCNs (DevoFlow [13]). Before showing the results, we briefly explain the behavior of Predictor, the baseline and DevoFlow.

In Predictor, bandwidth for intra-application communication is guaranteed at allocation time. For inter-application communication guarantees, we consider two modes of operation, as follows. The first one is called Proactive Inter-Application Communication (PIAC), in which tenants specify in the request all other applications that their applications will communicate with (by using the field $comm_a^{inter}$, as explained in Section 4). The second one is called Reactive Inter-Application Communication (RIAC), in which rules for inter-application traffic are installed by the controller by either reacting to new flows in the network or receiving communication requests from applications, as defined in Section 5.1. Note that both modes correspond to the extremes for inter-application communication: while PIAC considers that all inter-application communication is specified at allocation time, RIAC considers the opposite. Furthermore, we highlight that both modes result in the same number of rules in devices, but differ in controller load and flow setup time (results are shown below).

In the baseline, switches forward to the controller packets that do not match any rule in the flow table (we consider the default behavior of OpenFlow versions 1.3 and 1.4 upon a table-miss event). The controller, then, responds with the appropriate set of rules specifically designed to handle the new flow.

DevoFlow considers flows at the same granularity than the baseline, thus generating a similar number of rules in forwarding devices. However, forwarding devices rely on more powerful hardware and templates to generate rules for small flows without involving the controller. For large flows, DevoFlow has two modes of operation. DevoFlow Triggers requires switches to identify large flows and ask the controller for appropriate rules for these flows (i.e., only packets of large flows are forwarded to the controller). DevoFlow Statistics, in turn, requires forwarding devices to send the controller uniformly chosen samples (packets), typically at a rate of 1/1000 packets, so that the controller itself identifies and generates rules for large flows. In summary, both DevoFlow modes generate the same number of rules in devices, but differ in controller load and flow setup time.

Scalability metrics. We use four metrics to verify the scalability of Predictor in SDN-based datacenter networks: number of rules in flow tables, controller load, impact of reserved control plane bandwidth and flow setup time. These are typically the factors that restrict scalability the most [12, 17].

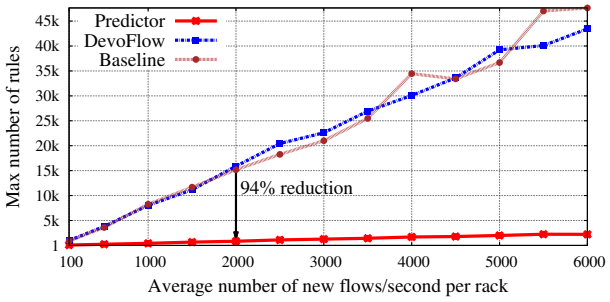
Reduced number of flow table entries. Figure 8 shows how network load (measured in new flows/second per rack) affects flow table occupancy in forwarding devices. More precisely, the plots in Figures 8(a), 8(b) and 8(c) show, respectively, the maximum number of entries observed in our experiments[9] that are required in any hypervisor, ToR and aggregation switch for a given average rate of new flows at each rack (results for core devices are not shown, as they are similar for all three schemes).

In all three plots, we see that the average number of arriving flows during an experiment affects directly the number of rules needed in devices. These results are explained by the fact that the number of different flows that pass through forwarding devices is large and may quickly increase due to the elevated number of end-hosts (VMs) and arriving flows in the network. Overall, the increase of the total number of flows requires more rules for the correct operation of the network (according to the needs of tenants) and enables richer communication patterns (representative of cloud datacenters [20]). Note that the number of rules for the baseline and DevoFlow is similar because (*i*) they consider flows at the same granularity; and (*ii*) the same default timeout for rules was adopted for all three schemes.

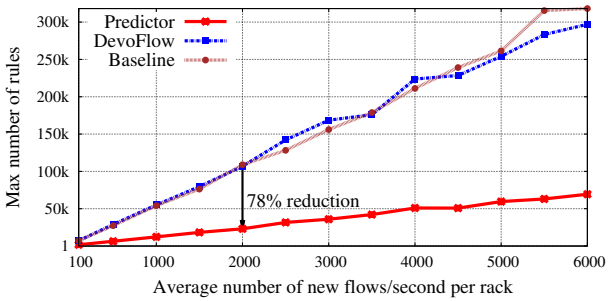
The results show that Predictor substantially outperforms DevoFlow and the baseline (especially for realistic numbers of new flows in large-scale DCNs, i.e., higher than 1,500 new flows/second per rack [17]). More importantly, the curves representing Predictor have a smaller growing factor than the ones for DevoFlow and the baseline. The observed improvement happens because Predictor manages flows at application-level and also wildcards the source and destination addresses in rules when possible (as explained in Section 7.2). Predictor reduces the number of rules up to 94% in hypervisors, 78% in ToRs and 37% in aggregation devices. In core devices, the reduction is negligible (around 1%), because (*a*) a high number of flows does not need to traverse core links to reach their destinations, thus the baseline and DevoFlow do not install many rules in core devices, while Predictor installs application-level rules; and (*b*) Predictor proactively installs rules for intra-application traffic (while other schemes install rules reactively).

Since Predictor considers flows at application-level and inter-application flows may require rules at a lower granularity (e.g., by matching MAC and IP fields), we now analyze how the number of inter-application flows affects the number of rules in for-

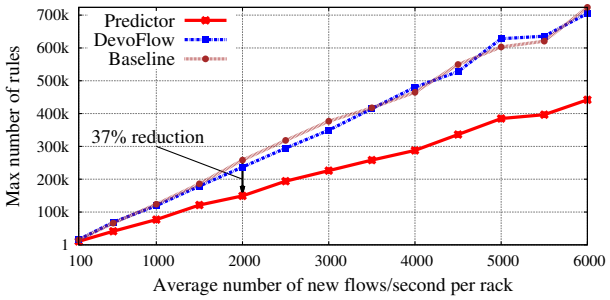
[9] Since flow table capacity of current available OpenFlow-enabled switches ranges from one thousand [56] to around one million entries [57], the observed values during the experiments are within acceptable ranges.



(a) In hypervisors.



(b) In ToRs.



(c) In aggregation switches.

Figure 8: Maximum number of rules (that were observed in the experiments) in forwarding devices.

warding devices for Predictor (previous results considered 20% of inter-application flows, a realistic percentage according to the literature [20]). Note that we only show results for Predictor because the percentage of inter-application flows does not impact the number of rules in forwarding devices for the baseline and DevoFlow.

Figure 9 shows the maximum number of entries in hypervisors, ToR, aggregation and core devices observed in our experiments (y-axis) for a given percentage of inter-application flows (x-axis), considering an average of 1,500 new flows/second per rack. As expected, we see that the number of rules in devices increases according to the number of inter-application flows. This happens because this type of communication often involves a restricted subset of VMs from different applications. Therefore, Predictor may not install application-level rules for these flows and may end up installing lower-granularity ones (e.g., by matching the IP field). Nonetheless, application-level rules address most of the traffic in the DCN.

Moreover, the number of rules in aggregation and, in particular, in core switches is higher than in ToR devices and in hypervisors. It is so because core switches interconnect several aggregation switches and, as time passes, the arrival and departure of applications lead to dispersion of available resources in the infrastructure. In this context, VMs from different applications (allocated in distinct ToRs) communicate with each other through paths that use aggregation and core switches.

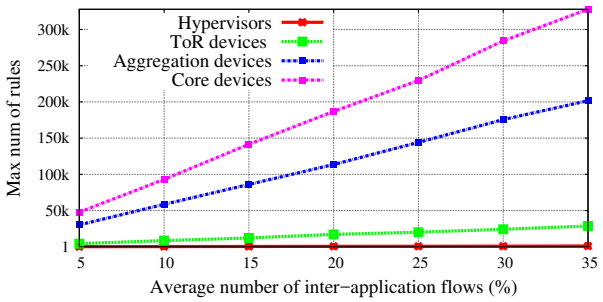


Figure 9: Maximum number of rules in forwarding devices for different percentages of inter-application flows for Predictor.

In general, Predictor reduces the number of rules installed in forwarding devices, which can (*i*) improve hypervisor performance (as measured by LaCurts et al. [22]); (*ii*) minimize the amount of TCAM occupied by rules in switches (TCAMs are a very expensive resource [15] and consume a high amount of power [58]); and (*iii*) minimize the time needed to install new rules in TCAMs, as measured in Section 2.

Low controller load. As DCNs typically have high load, the controller should handle flow setups efficiently. Figure 10 shows the required capacity in number of messages/s for the controller. For better visualization, the y-axis is represented in logarithmic scale, as the values differ significantly for different schemes. As expected, the number of messages sent to the controller increases according to the average number of new flows/s per rack (except for Predictor PIAC and DevoFlow Statistics). The controller must set up network paths and allocate resources according to arriving flows (flows without matching rules in forwarding devices).

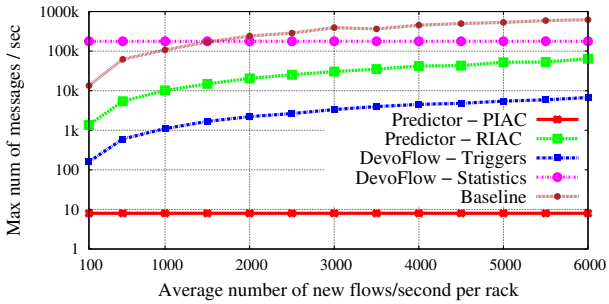


Figure 10: Controller load.

The baseline imposes a higher load to its controller than other schemes. DevoFlow Statistics requires a regular load to its controller, independently of the number of flows, as the num-

ber of messages sent by forwarding devices to the controller depends only on the amount of traffic in the network; in this scheme, devices send to the controller randomly chosen packet samples at a rate of 1/1000 packets. DevoFlow Triggers, in turn, only needs controller intervention to install rules for large flows (at the cost of more powerful hardware at forwarding devices). Thus, it significantly reduces controller load, but may also reduce controller knowledge of (*i*) network load and (*ii*) flow table state in switches. Predictor RIAC proactively installs application-level rules for intra-application communication at allocation time and reactively sends rules for inter-application traffic upon receiving communication requests, which reduces the number of flow requests when compared to the baseline ($\approx$ 91%) but increases it in comparison to DevoFlow Triggers ($\approx$ 8%). Finally, Predictor PIAC receives fine-grained information about intra- and inter-application communication at application allocation time, proactively installing the respective rules when needed. Therefore, controller load can be significantly reduced (i.e., the controller receives requests only when applications are allocated) without hurting knowledge of network state, but at the cost of some burden on tenants (as they need to specify inter-application communication at allocation time for Predictor PIAC).

Recall that the Predictor modes under evaluation correspond to extremes. Therefore, in practice, we expect that Predictor controller load will be between the results shown for PIAC and RIAC. Moreover, we do not show results for controller load varying the number of large flows because results are the same for both modes (and also for the baseline and DevoFlow Statistics). DevoFlow Triggers, however, imposes a higher load to its controller as the number of large flows increases (Figure 10 depicted results for a realistic value of 1% of large flows).

So, in both modes, the Predictor controller is aware of most of the traffic (at application-level) and performs fine-grained control. In contrast, DevoFlow Triggers has knowledge of only large flows (approximately 50% of the total traffic volume [16]) and DevoFlow Statistics has partial knowledge of network traffic with a high number of messages sent to the controller.

Impact of control plane bandwidth. SDN separates the control and data planes. Ideally, control plane communication is expected to be isolated from data plane traffic, avoiding cross-interference. In this context, the bandwidth it requires varies: the more dynamic the network, the more control plane traffic may be required for updating network state and getting information from forwarding devices. We evaluate the impact of reserving some amount of bandwidth (5%, 10%, 15%, 20%, 25% and 30%) on data plane links to the control plane and compare it with a baseline value of 0% (which represents no bandwidth reservation for the control plane). In other words, we want to verify how the acceptance ratio of applications (y-axis) is affected according to the amount of bandwidth reserved for the control plane (x-axis), since the network is the bottleneck in comparison to computing resources [21, 43]. Figure 11 confirms that acceptance ratio of requests decreases according to the amount of bandwidth available for the control plane (clearly, more bandwidth for the control plane means less bandwidth for the data plane). Nonetheless, this reduction is small, even for a worst-case scenario: reserving 30% of bandwidth on data plane links for the control plane results in accepting around 9% fewer requests.

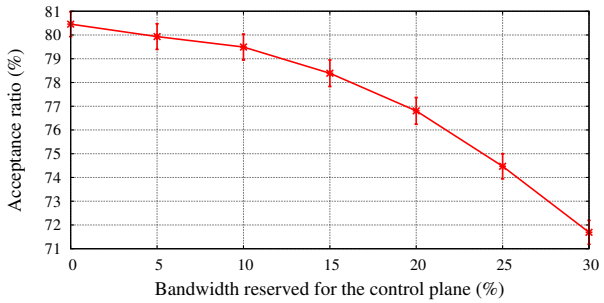


Figure 11: Impact of reserved bandwidth for the control plane on acceptance ratio of requests (error bars show 95% confidence interval).

Therefore, depending on the configuration, SDN may affect DCN resource utilization and, consequently, provider revenue. There are two main reasons: (*i*) it involves the control plane more frequently [13]; and (*ii*) switches are constantly exchanging data with the controller (for both flow setup and the controller to get updated information about network state). In this context, the amount of bandwidth required for the control plane for flow setup is directly proportional to the number of requests to the controller (Figure 10). In our experiments, switches were configured to send the first 128 bytes of the first packet of new flows to the controller (instead of sending the whole packet). With this configuration and for a realistic number of new flows/s per rack (i.e., 1,500 new flows), the bandwidth required by the controller for flow setup for each scheme was at most the following: 1 Mbps for Predictor PIAC, 15 Mbps for Predictor RIAC, 2 Mbps for DevoFlow Triggers, 173 Mbps for DevoFlow Statistics and 166 Mbps for the baseline. Even though Predictor may require more bandwidth for its control plane than DevoFlow in some occasions, it has better knowledge of current network state and does not need customized hardware at forwarding devices.

Reduced flow setup time. The SDN paradigm typically introduces additional latency for the first packet of new flows; traditional SDN implementations (e.g., baseline) delay new flows for at least two RTTs in forwarding devices (communication between the ASIC and the management CPU and between that CPU and the controller)[10] [13].

The results in Figure 12 show the minimum, average and maximum flow setup time (additional latency) for the schemes being compared, during the execution of the experiments. For a better comparison, latency values are normalized according to the maximum value of the baseline (i.e., the highest value in our measurements). Predictor PIAC proactively installs rules for (*a*) intra-application traffic at allocation time and (*b*) for inter-application traffic before the communication

[10]For a detailed study of latency in SDN, the interested reader may refer to Phemius et al. [59].

starts (due to the information provided in the application request). Thus, it has no additional latency for new flows. Predictor RIAC, in turn, presents no additional latency for most of the flows (due to the proactive installation of rules for intra-application communication). However, it introduces some delay for inter-application flows (maximum measured latency was around 80% of the maximum for the baseline). As both Predictor modes correspond to extremes, results will actually be somewhere between RIAC and PIAC. That is, the level of information regarding inter-application communication in application requests will vary, thus eliminating flow setup time for some inter-application flows.

In DevoFlow Statistics, forwarding devices generate rules for all flows in the network. In other words, most of the latency is composed of the time taken for communication between the ASIC and the management CPU when a new flow is detected. Later on, the controller installs specific rules for large flows when it identifies such flows. Thus, this scheme typically introduces low additional latency. In the case of DevoFlow Triggers, large flows require controller assistance (thereby increasing additional latency), while small flows are handled by the forwarding devices themselves (low additional latency).

Finally, the baseline requires that forwarding devices always ask for controller assistance to handle new flows, resulting in increased control traffic and flow setup time in comparison to Predictor and DevoFlow.

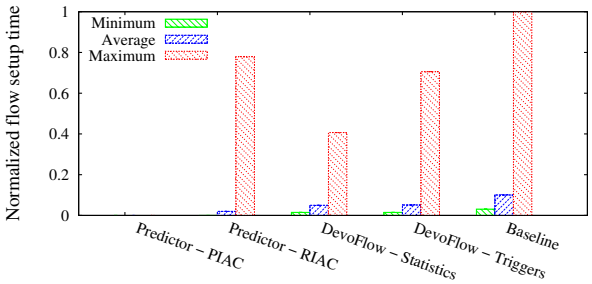


Figure 12: Flow setup time (normalized by the maximum flow setup time of the baseline) introduced by the SDN paradigm for new flows.

After verifying the feasibility of employing Predictor on large-scale, SDN-based DCNs (i.e., the benefits provided by Predictor, as well as the overheads), we turn our focus to the challenge of bandwidth sharing unfairness. In particular, we show that Predictor (*i*) proportionally shares available bandwidth; (*ii*) provides minimum bandwidth guarantees for applications; and (*iii*) provides work-conserving sharing under worst-case scenarios, achieving both predictability for tenants and high utilization for providers.

Impact of weights on proportional sharing. Before demonstrating that Predictor provides minimum guarantees with work-conserving sharing, we evaluate the impact of weights when proportionally sharing available bandwidth. More specifically, we first want to confirm that available bandwidth is proportionally shared according to the weights assigned to applications and their VMs.

Toward this end, Figure 13 shows, during a predefined period of time, three VMs from different applications allocated on a given server with same demands and guarantees, but different weights (0.2, 0.4 and 0.6, respectively). We verify that, in case that the sum of all three VM demands do not exceed the link capacity (1 Gbps), all VMs have their demands satisfied (e.g., between 1s – 86s and 118s – 197s), independently of their guarantees. In contrast, if the sum of demands exceed the link capacity, each VM gets a share of available bandwidth (i.e., more than its guarantees) according to its weight (the higher the weight, the more bandwidth it gets). Note that, in this case, the rate of each VM stabilizes (between 87s – 117s and 197s – 500s) because, as the sum of demands exceed the link capacity (and VMs have the same demands and guarantees), the only factor that impacts available bandwidth sharing is the weight. In general, the results show that the use of weights enables proportional sharing.

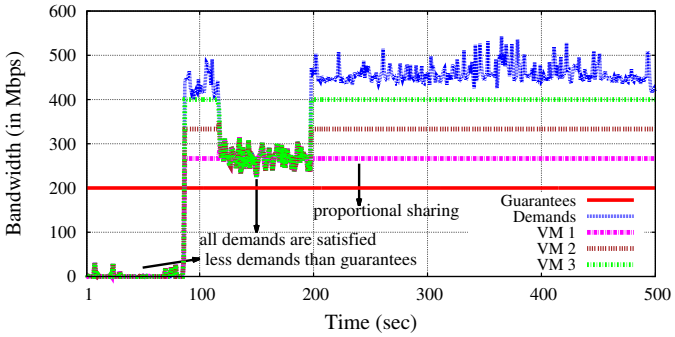
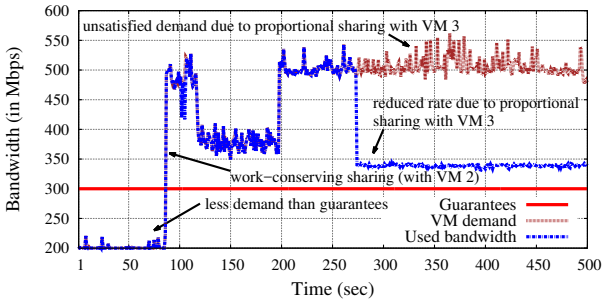


Figure 13: Proportional sharing according to weights (VM 1: 0.2; VM 2: 0.4; and VM 3: 0.6), considering the same guarantees (200 Mbps) and the same demands for all three VMs allocated on a given server connected through a link of 1 Gbps.

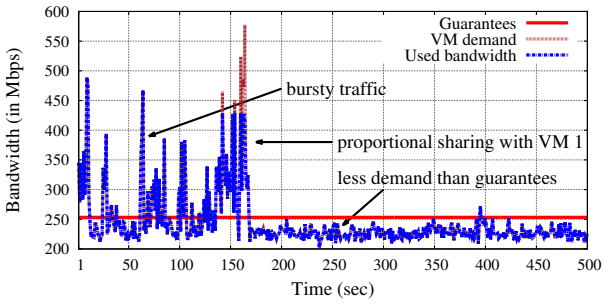
Minimum bandwidth guarantees for VMs. We define it as follows: the VM rate should be (*a*) at least the guaranteed rate if the demand is equal or higher than the guarantees; or (*b*) equal to the demand if it is lower than the guarantees. To illustrate this point, we show, in Figure 14, the set of VMs (in this case, three VMs from different applications) allocated on a given server during a predefined time period of an experiment. Note that VM 1 [Figure 14(a)] and VM 3 [Figure 14(c)] have similar guarantees, but receive different rates ("used bandwidth") when their demands exceed the guarantees (e.g., after 273s). This happens because they have different network weights (0.17 and 0.59, respectively), and the rate is calculated considering the demands, bandwidth guarantees, network weight and residual bandwidth. Moreover, we see (from Figures 13 and 14) that VMs may not get the desired rate to satisfy all of their demands instantaneously (when their demands exceed their guarantees) because (*i*) the link capacity is limited; and (*ii*) available bandwidth is proportionally shared among VMs.

In summary, we see that Predictor provides minimum bandwidth guarantees for VMs, since the actual rate of each VM is always equal or higher than the minimum between the demands and the guarantees. Therefore, applications have mini-

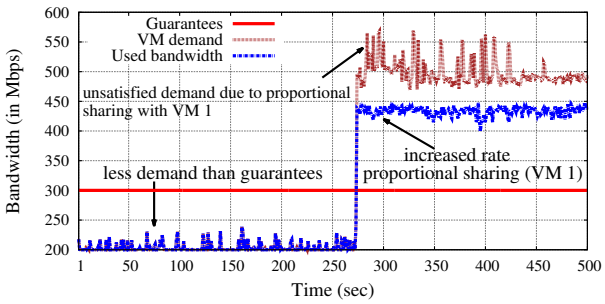
mum bandwidth guarantees and, thus, can achieve predictable network performance.



(a) VM 1.



(b) VM 2.



(c) VM 3.

Figure 14: Bandwidth rate achieved by the set of VMs allocated on a given server during a predefined period of time.

Work-conserving sharing. Bandwidth which is not allocated, or allocated but not currently used, should be proportionally shared among other VMs with more demands than their guarantees (according to the weights of each application, using Algorithm 2). Figure 15 shows the aggregate bandwidth[11] on the server holding the set of VMs in Figure 14. In these two figures, we verify that Predictor provides work-conserving sharing in the network, as VMs can receive more bandwidth (if their demands are higher than their guarantees) when there is spare bandwidth. Thus, providers can achieve high network utilization. Furthermore, by providing work-conserving sharing, Predictor offers high responsiveness[12] to changes in bandwidth requirements of applications.

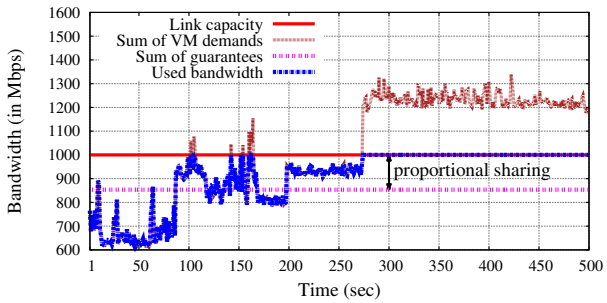


Figure 15: Work-conserving sharing on the server holding the set of VMs from Figure 14.

In general, Predictor provides significant improvements over DevoFlow, as it allows high utilization and fine-grained management in the network for providers and predictability with guarantees for tenants and their applications. As a side effect, Predictor may have higher controller load than DevoFlow (the cost of providing fine-grained management in the network without imposing to tenants the burden of specifying inter-application communication at allocation time).

8. Discussion

After evaluating Predictor, we discuss its generality and limitations.

Application-level flow identification. In our proof-of-concept implementation, Predictor identifies flows at application-level through the MPLS label (application ID with 20 bits). Therefore, it needs a MPLS header in each packet (adding four bytes of overhead). In practice, there are at least two other options to provide such functionality. First, when considering the matching fields defined by OpenFlow, application-level flows could also be identified by utilizing IEEE standard 802.1ad (Q-in-Q) with double VLAN tagging. The advantage of double tagging is a higher number of IDs available (24 bits), while the drawback is an overhead of eight bytes (two VLAN headers) per packet. Second, application-level flows could be identified by using OpenFlow Extensible Match (OXM)[13] to define a unique match field for this purpose. Nonetheless, this method is less flexible, as it requires (*i*) switch support for OXM; and (*ii*) programming to add a new matching field in forwarding devices.

Topology-awareness. Even though Algorithm 1 was specifically designed for tree-like topologies, the proposed strategy is topology-agnostic. Therefore, we simply need to replace Algorithm 1 to employ Predictor in DCNs with other types of interconnections. We used a tree-like placement algorithm in this

[11] Note that Predictor considers only bandwidth guarantees when allocating VMs (i.e., it does not take into account temporal demands). Therefore, even though the sum of temporal demands of all VMs allocated on a given server may exceed the server link capacity, the sum of bandwidth guarantees of these VMs will not exceed the link capacity.

[12] Responsiveness is a critical aspect of cloud guarantees [60].

[13] OXM was introduced in OpenFlow version 1.2 and currently is supported by several commercial forwarding devices.

paper for three reasons. First, currently most providers implement DCNs as (oversubscribed) trees, since they can control the oversubscription factor more easily with this type of structure (in order to achieve economies of scale). Second, by using an algorithm specially developed for a particular structure, we can enable better use of resources. Thus, we show more clearly the benefits and overheads of the proposed strategy. Third, we used tree topologies for the sake of explanation, as it is easier to explain and to understand how bandwidth is allocated and shared among VMs of the same application in this kind of topology (e.g., in Figure 5) than, for example, in random graphs.

Dynamic rate allocation with feedback from the network. The designed work-conserving algorithm does not take into account network feedback provided by the OpenFlow module. This design choice was deliberately made; we aim at reducing management traffic in the network, since DCNs are typically oversubscribed networks with scarce resources [21]. Nonetheless, the algorithm could be extended to consider feedback, which would further help controlling the bandwidth used by flows traversing congested links.

Application ID management. Predictor controller assigns IDs for applications (in order to identify flows at application-level) upon allocation and releases IDs upon deallocation. Therefore, ID management is straightforward, as Predictor has full control over which IDs are in use at each period of time.

Application request abstraction. Currently, Predictor only supports the hose model [61]. Nonetheless, it can use extra control applications (one for each abstraction) (*i*) to parse requests specified with other models (e.g., TAG [7] and hierarchical hose model [20]); and (*ii*) to install rules accordingly. With other abstractions, Predictor would employ the same sharing mechanism (Section 5). Thus, it would provide the same level of guarantees.

9. Related Work

Researchers have proposed several schemes to address scalability in large-scale, SDN-based DCNs and performance interference among applications. Proposals related to Predictor can be divided into three classes: OpenFlow controllers (related to scalability in SDN-based DCNs), and deterministic and non-deterministic bandwidth guarantees (related to performance interference).

OpenFlow controllers. DevoFlow [13] and DIFANE [18] propose to devolve control to the data plane. The first one introduces new mechanisms to make routing decisions at forwarding devices for small flows and to detect large flows (to request controller assistance to route them), while the second keeps all packets in the data plane. These schemes, however, require more complex, customized hardware at forwarding devices. Kandoo [19] provides a logically distributed control plane for large networks. Nonetheless, it does not scale when most communications occur between VMs located in different racks. It is so because the distributed set of controller instances needs to maintain synchronized information (strong consistency) for the whole DCN. This is necessary in order to route traffic through less congested paths and to reserve resources for applications. Lastly, Hedera [62] and Mahout [31] require precise statistics from the network with at most 500 ms of interval between them to efficiently route large flows and utilize available resources [63]. However, obtaining statistics with such frequency is impractical in large DCNs [13]. Predictor, in contrast, requires neither customized hardware nor precise statistics from the network with at most 500 ms of interval between them, and scales to the high dynamic traffic patterns of DCNs.

Deterministic bandwidth guarantees. They take advantage of rate-limiting at hypervisors [64], VM placement [65] and virtual network embedding [66] in order to increase their robustness. Silo [3], CloudMirror [7] and Oktopus [2] provide strict bandwidth guarantees for tenants by isolating applications in virtual networks. Unlike Predictor, these approaches do not provide complete work-conservation (which may result in underutilization of resources) and address only intra-application communication.

Proteus [21], in contrast to Predictor, must profile temporal network demands of applications before allocation, since it requires such information for allocating applications in the infrastructure. Such requirement may be unrealistic for some types of applications (e.g., ones that consume an excessive amount of resources or that have requirements which depend on external factors). EyeQ [67] attempts to provide bandwidth guarantees with work-conservation. However, different from Predictor, it cannot provide guarantees upon core-link congestion [68] (and congestion is not rare in DCNs [34]). Finally, Hadrian [20] introduces a strategy that considers inter-application communication, but it (*i*) needs a larger, custom packet header (hindering its deployment); (*ii*) does not ensure complete work-conservation, as the maximum allowed bandwidth is limited according to the tenant's payment; and (*iii*) requires switches to dynamically perform rate calculation (and enforce such rate) for each flow in the network. Unlike Hadrian, Predictor provides complete work-conservation and performs rate calculation at server hypervisors (freeing switches from this burden).

Non-deterministic bandwidth guarantees. Seawall [1] and NetShare [69] share the network proportionally according to weights assigned to VMs and tenants. Thus, they allocate bandwidth at flow- and link-level. FairCloud [35] explores the trade-off among network proportionality, minimum guarantees and high utilization. These proposals, however, may result in substantial management overhead (since bandwidth consumed by each flow at each link is dynamically calculated according to the flow weight, and large DCNs can have millions of flows per second [16]). Predictor, in contrast, reduces management overhead by considering flows at application-level.

Varys [9], Baraat [8] and PIAS [4] seek to improve application performance by minimizing average and tail flow completion time (FCT). Karuna [6], in turn, minimizes FCT for non-deadline flows while ensuring that deadline flows just meet their deadlines. Unlike Predictor, none of them provide minimum bandwidth guarantees.

QJUMP [5] explores the trade-off between throughput and latency. While being able to provide low latency for selected flows, it may significantly reduce network utilization (as opposed to Predictor, which can achieve high network utilization).

Finally, AC/DC TCP [34] and vCC [70] address TCP unfairness by performing congestion control at the virtual switch in the hypervisor. Consequently, they are mostly orthogonal to Predictor, since a DCN needs both congestion control (e.g., AC/DC TCP and vCC) and bandwidth allocation (e.g., Predictor).

10. Conclusion

Datacenter networks are typically shared in a best-effort manner, resulting in interference among applications. SDN may enable the development of a robust solution for interference. However, the scalability of SDN-based proposals is limited, because of flow setup time and the number of entries required in flow tables.

We have introduced Predictor in order to scalably provide predictable and guaranteed performance for applications in SDN-based DCNs. Performance interference is addressed by using two novel SDN-based algorithms. Scalability is tackled as follows: (*i*) flow setup time is reduced by proactively installing rules for intra-application communication at allocation time (since this type of communication represents most of the traffic in DCNs); and (*ii*) the number of rules in forwarding devices is minimized by managing flows at application-level. Evaluation results show the benefits of Predictor. First, it provides minimum bandwidth guarantees with work-conserving sharing (successfully solving performance interference). Second, it eliminates flow setup time for most traffic in the network and significantly reduces flow table size (up to 94%), while keeping low controller load (successfully dealing with scalability of SDN-based DCNs). In future work, we intend to evaluate Predictor on a testbed (such as CloudLab [71]).

Acknowledgments

This work has been supported by the following grants: MCTI/CNPq/Universal (Project Phoenix, 460322/2014-1) and Microsoft Azure for Research grant award.

References

[1] A. Shieh, S. Kandula, A. Greenberg, C. Kim, B. Saha, Sharing the data center network, in: USENIX NSDI, 2011.

[2] H. Ballani, P. Costa, T. Karagiannis, A. Rowstron, Towards predictable datacenter networks, in: ACM SIGCOMM, 2011.

[3] K. Jang, J. Sherry, H. Ballani, T. Moncaster, Silo: Predictable Message Latency in the Cloud, in: ACM SIGCOMM 2015 Conference, SIGCOMM '15, ACM, New York, NY, USA, 2015.

[4] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, C. Tian, Information-Agnostic Flow Scheduling for Commodity Data Centers, in: USENIX NSDI, 2015.

[5] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, J. Crowcroft, Queues Don't Matter When You Can JUMP Them!, in: USENIX NSDI, 2015.

[6] L. Chen, K. Chen, W. Bai, M. Alizadeh, Scheduling Mix-flows in Commodity Datacenters with Karuna, in: Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 174–187. doi:10.1145/2934872.2934888.

[7] J. Lee, Y. Turner, M. Lee, L. Popa, J.-M. Kang, S. Banerjee, P. Sharma, Application-driven bandwidth guarantees in datacenters, in: ACM SIGCOMM, 2014.

[8] F. R. Dogar et al., Decentralized Task-Aware Scheduling for Data Center Networks, in: ACM SIGCOMM, 2014.

[9] M. Chowdhury et al., Efficient Coflow Scheduling with Varys, in: ACM SIGCOMM, 2014.

[10] M. Chowdhury, Z. Liu, A. Ghodsi, I. Stoica, HUG: Multi-Resource Fairness for Correlated and Elastic Demands, in: USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), USENIX Association, Santa Clara, CA, 2016.

[11] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, Onix: a distributed control platform for large-scale production networks, in: USENIX OSDI, 2010.

[12] Y. Jarraya, T. Madi, M. Debbabi, A Survey and a Layered Taxonomy of Software-Defined Networking, IEEE Communications Surveys Tutorials PP (2014) 1–29.

[13] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, DevoFlow: scaling flow management for high-performance networks, in: ACM SIGCOMM, 2011.

[14] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, C. Guo, Explicit Path Control in Commodity Data Centers: Design and Applications, in: USENIX NSDI, 2015.

[15] R. Cohen, L. Lewin-Eytan, J. S. Naor, D. Raz, On the effect of forwarding table size on SDN network utilization, in: IEEE INFOCOM, 2014.

[16] T. Benson, A. Akella, D. A. Maltz, Network traffic characteristics of data centers in the wild, in: ACM IMC, 2010.

[17] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, J. R. Santos, ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing, in: ACM SIGCOMM, 2013.

[18] M. Yu, J. Rexford, M. J. Freedman, J. Wang, Scalable flow-based networking with DIFANE, in: ACM SIGCOMM, 2010.

[19] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: ACM HotSDN, 2012.

[20] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, G. O'Shea, Chatty tenants and the cloud network sharing problem, in: USENIX NSDI, 2013.

[21] D. Xie, N. Ding, Y. C. Hu, R. Kompella, The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers, in: ACM SIGCOMM, 2012.

[22] K. LaCurts, S. Deng, A. Goyal, H. Balakrishnan, Choreo: Network-aware task placement for cloud applications, in: ACM IMC, 2013.

[23] D. S. Marcon, M. P. Barcellos, Predictor: providing fine-grained management and predictability in multi-tenant datacenter networks, in: IFIP/IEEE IM, 2015.

[24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2008) 69–74.

[25] G. Judd, M. Stanley, Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter, in: USENIX NSDI, 2015.

[26] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, Proc. VLDB Endow. 3 (2010) 460–471.

[27] G. Wang, T. S. E. Ng, The impact of virtualization on network performance of amazon EC2 data center, in: IEEE INFOCOM, 2010.

[28] R. Shea, F. Wang, H. Wang, J. Liu, A Deep Investigation Into Network Performance in Virtual Machine Based Cloud Environment, in: IEEE INFOCOM, 2014.

[29] H. Shen, Z. Li, New Bandwidth Sharing and Pricing Policies to Achieve A Win-Win Situation for Cloud Provider and Tenants, in: IEEE INFOCOM, 2014.

[30] J. Guo, F. Liu, X. Huang, J. C. Lui, M. Hu, Q. Gao, H. Jin, On Efficient Bandwidth Allocation for Traffic Variability in Datacenters, in: IEEE INFOCOM, 2014.

[31] A. R. Curtis, W. Kim, P. Yalagandula, Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection, in: IEEE INFOCOM, 2011.

[32] P. Gill, N. Jain, N. Nagappan, Understanding network failures in data centers: measurement, analysis, and implications, in: ACM SIGCOMM, 2011.

[33] D. Abts, B. Felderman, A guided tour of data-center networking, Commun. ACM 55 (2012) 44–51.

[34] K. He, E. Rozner, K. Agarwal, Y. J. Gu, W. Felter, J. Carter, A. Akella,

AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks, in: Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 244–257. doi:10.1145/2934872.2934903.

[35] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, I. Stoica, FairCloud: sharing the network in cloud computing, in: ACM SIGCOMM, 2012.

[36] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data Center TCP (DCTCP), in: ACM SIGCOMM, 2010.

[37] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, The nature of data center traffic: measurements & analysis, in: ACM IMC, 2009.

[38] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, M. Thottan, Measuring Control Plane Latency in SDN-enabled Switches, in: ACM SOSR, 2015.

[39] S. A. Jyothi, M. Dong, P. B. Godfrey, Towards a Flexible Data Center Fabric with Source Routing, in: USENIX NSDI, 2015.

[40] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, S. Krishnamurthi, Participatory Networking: An API for Application Control of SDNs, in: ACM SIGCOMM, 2013.

[41] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, The Design and Implementation of Open vSwitch, in: USENIX NSDI, 2015.

[42] H. Moens, B. Hanssens, B. Dhoedt, F. De Turck, Hierarchical network-aware placement of service oriented applications in clouds, in: IEEE/IFIP NOMS, 2014.

[43] L. Chen, Y. Feng, B. Li, B. Li, Towards Performance-Centric Fairness in Datacenter Networks, in: IEEE INFOCOM, 2014.

[44] N. Chowdhury, M. Rahman, R. Boutaba, Virtual network embedding with coordinated node and link mapping, in: IEEE INFOCOM, 2009.

[45] Amazon EC2, 2014. Available at: http://goo.gl/Fa90nC.

[46] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, S. Katti, NUMFabric: Fast and Flexible Bandwidth Allocation in Datacenters, in: Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 188–201. doi:10.1145/2934872.2934890.

[47] L. Muller, R. Oliveira, M. Luizelli, L. Gaspary, M. Barcellos, Survivor: An enhanced controller placement strategy for improving sdn survivability, in: IEEE GLOBECOM, 2014.

[48] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, Reliability-aware controller placement for software-defined networks, in: IFIP/IEEE IM, 2013.

[49] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, G. Parulkar, Onos: Towards an open, distributed sdn os, in: ACM HotSDN, 2014.

[50] F. J. Ros, P. M. Ruiz, Five nines of southbound reliability in software-defined networks, in: ACM HotSDN, 2014.

[51] C. Chen, C. Liu, P. Liu, B. T. Loo, L. Ding, A Scalable Multi-datacenter Layer-2 Network Architecture, in: ACM SOSR, 2015.

[52] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: ACM HotSDN, 2012.

[53] D. Adami, B. Martini, M. Gharbaoui, P. Castoldi, G. Antichi, S. Giordano, Effective resource control strategies using openflow in cloud data center, in: IFIP/IEEE IM, 2013.

[54] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, A. Akella, Multi-resource Packing for Cluster Schedulers, in: ACM SIGCOMM, 2014.

[55] M. Moshref, M. Yu, R. Govindan, A. Vahdat, Trumpet: Timely and Precise Triggers in Data Centers, in: Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 129–143. doi:10.1145/2934872.2934879.

[56] Y. Nakagawa, K. Hyoudou, C. Lee, S. Kobayashi, O. Shiraki, T. Shimizu, Domainflow: Practical flow management method using multiple flow tables in commodity switches, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT '13, ACM, New York, NY, USA, 2013, pp. 399–404. URL: http://doi.acm.org/10.1145/2535372.2535406. doi:10.1145/2535372.2535406.

[57] Switching Made Smarter, 2015. Available at: http://noviflow.com/products/noviswitch/.

[58] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, CoRR (2014).

[59] K. Phemius, M. Bouet, Openflow: Why latency does matter, in: IFIP/IEEE IM, 2013.

[60] J. C. Mogul, L. Popa, What We Talk About when We Talk About Cloud Network Performance, SIGCOMM Comput. Commun. Rev. 42 (2012) 44–48.

[61] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, J. E. van der Merive, A flexible model for resource management in virtual private networks, in: ACM SIGCOMM, 1999.

[62] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: dynamic flow scheduling for data center networks, in: USENIX NSDI, 2010.

[63] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, M. Handley, Data center networking with multipath tcp, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, ACM, New York, NY, USA, 2010, pp. 10:1–10:6. doi:10.1145/1868447.1868457.

[64] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, A. C. Snoeren, Cloud control with distributed rate limiting, in: ACM SIGCOMM, 2007.

[65] J. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, Joint VM placement and routing for data center traffic engineering, in: IEEE INFOCOM, 2012.

[66] M. G. Rabbani, R. P. Esteves, M. Podlesny, G. Simon, L. Z. Granville, R. Boutaba, On Tackling Virtual Data Center Embedding Problem, in: IFIP/IEEE IM, 2013.

[67] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, A. Greenberg, EyeQ: practical network performance isolation at the edge, in: USENIX NSDI, 2013.

[68] J. Guo, F. Liu, D. Zeng, J. Lui, H. Jin, A cooperative game based allocation for sharing data center networks, in: IEEE INFOCOM, 2013.

[69] V. T. Lam, S. Radhakrishnan, R. Pan, A. Vahdat, G. Varghese, Netshare and stochastic netshare: predictable bandwidth allocation for data centers, SIGCOMM Comput. Commun. Rev. 42 (2012) 5–11.

[70] B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, I. Keslassy, Virtualized Congestion Control, in: Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 230–243. doi:10.1145/2934872.2934889.

[71] CloudLab, 2016. Available at : https://www.cloudlab.us.

Daniel S. Marcon is a Ph.D. student at the Institute of Informatics of Federal University of Rio Grande do Sul – UFRGS, Brazil. He holds a B.Sc. degree in Computer Science from University of Vale do Rio dos Sinos – UNISINOS (2011) and a M.Sc. degree in Computer Science from Federal University of Rio Grande do Sul – UFRGS (2013). His research interests include datacenter networks, cloud computing, network virtualization and software-defined networking (SDN). See `http://inf.ufrgs.br/~dsmarcon` for further details.

Fabrício M. Mazzola is an undergraduate Computer Science student at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil. His research interests include software-defined networking (SDN), network virtualization and datacenter networks. More information can be found at `http://lattes.cnpq.br/2243053098009957`.

Marinho P. Barcellos received a PhD degree in Computer Science from University of Newcastle Upon Tyne (1998). Since 2008 Prof. Barcellos has been with the Federal University of Rio Grande do Sul (UFRGS), where he is an Associate Professor. He has authored many papers in leading journals and conferences related to computer networks, network and service management, and computer security, also serving as TPC member and chair. He has authored book chapters and delivered several tutorials and invited talks. His work as a speaker has been consistently distinguished by graduating students. Prof. Barcellos was the elected chair of the Special Interest Group on Computer Security of the Brazilian Computer Society (CE-Seg/SBC) 2011- 2012. He is a member of SBC and ACM. His current research interests are datacenter networks, software-defined networking, information-centric networks and security aspects of those networks. He was the General Co-Chair of ACM SIGCOMM 2016 and TPC Co-Chair of SBRC 2016. More information can be found at `http://www.inf.ufrgs.br/~marinho`.