Diss. ETH No. 22127

Optimized Protocol Stack for Virtualized Converged Enhanced Ethernet

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

Daniel Crisan

Ing. Info. Dipl. EPF

born on 03.09.1982

citizen of Romania

accepted on the recommendation of

Prof. Dr. Lothar Thiele, examiner

Prof. Dr. Torsten Hoefler, co-examiner

Mitch Gusat, co-examiner

2014

to Sînziana

Abstract

Datacenter networking undergoes a silent transition driven by the emergence of Converged Enhanced Ethernet (CEE) and network virtualization.

CEE aims to converge all the traffic generated by the previously disjoint local, system and storage networks on a single physical infrastructure. Traditionally, Ethernet did not guarantee losslessness: packets were dropped whenever a buffer reached its maximum capacity. This behavior does not match the semantics of modern datacenter applications used for storage or low-latency communications. CEE segregates Ethernet frames into eight different hardware priorities. Each priority may be configured as either lossy or lossless. Within a lossless priority, Priority Flow Control (PFC) prevents buffer overflows in a hop-by-hop manner. In this thesis, we will show that lossless Ethernet clusters can improve the performance of on-line data intensive applications. In particular, lossless fabrics avoid TCP incast throughput collapse, and can reduce the completion times by up to an order of magnitude.

Virtualization aims to consolidate different applications on the same hardware, thus increasing the average utilization of both the servers and communication equipment. The drawback of virtualization is that the TCP/IP stack, which was originally created and optimized to run directly over the network hardware, now runs over a new stack of layers responsible for virtualization, isolation, and encapsulation. In this thesis we will show that is possible to deconstruct the TCP protocol and redistribute its functions between the guest OS and the hypervisor. We will show that it is possible to conserve the existing features but with a much lower overhead. In our proposed architecture the hypervisor takes over most of reliability, flow and congestion control functions from the guest OS.

In this work we will will provide a practical way of virtualizing CEE. We will show how current hypervisor software lags behind network hardware by arbitrarily dropping frames in the virtualization layers, despite the fact modern Ethernet provides lossless traffic classes. Therefore, we will take corrective actions and we will introduce the first CEE-ready virtual switch. Next we will design a hypervisor that prevents misconfigured or malicious virtual machines (VMs) from filling the lossless cluster with stalled packets and compromising tenant isolation. We will prove the benefits of our new network hypervisor using a prototype implementation deployed on production-ready datacenter hardware.

Résumé

Les réseaux de centres de données subissent une transition silencieuse entraînée par l'émergence de Converged Enhanced Ethernet (CEE) et la virtualisation du réseau.

L'objectif de CEE est de faire converger tout le trafic généré par les réseaux locaux, systèmes et de stockage, auparavant disjoints, sur une seule infrastructure physique. Traditionnellement, Ethernet n'était pas sans perte: des trames ont été supprimées chaque fois qu'un mémoire tampon a atteint sa capacité maximale. Ce comportement ne correspond pas aux applications de centres de données modernes utilisées pour le stockage ou les communications à faible latence. CEE sépare les trames Ethernet en huit priorités différentes. Chaque priorité peut être configuré soit avec ou sans perte. Dans une priorité sans perte, Priority Flow Control (PFC) empêche les débordements des mémoires tampons. Dans cette thèse, nous allons montrer qu'Ethernet sans perte peut améliorer les performances des applications en ligne. En particulier, les réseaux sans perte peuvent éviter « TCP incast » et peuvent réduire les délais d'exécution jusqu'à un ordre de grandeur.

Virtualisation vise à consolider les différentes applications sur le même matériel, augmentant ainsi le taux d'utilisation moyen des serveurs et des équipements de communication. L'inconvénient de la virtualisation est que la pile TCP/IP, qui a été initialement créé et optimisé pour fonctionner proche de matériel, fonctionne maintenant sur une nouvelle pile de couches responsables de la virtualisation, l'isolement et l'encapsulation. Dans cette thèse, nous montrons qu'il est possible de déconstruire le protocole TCP et de redistribuer ses fonctions entre le système d'exploitation de la machine virtuelle et l'hyperviseur. Nous montrons qu'il est possible de conserver les fonctionnalités existantes mais avec un cout beaucoup plus faible. Dans notre architecture proposée, l'hyperviseur prend les fonctions de fiabilité, de contrôle de débit et de congestion du système d'exploitation de la machine virtuelle.

Dans ce travail, nous allons fournir un moyen pratique pour virtualisation de CEE. Nous allons montrer comment les hyperviseurs courant suppriment arbitrairement des trames dans les couches de virtualisation, en dépit du fait qu'Ethernet fournit des classes de trafic sans perte. Par conséquent, nous allons prendre des mesures correctives et nous allons introduire le premier commutateur virtuel pour CEE. Ensuite, nous allons concevoir un hyperviseur qui empêche les machines virtuelles mal configurés ou malveillantes de remplir le cluster sans perte de paquets bloqués et de compromettre l'isolement. Nous allons prouver les avantages de notre nouvel hyperviseur de réseau à l'aide d'un prototype de l'application déployée sur le matériel du centre de données prêt pour la production.

Acknowledgements

First of all I would like to thank Prof. Lothar Thiele for giving me the chance to pursue a PhD at ETHZ, and for the discussions and practical comments that had an important contribution in shaping this thesis. Also I thank Prof. Torsten Hoefler for carefully reading my manuscript and providing feedback.

I am grateful that I had the chance to do this research work at IBM Zurich Research Laboratory (ZRL). I want to thank all the people from the Systems Department and particularly to all my colleagues from the System Fabrics Group. I'd like to express my appreciation to Cyriel Minkenberg and Ronald Luijten, my managers at ZRL. I am most indebted to Mitch Gusat, my advisor during these 4 years for his permanent encouragement, supervision, and support. I have learned a lot from him, from the design of Ethernet networks to subtleties of technical paper writing. Without Mitch's constant optimism many of the papers that are the basis of this thesis would have never been finalized.

I want to thank Robert Birke and Nikolaos Chrysos for their permanent help and challenging discussions. Special thanks go to Robert for being a great office-mate. Thanks to Andreea Anghel, Bogdan Prisacari, Fredy Neeser, German Rodriguez Herrera, for the myriad discussions on innumerable topics. Also I'm grateful to Charlotte Bolliger from the publications department for proofreading and correcting my papers.

Finally and most important I'm grateful to my wife Sînziana for her love and patience. She has been my constant support in the countless nights and long weekends of paper writing. Without her stubbornness in convincing me to keep going every time I wanted to abandon, I would have never arrived at the end of this work.

Contents

1. Introduction

Datacenter networking undergoes a silent transition driven by the emergence of new fabrics, protocols and workloads. On the hardware side, we assist to the introduction of Converged Enhanced Ethernet (CEE) [13, 11, 12]. CEE is motivated by the desire to reduce capital and operational expenditures by converging the cluster, storage and high-performance computing fabrics into a single physical network. On the software side, the established TCP/IP protocol stack is disrupted by virtualization and overlay virtual networks [88, 81, 110, 37]. Virtualization aims to cut costs through increasing the average utilization of hardware.

In parallel, we observe the rise of new workloads that increase the pressure on the datacenter network. Notable here are the Online Data Intensive (OLDI) applications [115] that need to obey tight latency constraints to provide a satisfactory user experience. Another class of emerging applications such as MapReduce [42], Hadoop, Dryad [64] aim to ease the processing of large quantities of data but also generate increased intra-datacenter traffic. How these new workloads behave in a virtualized environment remains an open problem.

In this thesis we aim to virtualize and exploit the HPC-like losslessness of modern Ethernet, while maintaining the socket programming interface. We will show that the emerging CEE networks convey performance benefits for cluster traffic, for example by eliminating TCP incast throughput collapse and by reducing the flow-completion time of latency-critical applications. On the other hand the virtualization mechanisms are responsible for new bottlenecks on the critically fast data-path. We will identify and quantify these bottlenecks by measuring the increases of flow completion times they produce.

Furthermore, we will show how current hypervisor software lags behind network hardware by arbitrarily dropping frames in the virtualization layers, despite the fact modern Ethernet supports the segregation of traffic in different lossy or lossless classes. Therefore, we will take corrective actions and we will introduce the first

CEE-ready virtual switch. Next we will design a hypervisor that prevents misconfigured or malicious virtual machines (VMs) from filling the lossless cluster with stalled packets and compromising tenant isolation. By taking advantage of the CEE-ready virtual switch, we will deconstruct the existing virtualized networking stack into its core functions and consolidate them into an efficient hypervisor stack.

1.1. Virtualized Networking Stack

The bulk of datacenter communications is based on the TCP/IP protocol stack, designed in the 70's. The end-to-end principle [106] suggested that functions placed on lower layers simply duplicate the functions placed at higher layers with little value added to the end-user compared to their implementation cost. The canonical example given is that of reliable delivery of messages between two end-points. According to the end-to-end argument, implementing reliability between the intermediate hops makes little sense because the end-points must also provide end-to-end reliability anyway. In the case of the TCP/IP stack the Transmission Control Protocol (TCP) is responsible for reliability, flow and congestion control, and runs on top of minimal network hardware performing packet forwarding.

The virtualization was driven by the need to reduce the capital and operational expenditures. This objective is achieved by consolidating different applications on the same hardware thus increasing the average utilization of both the servers and communication equipment. The applications belong to different tenants, and each tenant rents a slice of the datacenter hardware from the datacenter operator. The applications run in virtual machines (VM) that are fully functional operating systems (OS) under the full control of the tenant. Therefore in a virtualized datacenter we have multiple guest operating systems running on top of a host operating system or hypervisor. The guest OS can be arbitrarily modified by the tenants, that can change both the drivers and the network protocols. On the other hand tenants cannot tamper with the hypervisor application which is controlled by the datacenter operator.

In Chapter 5 of this thesis, we argue that the TCP/IP stack, designed and optimized to run on top of hardware, ended up running on top of a brand new hypervisor stack responsible for virtualization. The minimal functions performed by the hypervisor stack are forwarding, routing, multiplexing and demultiplexing of the VM traffic. Second, the same layers are often enriched with additional policy enforcing features, e.g., decide whether two VMs are allowed to communicate or not. The VMs can be instantiated, migrated or deleted at a much faster pace than the hardware can be added, moved or removed from the datacenter. The agility of the VMs can cause major trouble in the communication hardware, such as, forwarding or routing loops, loss of connectivity, or overflows of the switching or routing tables. Therefore the VMs must be isolated from the physical infrastructure and this is done mainly through

encapsulation as shown in [88, 37, 81, 110]. Encapsulation requires additional operations performed on every flow, or even packet, and significantly contributes to data-path performance penalties as will be detailed in Chapter 3.

The bandwidth available to each server is shared between different VMs. The more aggressive tenants of a virtualized datacenter have incentives to abandon TCP in favor of customized lighter protocols, derived from UDP [108, 68]. These typically TCP-unfriendly protocols can hog unfair bandwidth shares and harm TCP-based applications. To counteract the problems of misbehaving or malicious flows, VMs or tenants, recent solutions [68, 108, 22, 56, 99, 23, 74] stack up additional layers that perform flow and congestion control within the hypervisor. Therefore, we observe the rise of a *hypervisor transport* that replicates some functions of TCP, further increasing the per-packet processing overhead, while rendering the TCP-unfriendly transports fairer.

To sum up, the TCP/IP stack, which was originally created and optimized to run directly over the network hardware, now runs over a new stack of layers responsible for virtualization, isolation, and encapsulation. In this thesis we will deconstruct the TCP protocol and redistribute its functions between the guest OS and the hypervisor. We will show that it is possible to conserve the existing features but with a much lower overhead. In our proposed architecture the hypervisor takes over most of reliability, flow and congestion control functions from the guest OS. The TCP layer in the guest OS is left only with simple connection management and segmentation functions. This redistribution of functionality is compatible with the end-to-end argument, i.e., the functions move between different layers of the virtualized stack.

1.2. Converged Enhanced Ethernet

Also driven by the need to reduce capital and operational expenditures, the IEEE 802 Data Center Bridging task group recently standardized Converged Enhanced Ethernet (CEE). CEE aims to converge all the traffic generated by the previously disjoint local, system and storage networks on a single physical infrastructure. First generation 10G products are already on the market, and CEE fabrics at 40G, or even 100G, have been announced by several vendors.

Upcoming datacenter networks based on 802 CEE are short and fat: up to one million nodes are connected in a single Layer 2 domain with abundant multipathing across 10-100 Gbps links of a few tens of meters (at most). Typical round-trip times (RTTs) range from 1-2 μs up to a few tens of μs, except under hotspot congestion, when the end-to-end delays can grow by several orders of magnitude, reaching into tens of ms [57]. Unlike wide area networks, the datacenter RTT is dominated by queuing delays, which under bursty workloads [16, 27, 26, 33, 72, 111, 117, 42], lead to a difficult traffic engineering and control problem.

Traditionally, Ethernet did not guarantee losslessness: packets were dropped whenever a buffer reached its maximum capacity. This behavior does not match the modern semantics of datacenter applications, including High-Performance Computing (HPC) environments [41], storage (Fibre Channel over Ethernet [10]), or Remote Direct Memory Access (RDMA) over Ethernet [36]. This problem is corrected in CEE, that segregates Ethernet frames into eight different hardware priorities. Each priority may be configured as either lossy or lossless. Within a lossless priority, Priority Flow Control (PFC) [13] acts as the earlier 802.3x PAUSE, preventing buffer overflows in a hop-by-hop manner – except that a paused priority does not affect other priorities. PFC is matched with the Enhanced Transmission Selection (ETS) [12] mechanism that provides a framework to support bandwidth allocation to traffic classes with different bandwidth and latency requirements.

In Chapter 2 of this thesis, we will show that besides enabling network convergence, lossless Ethernet clusters can improve the performance of soft real-time, scale-out applications, that harness big-data. In particular, lossless fabrics avoid TCP incast throughput collapse, and can reduce the completion times by up to an order of magnitude. Motivated by these findings, in Chapter 4, we will extend the losslessness of Ethernet into the virtual domain and we will introduce a zero-loss Overlay Virtual Network (zOVN) built around a CEE compatible lossless virtual switch.

However, the benefits of PFC come at a price: besides the potential for deadlock in certain topologies and switch architectures, it introduces exposure to saturation tree congestion. To counteract the potentially severe performance degradation due to such congestion, IEEE has recently standardized a congestion control scheme, Quantized Congestion Notification (QCN) [11]. Another drawback of PFC is that it introduces head-of-line (HOL) blocking. For example, consider two flows that share a congested link in a lossless cluster. The first flow, i.e., 'culprit', targets a busy destination that can only receive packets at a fraction of the link speed. The second flow, i.e., 'victim', targets an uncongested destination. Unable to proceed towards the blocked destination, the packets of the culprit flow monopolize the shared buffer space in the upstream switches. The net result is that the throughput of the victimized flow drops to the level of the culprit. Using a similar strategy, a malicious tenant could easily fill the lossless cluster with stalled packets, thus compromising bandwidth sharing and tenant isolation.

While two priorities do not interfere, flows of the same priority can HOL-block each other. Obviously, the 8 priority levels of PFC cannot separate and isolate the potentially millions of active flows. As we will show later, QCN is also ineffective in solving HOL-blocking between VM-to-VM flows. In Chapter 5 we will introduce a new hypervisor stack that ensures reliable delivery of the messages and uses a VM-to-VM proactive buffer management scheme to avoid HOL blocking and the ensuing interference between VMs.

1.3. Research Questions

The results presented in this work aim to defend the following hypotheses:

1. *It is possible to reduce the flow completion times of latency sensitive applications by avoiding packet drops in the virtualized networking stack.*
2. *Furthermore, it is possible to simplify the heavy networking stack by moving functionality from the TCP stack in the guest OS to the hypervisor, and by exploiting the hardware link-level flow control from CEE.*

While defending these hypotheses we also answer to the following open research questions:

- *What is the influence of CEE protocols on the completion time of TCP based applications?*
- *How do latency sensitive applications perform in a virtualized environment? What are the main performance gating factors of overlay virtual networks?*
- *What is the cause of packet drops in virtualized networks? What is the performance penalty of the packet drops? How can they be avoided?*
- *Can we design a lighter virtualized stack that improves performance of socket-based application, running on top of CEE hardware?*

1.4. Contributions and Thesis Outline

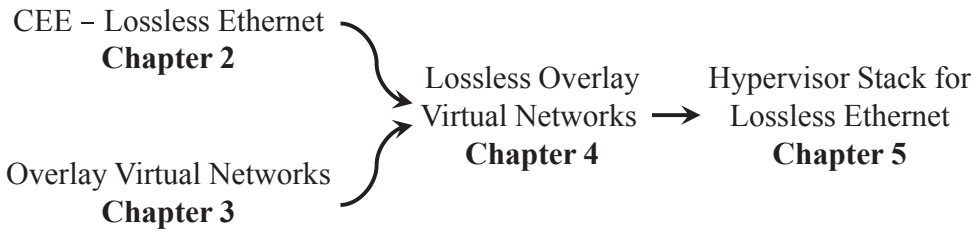


Figure 1.1.: Thesis structure.

The material is structured in four chapters organized as shown in Figure 1.1. The results introduced in Chapter 2 and Chapter 3 serve as basis for the proposals from Chapter 4, that are further extended in Chapter 5.

Chapter 2

In Chapter 2 we present the protocols implemented in Converged Enhance Ethernet. We perform the first evaluation of TCP performance in lossless CEE networks. First, we contribute the necessary parameter changes for TCP over 10Gbps CEE.

Next, we show evidence of PFC's benefits to TCP, leading to our recommendation to enable PFC also for TCP applications. In addition, we identify cases in which QCN is beneficial, respectively detrimental, to L4 transports. For this evaluation we extracted a full TCP stack from FreeBSD v9 and ported it to our simulation environment. Therefore we mix production-ready software with detailed L2 simulation models of CEE switches and adapters. On top of the TCP stack we run different workloads representative for datacenter traffic.

Next, we introduce the first source-based adaptive routing, proposed for CEE, using the established VLAN mechanism from Ethernet in conjunction with the recently standardized QCN. We introduce a simple *reactive route control* R^2C^2, and respectively, its higher performance counterpart, a *reactive route & rate control* R^3C^2, and we quantitatively examine the performance of our proposals against other routing algorithms.

Chapter 3

In Chapter 3 we present the virtualized network stack. We show that network virtualization requires new protocols responsible for the forwarding, filtering and multiplexing of traffic from different VMs. These layers are inserted between the TCP stack from the guest operating system and the CEE network hardware. The virtualization layers must ease the creation, deletion and migration of virtual machines. Furthermore the virtual networks connecting different VMs must be isolated from the physical network.

In this chapter we identify the main drawbacks of the overlay networks. We contribute the first completion time-based evaluation of partition/aggregate and 3-tier applications in a realistically virtualized datacenter network, using an actual TCP stack running over a detailed L2 CEE fabric model. We provide the first measurements of the OVN design parameters on the user-perceived performance of on-line and data-intensive applications.

Chapter 4

In Chapter 4 we show that the performance of applications in virtualized networks is harmed by the non-deterministic packet drops in the virtualization layers. We introduce the novel zero-loss Overlay Virtual Network (zOVN) that extends the CEE features described in Chapter 2 into the overlay virtual networks studied in Chapter 3.

We identify and characterize the problem of packet drops in overlay virtual networks. We show that virtual networks are affected by considerable and non-deterministic losses caused by the process and interrupt scheduling within the host OS. We implement the first zero-loss Overlay Virtual Network (zOVN) to address the problem of packet drops in converged multi-tenant datacenters. We quantitatively verify

how zOVN improves the standard TCP performance for data-intensive applications. Testing Partition-Aggregate on top of zOVN, we achieved up to 15-fold reductions in flow completion times using two distinct testbeds with 1G and 10G Ethernet respectively, and three standard TCPs. Finally, we investigate the scalability of zOVN by means of accurate full system cross-layer simulations.

Chapter 5

In Chapter 5 we deconstruct the existing TCP stack from the VMs kernel and consolidate its functions into zFabric, a new hypervisor build around the lossless virtual switch introduced in Chapter 4. We redistribute congestion management responsibilities from the guest OS, which does less, to the hypervisor and virtual NIC, which does more. Next we replace reactive schemes, used in prior work, with a VM-to-VM proactive credit-based buffer reservation scheme, better suited to lossless networks to avoid stalled frames potentially blocking the network.

We contribute a slim hypervisor stack, named *zFabric*, optimized for lossless Ethernet. It avoids HOL blocking – and the ensuing interference between VMs – by managing the buffers between each vNIC communication pair through a VM-to-VM credit-based scheme. For the reliable delivery of both user data and credit messages, zFabric implements a thin reliability scheme on top of the lossless CEE hardware. A deployment of zFabric requires no changes to the applications and to the CEE hardware. We propose *TCPlight*, a slim replacement for the TCP sockets. Although zFabric works with any user transport, optimal results are obtained with the newly introduced lightweight *TCPlight* socket, which is responsible for connection handling and data segmentation. We build a working zFabric prototype and evaluate it using long throughput-bounded transfers and short latency sensitive flows.

2. Converged Enhanced Ethernet: Application Performance Booster

In this chapter we present the new standards implemented in Converged Enhanced Ethernet (CEE) responsible for hardware flow and congestion control. We begin with an evaluation of the influence of these new features on the performance of TCP application. Next, we show that CEE enables the use of innovative source routing schemes that make routing decisions based on information obtained from the load sensors of the hardware congestion control.

Losslessness is one of the consequential new features of emerging datacenter networks, achieved by means of Priority Flow Control (PFC). Despite PFC's key role in the datacenter and its increasing availability – supported by virtually all CEE products – its impact remains largely unknown. This has motivated us to evaluate the sensitivity of three widespread TCP versions to PFC, as well as to the more involved Quantized Congestion Notification (QCN) congestion management mechanism.

As datacenter workloads we have adopted several representative commercial and scientific applications. A somewhat unexpected outcome of this investigation is that PFC significantly improves TCP performance across all tested configurations and workloads, hence our recommendation to enable PFC whenever possible. In contrast, QCN can help or harm depending on its parameter settings, which are currently neither adaptive nor universal for datacenters. To the best of our knowledge this is the first performance evaluation of TCP performance in lossless CEE networks.

Next, we propose two novel source-based adaptive routing schemes exploiting the features of CEE-based. We develop a basic source-driven Reactive Route Control (R^2C^2) adaptive routing scheme. In response to congestion notifications, the source activates additional paths to re-route traffic around potential congestion points. Using industry standard virtual local area networks (VLANs), a source node can effectively control the path choices in the network. This approach goes beyond conventional QCN limitations by replacing its reaction point with a VLAN-based multipath route controller.

Finally, we combine R^2C^2 with the QCN reaction point, resulting in the higher performance Reactive Route & Rate Controller (R^3C^2). In case of persistent or multiple hotspots when VLAN route selection alone is insufficient, the R^3C^2 source

will throttle its packet injection rates individually along each congested route of a multipath bundle. Detailed simulations against established datacenter and HPC benchmarks show the practical benefits in performance and stability.

2.1. Introduction

Besides power, the top three technical obstacles, according to [20], hindering the adoption and growth of Cloud computing are directly related to networking. The first obstacle is confidentiality of private data, such as medical, corporate, or government records. This data is most vulnerable on the network, where security concerns are typically addressed by encryption, VLANs, packet filtering, firewalls, etc. The second obstacle is data transfer bottlenecks, both within and between datacenters. In particular, for latency sensitive applications such as HPC and financial trading, a difference of a few microseconds can significantly affect the cost/performance ratio of the network. When compared against widely used alternatives such as overnight hard disk shipping, datacenter networks need a two orders of magnitude improvement in cost/performance. This requirement affects the entire hierarchy of protocol stacks, communication libraries, adapters, switches, and routers exceeding 1 Gbps. The third obstacle is performance unpredictability, resulting from I/O sharing and interference between scheduling virtual machines. These issues make networking critical for the future of datacenter and Cloud.

Nowadays datacenter networks and HPC installations are composed of multiple disjoint networks: (i) a Local Area Network – Ethernet or Gigabit Ethernet, (ii) a System Area Network – Myrinet, Quadrics or InfiniBand, and (iii) a Storage Area Network – FibreChannel or InfiniBand. Yet the currently distinct networks remain expensive to buy, wasteful to operate, and, complex to deploy/upgrade and manage. While for now 1-10 Gbps Ethernet, 10-40 Gbps Infiniband, 2-8 Gbps Fibre Channel, Quadrics and Myrinet may still coexist in the same Cloud, eventually their traffic will be aggregated on a single network. Consolidation into a single network is the only practical solution to reduce Cloud's cost, complexity and power consumption – a promise now starting to materialize.

The technology recently promoted by the industry and standardized by IEEE as universal network fabric is the Converged Enhanced Ethernet. It provides a unified Layer 2 network that carries all the traffic generated by the applications running in a datacenter using a single physical infrastructure. Upcoming datacenter networks based on 802 CEE are short and fat: up to one million nodes are connected in a single Layer 2 domain with abundant multipathing across 10-100 Gbps links of a few tens of meters (at most). Typical round-trip times (RTTs) range from 1-2 μs up to a few tens of μs, except under hotspot congestion, when the end-to-end delays can grow by several orders of magnitude, reaching into tens of ms [57]. Unlike wide area networks, the datacenter RTT is dominated by queuing delays, which under bursty workloads [16, 27, 26, 33, 72, 111, 117, 42], lead to a difficult traffic engineering and control

problem. Hence the recent surge in research and standardization efforts addressing the new challenges in datacenter virtualization, flow and congestion control, routing and high performance transports.

One of CEE core new features is Layer 2 (L2) losslessness, achieved via per priority link-level flow control as defined by 802.1Qbb PFC [13]. It enables convergence of legacy and future datacenter applications, such as Fibre Channel over Ethernet (FCoE), business analytics, low latency algorithmic trading, high performance network storage, and MPI workloads currently still running on Myrinet and InfiniBand networks. However, the benefits of PFC come at a price: besides the potential for deadlock in certain topologies and switch architectures, it introduces exposure to saturation tree congestion. To counteract the potentially severe performance degradation due to such congestion, IEEE has recently standardized a new L2 congestion control scheme, Quantized Congestion Notification (QCN, 802.1Qau) [11].

2.1.1. Guiding Questions

The bulk of datacenter communications is based on Layer 4 (L4) transports, i.e., predominantly TCP with some notable RDMA, SCTP and UDP exceptions. TCP, has traditionally relied on *loss* as congestion feedback from *uncorrelated single* bottlenecks, whereas in CEE, loss and congestion are avoided by PFC and QCN respectively. Therefore an overarching question is: How disruptive are the new CEE features? We aim to find how do the typical datacenter applications perform in a CEE environment. To this end, we ask the following question:

(Q1) How does TCP perform over CEE networks? Is PFC – with its potential saturation trees – beneficial or detrimental to TCP? Is QCN beneficial or detrimental to TCP?

Next, we observe that although Ethernet is not source routed, alternative paths are possible, with the condition that they belong to different VLANs. The source can determine the route towards destination by pre-selecting the VLAN number before injecting a new frame. We aim to answer the following question:

(Q2) Assuming QCN-compliant adapters will follow an accelerated CEE-adoption curve, can we reap any sizable benefits by combining VLAN-based route control with QCN rate control? Our aim is to design a "route & rate" controller using QCN's feedback as the congestion price, while avoiding the interference and potential instabilities of two intercoupled control loops, with their respective timescales. QCN is a rate, not route, control feedback loop, therefore stability and performance of source-driven route changes are unknown in 10-100Gbps. Next, can a source-based adaptive routing scheme show performance benefits with simpler adapters that do not implement QCN?

In addressing these questions we hope to provide some useful guidance to datacenter architects, network vendors, as well as operating system, hypervisor and application designers.

2.1.2. Contributions and Structure

The contributions of this chapter are as follows:

1. We extracted a full TCP stack from FreeBSD v9 and ported it to our simulation environment. Therefore we mix production-ready software with detailed L2 simulation models of CEE switches and adapters. On top of the TCP stack we run different workloads representative for datacenter traffic.
2. We perform the first evaluation of TCP performance in lossless CEE networks. First, we contribute the necessary parameter changes for TCP over 10Gbps CEE. Next, we present evidence of PFC's benefits to TCP, leading to our recommendation to *enable PFC also for TCP* applications. In addition, we identify cases in which QCN is beneficial, respectively detrimental, to L4 transports.
3. To the best of our knowledge, this is the first source-based adaptive routing, proposed for CEE, using the established VLAN mechanism in conjunction with the recently standardized QCN. We introduce a simple *reactive route control* R^2C^2, and respectively, its higher performance counterpart, a *reactive route & rate control* R^3C^2.
4. Finally we quantitatively examine performance of our proposals against previously introduced routing algorithms for CEE.

The remainder of this chapter is structured as follows: The datacenter network stack is briefly described in Section 2.2. In Section 2.3 we review the routing schemes selected as candidates for CEE-based datacenter interconnects. Next, in Section 2.4 we introduce our proposed adaptive routing algorithms. Section 2.5 describes the simulation environment, network models, and the workloads and traffic scenarios, that we have used to validate, and to quantitatively examine performance at application level. We answer to the first guiding question in Section 2.6 and to the second guiding question in Section 2.7. Finally, we present a selection of some related work in Section 2.9, after which we conclude in Section 2.10.

2.2. Datacenter Network Stack

2.2.1. Layer 1 Topology - Fat-Trees

Future datacenters will typically consist of 1K–100K processing nodes interconnected by a blend of 1-10-100Gbps Ethernet networks. The classical topology for datacenter networks follows a tiered approach as shown in Figure 2.1. The processing nodes are connected to edge switches that provide the connectivity between the nodes collocated in the same rack. The edge switches in turn are connected to an intermediate layer of aggregation switches that connect different racks together to

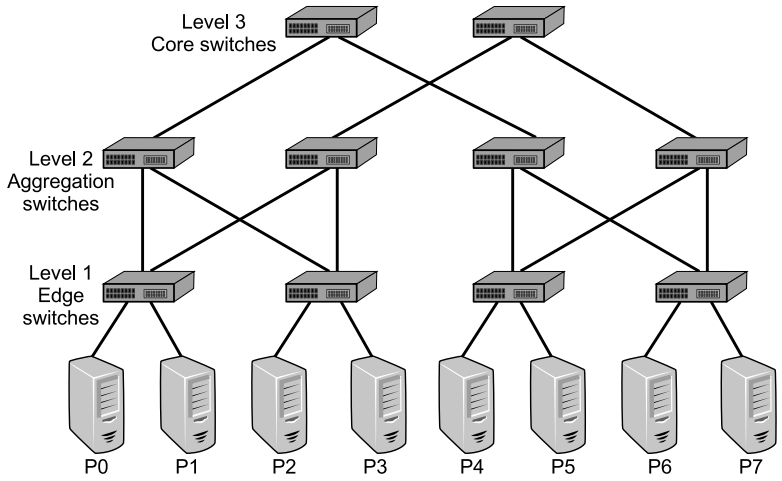


Figure 2.1.: Multi-tiered datacenter with edge, aggregation and core switches.

form a cluster. The clusters can be further linked through another layer of core switches [14, 89].

The packets generated by one of the processing nodes have to traverse a few levels of switches before reaching the destination nodes. A common multi-stage interconnect is the fat-tree or k-ary n-tree.

A k-ary n-tree consists of $N = k^n$ processing nodes and $n \cdot k^{n-1}$ switch nodes. The switch nodes are organized on n levels, each level having k^{n-1} switches. All switches have the same arity $2k$, excepting the top switches, which have arity k. This type of network has full bisection bandwidth and path redundancy [94]. The k-ary n-trees can be slimed by populating the upper layers with less than k^{n-1} switches. A slimmed fat-tree is cheaper to build because it requires fewer core and aggregation switches but it doest not provide full bisection bandwidth. An example of a 2-ary 3-tree can be seen in Figure 2.2, and a slimed versions of the same tree is shown in Figure 2.1.

The k-ary n-trees and their slimmed versions belong to the family of extended generalized fat-trees (XGFT) as described in [91]. An XGFT $(h; m_1, ...m_h; w_1, ...w_h)$ has $h+1$ levels of nodes. The nodes on level 0 are called leaf-nodes and the nodes on level h are called root-nodes. The processing nodes occupy the $\prod_{i=1}^{h} m_i$ leaf-nodes on level 0. The switches occupy the other nodes on levels 1 to h. Each non-leaf node on level i has m_i child nodes and each non-root node on level j has w_{j+1} parent nodes.

A deadlock free path in a fat-tree is computed by selecting an intermediate switch from the set of Nearest Common Ancestors (NCA) of the source and the destination node [93]. A NCA is a common root of both the source and the destination located at the lowest possible level. Packets are following an up-phase from the source to the NCA, and then a down-phase from the NCA to the destination node.

For example in Figure 2.5, to send data from source P0 to source P7, there are two

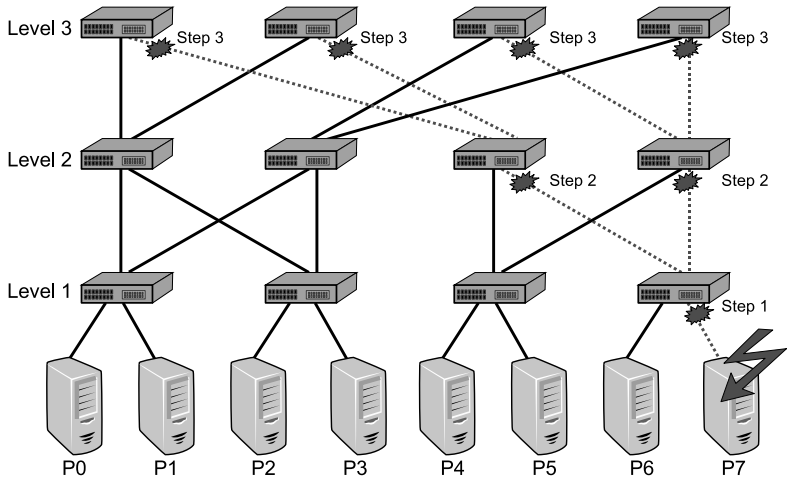


Figure 2.2.: Saturation tree formation in a fat-tree network. End-node P7 is slow in processing incoming packets sent from nodes P0 ... P3. In the first step, congestion appears on the edge link connecting P7 with the corresponding level 1 switch. In the second step, the dashed links from level 2 to level 1 are affected. In the third step, congestion propagates up to root level. Other flows that do not even target P7 are now potentially affected (e.g. P3 → P4). If the initial hotspot persists long enough, the domino effect created by the link-level flow-control continues and the congestion propagates further back to level 2 and level 1 (not drawn). The network experiences a catastrophic loss of throughput affecting all the nodes.

NCAs: the two switches on level 3. Packets leaving source P0 travel upward until they reach one of the level 3 switches, then downward to destination P7.

2.2.2. Layer 2 - Converged Enhanced Ethernet (CEE)

There is a growing interest in consolidated network solutions that meet the requirements of datacenter applications, i.e., low latency, no loss, burst tolerance, energy efficiency etc. One possible universal datacenter fabric is Converged Enhanced Ethernet (CEE) with the following key features: (i) per-priority link-level flow-control and traffic differentiation, i.e., Priority Flow Control (PFC; 802.1Qbb) [13]; (ii) congestion management, i.e., Quantized Congestion Notification (QCN, optional in CEE; 802.1Qau) [11]; (iii) transmission scheduling, i.e., Enhanced Transmission Selection (ETS; 802.1Qaz) [12].

2.2.2.1. Priority Flow Control (PFC)

Traditionally Ethernet does not guarantee lossless frame reception; instead, packets will be dropped whenever a receive buffer reaches its maximum capacity. Reliable upper-layer transports such as TCP interpret this event as implicit congestion feedback triggering congestion window or injection rate corrections. This lossy network

behavior, however, does not meet the semantics of applications such as Fibre Channel over Ethernet, MPI or low-latency messaging for Business Analytics.

CEE networks are specifically designed to prevent frame losses, by using a link-level flow-control mechanism named Priority Flow Control (PFC), defined in the IEEE standard 802.1Qbb [13]. It works by pausing the transmission on an input link when the corresponding buffer occupancy exceeds a certain maximum threshold. The transmission is paused using a special XOFF control frame sent to the upstream device. When the buffer occupancy drops below a minimum threshold, the transmission is resumed using a XON control frame sent to the upstream device.

The desired effect is that the congestion information is propagated from the congestion point to the upstream devices. Hence, eventually the core congestion is pushed to the network edge. On the other hand, when a link is paused, the buffers of the upstream device fill up and new upstream links will have to be paused. This has the potential to continue recursively affecting more and more devices. Therefore if the congestion persists, it can spread from one network device to another forming a congestion tree [96, 97]. Previous studies [96] showed that a congestion tree can fill all the buffers in only a few round-trip times, too fast for software to react. An example about how a congestion tree can form in a network is shown in Figure 2.2. This undesired effect of link-level flow-control can cause a catastrophic loss of throughput of the entire network. To make the situation worse, after the original congestion subsides, the congestion tree dissipates slowly, because all the buffers involved must first drain [57].

PFC divides the traffic into 8 priority classes based on the IEEE 802.1p Class of Service field. Each priority class has its own buffer and link-level flow-control protocol. Congestion generated by a priority class does not influence the other priority classes.

2.2.2.2. Quantized Congestion Notification (QCN)

QCN, a congestion management scheme that attempts to match a source node's injection rate to the available capacity, as signaled by the QCN-monitored switches, see [79, 83] for a full description.

QCN is defined in the IEEE standard 802.1Qau [11]. The final version of the standard provides a set of protocols and procedures for congestion management of long-lived data flows. QCN-compliant switches can detect and signal congestion to the end-nodes via explicit congestion feedback. The QCN-capable end-nodes respond to the congestion information by limiting their transmission rate to the available network capacity.

QCN consists of two algorithms:

(i) Congestion detection: An instantaneous queue load sensor is implemented at each potential congestion point, e.g. switch buffer. Each congestion point sensor observes

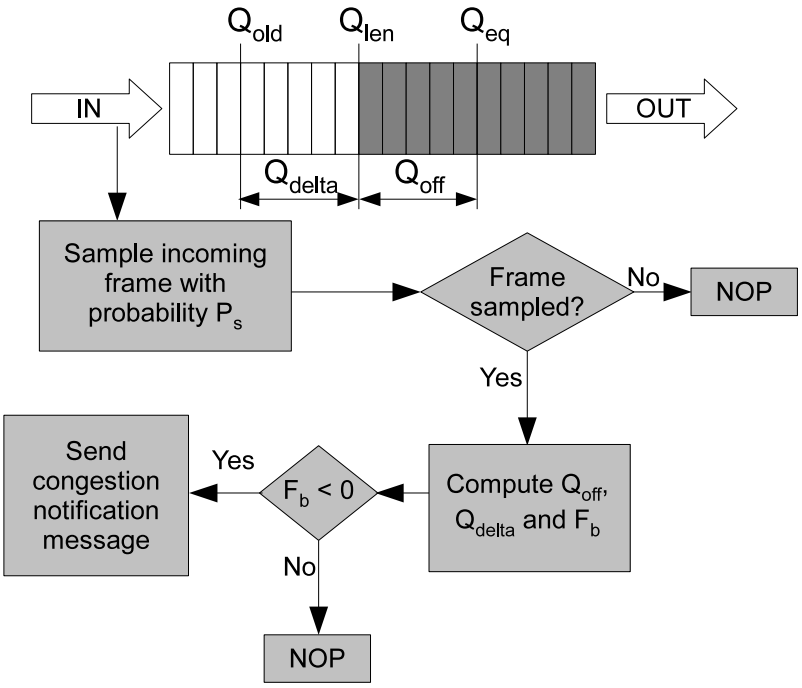


Figure 2.3.: QCN load sensor mechanism. The sampling rate P_s is a function of the measured feedback F_b. For low congestion levels, one every 100 received frames is sampled. Sampling rate increases linearly with the severity up to one every 10 received frames.

the state of the network and detects congestion by sampling the incoming frames with a variable and randomized sampling rate (see Figure 2.3). For each frame sampled, the switch measures the output queue occupancy and computes a feedback value F_b. The feedback value is computed as $F_b = -(Q_{off} + w \cdot Q_{delta})$. Q_{off} is the difference between the measured queue occupancy Q_{len} and an equilibrium queue occupancy Q_{eq}, considered normal during the operation: $Q_{off} = Q_{len} - Q_{eq}$. Q_{delta} is the change of the queue occupancy from the preceding sample instant: $Q_{delta} = Q_{len} - Q_{old}$. If the computed F_b is negative, the switch generates a 64B congestion notification message, sent back to the end-point that generated the sampled frame, i.e., culprit source. The congestion notification informs the source about the hotspot, essentially conveying its location via the Congestion Point ID, whereas the feedback value F_b provides a 6-bit quantitative indication of how severe the bottleneck is. When a higher precision is required, the congestion notification also entails two 16-bit values, i.e., the raw queue offset and delta.

(ii) Source reaction: This is a mechanism by which the source limits its transmission rate in response to the congestion notifications received from the QCN-enabled switches. When a notification is received, the source instantiates a Rate Limiter (RL) that adjusts the transmission rate according to the feedback received: the higher the feedback, the higher the rate reduction. The RL also provides a way to recover: if no congestion notification has been received for a certain number of sent frames, it can be assumed that congestion has vanished and the source can increase

its transmission rate.

The above description shows that the QCN algorithm matches the transmission rate of an end-node with the available bandwidth in the network. Unlike in earlier proposals (Ethernet Congestion Management – see [83] for a full description) there is no positive feedback in QCN. Hence, the source has to recover the bandwidth autonomously. This rate recovery is performed in three phases. In the first phase (Fast Recovery), a few binary increase steps are performed similar to BIC-TCP [120]. In the second phase (Active Increase), several linear increase steps take place, followed by an optional superlinear increase regime (Hyper-Active Increase) in the third phase.

Despite the merits of a datacenter-tailored congestion management solution, QCN has also been criticized because of its:

(i) **Increased switch and adapter complexity**: QCN's congestion point sensor runs at line speed, 10-100 Gbps, at each monitored queue ($ports \times priorities$). The hundreds of high-speed congestion point engines per CEE switch add complexity, power and costs. Similarly, QCN's reaction point requires a line-speed rate controller, hence a new scheduling stage – a potentially large set of Rate Limiters (RL) in QCN terminology – to be added per priority to every network adapter. The tens to thousands of RL engines per adapter increase the cost and power budget, expected to grow with virtualization.

(ii) **Lack of application control**: HPC applications assume end-nodes' direct control of task placement and routing, without any interfering form of rate control by congestion management schemes such as QCN. Increasingly often since MapReduce, Hadoop, Dryad, the datacenter application developers prefer to explicitly control their workload, its injection, routing and service level agreement (SLA) monitoring. At least for the critical traffic classes (analytics, trading, mission-critical applications mapped to high priorities), one must ensure that the Layer 2 traffic management modules do not conflict with the higher-layer SLA. Often, to prevent interference with the main application and its communication stack, the QCN reaction point is disabled – thus orphaning the investments in QCN-compliant networks. In either case the source end-node must actively control the routing and/or the injection rate, ideally in conjunction with the application, communication library, operating system and virtual machine. Currently the QCN operation on Layer 2, albeit fast, remains opaque from an application and virtual machine perspective.

2.2.2.3. Enhanced Transmission Selection (ETS)

The Enhanced Transmission Selection mechanism provides a framework to support bandwidth allocation to traffic classes. This is needed because different classes have different bandwidth and latency requirements. For time-sensitive applications requiring minimum latency, a strict priority scheduling is needed. Active priorities that generate bursty traffic can share bandwidth. When a priority is not using its

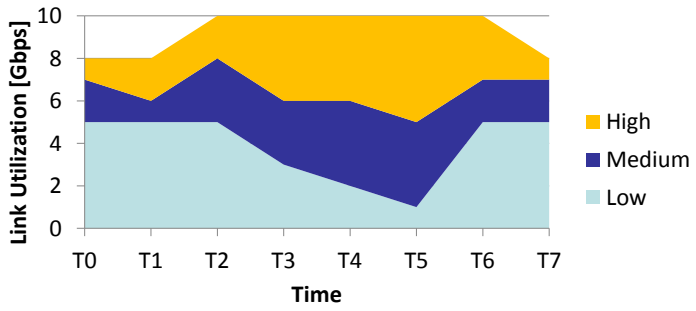


Figure 2.4.: Enhanced Transmission Selection bandwidth allocation for a 10Gbps link with three priorities: low, medium and high. The offered traffic intensity for the low-priority is constant: 5Gbps. Between T2 and T6 the intensity of the medium and high-priority traffic is higher than 5Gbps, thus reducing the capacity left to the low-priority. The low-priority bandwidth share shrinks under the pressure of the higher priorities. The flows mapped to the low-priority encounter an artificial bottleneck.

allocation other priorities can use the bandwidth. An example of the operation of ETS in a datacenter with three priority classes is depicted in Figure 2.4. Note how the low-priority traffic bandwidth slice shrinks under the pressure of the high-priority traffic.

2.2.3. Layer 3 - Explicit Congestion Notifications (ECN)

Random Early Detection [49] (RED) is a Layer 3 Active Queue Management (AQM) congestion avoidance technique for packet-switched networks. Unlike QCN whose congestion estimation is based on instantaneous queue size measurements, RED detects network congestion by computing the average queue length and comparing it with a given threshold. The RED-enabled queue has a minimum and a maximum threshold. If the average queue length is below the minimum threshold, all incoming packets are forwarded unchanged. If the average queue length is above the maximum threshold, all the incoming packets are dropped. Finally, if the average queue length is between the threshold values, some of the incoming packets are dropped according to a linear probability which is function of the average queue length. RED allows the network to absorb a limited amount of bursty traffic with little performance degradation. Unlike the QCN load sensor mechanism which uses the *instantaneous* queue length, RED uses the *average* queue length to detect congestion, therefore being more tolerant to bursts as confirmed later in this chapter.

Explicit Congestion Notification (ECN) is a Layer 3-4 end-to-end congestion management protocol defined in RFC 3168. In order to be operational and efficiently used, ECN has to be supported by both endpoints and also by the intermediate network devices. ECN uses the two least significant bits of the Differential Services (DiffServ) field in the IP header. Once the communicating end-nodes have negotiated ECN, the transport layer of the source node sets the ECN-capable code in the IP header of the packet and sends the packet to the destination. When the ECN-

capable packet arrives at a RED-enabled queue that is experiencing a congestion, the router may decide to mark the packet instead of dropping it. Upon receiving the marked packet, the destination sends back to the source an ACK packet with the ECN-Echo bit set in the TCP header. The destination repeats sending the ECN-Echo bit until the source acknowledges having received the congestion indication.

2.2.4. Layer 4 - TCP Congestion Avoidance Algorithms

We have selected three representative TCP congestion avoidance algorithms:

1. TCP New Reno [48] - the most studied and the most implemented version;
2. TCP Cubic [60] - the default congestion avoidance in today's Linux kernels;
3. TCP Vegas [30] - uses delay probing for congestion window adjustments.

TCP New Reno, like Reno, includes the slow-start, congestion avoidance, and fast recovery states. Its congestion feedback is either packet loss and/or ECN-marked packets. TCP New Reno outperforms TCP Reno in the presence of multiple holes in the sequence space, but performs worse in case of reordering due to useless retransmissions. It was the default TCP algorithm in Linux kernels till version 2.6.8.

TCP Cubic is RTT-independent. It has been optimized for high speed networks with high latency (due to flight delays) and is a less aggressive derivative of TCP BIC (Binary Increase Congestion control). TCP BIC uses a binary search to probe the maximum congestion window size. TCP Cubic replaces the binary search with a cubic function. The concave region of the function is used to quickly recover bandwidth after a congestion event happened, while the convex part is used to probe for more bandwidth, slowly at first and then very rapidly. The time spent in the plateau between the concave and convex regions allows the network to stabilize before TCP Cubic starts looking for more bandwidth. TCP Cubic replaced TCP BIC as the default TCP implementation in Linux kernels from version 2.6.19 onwards.

While both TCP New Reno and TCP Cubic rely on losses to detect congestion and react accordingly, Vegas avoids congestion by comparing the expected throughput in the absence of congestion with the actually achieved throughput and then it adjusts the transmitter window size accordingly. TCP Vegas is representative for the delay-probing class of TCPs similar to Adaptive Reno and Compound TCP.

2.3. Routing in Converged Enhanced Ethernet

In the following paragraphs we will describe the routing schemes selected as the most promising for Cloud apps. We assume networks based on k-ary n-trees.

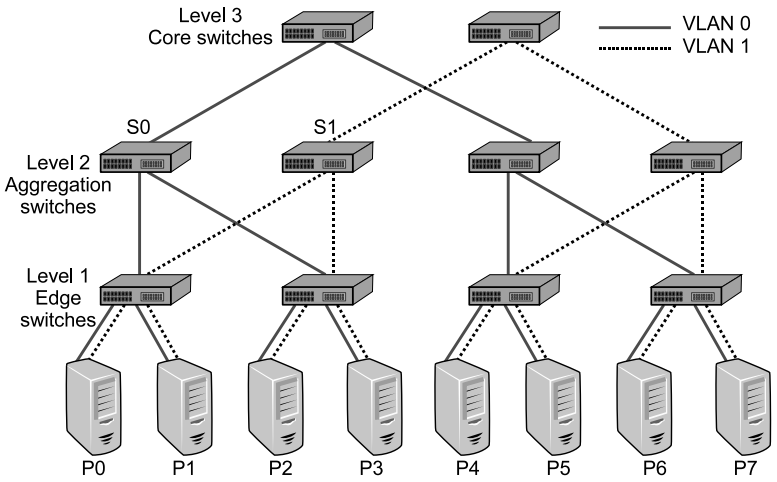


Figure 2.5.: VLAN source routing. To send data from source P0 to destination P3 two paths are available: through switch S0 or through S1. Source P0 selects the path via VLAN 0 or VLAN 1.

2.3.1. Source Routing Using Virtual LANs

Traditional Ethernet networks guarantee loop-free routing using the Spanning Tree Protocol (STP). The STP algorithm transforms a physical network that may contain loops into a loop-free single-rooted tree graph by disabling the loop-inducing edges. Hence, even if the original topology has path diversity, e.g. fat-tree (inherently loop-free with up/down routing), STP will reduce it to a unique path network, thus also reducing its performance and reliability.

We remove this limitation by using the VLAN mechanism defined by IEEE standard 802.1q. This enables the coexistence of multiple spanning trees, one per VLAN. Thus alternative paths are possible, with the condition that they belong to different VLANs.

A method of VLAN assignment and route selection in k-ary n-trees was introduced in [73, 87], which assigns a VLAN number to each top-level switch. The corresponding VLAN contains that particular top-level switch, plus all its 'children': switches accessible from it through descending links, including the end-nodes.

As depicted in Figure 2.5 each end-node is configured as member of all the VLANs. Before injecting a new frame, a source has to pre-select the desired VLAN, which determines the route to destination. Thus one can implement source routing without changing the Ethernet frame format, nor the switch VLAN routing mechanism.

With 4096 VLANs, assuming future CEE switches with arity 64, the network scales up to 100K nodes. Next, 802.1ad provides two VLAN tags per Ethernet frame, scaling beyond 1M nodes; such physical scalability, however, is limited by other constraints, e.g. power and cooling.

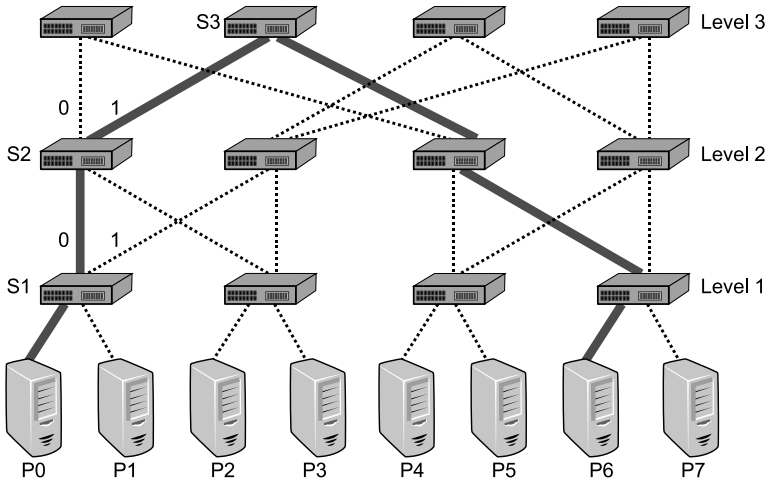


Figure 2.6.: *D-mod-k* routing in a 2-ary 3-tree ($k = 2$). Source P0 sends a packet for destination P6 ($D = 110$). The packet arrives at switch S1 at level 1, which computes $\left\lfloor \frac{D}{k^0} \right\rfloor \bmod k = 0$ and selects parent #0. Then the packet arrives at switch S2 at level 2, which computes $\left\lfloor \frac{D}{k^1} \right\rfloor \bmod k = 1$ and selects parent #1. The packet reaches the root switch S3, which is NCA for P0 and P6. From the NCA, there is a single downward path available to the destination P6.

2.3.2. Deterministic Routing

Deterministic routing always uses a single fixed path from a given source S to a given destination D. The choice of paths is done such that the load is distributed evenly across the switches that act as Nearest Common Ancestors (NCA) between different sources/destinations.

An extensively studied deterministic routing technique is the modulo-based *D-mod-k* routing [78, 69, 121] also know as Stage And Destination Priority - SADP [52, 53]. To establish a path $S \rightarrow D$, the algorithm chooses the parent $\left\lfloor \frac{D}{k^{l-1}} \right\rfloor \bmod k$ at the level l in the upwards phase of the routing until a NCA is reached. An example of *D-mod-k* routing is given in Figure 2.6. The NCA choice is dictated by the destination address. Consequently flows with different destinations use different NCAs and the traffic sent to different destinations is distributed statically over alternative paths.

Another approach uses the source address in the choice of the NCA. This is accomplished by the modulo-based *S-mod-k* routing that chooses the parent $\left\lfloor \frac{S}{k^{l-1}} \right\rfloor \bmod k$ at every level l in the upwards phase. Using this algorithm the flows with different sources use different NCAs.

Various studies [52, 104] proved that *D-mod-k* is one of the best performing deterministic routing algorithms. Additionally it has the advantage of in-order delivery, hence no need for resequencing buffers at the destination. Nonetheless, its throughput can suffer because of resource conflicts and head-of-line blocking. It is always possible to find particular traffic patterns under which two or more flows contend on the same link. Such conflicts are unavoidable owing to the static nature of the

algorithm [104, 53].

2.3.3. Random Routing

Random routing [114, 47, 76] uses all available paths from S to D with equal probability. This approach distributes the loads across the different links and switches.

The Valiant routing algorithm [114] states that, in a network with an arbitrary topology, for each packet from S to D, a random intermediate node R must be selected and the packet is then routed along the path $S \rightarrow R \rightarrow D$. In fat-trees the role of the intermediate nodes is taken by the NCAs. To route a packet from S to D, a random NCA is chosen and the packet sent through that NCA. The choice of NCA can be done at the source, as in [47], or at each step of the upward phase as in the Connection Machine CM-5 [76].

Misordering is possible, hence any traffic type that requires in-order delivery needs a resequencing buffer at the destination. The throughput may be reduced by uneven loading of the alternative paths. If one of the NCAs is loaded more than others, still it will receive the same amount of traffic, because the division is statical.

2.3.4. Hash-based Routing

Hashed routing is a special case of Equal-Cost Multi-Path routing detailed in [63, 113]. In hashed routing, each flow from S to the D is characterized by a flow identifier. The source uses a hash function that takes as input a flow identifier and outputs a path selected from the set of alternative paths to the destination. The flow is usually identified by a 5-tuple containing the source and destination address (Layer 2 or 3), the source and destination port, and the protocol number.

For fat-trees, the number of alternative paths is determined by the number of NCAs. Hence the hash function has to select an NCA for each flow identifier given as input. This distributes the flows evenly over different links. Since all packets of a flow follow the same path, as in deterministic routing, no resequencing is required. Care is still needed for a flow level ordering.

Hashed routing performs similar to random routing for sources that generate a large number of “mice” flows, which will select different paths because of hashing, thus the load is distributed across the network. On the other hand, if the number of flows between S and D is small, hashing degenerates into deterministic routing and a single path will be used.

2.3.5. Switch Adaptive Routing

Switch Adaptive Routing (Switch AR) [84, 86] uses the QCN congestion information to steer the traffic. Switches are QCN-enabled and continuously monitor the status

of their outbound queues. If congestion is detected in one of these queues, the switch generates a congestion notification that travels upstream from the congestion point to the originating source of the packets deemed as culprits.

Congestion notifications are snooped by the upstream switches, which thus learn about the downstream congestion. When a switch detects congestion, it can reroute the traffic to alternative paths, to avoid the hotspots, and to allow the congested buffers to drain. In this way, the path diversity is exploited, theoretically better than by the load-oblivious schemes.

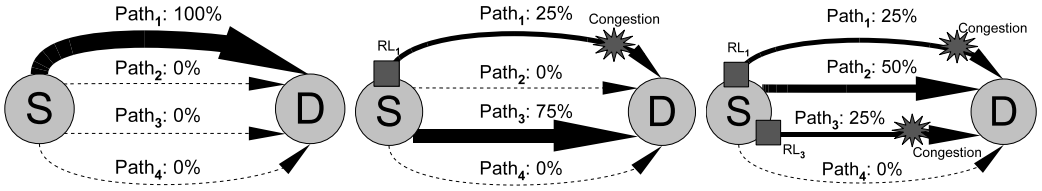
If the new path, however, is also congested, the Switch AR algorithm will revert to the original path, hence oscillations are possible. These are likely to appear in networks with multiple hotspots or, when multiple flows contend as observed in [51].

The switch uses the snooped congestion information to annotate its routing table with a congestion level for each port through which a given destination is reachable. When a frame for this destination arrives, it will be routed through the port with the minimum congestion level. By marking the ports as congested with respect to each destination, the switch reorders its routing preferences to favor the uncongested ports. The algorithm must ensure that all the upstream switches learn about congestion. Congestion notifications are routed randomly towards the culprit source. Thus all the upstream switches of all the alternative paths can detect the hotspot.

The Switch AR algorithm is using binary route split ratios. When congestion is detected on one path, the entire traffic flow is switched to an alternative path. The advantage is in simplicity and low-cost. The resources required are minimal because only some additional per-port data needs to be stored [84, 86]. Another advantage is that the switch forwarding logic is unchanged. Only the routing table is updated in response to congestion notifications.

One can argue that the binary split can lead to oscillations. In order to avoid oscillations, fractional route split ratios can be used: in response to congestion only a fraction of the traffic is rerouted to an alternative path. However, this comes at a higher cost, as the switch will have to store per-flow information. Extensive changes to the forwarding logic are required. For example, in order to divert 20% of the traffic to an alternative path, every 10 packets, 2 packet have to be rerouted. We are currently working to improve Switch AR's stability by a less aggressive load re-routing decision while keeping the hardware requirements as low as possible.

Since Switch AR is a route-only scheme, we also want to test its coupling with a congestion management scheme such as the QCN. Therefore we devise a new version, called Switch AR with Rate Limiters, whereby QCN compliant rate limiters can be instantiated at each source, for each "culprit" flow. These flows are identified solely using the destination address carried by the congestion notifications. When a congestion notification is received, a rate limiter is instantiated at the flow source. The algorithm used for rate limitation is identical with the one described in the 802.1Qau standard.



(a) (b) (c)

Figure 2.7.: R^3C^2 – Reaction to congestion. Four paths are available between S and D: $Path_1, Path_2, Path_3$ and $Path_4$. The default path that is $Path_1$.In the absence of congestion, 100% of the traffic from S to D takes the default path as in **(a)**. In case of congestion this path's load must be reduced, done by instantiating the rate limiter, RL_1, according to the received congestion notification feedback. An additional path, $Path_3$, is activated for the excess load (see **(b)**) which should eliminate the original bottleneck on $Path_1$. Assuming that the load on $Path_1$ was reduced to 25%, the excess 75% is routed on $Path_3$. However this may in turn, generate a hotspot on $Path_3$. Hence, RL_3 must be instantiated while a new alternative, $Path_2$, is activated as in **(c)**.

2.4. Reactive Source-based Adaptive Routing

In this section we describe the proposed R^2C^2, respectively the R^3C^2 algorithms. The former is a route-only scheme that in response to congestion notifications can exploit the available path diversity of the network – even when the source adapter is not QCN-compliant. The latter scheme, R^3C^2, combines the R^2C^2 scheme with QCN rate control. Both algorithms are reactive: traffic sources react to hotspots, signaled by QCN-compliant switches via congestion notifications. Route reaction is by steering the traffic away from the overloaded paths toward the remaining, if any, uncongested paths.

2.4.1. Concept and Assumptions

At initialization time, the traffic sources learn the network topology to discover alternate paths. In the absence of congestion, a unique default path is used – selected using a deterministic algorithm with static load balancing. Whenever a congestive event is signaled, the notified source will first attempt to re-route, i.e., its excess demand will be transferred to the additional alternative paths. Therefore, under persistent congestion, new paths are *incrementally* activated by the 'culprit' source – thus diverting its load away from the initial hotspot, as characterized by a congestion-point ID (location) and feedback value (severity). When all the paths are exhausted, if necessary, rate control will be activated: a basic form of application flow control or TCP for R^2C^2, or, selective QCN for R^3C^2.

The R^3C^2 scheme benefits from QCN-compliant adapters with Rate Limiters (RL). In addition to re-routing, it can also activate a RL for each path in use that is

Algorithm 2.1: R^2C^2 transmission from source S to destination D

```
globals: Q, AvailablePaths, DefaultPath, PathsInUse, CurrentRate
on initialization
    Q.initialize()
    AvailablePaths ← Routing(D)
    DefaultPath ← DeterministicRouting(AvailablePaths)
    PathsInUse ← {DefaultPath}
end
on congestion notification arrival
    if AvailablePaths\PathsInUse not empty then
        NewPath ← SelectPath(AvailablePaths \ PathsInUse)
        PathsInUse ← PathsInUse ∪ {NewPath}
    end
end
on frame received from upper layers
    Q.enqueue(frame)
end
on Path_i ready to send
    if not Q.empty then
        Frame ← Q.dequeue
        send Frame on Path_i
    end
end
on timer
    OldRate ← CurrentRate
    CurrentRate ← flow S → D rate
    if OldRate > CurrentRate then
        reduce number of paths
    end
end
```

signaled as congested. The injection rate can be controlled independently per path.

To conserve resources, the R^2C^2 deactivates its paths based on load reduction, after a timeout. The R^3C^2 attempts to deactivate paths based on their respective RL state.

2.4.2. Pseudocode

Source S sends to destination D. S stores the packets to be sent to D in a transmission queue Q. By calling the routing function *Routing*, S obtains the set $AvailablePaths = Path_1, Path_2, ..., Path_n$. Initially, the *DeterministicRouting* function will select a $DefaultPath$ from the set. The paths currently in use by S are stored in the $PathsInUse$ subset. Unlike R^2C^2, whose global rate control is performed by application's flow control, the R^3C^2 algorithm can instantiate independent rate limiters RL_i for each $Path_i$.

The pseudocode is shown in Algorithm 2.1 and Algorithm 2.2, together with an

Algorithm 2.2: R^3C^2 transmission from source S to destination D

globals: Q, AvailablePaths, DefaultPath, PathsInUse, RL
on *initialization*
  Q.initialize()
  AvailablePaths ← Routing(D)
  DefaultPath ← DeterministicRouting($AvailablePaths$)
  PathsInUse ← $\{DefaultPath\}$
  foreach $Path_i \in AvailablePaths$ **do**
    RL_i ← null
  end
end
on *congestion notification arrival from* $Path_i$
  if RL_i **is** *null* **then**
    RL_i.initialize()
  end
  if ($AvailablePaths \backslash PathsInUse$ **not** *empty*) **and** ($\forall Path_i \in PathsInUse$, RL_i **not** *null*) **then**
    NewPath ← SelectPath($AvailablePaths \setminus PathsInUse$)
    PathsInUse ← PathsInUse ∪ {NewPath}
  end
end
on *frame received from upper layers*
  Q.enqueue($frame$)
end
on $Path_i$ ***ready to send*** **and** (RL_i **is *null*** **or** (RL_i **not *null*** **and** RL_i ***ready to send***))
  if not *Q.empty* **then**
    Frame ← Q.dequeue
    send Frame on $Path_i$
  end
end
on *timer*
  if $|PathsInUse| > 1$ **and** $\exists Path_i \in PathsInUse$ *such that* RL_i **not** *null* **then**
    reduce number of paths
  end
end

R^3C^2 example in Figure 2.7. In response to congestion notifications, the R^3C^2 algorithm updates two variables: (i) *number of active paths,* increased when congestion notifications are received / decreased with a time-out after the last notification received; (ii) *per path injection rate*, controlled by the respective RL_i. The simpler R^2C^2 can change only the number of active paths.

2.4.3. Hardware Requirements

Traffic sources need to be topology-aware, namely to store the set of VLAN IDs usable to reach the required destinations. As this feature is also required for other functions, e.g. security and traffic segregation, the additional cost is modest. Next, for both proposals, an adapter must detect the new EtherType of the

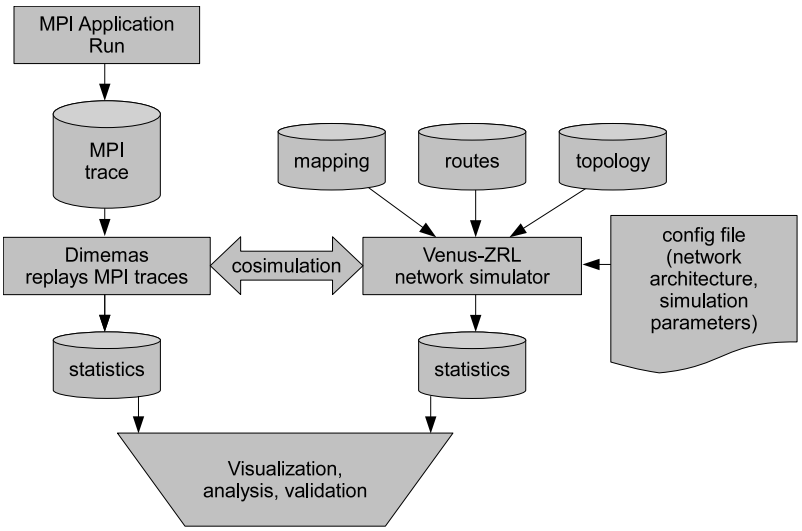


Figure 2.8.: The structure of the Venus [85] simulation environment. The MPI applications are run on a real parallel machine. Traces of the MPI calls are stored in files, which are replayed by the Dimemas simulator. Venus initializes the simulation using the provided topology, routes, mappings and configuration file. The messages generated by Dimemas are transported by the network simulated in Venus and eventually returned to Dimemas. Both simulators output statistics that can be visualized with specific tools and used for analysis and validation.

incoming QCN frames, and implement the route control mechanism. Whereas an R^2C^2 adapter needs no other QCN support, thus sparing the cost and delays of QCN rate limiters, the R^3C^2 adapter relies on the rate limiters for rate control ($\#active\,flows \times average\,path\,diversity$). No changes are made to the switching fabric and the Ethernet frame format.

2.5. Evaluation Methodology: Environment, Models, Workloads

Our simulation environment, entails two simulators coupled in an end-to-end framework: Dimemas and Venus [85]. We have decided to port a full TCP stack from a production-ready operating system to our environment. Next, we calibrate the parameters of the resulting model agains the actual OS stack running on real hardware.

2.5.1. Simulation Environment

Venus is an event-driven simulator developed at IBM Research – Zürich, capable of flit level simulations of processing nodes, switches and links. It is based on OMNeT++ [116], an extensible, C++ simulation library. It was developed as an exten-

sion of the Mars network simulator [43]. Venus supports various network topologies such as fat-trees, tori, meshes, and hypercubes. It can simulate different network hardware technologies, such as Ethernet, InfiniBand, and Myrinet. Additionally, it can also model irregular networks topologies and new types of hardware.

Dimemas is a Message Passing Interface (MPI) simulator, developed at Barcelona Supercomputing Center, that models the semantics of the MPI calls. The two simulators communicate through a co-simulation interface. When an MPI message is produced, Dimemas passes the message to Venus, which models the segmentation, buffering, switching, routing, reassembly and eventually delivers the message back to Dimemas.

A brief scheme of the simulation environment is shown in Figure 2.8; a more detailed description can be found in [85]. Moreover, Venus can operate as a standalone simulator; in this case the traffic is generated by synthetic traffic sources used to simulate various traffic patterns, such as Bernoulli, bursty, on/off or Markov traffic. Also we can simulate communication patterns ranging from simple permutations to more complex sweeping hotspot scenarios. The simulation environment has already been tested in InfiniBand, Myrinet and 802/CEE simulations.

2.5.2. Network Models

The network we are modeling in Venus has the following components:

(1) Processing nodes – The processing nodes are the sources and destinations of the network traffic. They are assumed to have an infinite bandwidth link with the network adapters. In the processing nodes, we gather statistics such as the delay. They are computed as the difference between the time the packet was generated at the source and the time it was received at the destination. Thus we make sure the simulator also accounts for the time spent by the packets waiting before entering the network.

(2) Network adapters – The network adapters are responsible for link-level flow control and the source reaction algorithm for congestion management. Out-of-order arrivals are resequenced in the receive buffer. The input adapters provide one virtual output queue (VOQ) for each destination. This avoids most of the head-of-line blocking, which can be further exacerbated by the QCN rate limiters.

(3) Switches – The switches are network devices that transfer packets from their input links to the output links. They are responsible for link-level flow control, contention resolution and congestion detection. In switch-controlled routing schemes, also the routing decisions are made by the switches.

Switches are modeled as ideal input-buffered output-queued switches (IBOQ) – consistent with 802.1Qau's choice. When a packet arrives on an input link, it is buffered in the input buffer associated with that link. Simultaneously, an output port is selected according to the routing algorithm in use. The incoming packet is enqueued

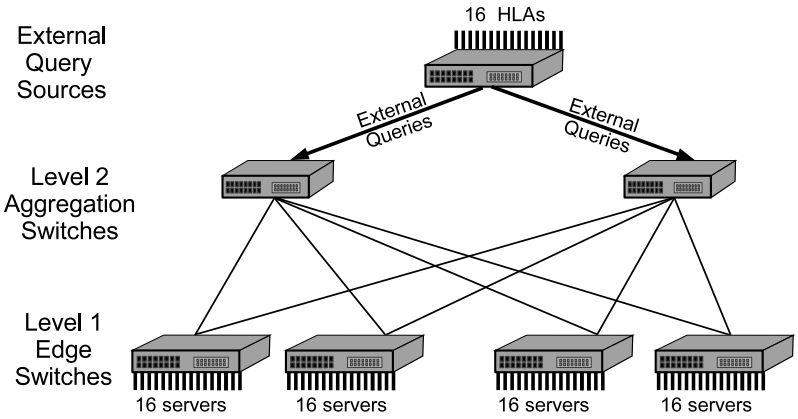


Figure 2.9.: Two-tiered data center network topology with edge and aggregation switches and 64 end-nodes distributed in 4 racks. This topology is an *XGFT(2;16,4;1,2)*. The 16 external query sources are acting as the HLAs for the Scatter/Gather communication pattern generated by the commercial application traffic. These external sources inject TCP queries in the datacenter network through the Level 2 aggregation switches.

in the output queue associated with the output port selected. If the output queue is empty, the packet will be sent out immediately. If there is contention on the output port, the packets will be sent out in FIFO order. The switch we model implements a cut-through switching policy.

There are two main differences between the ideal switch we are using and a real switch:

- $N\times$ speedup: The input bandwidth in each switch output queue is N times the line rate. Hence, it is possible for all input ports to simultaneously enqueue a packet in an output queue. In a real system, for a high arity switch, this is unrealistic because of physical limitations.
- Full buffer sharing: The size of each switch output queue is only bounded by the sum of all the input buffers for all ports. In a real system, the size of the output queue is bounded to a smaller value than the sum of all the buffers capacities in the device. Hence a single output queue can not use the entire buffer memory in the device.

2.5.2.1. Datacenter Topology

The first part of the evaluation will outline the impact that the newly introduced CEE protocols have on TCP performance in three scenarios: *(i)* commercial application over TPC, *(ii)* commercial application over TCP mixed with UDP, *(iii)* scientific workloads over TCP. For the commercial application we use two practical, albeit scaled down in size, extended generalized fat tree (XGFT) [91] topologies: *XGFT(2;32,4;1,2)* and *XGFT(2;16,4;1,2)*. The latter is shown in Figure 2.9. In the first scenario, described in Section 2.6.2, we inject solely TCP traffic in the *XGFT(2;16,4;1,2)* network. In the second scenario, used in Section 2.6.3, we inject

both TCP and UDP traffic in the *XGFT(2;32,4;1,2)* network. In the third scenario, for the scientific workloads in Section 2.6.4, we used two slightly different topologies: *XGFT(2;16,7;1,2)* and *XGFT(2;32,7;1,2)*.

The second part of the evaluation will measure the impact of the proposed routing schemes on scientific workloads running over UDP. Here we have simulated a 2-ary k-tree of 5 to 8 levels, with 32 up to 256 end-nodes. This simulation model we consider representative – with respect to RTT and average hop count – for an average datacenter of 10-50K nodes, with 32...64-port CEE switches.

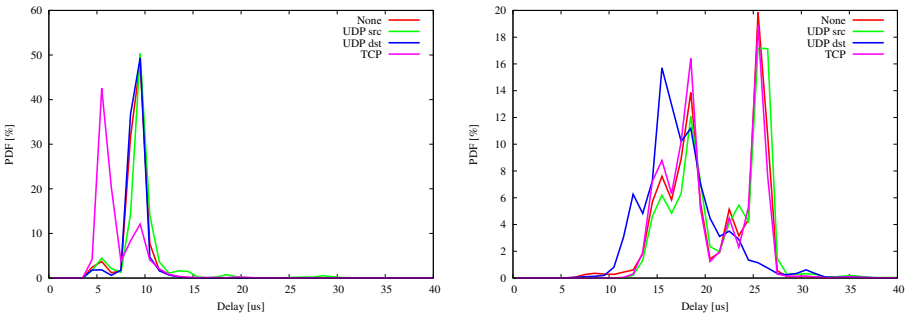
2.5.3. TCP Transport Model

We extended the existing Venus network simulator with a model of the TCP transport. To be as close as possible to reality, we ported the TCP implementation code directly from the FreeBSD v9 kernel into our simulation framework, performing only compulsory (minimal) changes. They are mostly related to the allocation and deallocation of segments. Different TCP sockets are served in round-robin order to prevent one socket from monopolizing the entire network adapter memory. The FreeBSD v9 kernel has two important features: (i) *connection cache*: the congestion window and the RTT estimation are inherited from one connection to the next; (ii) *adaptive buffers*: the receive and transmit buffers are increased in response to an RTT increase.

Based on our network measurements and on previous work [117, 33] we modified the following parameters of the TCP stack:

(i) **Kernel timer quanta**: During the calibration runs we noticed that the typical RTT of our network was in the range of tens of microseconds. This RTT was two order of magnitude smaller than the kernel timer quanta – by default 1 ms. With this setup both the Retransmission Time-Out (RTO) estimator and the TCP Vegas RTT measurement were ineffective. The accuracy of the RTT estimation is critical especially for delay-probing TCP protocols such as Vegas and Compound TCP [112]. TCP Vegas relies on fine resolution timers for accurate timing measurements needed to compute the actual and the expected throughputs, and to accordingly adjust its congestion window. Therefore for all the experiments we reduced the timer granularity to 1 μs.

(ii) **RTO defaults**: Next, we reduced the value of the minimum RTO (*RTO min*) from 30 ms to 2 ms, based on our network measurements and [117, 33]. In the absence of updated information, the kernel defaults to RTO = 3 s. Thus a loss of the initial SYN segment will drastically penalize the flow completion time, which occurred with PFC disabled. We reduced the default RTO (*RTO base*) from 3000 ms to 10 ms, larger than the maximum RTT of our network. The RTO is computed using Jacobson's estimator; a constant term is added to the estimation, accounting for the variance in segment processing at the end-point kernels. In FreeBSD this



(a) Transmission delays. (b) Reception delays.

Figure 2.10.: Linux kernel TCP stack delay measurements. The measurements were performed with different types of background traffic: None, TCP, or UDP on the source or on the destination.

term is conservatively set to 200 ms, accommodating slower legacy machines. We set the RTO variance (*RTO slop*) to 20 ms to match current processors.

2.5.3.1. TCP Stack Delay Evaluation

When a packet is sent from an application, it is first copied to the kernel memory space, then appended to the TCP transmission buffer. Next, it is pushed to the IP layer and, if the transmission queue *tx_qdisc* is not full, a pointer to the packet is enqueued in the *tx_qdisc*. The NIC device driver removes the packet from the *tx_qdisc* and maps it into a transmission circular buffer called *tx_ring*. Finally, if resources are available, the packet is transferred using DMA to the memory of the network adapter that sends it out on the wire.

When a packet is received, the NIC transfers the packet via DMA into a free slot in the the reception circular buffer called *rx_ring*. After the DMA transfer an interrupt is raised to signal the new packet to the device driver. The device driver takes the packet out of the *rx_ring* and sends it to the network stack. The packet is first analyzed by the IP layer and then, if it is destined to the local stack, it is appended to the TCP reception buffer. Finally the packet is eventually copied to user space and consumed by the application.

We determined the delay of the TCP stack by modifying a Linux 2.6.32.24 kernel running on an Intel i5 3.2GHz machine with 4 GB of memory. The transmission delay was measured as the time elapsed from the moment the application sends a packet, to the moment the packet is enqueued in the *tx_ring*. Similarly, the reception delay was measured as the time elapsed from the moment a frame was taken out of the *rx_ring*, to the moment the application receives the data. The delay introduced by the DMA transfer and the hardware operation is not measurable from software. We instrumented the E1000e Ethernet device driver of an Intel Gigabit Ethernet controller 82578DM to timestamp each frame. In parallel with the data flow subject

Table 2.1.: Simulation parameters

Parameter	Value	Unit	Parameter	Value	Unit
TCP					
buffer size	128	KB	TX delay	9.5	μs
max buffer size	256	KB	RX delay	24	μs
RTO base	10	ms	timer quanta	1	μs
RTO min	2	ms	reassembly queue	200	seg.
RTO slop	20	ms			
ECN-RED					
min thresh.	25.6	KB	W_q	0.002	
max thresh.	76.8	KB	P_{max}	0.02	
QCN					
Q_{eq}	20 or 66	KB	fast recovery thresh.	5	
W_d	2		min. rate	100	Kb/s
G_d	0.5		active incr.	5	Mb/s
CM timer	15	ms	hyperactive incr.	50	Mb/s
sample interval	150	KB	min decr. factor	0.5	
byte count limit	150	KB	extra fast recovery	enabled	
PFC					
min thresh.	80	KB	max thresh.	97	KB
Network hardware					
link speed	10	Gb/s	adapter delay	500	ns
frame size	1500	B	switch buffer size/port	100	KB
adapter buffer size	512	KB	switch delay	100	ns

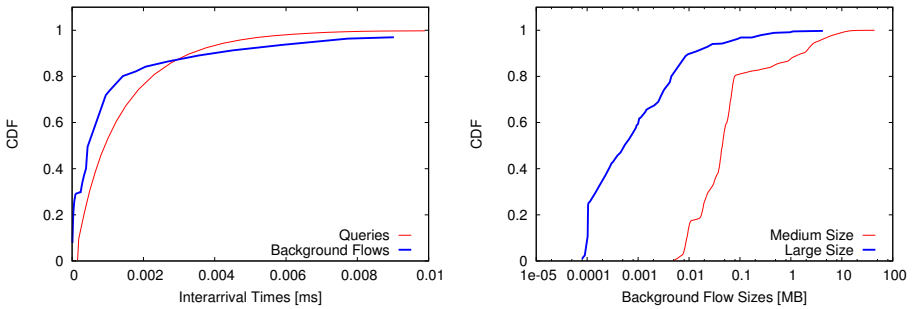
to measurements we injected TCP or UDP background traffic. The results of the experiments repeated for 10K packets are reported in Figure 2.10.

2.5.4. Simulation Parameters

The 802 congestion management algorithm implemented is QCN 2.4. The CEE-based network is 10Gbps, with packet size 1500B. The switches have QCN congestion-points and support ECN/RED active queue management. Table 2.1 contains the key parameters of the network.

2.5.5. Applications, Workloads and Traffic

We have selected a few datacenter applications, divided in two groups: commercial and scientific workloads. These application generate congestive traffic patterns that could benefit from the new features of CEE.



(a) Inter-arrival distributions. (b) Flow size distributions.

Figure 2.11.: Flow inter-arrival and size distributions. For background flows we use the inter-arrival time and flow size distributions given in [16, 26]. The queries (foreground traffic) follow the inter-arrival time distribution from [16] accelerated 100×.

2.5.5.1. Commercial Applications

We have designed our commercial traffic generator based on findings from a few recent papers. In [27] the authors instrumented 19 datacenters to find evidence of On-Off traffic behavior. In [26] they perform an in-depth study of the spatial and temporal distribution of the flows in 10 production datacenters. In [16] the authors use a similar approach to measure the size and inter-arrival time distribution of the flows. Another study [72] observed that modern applications use a Scatter/Gather communication pattern. The traffic study from [16] confirms that finding.

We place the High Level Aggregators (HLA) as in Figure 2.9. The HLAs execute queries triggered from external HTTP requests. The queries have an inter-arrival time as in [16]. When an HLA launches a query it contacts some randomly chosen Mid-Level Aggregators (MLA) – one in each rack – and sends them a subquery. An MLA that receives a subquery will distribute it to all the other servers in the same rack – and then, later, it will collect the partial results. When *all* the results have been received, the MLA will send back its aggregated response to the HLA. Using the real-life data from [16, 26] we have created a traffic generator that injects a *foreground traffic* matrix of queries ('mice') on top of a *background traffic* matrix of longer lived flows ('elephants'). The queries are generated as outlined in the previous paragraph, have a fixed size of 20 KB and the inter-arrival time distribution shown in Figure 2.11a. For the background flows each source randomly chooses a destination in order to match the ratio of intra-rack to inter-rack traffic of 30%. Then each source draws from the inter-arrival time (Figure 2.11a) and flow size distributions (Figure 2.11b) and sends the data. For the queries as well as for the background flows we collect the completion time as an application level metric [46]. Each experiment lasts until 10K queries are completed.

2.5.5.2. Scientific Applications

We have selected nine MPI applications. Five of them belong to the NAS Parallel Benchmark [21]: BT, CG, FT, IS and MG. This benchmark aims to measure the performance of highly parallel supercomputers. In addition we used another 4 applications for weather prediction (WRF), parallel molecular simulations (NAMD) and fluid dynamics simulations (LISO and Airbus). All the above applications were run on the MareNostrum cluster at the Barcelona Supercomputing Center; during the run, the MPI calls of the applications were recorded into trace files. The collected traces were then fed into our end-to-end simulation environment; for a detailed description of this methodology, please refer to [85]. We assume that the MPI library on each processing node uses TCP or UDP sockets as underlying transport. The collected traces are in the order of a few seconds. When using TCP, the connection between a source and a destination of an MPI communication is established only once, when the first transfer occurs and it is kept open during the entire run of the trace.

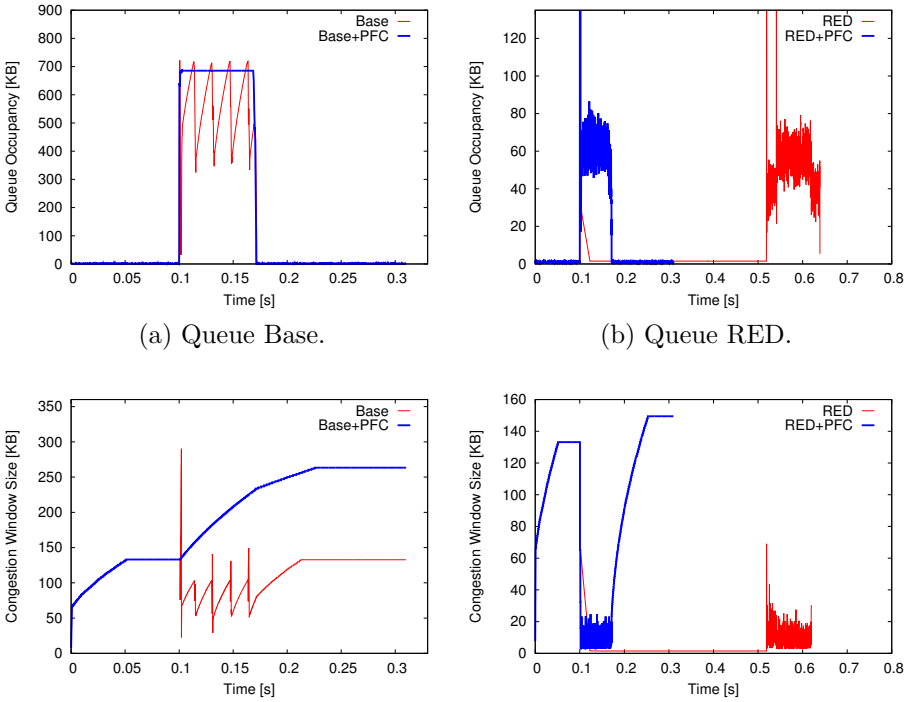
2.6. CEE and TCP Simulation Results

In this section, we aim to evaluate the impact of different L2, L3 and L4 protocols on performance measured at application level, centered on revealing the TCP sensitivities to the two new CEE features: PFC and QCN. We use the following notations: *Base* – no congestion management scheme enabled; *QCN 20/66* – Quantized Congestion Notifications (QCN) enabled with Q_{eq} = 20KB or 66KB respectively; *RED* – Random Early Detection with Explicit Congestion Notifications (ECN-RED) enabled. We run each of these congestion management schemes with or without PFC enabled and with different TCP congestion control algorithms: New Reno, Vegas, and Cubic.

2.6.1. Congestive Synthetic Traffic

To debug the simulation model and calibrate our expectations, we initially use TCP New Reno in a congestive synthetic traffic scenario, described in Figure 2.12, derived from the 802.1Qau input-generated hotspot benchmark.

Base Figure 2.12a shows the evolution of the congested queue. Figure 2.12c shows the congestion window of one of the TCP sources. Without PFC (red) we observe the typical sawtooth graph. TCP increases the window size until the first segment is dropped; detected via duplicate ACKs and handled by the Fast Retransmit. With PFC (blue) there are no losses, therefore at t_1 the RTT increases abruptly; detected by the automatic buffer resizing mechanism that increases the receiver buffer size to allow further augmentation of congestion window.



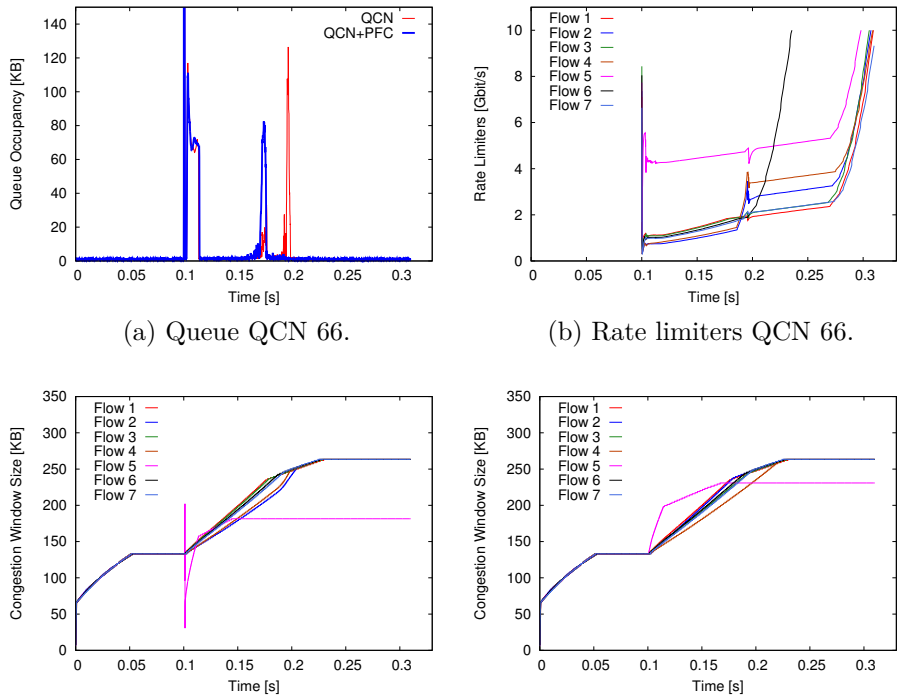
(a) Queue Base. (b) Queue RED.

(c) Cwnd Base. (d) Cwnd RED.

Figure 2.12.: Congestive synthetic traffic: many to one. 7 TCP sources send to the same destination. From $t_0 = 0$ ms to $t_1 = 100$ ms admissible offered load. t_1 to $t_2 = 110$ ms burst: all 7 sources inject a 4× overload of the destination sink capacity. After t_2 admissible load. The congestive event extends past t_2 due to backlog draining.

ECN-RED Figure 2.12(b,d) show the congested queue and one TCP source congestion window evolution, respectively. With PFC enabled (red) the behavior is similar to the base. The difference is in the much lower queue occupancy during the congestive event which extends past t_2 due to backlog draining. Source reduces the injection rate based on ECN feedback. Disabling PFC leads to lower ECN-RED performance. When the queue fills, the TCP sources receive ECN feedback and enter Congestion Recovery. This however is too late to avoid loss, as the load has already been injected. The received duplicate ACKs are ignored since the sources are already in recovery, hence no Fast Retransmit before the retransmission timeout, resulting in throughput loss between 0.17s and 0.53s. This is a situation in which additional feedback, i.e., ECN leads to a wrong decision.

QCN 66 Figure 2.13a shows the congested queue, while Figure 2.13b shows the evolution of the QCN rate limiters. QCN's unfairness [44] causes a single flow to monopolize more than 40% of the capacity: the 'winner' flow ends its transmissions first. However, the other flows still cannot increase their injection rate because of the recovery phase. Figure 2.13(c,d) show the congestion window with and without PFC, respectively. QCN per se is capable of avoiding all the losses but one (of the 'winner').



(a) Queue QCN 66.

(b) Rate limiters QCN 66.

(c) Cwnd QCN 66.

(d) Cwnd QCN 66 + PFC.

Figure 2.13.: Congestive synthetic traffic: many to one. Same traffic pattern as in Figure 2.12. The rate limiters for the QCN+PFC configuration are not shown because they exhibit the same unfairness as those without PFC i.e. flow 5 gets more than 40% of the bandwidth.

2.6.2. Commercial Workload with TCP Background Traffic

The traffic pattern is described in Section 2.5.5.1. Figure 2.14, 2.15a and 2.15b show the average flow completion time using 3 background traffic flow sizes. Figure 2.14 corresponds to query traffic ('mice') without background flows; Figure 2.15a and Figure 2.15b show the same query traffic with medium, large sized 'elephants' respectively, as background traffic.

TCP Vegas Despite the different configurations, Vegas does not reveal significant impact in flow completion time. It adjusts the congestion window based on the measured delays. Since PFC is effective only when flows experience drops which Vegas avoids, PFC plays a secondary role here. Ditto for the other L2 and L3 congestion management schemes.

TCP Cubic Cubic [60] performs worse than New Reno and Vegas in this environment. We observed that the aggressive increases in the congestion window generate more losses than New Reno, therefore penalizing the query completion time. This is corroborated with Cubic's RTT independence, leading to increased losses and poor performance in a datacenter environment.

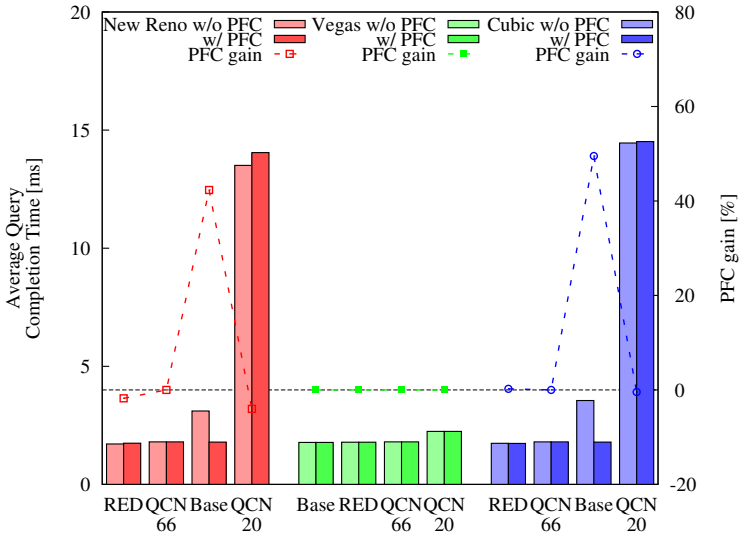


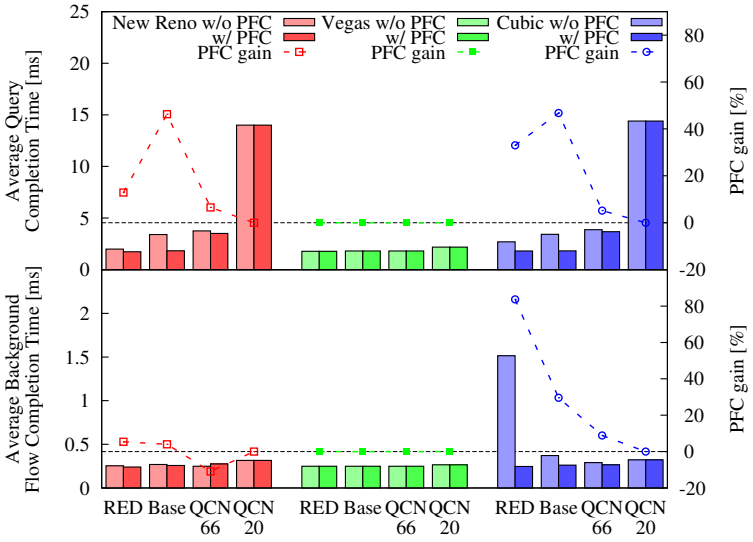
Figure 2.14.: Commercial Workload average query completion time without background traffic. The bars are grouped in three categories, based on the TCP version. Within a category bars are sorted increasing with average query completion time without PFC.

PFC In all the tests, PFC reduces the flow completion time, with the exception of QCN 20 configuration. We attribute the PFC gains to avoiding TCP stalls waiting for retransmissions. They are caused by Jacobson's RTO estimator, in the datacenter environment, where the RTT is dominated by queuing, instead of flight delays [57]. Whereas link delays are constant, the datacenter's queuing delays are extremely dynamic: they can increase 100 to 1000 fold within milliseconds. The original RTO estimator, however, reacts slowly. This is compounded with its kernel calculation: a constant term is added that accounts for the cumulated variances in the segment processing at the two end-point kernels. This constant is orders of magnitude higher than the typical datacenter RTT.

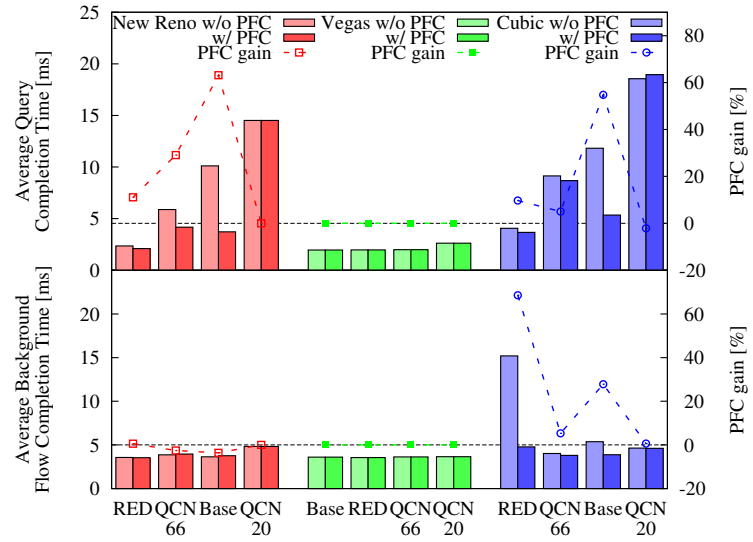
QCN For comparison with [16] we choose two Q_{eq} setpoints : (i) the value recommended by the IEEE standards committee – 20% of the queue size, i.e. 20 KB, (ii) an experimental value i.e. 66 KB. QCN 66 is always better than QCN 20 in Figure 2.15 mostly due to its higher tolerance to the intrinsic burstiness of the transport layer. The TCP source sends a burst of segments until either the congestion window or the receiver window is exhausted. The first burst of segments will trigger a reverse burst of ACKs, which in turn will produce a second burst of segments etc. This is supported by measurements in [27]. Generally QCN is highly sensitive to burstiness.

ECN-RED With this workload ECN-RED delivers the best performance, further improved by enabling PFC. ECN-RED outperforms QCN because:

(i) *burst sensitivity* - ECN-RED congestion feedback is based on averaged, whereas QCN is based on instantaneous, queue length. Therefore a transient burst will not trigger a reduction of the injection rate with RED.



(a) Medium sized background flows.



(b) Large sized background flows.

Figure 2.15.: Commercial Workload with TCP Background Traffic. The upper part of the graphs shows the average query completion time, while the lower part shows the average completion time of the background flows. The bars are grouped in three categories, based on the TCP version. Within a category bars are sorted increasing with average query completion time without PFC.

(ii) *interaction with L4* - the congestion notifications generated by RED are processed directly at the transport layer which adjusts the congestion window accordingly. On the other hand, TCP remains oblivious of Layer 2 congestion feedback.

(iii) *data/control differentiation* - RED can generate congestion notifications only for segments carrying data. The reduction of the congestion window only affects the data flow while the control segments can still move freely. On the other hand, QCN's rate limiters can not distinguish between control and data. For example, we found that some queries were delayed because the initial SYN segments were throttled by the rate limiter.

2.6.3. Commercial Workload with UDP Background Traffic

We also tested mixed TCP-UDP performance. In addition to the previous section, TCP has to compete against aggressive UDP background sources ('elephants'). Therefore, we double the number of end-nodes: half of the end-nodes are TCP, while the other half are UDP sources. UDP sources inject bursty traffic with average burst sizes of 28 KB and 583 K. The UDP burst sizes are selected according to the background flow size distributions from Figure 2.11b.

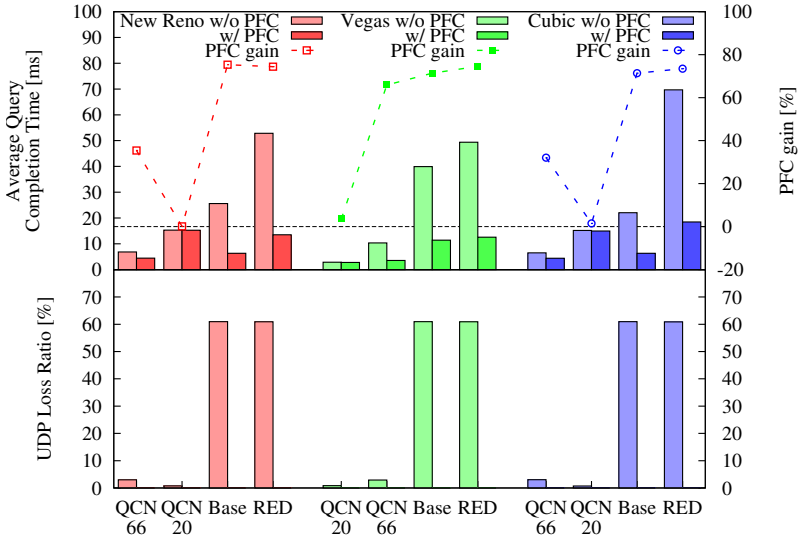
The average flow completion time for the TCP queries are shown in Figure 2.16. We also measure the loss ratio for TCP and UDP flows – the loss ratio is computed as the percentage of dropped bytes vs. the total injected bytes (see Figure 2.16 – lower half).

Most of the dropped bytes are UDP. This is because TCP reduces its window whenever losses are detected. In contrast with the previous section, here we observe that Vegas is sensitive to ECN and QCN. Again, enabling PFC improves performance. Overall, the best performer is QCN 66. When we introduce non-cooperative UDP sources, only QCN's rate limiters can restore some of the fairness lost by TCP in competing against UDP.

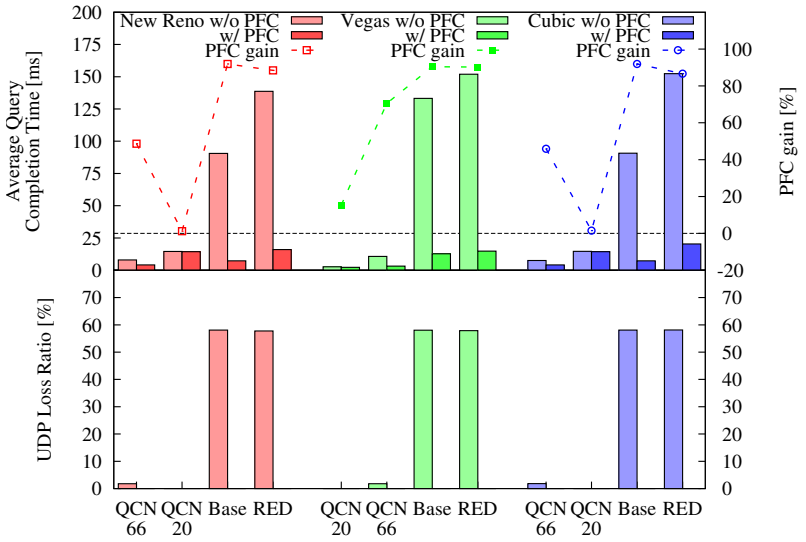
2.6.4. Scientific Workloads

The simulated MPI traces are described in Section 2.5.5.2. Initially we run each benchmark on a reference system where we assume we have a perfect hardware accelerated transport and lossless network. We run every benchmark on each configuration while measuring the execution times. Then we compute the relative slowdown of each benchmark vs. the ideal reference. Finally we average all the slowdowns across the nine benchmarks, plotted in Figure 2.17.

Enabling PFC improves performance across all the configurations. The previous observations from Section 2.6.2 apply also to this workload. The best performer is ECN-RED with PFC enabled. With PFC disabled, however, QCN 20 produces better results. In contrast with the commercial traffic, QCN 20 was the worst



(a) Medium sized background flows.



(b) Large sized background flows.

Figure 2.16.: Commercial Workload with UDP Background Traffic. The upper part of the graphs shows the average query completion time, while the lower part shows the loss ratios of the background UDP flows. Bars are grouped in three categories based on the TCP version. Within a category bars are sorted increasing with average query completion time without PFC.

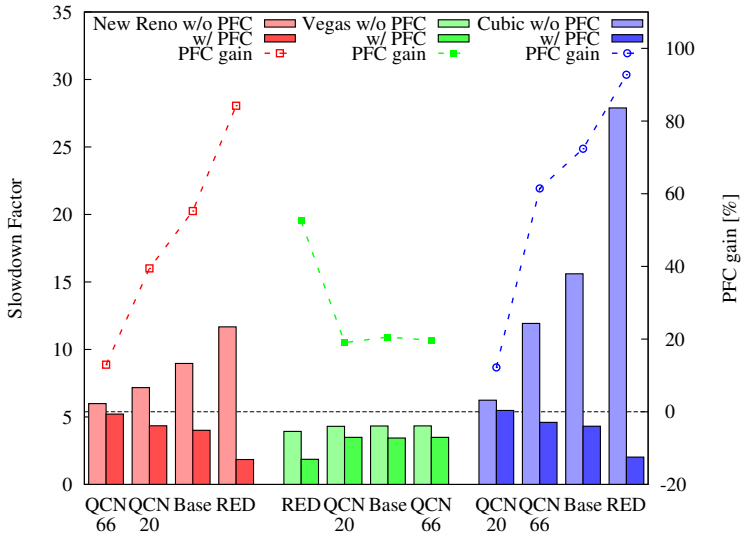


Figure 2.17.: Scientific Workload: MPI Traces relative slowdowns. Bars are grouped in three categories based on the TCP version. Within a category bars are in increasing order of the relative slowdown factors with PFC disabled.

performer. Commercial workloads exhibit only sparse transient congestive events, whereas in the scientific workload the congestive events are sustained and involve all the end-nodes. The MPI applications use barriers to synchronize between execution phases. All the nodes start communicating almost at the same time and this generates heavy congestion. The aggressive Q_{eq} setpoint of QCN 20 effectively mitigates such congestive cases.

2.7. CEE Routing Simulation Results

In this section, we aim to evaluate the influence of different routing schemes on performance measured at application level. In Section 2.6 we showed that PFC reduced completion times across all configurations. Therefore in this section we will enable PFC for all experiments. Activation of PFC allows us to run all the benchmarks over UDP. The idea of using PFC to simplify the transport layer will be further developed in Chapter 5 in a virtualized datacenter context.

We extended the Venus simulator to include the routing algorithms described in Section 2.3 and Section 2.4. From the load-oblivious class we assess the random and the hashed routing. From the deterministic class we evaluate the best performing algorithm, i.e., *D-mod-k* routing. In the adaptive class, we also assess the switch-based adaptive routing. For these algorithms we consider a version *with,* and another version *without,* rate limiters, the former bearing the suffix "**RL**". All are compared against the R^2C^2 and R^3C^2.

2.7.1. Congestive Synthetic Traffic

Some of the synthetic traffic patterns we use in the following subsections are part of the Hotspot Benchmark used by IBM Research and 802 Task Groups. One or more sources can generate a hotspot at a given location in the network by injecting a predetermined amount of (in)admissible traffic for that location. Flows that pass through the hotspot are referred as *hot flows*, while the others are referred as *cold flows*.

Hotspots are classified using the following criteria:

- **Type**

  – *Input Generated* – The inputs (sources) require more bandwidth than available in the network. Typical examples are the patterns when flows from different sources converge into the same link exceeding its capacity.

  – *Output Generated* – An output (network device) is slow in processing incoming packets. For example, a traffic destination can be slowed down because of a CPU overload. Another possible cause can be a switch servicing traffic from different priorities. In this case, output generated hotspots can appear because a part of the available bandwidth is reserved for the higher priorities.

- **Severity** – measures the ratio between offered and accepted traffic (the drain rate of the bottleneck link during congested phase).

  – *mild* – smaller than 2

  – *moderate* – between 2 and 10

  – *severe* – higher than 10

- **Degree** – the fan-in of the congestive tree at the hotspot (i.e. the percentage of all sources that inject hot flows into the hotspot).

  – *small* – less than 10%

  – *medium* – 20% to 60%

  – *large* – more than 90%

We use a network with 32 processing nodes connected by a 2-ary 5-tree network. This network has the round-trip time and average hop count of a large datacenter interconnect. In an average datacenter, the number of nodes is on the order of 10K, but also the arity of switches is much higher (32 to 64 ports). Because of these factors, a large interconnect will still have a small number of levels (3-5), as in our simulation.

Next, we continue with three types of synthetic traffic: permutation patterns, input-generated hotspots, and output-generated hotspots.

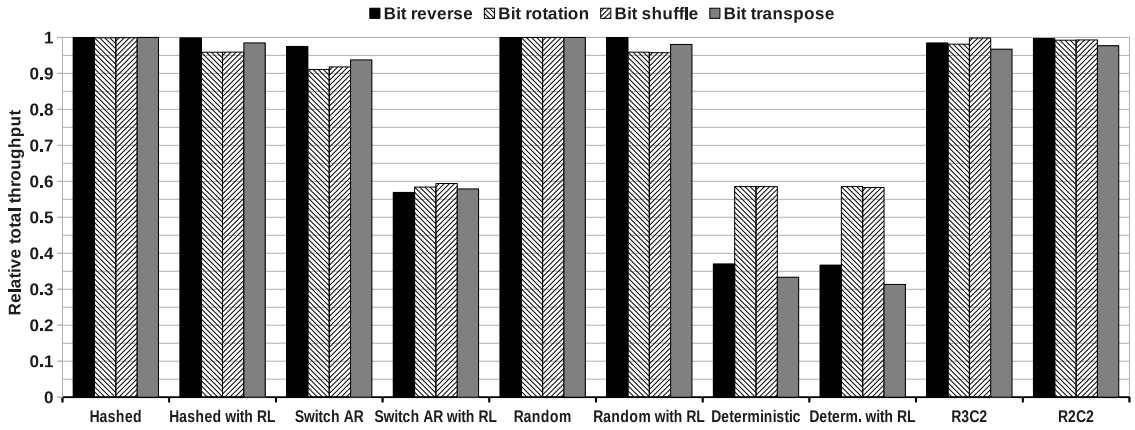


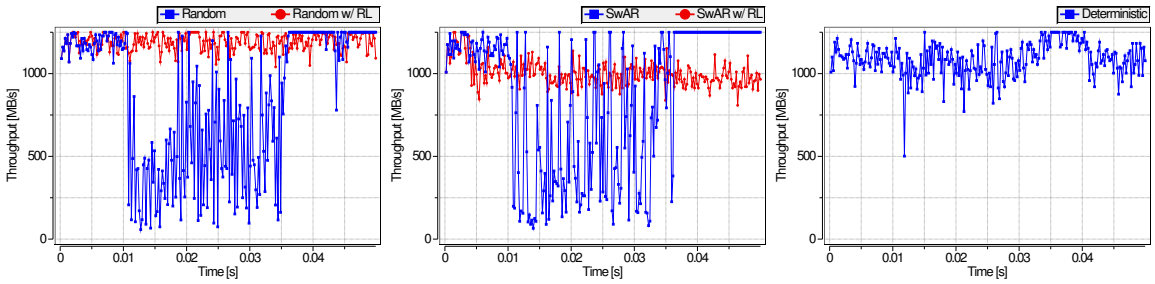
Figure 2.18.: Permutation traffic. All end-nodes communicate in a one-to-one permutation pattern. The sources inject traffic in the network at 90% of the link capacity. Simulation length is 30 ms. The chart above displays the relative throughput of each routing scheme. As reference we consider the throughput of the same traffic pattern using an ideal crossbar. R^2C^2 and R^3C^2 achieve 98% of the ideal throughput. Idem for random and hashed routing. For this simulation, the hashing-key is computed per-packet; hence, random and hashed routing show identical results. Deterministic routing (static) cannot reroute the traffic if two flows collide, therefore it looses up to 70% in throughput. Switch AR with RLs looses 40% owing to multiple hotspots. A flow that is being rerouted to avoid a congested link could still hit another bottleneck. Eventually this will activate the rate limiters.

2.7.1.1. Worst-case Scenario – Permutation Traffic

We have initially used uniform traffic generators. Since all the selected routing candidates successfully handle the uniform distributions up to loads of >98%, we omit these results. The permutation traffic pattern is a worst-case scenario. Results are shown in Figure 2.18. We expect large multi-tenant virtualized datacenters to benefit from statistical multiplexing. Hence, the network nodes will not *always* synchronize their communication patterns – except the HPC and special operations, such as synchronization primitives, large MapReduce computations, distributed in-memory databases etc.

2.7.1.2. Input-generated Hotspot at Edge Links

The objective of this test is to check whether the routing algorithms are generating congestion trees. We create an input-generated hotspot of mild severity and small degree. To achieve this, we direct 45% of the traffic from 4 different nodes to a single destination node for 20 ms. The entire simulation lasts 50 ms. The congestion tree evolution can be directly visualized by inspecting the length of the queues for different switches during simulation, or indirectly by measuring the throughput of a cold flow.



(a) Tput Random. (b) Tput Switch AR. (c) Tput Deterministic.

Figure 2.19.: Input generated hotspot at edge links: victim flow throughput. The ideal throughput is 95% of link speed (1187MB/s). Random and switch adaptive schemes without RLs generate large congestion trees. Consequently, the victim flow's throughput drops during the congestive pattern. Deterministic routing limits the expansion of the congestion tree.

We adopt the second method; in Figure 2.19 we show the throughput of the "victim" cold flow when using different routing schemes. The hot flows converge only on the edge link connecting the destination with the network. The cold flow does not pass through that link, hence its throughput should not be affected by the bottleneck. However, when a congestion tree is formed, many other links can saturate, as shown in Figure 2.2. Hence the cold flow can be indirectly affected by the secondary hotspots belonging to the same congestion tree.

As seen in Figure 2.19a and Figure 2.19b, both random routing without RL and switch adaptive routing without RL can generate congestion trees. On the other hand, activation of rate limiters will eliminate this issue. We observe from Figure 2.19c that deterministic routing is less strongly affected. This is because random routing and switch adaptive use of all the available alternative paths. Hence, they tend to spread the congestion. Deterministic routing uses a single path all the time, hence congestion is limited to that path. These results confirm the observations from [53] about the undesired effects of adaptivity.

2.7.1.3. Output-generated Hotspot at Root Links

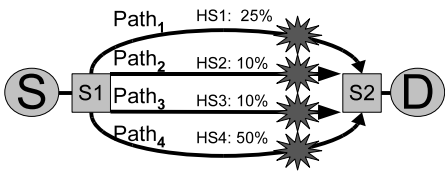
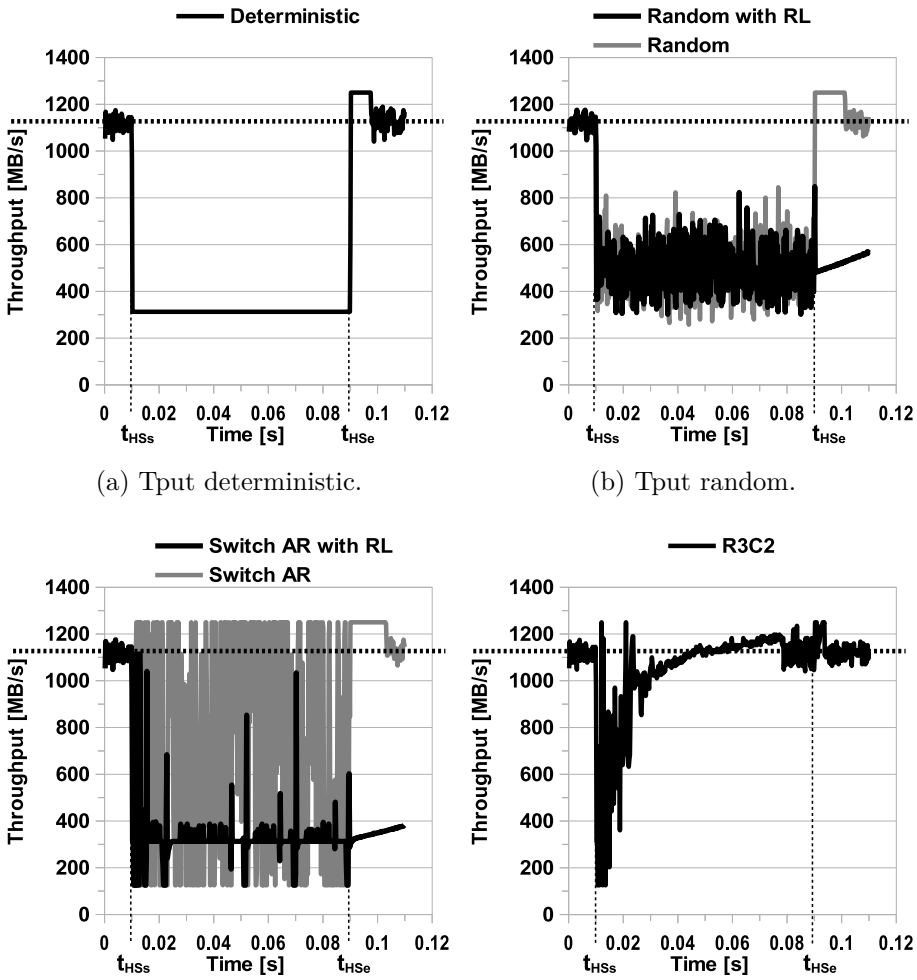


Figure 2.20.: Congestion scenario: hotspots on each of the four paths between S and D. The hotspots decrease the capacity as shown.

The objective is to study the effects of multiple output-generated hotspots of *different* severities. Multiple hotspots can appear in datacenters running different



(a) Tput deterministic.

(b) Tput random.

(c) Tput Switch AR.

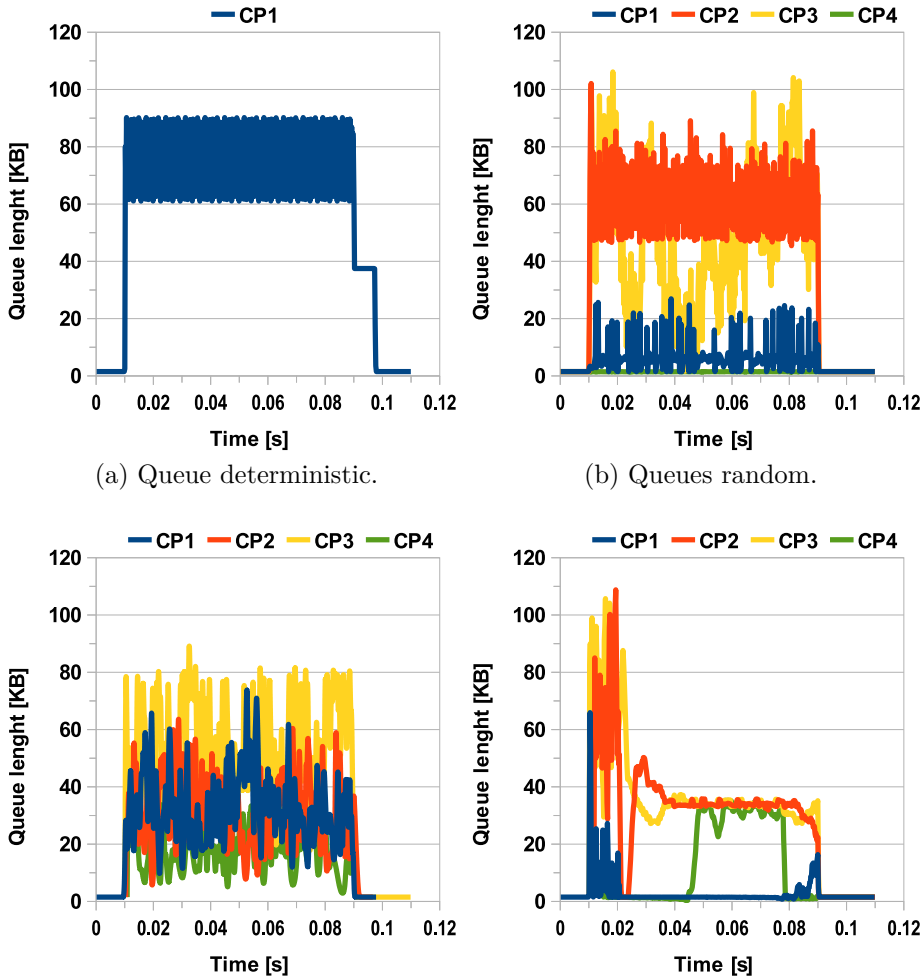
(d) Tput R^3C^2.

Figure 2.21.: Throughput evolution for the congestive pattern shown in Figure 2.20. The ideal throughput is 90% of link speed (1125MB/s) – horizontal dashed line. Congestion lasts 80 ms from $t_{HSs} = 10$ ms to $t_{HSe} = 90$ ms – vertical dashed lines.

applications on different priority levels that might employ distinct routing strategies. We simulate this scenario by reducing the service rate of root level links. This is equivalent to providing multiple paths, each path of different capacity. As shown in Figure 2.20 there are four paths between the source and the destination. On each path we place a different bottleneck, hence multiple hotspots. Path capacities are 25%, 10%, 10% and 50%, respectively.

S injects packets at 90% of the link capacity. In Figure 2.21(a,b,c,d) we observe the throughput at D. The ideal throughput is 90% of the 10Gbps Ethernet link speed (1125MB/s). The congestive pattern lasts 80 ms.

Deterministic routing Figure 2.21a uses $Path_1$ only, hence, achieves only 25% (312MB/s). It is outperformed by R^3C^2 that adaptively avoids the most severe bottlenecks by



(a) Queue deterministic.

(b) Queues random.

(c) Queues Switch AR.

(d) Queues R^3C^2.

Figure 2.22.: Queues evolution for the congestive pattern shown in Figure 2.20. Congestion lasts 80 ms from $t_{HSs} = 10\ ms$ to $t_{HSe} = 90\ ms$ – vertical dashed lines.

re-routing most of the traffic on the higher capacity links.

Random routing Figure 2.21b is limited by head-of-line blocking in switch S1 to ca. 40% throughput (500MB/s). At ingress, the packets are uniformly distributed across the four paths. The low capacity paths (2 and 3) slow down the high capacity ones (1 and 4) hence the 40% throughput. This is confirmed by inspecting the queue lengths at the congestion points in Figure 2.22b. The queues for $Path_2$ and $Path_3$ are congested, while the queues 1 and 4 are nearly empty generating misordering and delay jitter. The activation of rate limiters brings little benefit. The R^3C^2 outperforms the random: not only it uses multiple paths in parallel, but it can also match the injection rate to the current hotspot severity.

Switch AR achieves on average 60% throughput (750MB/s). In Figure 2.21c oscillations are visible, caused by switching the full load and further amplified by the

resequencing process at the destination. We observe that the queue occupancies are comparable (Figure 2.22c). Switch AR manages to discover the severity of each hotspot, albeit it is partly penalized by oscillations. Subject of ongoing research, Switch AR's instability could be improved by adopting fractional split ratios, instead of switching the full load from one path to another. However, the current Switch AR is outperformed by R^3C^2, which despite its higher control loop delays, is more stable.

R^3C^2 throughput is shown in Figure 2.21d. During the first 15 ms it oscillates as its switch-based counterpart. Sufficient congestion notifications are needed by the source to trigger the activation of alternative paths – route control. Once the paths are activated, further notifications are needed for rate control to adjust the injection rate to the individual hotspot severity. During the learning phase, similar to Switch AR, our proposal will flood each of the alternate paths until it eventually converges to a stable rate. This is reflected by the queues' occupancies (in Figure 2.22d) that drain and converge to the Q_{eq} value set by the QCN configuration.

To sum up, R^3C^2 wins owing to three features. (i) It adapts the routes, unlike the deterministic routing. (ii) It has built-in rate adaptivity, which allows it to outperform random routing. (iii) Finally, it reacts less aggressively than Switch AR to congestion notifications, evincing improved stability.

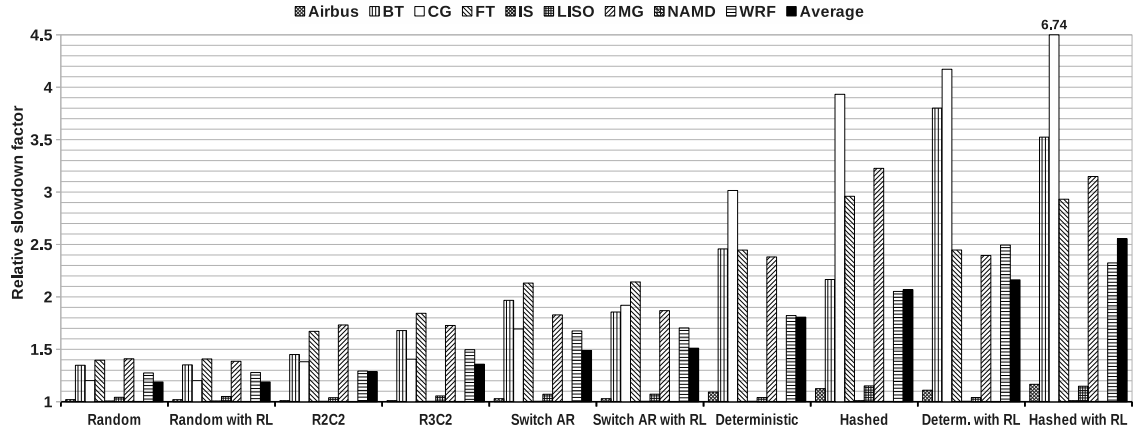
2.7.2. Scientific Workloads

2.7.2.1. MPI Traces

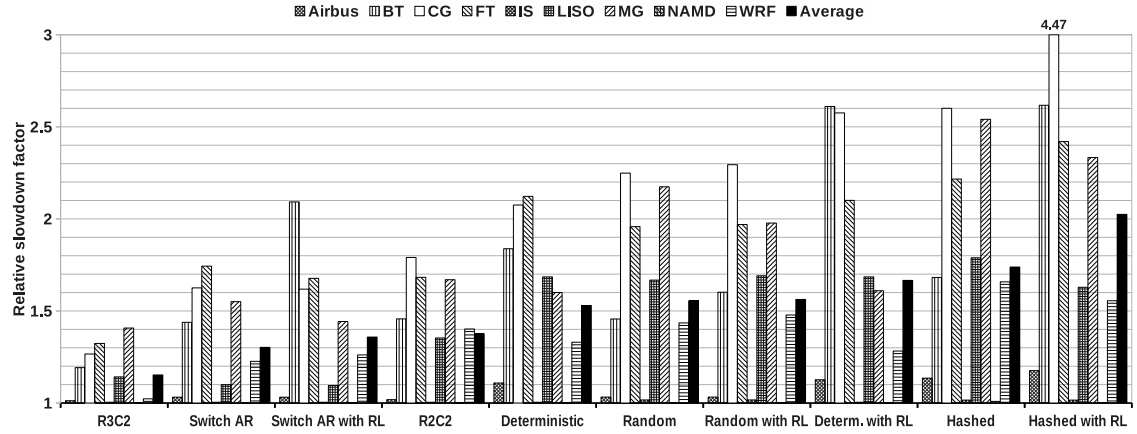
Evaluation results are plotted in Figure 2.23a and Figure 2.23b. In order to facilitate the comparison, we display the relative slowdown of each routing scheme. The time needed to run a trace on an ideal crossbar is our base reference for this Section's figures (base=1.0) and relative percentages (normalized to base=100%). For all simulations we employed random task placement. In a virtualized datacenter we expect tasks will be arbitrarily assigned to nodes, depending on factors such as the load, priority, cluster fragmentation, service level agreement, power constraints etc. We consider two scenarios.

In the first one (see Figure 2.23a) we run the application on an empty un-impaired network, with no other traffic present. The application can use each link's full capacity.

In the second, more realistic, scenario (see Figure 2.23b) we reduce the capacity of a level-two link, hence simulate an output-generated hotspot. We thus measure the performance of the medium/low priority traffic, assuming per-priority flow-control and Enhanced Transmission Selection are implemented. The high priority traffic is preemptively scheduled, it has a guaranteed fraction of the capacity. The low priorities can use this fraction only when no high priority traffic is active. Else, as the most common case in multi-tenant datacenters, the low priority traffic will be



(a) Without high priority traffic. Random routing produces the smallest execution times followed by R^2C^2 and R^3C^2. The switch adaptive losses throughput because of frequent oscillations. Deterministic and hashed lack adaptivity.



(b) Simulated high priority traffic. R^3C^2 provides the smallest execution time followed by Switch AR. Adaptive routing schemes can detect the hotspot and avoid it while random routing continues to send packets through the congested link. Deterministic does not use the congested link for half of the connections, in average. Hence it outperforms the random.

Figure 2.23.: HPC workloads relative slowdowns (smaller is better). As reference we use the execution time of the same respective traces running on a single-stage ideal crossbar (base=1.0). Results are sorted by their average slowdowns. In (a) the HPC application can use the full capacity of each link, whereas in (b) we simulate the impact of the high priority on the rest of the traffic. In this particular case we reduce a single link's capacity to 33%.

scheduled as if the links have variable, lower, capacities – modulated by the high priorities active above.

Figure 2.23a shows that R^3C^2 improves the performance: up to 161.3% over deterministic, on average 45.2%; 28.8% over Switch AR, on average 13.3%. Nonetheless, random routing is, on average, 17.2% faster than R^3C^2. Random immediately uses all the available paths, whereas R^2C^2 and R^3C^2 must first wait for sufficient congestion notifications to arrive, before activating any alternative paths. Therefore, in a dedicated single application per cluster, the random scheme still leads in price-performance.

Nevertheless, when we introduce output-generated hotspots, e.g. a link capacity reduction (by 66%), R^3C^2 takes the lead with the shortest execution time (Figure 2.23b): up to 98.2% faster than random, on average 40.2%; up to 36.4% faster than Switch AR, on average 14.9%. Generally, with the slightest asymmetry in link capacity, random is exposed to head-of-line blocking and throughput loss. We believe Switch AR still has ample room for improvement, although beyond our scope here.

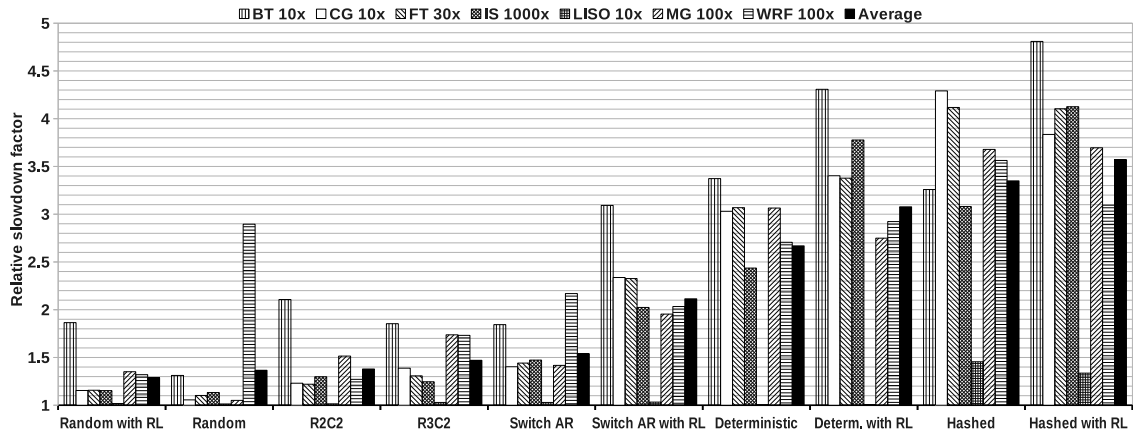
Our proposal is up to 81.1% faster than deterministic, and up to 133.5% faster than hashed. Deterministic shows execution delays, missing adaptivity and load balancing. This is particularly visible with applications that generate long messages, e.g., CG produces 750KB messages. The delay can be substantially increased by a conflict between two long messages. Ditto for the hashed routing, which also shows slowdowns.

2.7.2.2. Scaled MPI Traces

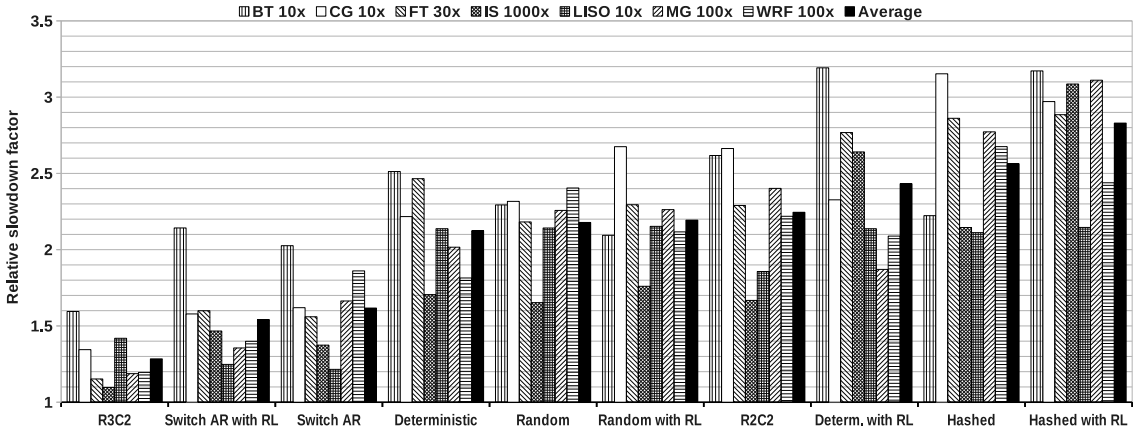
For some of the applications listed in Section 2.5.5.2, we observed that they do not put pressure on the communication network. For others such as IS or LISO, the differences observed between the routing schemes were minor. This was due to the fact the applications rarely exchanged small messages. For such applications, contention was infrequent, hence the adaptive routing algorithms could not provide any benefit. This is confirmed by previous work. In [53] the authors showed that the adaptivity does not provide any improvement for some categories of workloads.

To stress the communication network more and to emphasize the differences between the various routing schemes, we scaled the trace files. When a trace is scaled the size of all the messages generated by its execution is multiplied with a given scale factor. We used different scale factors to ensure that the communication demands of the application are high enough to generate contention. Table 2.2 lists the scale factors used and the average and maximum message size the scaled application uses. For most of the applications the maximum message size is on the order of MB.

The results are in Figure 2.24a and Figure 2.24b. Scaling of traces does not radically changes the ranking of the routing schemes, but will accentuate the differences



(a) Without high priority traffic.



(b) Simulated high priority traffic.

Figure 2.24.: Relative slowdowns of scaled HPC workloads (smaller is better). The reference is the runtime needed to execute the trace on a single-hop ideal crossbar. Results are sorted by their average slowdown. In (a) the HPC applications can use the full bandwidth of each link, while in (b) we simulate the impact of high-priority traffic by reducing the bandwidth of a single link to 33% (by 66%). Same observations as for Figure 2.23 apply. The rate-limited versions or random routing and Switch AR, respectively, produce the shortest execution times because they avoid the end-point congestion generated by scaling.

Table 2.2.: Trace scaling factors

Trace	Scale factor	Mean size	Max size
BT	10x	69.4 KB	1.23 MB
CG	10x	1.61 MB	7.5 MB
FT	30x	1.31 MB	3.9 MB
IS	1000x	158 KB	2.06 MB
LISO	10x	40 KB	121 KB
MG	100x	2.96 MB	13.5 MB
WRF	100x	0.93 MB	9.7 MB

between them. For empty network the random routing is still the best followed by R^2C^2 and R^3C^2, Switch AR, deterministic and hashed routing. As in the previous section, for a network where high-priority traffic is present, the R^3C^2 provides the shortest execution time followed by the Switch AR, deterministic and random routing.

We can notice that the rate-limited versions of random routing and Switch AR perform better than the versions without RLs. As a side effect of trace scaling end-point congestion is generated in workloads like MG or WRF. The adverse effects of this end-point congestion are eliminated by the use of RLs as explained in Section 2.7.1.2.

2.8. Results Analysis

We summarize the results discussion by answering our initial questions from Section 2.1.1.

(**Q1**) How does TCP perform over CEE networks? Is PFC beneficial or detrimental to TCP? Is QCN beneficial or detrimental to TCP?

The delay-probing TCP Vegas performs the best, requiring arguably minimal changes, i.e., high resolution timers. By contrast, RTT-independent TCP Cubic entails the most adaptation effort for datacenter environments, eliciting exhaustive parameter retuning and potentially core algorithm changes. In our experiments, Cubic suffers from aggressivity and slow convergence of congestion windows. New Reno lies in between, requiring more parametrical retuning than Vegas, but no invasive changes such as Cubic. Whether RTT independence, as in BIC and Cubic, is actually harmful in CEE networks with a wide dynamic range of queuing delays (sub-μs up to tens of ms) remains an open research problem.

Next, despite our contrary expectations, PFC has consistently improved the performance across *all* the tested configurations and benchmarks. The commercial workload completion time improves by 27% on average, and up to 91%. Scientific workloads show higher gains by enabling PFC: 45% on average, and up to 92%.

On the *positive* side, properly tuned for commercial TCP with UDP applications, QCN 66 with PFC improves performance on average by 49%, up to 70%. When we introduce non-cooperative UDP flows in the network QCN 66 keeps congestion under control regardless of the upper layer protocols. For scientific workloads, QCN 20 without PFC – currently an uncommon HPC configuration – improves performance on average by 31%, up to 59%. HPC applications typically exhibit alternating phases of computation and communication. During the latter, typically all nodes start communicating quasi-simultaneously, which can generate overload episodes and hotspots – especially in slim networks as reproduced here. The aggressive Q_{eq} setpoint of QCN 20 effectively mitigates such congestive events. On the *negative* side, mistuned QCN can *severely* degrade performance. E.g., in commercial workloads relying exclusively on TCP – without competing UDP traffic sources – QCN 20 without PFC degrades performance on average 131%, up to 311% for New Reno and 321% for Cubic. For scientific workloads QCN 66 with PFC degrades performance on average by 5.4%, up to 8.2% – hence leaving QCN enabled is acceptable whenever its Q_{eq} is set 2× to 4× higher than the standard recommendation.

Our results show that RED handles the transient congestion episodes generated by commercial applications better than QCN. This reveals a preventable (by careful tuning) QCN weakness: burst sensitivity. A properly configured RED is less sensitive to burstiness, mainly because it relies on smoothed (low pass filtered) queue length. This can reduce the query completion time by up to 76%. Aggravating the performance penalty with bursty commercial workloads, QCN suffers from inherent unfairness: it tends to arbitrarily favor some 'winner' flows over the others, harming the average completion time.

(Q2) Can we reap any benefits by combining VLAN-based route control with QCN rate control? Can a source-based adaptive routing scheme show performance benefits with adapters that do not implement QCN?

Our evaluations showed that for HPC benchmarks, combined VLAN-based route control with QCN rate control (R^3C^2) can be up to 98% faster than random routing, on average 40%. Compared to deterministic and hashed routing schemes, it reduces the execution time up to 133%. The synthetic traffic benchmarks have also confirmed its stability gains. Finally, we showed that a performant and stable source adaptive routing (R^2C^2) is indeed possible for CEE-based networks, even when the lower cost end-nodes are not fully QCN compliant, i.e., they don't implement rate limiters.

2.9. Related Work

Our work is at the confluence of established, e.g., TCP, and emerging research areas, such as datacenter workload analysis and new L2 networking protocols. Our commercial workload traffic generator is based on [27, 26, 16, 72]. In [83] the authors provide an overview of the congestion management schemes proposed for CEE. QCN

is standardized in [11]. Its unfairness and lack of RTT adaptivity [16] have been addressed by E^{2}CM [59]. An alternative solution is proposed in [70]. While to the best of our knowledge this is the first comparative evaluation of 'short-fat' TCP for datacenters, a few performance reviews of modern TCP variants for 'long-fat' networks are available, e.g. in [77, 32].

The TCP Incast problem has been analyzed in [33, 117], where a $10 - 1000\times$ RTO reduction and high resolution timers have been proposed. Another TCP Incast solution is DCTCP [16], using a modified RED/ECN and a new multibit feedback estimator that filters the incoming single-bit ECN stream. This compensates the stiff adaptive queue management setup in the DCTCP congestion point (partly similar to QCN's sensor) with a smooth congestion window reduction function, reminiscent of QCN's rate decrease – hence departing from TCP's halving of the congestion window. Closely related is [44] which analyzes the TCP Incast problem in a QCN-enabled *lossy* network – arguably in conflict with default assumption of lossless CEE. The main drawbacks are the use of NS-2 simulations and the overly aggressive sampling proposal. TCP Incast is studied in [111] using SCTP and ECN with hardware experiments. The performance improvements range from 5.5% to 7.9%, limited by the experimental platform.

An extensive overview of routing mechanisms is presented in [41] Chapter 8 to 11 and [45] Chapter 4. Deterministic routing was analyzed in [53]. Random routing was studied in [114, 76]. Hashed routing is presented in [63] as a particular case of Equal-Cost Multi-Path routing described in [113]. Switch adaptive routing was introduced in [84]. An alternative, delay-based congestion management scheme is introduced in [57]. [103] introduces a pattern-aware routing scheme suitable for HPC environments whereby the workload is a priori known. Closest to our proposal is the Distributed Routing Balancing algorithm [50] that creates alternative paths in a 2D mesh and balances the traffic across them, using a delay-based estimation of the path congestion. This is extended in [80], where a congestion management scheme is evaluated. The method requires changes to the switching fabric and, does not address the endpoint congestion.

2.10. Discussion

We showed that PFC significantly improves TCP performance across all tested configurations and workloads, hence our recommendation to enable PFC whenever possible. QCN on the other hand elicits further investigation and improvement, particularly with respect to its lack of adaptivity and fairness. Meanwhile we recommend that QCN should be conservatively tuned and enabled whenever heterogeneous transports – e.g., TCP with UDP, RDMA, FCoE, RoCEE etc. – will be expected to share, even briefly, the same hardware priority in the CEE datacenter network. Finally, we proved that a judicious combination of VLAN-based routing with with QCN's rate limiters (R^3C^2) can improve both performance and stability

beyond the current state of the art routing in datacenters. Overall, we have shown that a control shift from the network core toward the edge can benefit the Cloud and HPC applications, while improving their performance and stability.

Summing up, we have contributed: *(1)* A simulation environment combining a real TCP stack with detailed L2 simulation models of CEE switches and adapters, *(2)* the first evaluation of TCP performance in lossless CEE networks, *(3)* a novel source-based adaptive routing algorithm for CEE, using the established VLAN mechanism in conjunction with the recently standardized QCN, and, *(4)* a quantitative performance comparison of the datacenter routing algorithms.

A few aspect of this work could be certainly improved. Both R^2C^2 and R^3C^2 introduced in this chapter are reactive. By waiting on the QCN feedback from congested switches, any reactive scheme will inherently incur additional, potentially destabilizing, delays. Another destabilizing factor is QCN's centralized multi-rate sampling, adaptive with the congestion severity, but not with the flow RTT. The main limitation of our model is the canonical implementation of queries with a strict synchronization barrier; this exacerbates QCN's unfairness, thus further degrading performance.

3. Overlay Virtual Networks: New Layer Between TCP and CEE

In Chapter 2 we focused on the performance of TCP applications running in a traditional non-virtualized environment. In this chapter we will show that datacenter-based Cloud computing has induced new disruptive trends in networking, key among which is network virtualization. We will present the architecture of overlay virtual networks (OVN) that introduce a new layer between the TCP software stack running in each virtual machine (VM) and the Converged Enhanced Ethernet (CEE) hardware.

Overlays aim to improve the efficiency of the next generation multitenant datacenters. While early overlay prototypes are already available, they focus mainly on core functionality, with little being known yet about their impact on the system level performance. Using query completion time as our primary performance metric, we evaluate the overlay network impact on two representative datacenter workloads, Partition/Aggregate and 3-Tier. We measure how much performance is traded for overlay's benefits in manageability, security and policing. Finally, we aim to assist the datacenter architects by providing a detailed evaluation of the main overlay choices, all made possible by our accurate cross-layer hybrid/mesoscale simulation platform.

3.1. Introduction

In addition to faster line rates, new features such as multicore CPUs, Software-Defined Networking (SDN), virtualization, workload-optimized datacenter transport protocols, and, 40G–1Tbps Converged Enhanced Ethernet (CEE) fabrics, are independently responsible for arguably a networking revolution. Future datacenters are expected to scale beyond millions of VMs, assuming servers with 32-512 cores/server [2] and thousands of simultaneous tenants. This, together with the extreme variety of solutions – from physical fabrics up to hypervisors – renders the evaluation of virtualized datacenters an exceedingly complex task.

Traditional datacenters consist of lightly utilized servers running a "bare-metal" operating system or a hypervisor with a small number of virtual machines (VMs) running the applications. Their networks are static and manually managed – a costly and unsustainable mode of operation. The modern multitenant datacenters

are transitioning towards a dynamic infrastructure, including highly utilized servers running many VMs. While server virtualization has been used in IBM mainframes since the early '60s, it has only recently become widely available on commodity x86 servers. The same holds for storage virtualization.

3.1.1. Obstacles to Network Virtualization

From networking perspective, the main obstacles of large scale virtualization are: state explosion, arbitrary addressing constraints, and management difficulties. The upcoming multitenant datacenters contain up to multi-million physical servers, each potentially hosting tens, soon hundreds, of VMs. They impose new requirements on the datacenter network, which must now cope with multi-tenancy and automated provisioning, deletion, and migration of VMs.

The simplest solution would start with a large flat L2 network for each tenant. However, this approach does not scale within the practical constraints of current network devices, switches and routers. Each VM has its own virtual MAC, thus the size of the switch forwarding tables has to grow accordingly with one – soon two – orders of magnitude. Also the dynamic VM management stresses the broadcast domains. Using Q-in-Q or MAC-in-MAC encapsulation to overcome today's insufficient 4K Ethernet VLANs limit, squares the amount of state in each network switch. Furthermore, the datacenter network must support automatic provisioning and migration of VMs and virtual disks without imposing arbitrary constraints, such as VLAN/IP/MAC address, allowing tenants to choose their preferred addressing scheme without interfering with each other. Finally, network devices must be reconfigured at each new provisioning, deletion or migration of VMs. In a large heterogeneous network, containing different generations of devices, possibly from different vendors, the reconfiguration is both complex and error-prone requiring the management of multiple intricate scripts and configuration files.

3.1.2. Overlay Virtual Networks (OVN)

OVNs, prominent application of Software Defined Networking (SDN), are the emerging solution to overcome the problems outlined above and reach the next and final step: full network virtualization. Although a number of different overlays have been recently proposed [55, 110, 81, 88, 37, 24], their key architectural abstraction lies in the separation of virtual networking from the underlying physical infrastructure. Overlays enable the arbitrary deployment of VMs within a datacenter, independent of the physical network – even at runtime – without changing or reconfiguring the existing hardware.

The current overlays are predominantly built using Layer 2 to 4 encapsulation in UDP, whereby the virtual switches – typically located within the physical hosts – intercept the VM traffic, perform en-/de-capsulation and tunnel it over the physical

network. Each VM has an associated network state residing only in the adjacent virtual switch. Upon VM migration, virtual switches update their forwarding tables to reflect the new location. Using encapsulation over IP [88, 81, 37, 24], the VM locations are neither limited by the L2 broadcast domains, nor by VLAN exhaustion. Instead, the full IP functionality is preserved, including QoS and load balancing, independent of location, domains and the physical networking capabilities. Thus, virtual switches are similar to the traditional hypervisor switches, but with additional functions as overlay nodes. Virtual switches separate the datacenter tenants, both from each other and from the network devices, reducing the amount of state and simplifying the configuration.

There are a few aspects in which OVNs influence the datacenter networking. Firstly, on the data plane: OVNs use encapsulation to build tunnels between the virtual switches that host a connection's source and destination. Current encapsulation solutions such as, VXLAN [81] and NVGRE [110], solve the original VLAN limitation while reducing the management overhead. However, no performance evaluation of these encapsulation techniques has been published so far. Secondly, on the management plane: Network configuration, distribution and learning protocols are necessary for tunnel creation at each virtual switch. To create a tunnel, the overlay switch needs to map the destination address to its physical location. The overlay configuration management can be performed either by learning or in a centralized fashion. The *learning* approach, chosen for VXLAN [81], floods the packets with unknown destinations. In the *centralized* approach, virtual switches are responsible for retrieving the information required for encapsulation. In NetLord [88], this information is learnt by switches through communication with each other, and, from a central configuration repository. In Distributed Overlay Virtual Ethernet (DOVE) implementation [24, 37], this configuration information is retrieved from a centralized database.

Both the central configuration repository in NetLord and the centralized database in DOVE must be highly available and persistent, which poses a challenge for the multi-million node datacenters – thus raising the future third option of a distributed repository approach and its coherency protocols. For now the former two approaches, learning and centralized, are simpler to design and manage. Notably, the centralized method also inherently prevents flooding - unavoidable with learning. For our DOVE-like overlay investigation we have adopted and extended the centralized approach.

3.1.3. Why a per-Workload, Cross-layer OVN Study?

Given their rapid deployment, the SDN-based overlays throw the proverbial wrench in the datacenter network stack of socket-based transports (TCP and UDP) and congestion controls such as Random Early Detection (RED) and Quantized Congestion Notification (QCN). Despite their obvious benefits, some overlay-related drawbacks must be explicitly identified, investigated and addressed.

1. *Heavier* networking stack: An overlay introduces additional protocol layers that touch every flow, or even every packet, with the corresponding scalability implications for 10-100G Ethernet.
2. Transports, sensitivity to *congestion* and packet loss: The datacenter tenants aim to harvest maximum performance from the rented infrastructure. Therefore, we expect an increasing number of applications based on proprietary transports, many of them UDP based, which are neither congestion sensitive, nor react to losses in a TCP-friendly manner. This shifts the balance between TCP – approx. 97% of all the current datacenter traffic – and the other transports. How does this affect the future of congestion control?
3. Practically scalable and efficient implementations of the overlay *control*: State acquisition, distribution, communication, fault tolerance are yet to be proven. The same holds for state caching, which constitute a key design parameter studied in more detail here.
4. Legacy protocols getting "*lost in translation*": As one example, the Explicit Congestion Notification (ECN) markings are lost unless they are extended across the OVN. As this has critically affected our study, we have addressed the issue by contributing a simple, yet effective, OVN-aware ECN translation protocol.

Therefore, researchers as well as datacenter operators, and application writers have justified concerns about OVNs.

3.1.4. Workloads, Metrics and Guiding Questions

The above motivates us to investigate two well-known datacenter workloads. One is Partition/Aggregate (PA), a network- and protocol-sensitive application, the core of MapReduce and Hadoop, notoriously exposed to a particular form of congestion known as TCP incast [33, 117]. The other workload is a common datacenter workload, namely the 3-Tier (3T) application. Using query completion time as our primary metric, we study the OVN impact on these two applications' behaviors, their performance bounds with overlays, and whether RED or QCN can be adapted to this new environment.

Anecdotal evidence suggests performance degradation for the datacenter applications in a virtualized environment. Here we ask three guiding questions related to the above OVN issues:

1. What is the influence of TCP parameters on the application performance in an OVN environment?
2. Does a DOVE-like OVN impact the performance of our two workloads? What are the primary performance gating factors of a datacenter's overlay?
3. What are the best metrics to quantify the saturation of network resources in overlay virtual networks?

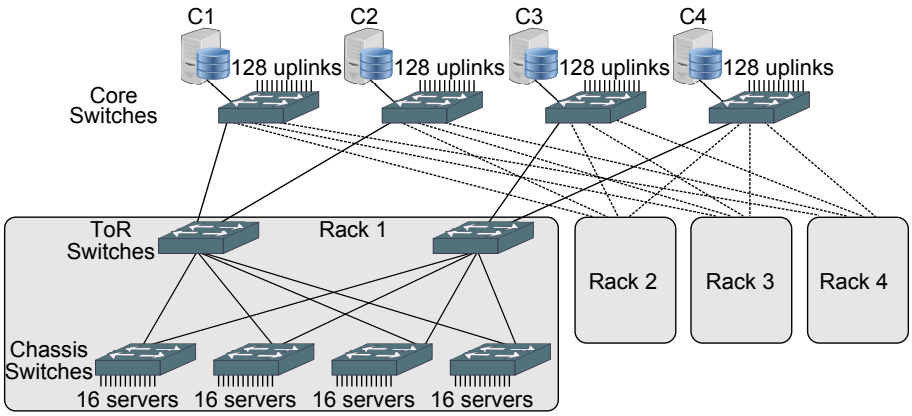


Figure 3.1.: Datacenter network topology with three layers of switches forming an Extended Generalized Fat Tree [91] *XGFT(3;16,4,4;1,2,2)*. The top ISP uplinks are used by the external clients to inject HTTP queries that are served by the tenants' VMs. The bottom 256 servers, grouped in 4 racks each rack containing 4 chassis, are virtualized holding up to 16 VMs each. The overlay controller is distributed and attached to the core switches.

In addressing these issues we hope to provide insights and guidance for the datacenter and overlay architects.

3.1.5. Contributions and Structure

The contributions of this chapter are are twofold:

1. We provide the first completion time-based evaluation of Partition/Aggregate *and* 3-Tier applications in a realistically virtualized datacenter network, using an *actual* TCP stack running over a *detailed* L2 CEE fabric model.
2. We measure the influence of the OVN design parameters on the user-perceived performance.

The rest of the chapter is structured as follows: In Section 3.2 we present the virtualized datacenter network stack, including a description of the overlay virtual networks. We describe the selected applications in Section 3.3 and the evaluation methodology in Section 3.4. We discuss the results and answer the three guiding questions in Section 3.5, Section 3.6 and Section 3.7, respectively. Finally, we present a selection of some related work in Section 3.8, and we conclude in Section 3.9.

3.2. Virtualized Datacenter Network Stack

In this section we present the networking stack of a virtualized datacenter. In contrast with the traditional non-virtualized stack presented in Section 2.2, a new layer is inserted between TCP and the network hardware. Table 3.1 summarizes all the main parameters of this section.

Table 3.1.: Model parameters

Parameter	Value	Unit	Parameter	Value	Unit
(L2) ***Network hardware***					
link speed	10	Gb/s	adapter delay	500	ns
frame size	1518	B	switch buffer size/port	100	KB
adapter buffer size	512	KB	switch delay	500	ns
(L2) ***Congestion management (QCN)***					
Q_{eq}	66	KB	fast recovery thresh.	5	
W_d	2		min. rate	100	Kb/s
G_d	0.5		active incr.	5	Mb/s
CM timer	15	ms	hyperactive incr.	50	Mb/s
sample interval	150	KB	min decr. factor	0.5	
byte count limit	150	KB	extra fast recovery	enabled	
(L3) ***ECN-RED***					
min thresh.	25.6	KB	W_q	0.002	
max thresh.	76.8	KB	P_{max}	0.02	
(L2-4) ***SDN Overlay***					
request size	64	B	encap. overhead	54	B
reply size	64	B	request RTO	10	ms
delay	20	μs			
(L4) ***TCP***					
buffer size	128	KB	TX delay	9.5	μs
max buffer size	256	KB	RX delay	24	μs
timer quanta	1	μs	reassembly queue	200	seg.
(L4) ***UDP***					
TX delay	9.5	μs	buffer size	128	KB
RX delay	24	μs			

3.2.1. Layer 2: Converged Enhanced Ethernet

We use an Extended Generalized Fat-Tree (XGFT) [91], which is a common topology in today's datacenters [14, 89, 54]. The details of the topology are given in Figure 3.1. The fabric is based on 10G Ethernet. Each network adapter uses virtual output queues (VOQ), one for each destination, to avoid the primary head-of-line blocking. Switches use an input-buffered output-queued architecture to store the incoming frames in the input buffer, and in parallel also enqueue them in the output queue. We assume N-fold (ideal) internal speedup and full buffer sharing for optimal performance. Although idealized, some of these characteristics are available in high-end products e.g. [1].

Adapters and switches optionally support QCN [11] as a L2 congestion management scheme. The QCN load sensors, placed in each switch output port, sample the *instantaneous* output queue occupancy. If the queue is deemed congested, feedback

is sent back to the traffic sources identified as culprits. Here the associated QCN reaction points will control the injection rate to match the available network capacity.

3.2.2. Layer 3: RED and ECN

Random Early Detection [49] (RED) detects congestion based on the *average* queue length. Unlike the QCN load sensor, a well tuned [111] RED is burst-tolerant by design. The onset of congestion is usually notified though Explicit Congestion Notification (ECN), which marks the suspected culprit packets at L3 and provides feedback to the originating source at L4. The ECN marking is done only if the IP packet is flagged as ECN capable.

In an overlay network environment the original packets are encapsulated. Thus the two IP headers can potentially confuse the switches and routers that perform ECN marking. In a first approach, switches must correctly detect the encapsulated frames, parse the headers and mark the inner IP header. This method lacks flexibility and requires hardware support. Therefore we adopted a second approach in which physical switches mark the encapsulation header only. The virtual switches must transfer the ECN bits from the inner IP header to the encapsulation IP header and back. We include in the model this transfer mechanism, similar to [31], in order to preserve the ECN information during en-/de-capsulation across the overlay.

3.2.3. Overlay Network

In between the Ethernet hardware and the TCP stack we add a model of the overlay network. The generic overlay network architecture—inspired by DOVE [37, 24]—contains at least (i) a controller, and, (ii) a set of virtual switches. Each server is virtualized, hosting several VMs and a virtual switch. The VMs may belong to different applications, potentially owned by different tenants. The virtual switch extends a traditional hypervisor switch with added overlay functionality. This can be also located within a physical SDN-enabled switch, acting as gateway for legacy networks and appliances.

An example of the basic operation of the virtual overlay network is presented in Figure 3.2. The administrator defines the overlay connectivity and policy configurations rules. The controller stores these rules, which are thereafter enforced by the designated virtual switches. Each VM interface is assigned by the switch to a specific overlay instance by the management plane and this is reflected in the configuration of the virtual switch hosting the VM instance.

More in detail, the SDN overlay switches are placed at the edges of the physical network and host the endpoints of the overlay. These switches can be both SDN-enabled physical switches or, as mentioned previously, placed inside the hypervisor hosting virtual machines. The OVN switches must: (i) make the overlay transparent

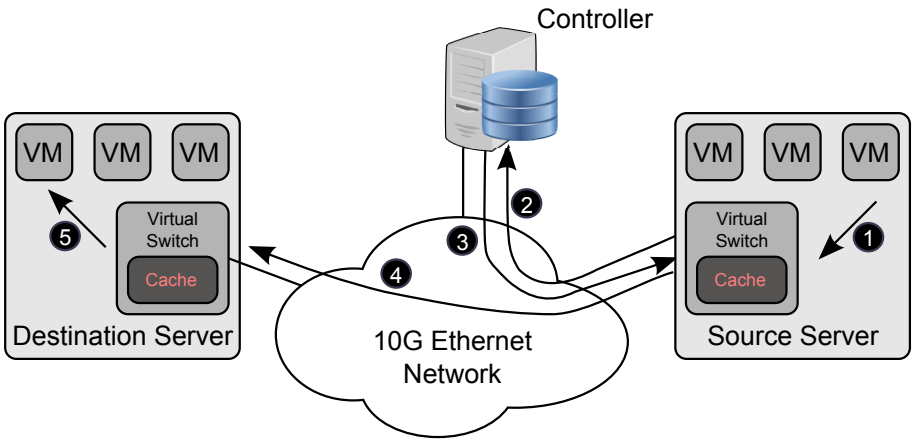


Figure 3.2.: Overlay network operation. When receiving a packet from source VM (1), the virtual switch must first learn the location (trivial if local) and policies of the destination VM. First the switch checks the cache. If miss, the switch will query the overlay controller (2), by sending a request. The controller will eventually send a reply (3) containing the L2/L3 address of the destination server. Upon receiving the reply, the requesting switch caches the location and policy of the destination VM for further use. If hit, or after the controller reply was received, the triggering data packet is tunneled across the physical network, encapsulated within a new Ethernet header, IP header, UDP header and encapsulation header (4). The destination switch terminates the tunnel and delivers the data packets to the destination VM (5).

to the endnodes, and, (ii) enforce the overlay connectivity and policy rules. Both are achieved using tunneling. Tunnel encapsulation allows full network virtualization, but incurs an additional per frame overhead. The encapsulation headers are used to identify the endpoints on the overlay and to send the encapsulated packets to the overlay switch hosting the destination endnode under the constraints of the OVN policy rules. To avoid the harmful effects of fragmentation, we assume that the MTU on the endnodes is decreased proportionally to the added encapsulation overhead.

The OVN controller is used to store the connectivity configuration (address mappings) and policy rules for the SDN overlay. The controller is queried by the OVN switches. As we focus on the data-plane rather than on the control plane: (i) we consider a centralized overlay controller and (ii) we do not try to enforce particular policy rules. Overlay switches communicate with the controller over UDP. A retransmit timer is used to recover lost queries.

3.2.3.1. Encapsulation and Tunneling

The L2 traffic crossing the physical network is encapsulated to provide traffic isolation between different virtual networks and multi-tenancy support: e.g., preserve tenants' original IP and MAC addressing schemes. The encapsulation per se adds at least the cost of an additional per frame header, and possibly more. A few different encapsulation protocols have been proposed prior to DOVE, which itself is

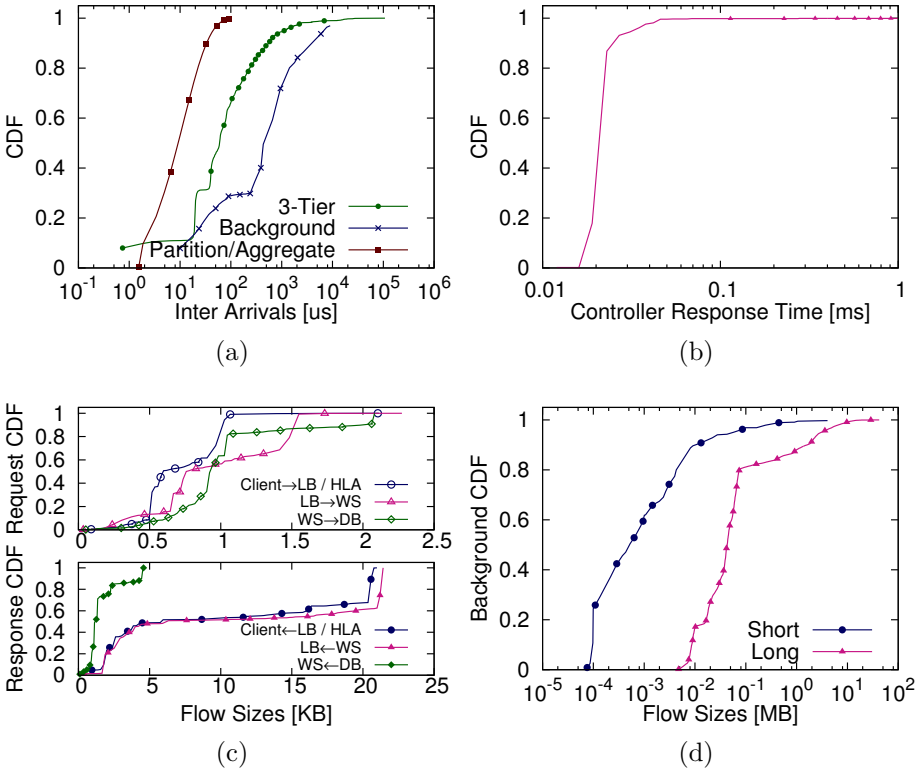


Figure 3.3.: Traffic parameters. (a) Background flows and HTTP queries inter-arrivals for 3T and PA. (b) Overlay controller delays. (c) Flow size distributions for 3T and PA workloads. (d) Background flows size distributions given in [16, 26].

encapsulation-agnostic. In our overlay model, however, we add 54B reflecting a VXLAN-type encapsulation: 18B outer Ethernet header + 20B outer IP header + 8B UDP header + 8B VXLAN header. To avoid fragmentation we decrease accordingly the MTU from 1500B to 1446B. Modern CEE hardware can instead increase their physical MTUs by a few tens to a hundreds of bytes, thus preserving the default software application settings. Host-local traffic is not encapsulated in our model.

3.2.3.2. Overlay Cache

In order to retrieve the connectivity and policy configurations, virtual switches send queries to the controller. To avoid per frame queries, each switch has a *cache*, in which we have implemented 5 *eviction* strategies. Depending on its implementation in either hardware or software, the cache can be either fast and small, or slow and big, respectively. To capture the cache influence on the workload completion times, we have artificially downscaled the cache sizes to reflect our proportionally smaller environment. All the controller queries are sent over UDP. A configurable retransmission timer is used to recover lost queries/replies. For the overlay configuration entries, we use 64B queries and replies. Queries are replied after a processing de-

Table 3.2.: TCP RTO parameters

Parameter	Value	Unit	Parameter	Value	Unit
Default configuration					
RTO base	3	s	RTO slop	200	ms
RTO min	30	ms			
Tuned & RED configurations					
RTO base	20	ms	RTO slop	20	ms
RTO min	2	ms			

lay, extracted from the distribution shown in Figure 3.3b. The average measured response time is 0.03 ms with a standard deviation of 0.07 ms. This we have measured from a threaded UDP daemon with in-memory hash tables for fast lookups running on a Intel i5 3.2GHz machine with 4 GB of memory and an Intel e1000e 1G Ethernet NIC. This values account only for the software delays, thus disregarding the network. These samples were collected under low load to reflect the infinite CPU capacity assumption of the simulator, i.e. query response times independent of the controller load.

3.2.4. Layer 4: TCP Stack

We consider three TCP versions: NewReno, CUBIC and Vegas. NewReno is an improvement of Reno, also the default in Linux kernels up to version 2.6.8. CUBIC [60] has been optimized for fast networks with high delay (due to flight lags in WANs); it is an RTT-independent scheme, namely a less aggressive derivative of Binary Increase Congestion control (BIC) using a cubic function to probe for the maximum congestion window. This provides faster bandwidth recovery after congestion. From version 2.6.19 onwards, CUBIC is the Linux kernel default. Both NewReno and CUBIC rely on lost and/or ECN-marked packets as congestion feedback. In contrast, Vegas [30] uses the RTT delay as primary congestion measure and represents the delay-probing class of TCPs, including Compound TCP and Adaptive Reno. Vegas avoids congestion by comparing the expected throughput in the absence of congestion with the actually achieved throughput.

We ported the TCP stack from a FreeBSD v9 kernel into Venus with minimal changes, mostly related to memory management. As we focus on the network, we do not model the CPUs, assuming that the endnodes can process data segments as fast as they arrive, and that the applications can immediately reply - hence an idealized computational model that increases the communication pressure. The TCP stack adds only a *fixed* delay to each segment, calibrated from our prior test-bed experiments. Even if idealized, these assumptions are consistent with our network-centric methodology. To calibrate the model, we instrumented the TCP stack running on the same hardware as previously described.

The original set of TCP parameters was conservatively chosen to accommodate the link speeds and router hardware of two decades ago. During the calibration runs we noticed that the minimum RTT was much smaller than the kernel timer quanta, which equals 1 ms by default. With this setup the retransmission time-out (RTO) estimator was ineffective. Therefore for all the experiments we reduced the timer granularity to 1 μs. Based on [117, 33] and our own measurements we found that the choice of the RTO parameters can strongly impact the performance. The BSD kernel has three such parameters: the default RTO (*RTO base*), the minimum RTO (*RTO min*) and the RTO variance (*RTO slop*). *RTO base* is used by the kernel in the absence of information when a connection is initialized. *RTO slop* is a constant factor, accounting for the variable delays in segment processing at the endnode kernels. It is added to the RTO value computed in real time using Karn's algorithm and Jacobson's algorithm [65].

We used the two sets of RTO parameters from Table 3.2. The first set, called *Default,* is using the default parameters from the BSD kernel i.e. RTO base = 3 s, RTO min = 30 ms and RTO slop = 200 ms. The *Default* set is built conservatively to accommodate slow dial-up links and legacy hardware. We constructed a second set named *Tuned* to match the modern processors and the delays found in current datacenters i.e. RTO base = 20 ms, RTO min = 2 ms and RTO slop = 20 ms. The base RTO for the *Tuned* parameter set is chosen to be larger than the worst case RTT of network – less than 10 ms – to which we added the RTO of the OVN request – another 10 ms.

Table 3.1 and Table 3.2 summarize the main parameters for the TCP stack used in our simulations. The simulator also incorporates a thin UDP layer used for background flows, performing simple segmentation and encapsulation of the application data.

3.3. Application Models

We evaluate the cross-layer performance impact of different L2, L3 and L4 protocols. As mentioned before we selected Partition/Aggregate and 3-Tier web server systems. While the 3T applications are widely used, [72] and [16] show that the scatter/gather communication pattern, as in PA, is representative of modern datacenter applications.

We have designed our commercial traffic generator based on findings from a few recent papers. [26] presents an in-depth study of the spatial and temporal distribution of the flows in ten production datacenters. [16] uses a similar approach to measure the size and inter-arrival time distribution of the flows. [72] observed that modern applications use a Scatter/Gather communication pattern. The traffic study from [16] confirms that finding.

In a realistic environment, multiple applications run in parallel and the multitenant

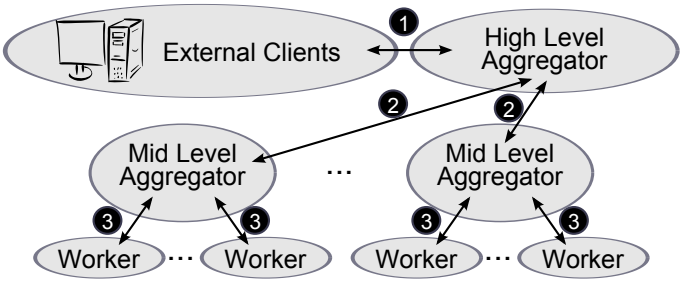


Figure 3.4.: Partition/Aggregate operation: External clients produce HTTP requests served by the High-Level Aggregators (HLA) randomly distributed within the datacenter (1). Upon the reception of such a request, the HLA contacts randomly selected Mid-Level Aggregators (MLA)—one per chassis—and sends them a subquery (2). The MLAs further split the subquery across their workers, one in each server from the same chassis (3). Eventually each worker replies to the MLA by sending back a response. The MLA collects the partial results from workers. When *all* the results have been received, the MLA sends back its aggregated response to the HLA. The query is completed when the HLA receives the aggregated response from each MLA.

performance inelasticity/isolation is subject of ongoing research [39]. Using the real-life data from [16, 26] we created a traffic generator that injects a *foreground traffic* matrix of queries on top of a *background traffic* matrix of random flows. The queries are generated as outlined in the following two sections. For the background flows, each source randomly chooses a destination constrained by the ratio of intra-chassis to inter-chassis traffic of 30% reported in [26]. Then each source draws from the inter-arrival time (Figure 3.3a) and flow size distributions (Figure 3.3d) and sends the background traffic over a TCP or UDP socket. We use two background flow size distributions – short and long – to model a low loaded network as well as a medium loaded one.

The simulation is ended after a fixed number of queries are processed. For the queries as well as for the background flows we collect the completion time as an application level metric [46].

3.3.1. Partition/Aggregate Workload (PA)

The operation of a PA web server is described in Figure 3.4. For simplicity all messages exchanged between High-Level Aggregators (HLAs), Mid-Level Aggregators (MLAs) and workers have a fixed size of 20KB in both directions. Each of the 256 servers from Figure 3.1 contains VMs that allow it to simultaneously act as HLA, MLA or worker for different queries. The external client HTTP queries are injected with inter-arrival times shown in Figure 3.3a and with the flow sizes shown in Figure 3.3c. The inter-arrival distribution is based on [16] but accelerated 100-fold to reflect our environment, i.e., faster links, larger switch buffers and lower delays.

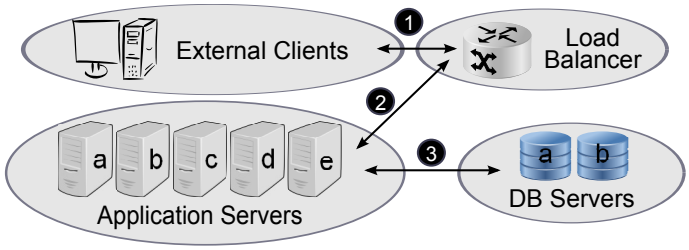


Figure 3.5.: 3-Tier operation: The entry points for external client HTTP requests are the load balancers (1), which forward the request to a random web server (2). 28% of the requests require additional dynamic content that is retrieved from a database server (3). Once the SQL result has been received, the web server assembles the HTTP response and sends it back to the load balancer, which forwards it to the originating client.

3.3.2. 3-Tier Workload (3T)

Similar to the PA, the 3T application receives HTTP requests from external users. However, instead of splitting and processing them in parallel as in PA, here, at each level, the request is handled by a single VM. As shown in Figure 3.5, three types of VM servers are involved: load balancer (LB), web server (WS) and database server (DB). We assume that 256 tenants share the datacenter, each one deploying a 3T application with 8 VMs: 1 load balancer, 5 web servers, and 2 database servers (see Figure 3.5). We *anti-collocate* these 8 VMs by randomly placing them across physical servers, with up to 16 VMs/server, and no 2 VMs of a tenant hosted on the same server. Thus the primary bottlenecks are the physical datacenter fabric and the overlay. Furthermore, each tenant receives external queries through a dedicated uplink, thus avoiding ISP-related congestion. The size of the requests and replies are drawn from the distributions shown in Figure 3.3c. We have extracted these distributions by instrumenting a 3T workload based on RUBiS v1.4.3 [9]. The instrumented system was installed on four physical machines, one for each tier, plus one for RUBiS emulating 80 external clients. The inter-arrival distribution for the 3T application is also measured using the instrumented system previously described. As before, we apply a 50-fold acceleration factor.

3.4. Methodology

3.4.1. Simulation Environment

For our performance evaluation we rely on cross-layer simulations augmented with experimental hardware results as described below. Simulation-base modeling has the following merits. (a) *Observability*: The path of each packet can be traced, every link and queue can be measured. (b) *Repeatability*: Traffic patterns and event ordering are deterministic. (c) *Flexibility*: Can implement new L2 features that re-

quire changes of protocols, scheduling or queuing disciplines. (d) *Availability*: One can test CEE-hardware with functions that are still in early-design phases, expensive or not commercially available. However, simulations may lack (a) *Accuracy*: Models, however accurate, may not reproduce all the subtle interaction facets of a real implementation; and (b) *Speed*: Event-based simulations require the *serial* processing of many billions of events, hence a 2 to 4 orders of magnitude slowdown compared to hardware FPGA emulators or ASIC-based product implementation, respectively.

Although ns2/ns3 [6] are well established in the research community, they focus on higher abstraction levels. Our research requires realistic L2 simulations of CEE switch and adapter micro-architectures including management and monitoring of the queues and buffers, scheduling, link-level flow control and memory management. Furthermore the TCP models implemented in NS are different from the actual TCP versions implemented in BSD, AIX and Linux kernels. The NS-3 TCP libraries are streamlined and simplified, thus trading accuracy for simulation efficiency.

Hence we settled on a custom-built L2 network simulator called Venus [85] and described in Section 2.5.1 and Section 2.5.2. The simulator is extended with a port of the TCP/IP stack extracted from the FreeBSD v9.0 kernel as described in Section 2.5.3.

3.4.2. Experiments

The experiments are arranged in three groups:

1. In Section 3.5, we begin with an investigation of the performance impact of TCP parameters in an overlay network environment.
2. Next, in Section 3.6, we analyze the influence of the overlay network components on the flow completion times of the 3T and PA applications.
3. Finally, in Section 3.7, we study the saturation of network resources in overlay virtual networks.

3.5. TCP Parameters Influence

In this section we use the following names. *Default*, *Tuned* and *RED* identify the TCP configuration. *Default* and *Tuned* use the two sets of TCP parameters described in Section 3.2.4. *RED* has the same TCP parameters as the *Tuned* configuration in addition with ECN/RED enabled in the network.

No OVN, *Per flow Cache, 8-entry Cache*, and *Infinite Cache* are the OVN configurations. The *No OVN* configuration is our baseline running without SDN-based overlay virtual networking. When running with OVN, we tested three different

Background Flows		Cache	Default mean [ms]	Default std [ms]	Default OVN cost [%]	Default losses [‰]	Tuned mean [ms]	Tuned std [ms]	Tuned OVN cost [%]	Tuned losses [‰]
Short		**No OVN**	6.16	71.21	0.00	0.14	3.65	14.40	0.00	0.25
		8-entry	8.97	123.35	45.62	0.09	5.16	10.68	41.37	0.13
		Per Flow	10.86	129.35	76.30	0.12	4.39	17.51	20.27	0.24
		Infinite	9.73	119.60	57.95	0.12	3.70	13.50	1.37	0.26
Long		**No OVN**	20.88	178.63	0.00	0.85	11.11	28.65	0.00	2.83
		8-entry	27.15	237.11	30.03	1.01	10.11	16.49	-9.00	2.75
		Per Flow	22.54	198.13	7.95	0.86	11.94	30.68	7.47	2.58
		Infinite	25.99	227.02	24.47	0.95	11.50	29.97	3.51	2.85

Background Flows		Cache	RED mean [ms]	RED std [ms]	RED OVN cost [%]	RED losses [‰]
Short		**No OVN**	1.85	1.95	0.00	0.01
		8-entry	3.57	2.91	92.97	0.02
		Per Flow	3.15	10.12	70.27	0.03
		Infinite	1.93	2.75	4.32	0.02
Long		**No OVN**	2.60	5.13	0.00	0.07
		8-entry	4.66	5.88	79.23	0.08
		Per Flow	3.36	5.87	29.23	0.09
		Infinite	2.75	7.25	5.77	0.08

Table 3.3.: Query completion time (mean / standard deviation) and losses.

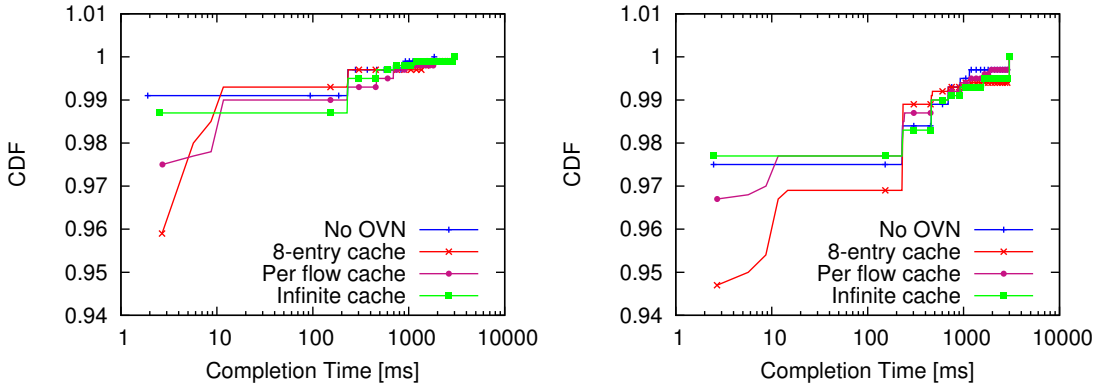
cache policies for the OVN cache. The *Per flow Cache* makes a single request to the controller for each TCP connection. This approach is similar with OpenFlow [82] where the switch tables are populated per flow. This cache penalizes the completion time of short flows that have to wait for the reply of the controller before sending a few packets. On the other hand, for the long flows, the delay of the request is amortized over a large number of packets.

The *8-entry Cache* configuration uses a small cache with 8 entries with FIFO eviction. This cache is expected to trigger frequent misses for the workloads that contact more than 8 destinations in parallel. The *Infinite Cache* assumes a cache large enough to store all the possible destinations. Therefore it will issue a request only for the first packet addressed to each destination.

The results are gathered in Table 3.3, which shows the mean and standard deviations of the query completion times in the above configurations. In addition we compute a OVN cost i.e. the degradation of the query completion time with respect to the *No OVN* configuration. The OVN cost includes both the cache price and the encapsulation overhead. Figure 3.6, Figure 3.7 and Figure 3.8 show the cumulative distributions of the query completion times for small and large background flows respectively.

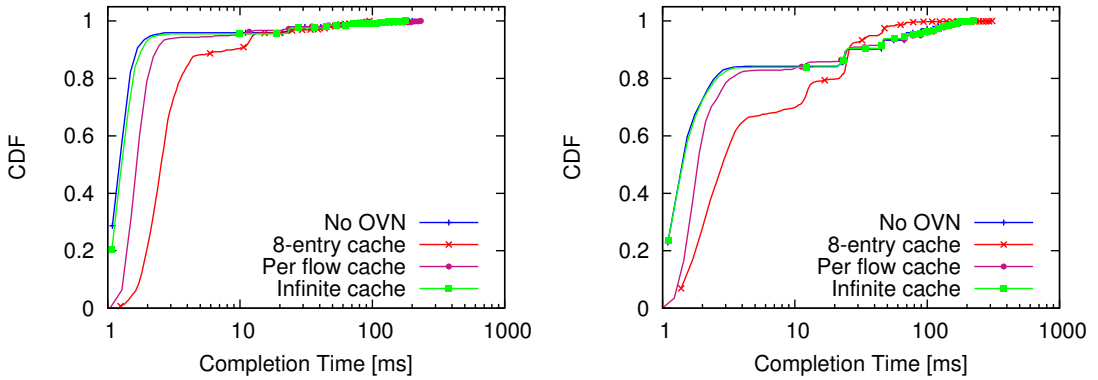
3.5.1. TCP Configuration Impact

The first observation is that the *Default* TCP parameters were unsuitable for the simulated datacenter networks. When we compare *Default* with *Tuned* in the *No OVN* configuration we find that the average completion time deteriorated by 68%



(a) Short background flows. (b) Long background flows.

Figure 3.6.: Partition/Aggregate query completion time with *Default* TCP.

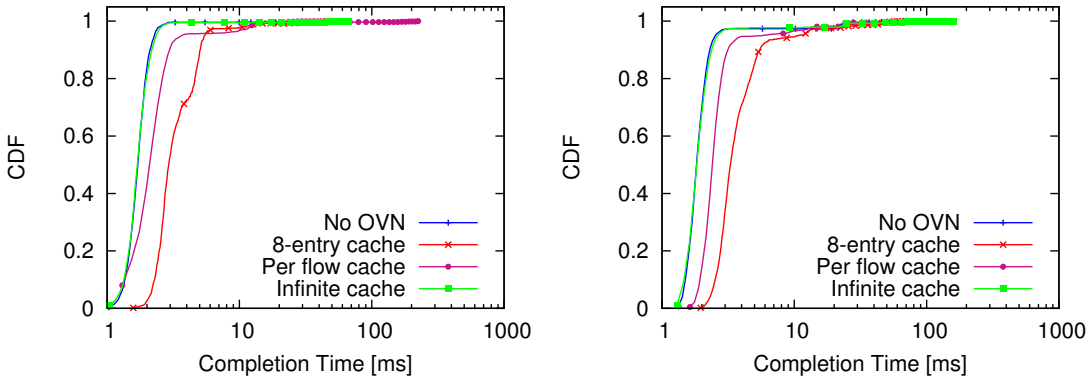


(a) Short background flows. (b) Long background flows.

Figure 3.7.: Partition/Aggregate query completion time *Tuned* TCP.

and 87%. Importantly, the standard deviation was one order of magnitude larger than the average. This happened because the very few transactions that lost a SYN or SYN/ACK segment had to wait as long as 3s for the RTO before resuming. The distributions from Figure 3.6a and Figure 3.6b have long tails. We conclude that the *Default* configuration is highly unreliable.

The activation of the OVN had no effect *Tuned* behaving consistently better than *Default*. We observed that the large standard deviation compared to the average still persisted despite the tuning. The problem was solved by activating *RED*: with *RED*, standard deviation values were much closer to the average compared to the other configurations. Note that *RED* also improved OVN performance. These results demonstrate that the OVN must offer support for RED, REM or other L3 congestion management techniques.



(a) Short background flows.

(b) Long background flows.

Figure 3.8.: Partition/Aggregate query completion time with *RED* TCP.

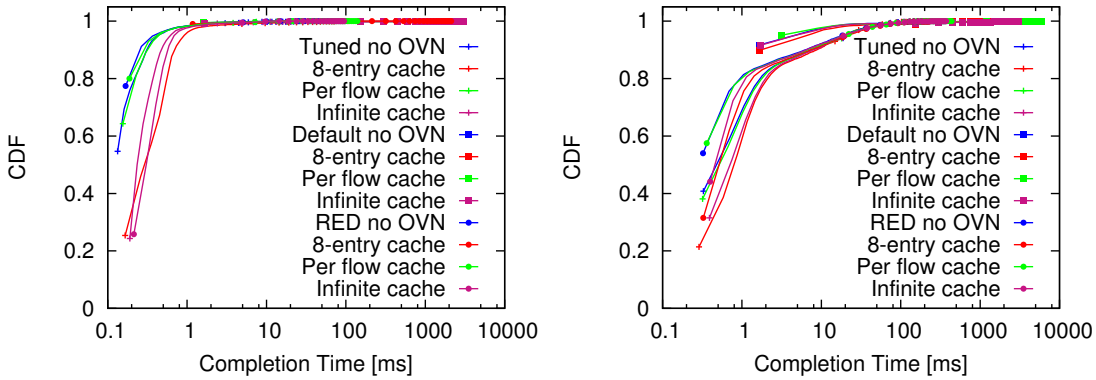
3.5.2. OVN Performance Impact

Generally the activation of the OVN led to a degradation of the performance. The partition/aggregate applications were heavily penalized by the 8-entry cache as visible for example in Figure 3.7a where we see a deterioration with 41% of the average completion time in the *Tuned 8-entry* configuration with short flows. The worst results are in the *RED 8-entry* configuration where the completion time almost doubled – up to 92% deterioration. This is because the partition/aggregate has to contact in parallel all the other 15 workers in the same rack. This leads to frequent evictions from the insufficient cache and lot of extra traffic directed towards the OVN controller.

An unexpected result is the 9% improvement caused by the 8-entry cache in the *Tuned* configuration with long background flows. The partition/aggregate workloads produce the TCP incast congestion because in most of the cases the workers send their responses synchronized. Because of the small cache some of the workers have to wait for the OVN controller to reply before answering to the MLA. Therefore the workers are desynchronized and the TCP incast congestion is reduced.

The per flow cache in general exhibited a better performance than the 8-entry cache. But still the partition/aggregate suffers a lot because it generates short 20KB flows. As expected, the clear winner was the infinite cache configuration. We observe that in *Tuned* and *RED* configurations in Figure 3.7 and Figure 3.8 the difference between the baseline *No OVN* and *Infinite Cache* is negligible. The OVN controller is interrogated only for the first partition/aggregate query and subsequent queries use the already cached information. Hence the performance degradation can mostly be attributed to the lower efficiency induced by the encapsulation overhead.

This results confirmed our expectation that the performance degradation induced by the OVN is caused by two factors: OVN controller request/reply delays and encapsulation overhead. We discovered that the delays induced by OVN requests



(a) Short flows.

(b) Long flows.

Figure 3.9.: Background flow completion time.

can be alleviated using proper caching whereas the encapsulation overhead delays are negligible.

3.5.3. Background Flows

The background flows are by nature desynchronized whereas the queries are synchronized. Therefore, the TCP incast phenomenon observed for the queries did not affect the background flows. This was the cause of the almost identical completion times that we observed for both short and long background flows in Figure 3.9a and Figure 3.9b (note the logarithmic scale). In the *Default* configuration we observed the long tailed distribution of completion times similar to what we observed for the queries.

3.6. Overlay Network Evaluation

We run the two datacenter applications (*3T* and *PA*) described in Section 3.3 in different setups. In this section we use the following naming convention. The congestion management (CM) scheme in use is either absent (*w/o CM*), *RED* or *QCN*. Our applications run over a background traffic matrix that can be either absent (*w/o BKGD*), *TCP*-based or *UDP*-based. The TCP versions are *NewReno*, *Cubic* and *Vegas*. As shown in the previous section, the *Tuned* parameters reduces flow completion times, therefore in this section we will use exclusively these parameters. In addition, we vary the parameters of the overlay network as described below.

BKGD	Overlay	RED					QCN				
		5-p. [ms]	median [ms]	95-p. [ms]	cost [%]	loss [‰]	5-p. [ms]	median [ms]	95-p. [ms]	cost [%]	loss [‰]
w/o	**Base**	3.66	25.82	69.38	0.0	3.95	5.38	47.54	90.06	0.0	7.99
	Encap	3.99	46.93	69.97	**81.74**	7.02	24.96	47.96	90.22	**0.89**	11.24
	Inf	4.28	47.40	70.48	**83.56**	7.25	25.07	48.20	91.60	**1.38**	11.16
TCP	**Base**	5.44	28.10	70.71	0.0	5.07	26.62	49.88	76.11	0.0	9.91
	Encap	6.23	48.23	71.28	**71.62**	8.33	26.97	50.41	93.35	**1.07**	12.91
	Inf	25.57	48.70	72.05	**73.31**	8.47	27.67	51.17	94.20	**2.59**	12.98
UDP	**Base**	25.93	48.45	92.82	0.0	7.27	25.85	49.57	114.64	0.0	11.96
	Encap	26.39	49.49	93.74	**2.15**	10.12	26.15	51.29	115.88	**3.46**	15.34
	Inf	26.58	49.74	96.55	**2.67**	9.88	26.27	49.62	93.43	**0.09**	14.42

Table 3.4.: Partition/Aggregate: Query completion time and losses with TCP NewReno.

BKGD	Overlay	RED					QCN				
		5-p. [ms]	median [ms]	95-p. [ms]	cost [%]	loss [‰]	5-p. [ms]	median [ms]	95-p. [ms]	cost [%]	loss [‰]
w/o	**Base**	0.91	1.70	5.04	0.0	2.93	0.84	1.51	47.10	0.0	8.77
	Encap	0.92	1.85	24.04	**9.43**	6.81	0.87	1.66	73.01	**10.48**	13.43
	Inf	0.93	1.87	24.16	**10.32**	7.02	0.87	1.65	79.33	**9.75**	15.36
TCP	**Base**	1.20	2.97	48.45	0.0	12.02	1.36	28.60	526.61	0.0	17.52
	Encap	1.23	3.22	70.92	**8.39**	19.22	1.37	34.72	555.40	**21.40**	18.53
	Inf	1.22	3.21	71.08	**7.92**	19.66	1.38	38.20	540.41	**33.57**	18.14
UDP	**Base**	1.09	3.29	73.21	0.0	50.39	1.06	5.91	246.89	0.0	41.44
	Encap	1.17	4.24	97.33	**28.94**	68.59	1.09	12.23	313.17	**106.96**	50.50
	Inf	1.15	3.78	94.44	**14.87**	61.68	1.07	9.72	274.50	**64.42**	49.49

Table 3.5.: 3-Tier: Query completion time and losses with TCP NewReno.

3.6.1. Overlay Network Performance Impact

The overlay network introduces two types of overhead: encapsulation and discovery overheads. The encapsulation overhead is caused by the additional 54B header. Each overlay cache miss incurs a discovery overhead. In this case, the virtual switch has to query the central controller to find out the physical IP and MAC address of the destination VM. To estimate the two overheads we start with a baseline (*Base* configuration) without encapsulation and discovery overhead. This is equivalent to a non-virtualized network where the virtual switches forward raw packets from one interface to another. Next we add encapsulation (*Encap* configuration) to separate the tenants from each other. Finally we account also for the discovery overhead (*Inf* configuration). In this case we assume – an idealized – infinite cache scenario.

In Table 3.6 and 3.5 we print the completion times of the client queries for both the PA and 3T application. In Figure 3.11 we plot the cumulative distributions of completion times for PA and 3T only in the scenarios without congestion management (space constraint). As seen in Figure 3.11 the result distributions are non-normal, hence we report the 5-percentile, the median (50-percentile) and the 95-percentile of the completion times rather than the mean and standard deviations. We observe the completion times for PA are 10× to 50× larger than for 3T. In PA, packets

BKGD	Overlay	Partition/Aggregate					3-Tier				
		5-p. [ms]	median [ms]	95-p. [ms]	cost [%]	loss [‰]	5-p. [ms]	median [ms]	95-p. [ms]	cost [%]	loss [‰]
w/o	**Base**	3.81	47.53	70.55	0.0	7.94	0.84	1.51	24.12	0.0	13.14
	Encap	24.81	47.80	71.46	**0.58**	11.10	0.89	1.69	28.14	**12.17**	25.46
	Inf	25.24	48.23	91.99	**1.48**	11.37	0.89	1.71	30.12	**13.40**	26.30
TCP	**Base**	26.87	49.94	92.10	0.0	10.66	1.24	3.79	120.96	0.0	46.57
	Encap	27.41	50.54	93.69	**1.19**	14.23	1.25	4.42	146.98	**16.74**	53.01
	Inf	27.89	51.49	94.19	**3.11**	13.59	1.24	4.35	144.70	**14.89**	43.24
UDP	**Base**	26.15	49.61	95.41	0.0	11.68	1.03	3.15	95.13	0.0	79.91
	Encap	26.51	50.70	95.04	**2.20**	15.10	1.08	4.17	122.21	**32.50**	103.62
	Inf	26.94	58.66	113.00	**18.25**	14.81	1.06	3.67	117.93	**16.50**	96.75

Table 3.6.: Without congestion management: Query completion time and losses with TCP NewReno for Partition/Aggregate and 3-Tier applications.

are dropped in each of the two aggregate phases, whereas for 3T the congestive events are rare. Therefore the completion times for PA incorporate more RTOs, of at least 20ms each. As figure of merit, we report the overlay performance cost with respect to the Base configuration, computed on the median completion times as $\frac{CT - CT_{Base}}{CT_{Base}} \cdot 100$.

For PA without congestion management, we observe that the encapsulation overhead affects the median completion times with an increase from 0.58% up to 2.20%. Similar results are obtained with QCN where the cost is at most 3.46%. Enabling RED reduces the completion times by over 40% with respect to the Base case. Activating the encapsulation increases the completion time up to 81.74%. The RED control loop is destabilized by encapsulation because the TCP control segments – not RED controlled – double up in size. Ditto for the 3T workload. Without congestion management the 3T performance impact ranges from 12.17% to 32.50%. The PA application performance is mainly impacted by TCP *incast* when up to 16 flows simultaneously target the same bottleneck. On the other hand, the 3T workload is mainly impacted by the effective flow size increase, therefore the effect of encapsulation is more visible here.

Next we add the infinite cache. As expected for the PA application, with and without RED, the cache slows down the queries even more than in the previous case. This additional delay is around 2% in most cases, up to 16.05% (see black vs. blue figures in Table 3.6 and 3.5). The unexpected result is the slight *gain* in performance experienced by the 3T application when the background flows are active (see black vs. red figures in Table 3.6 and 3.5). The background traffic is stalled by the cache queries, therefore its injection rate is lowered leading to reduced completion times. We observe a similar phenomenon for the PA with QCN *and* UDP background flows.

The main cause of the performance degradation is the increase in traffic caused by encapsulation. In Figure 3.11 we can see that the line corresponding to infinite cache almost coincides with the encapsulation only line. Thus, for such workloads where the sources and destinations of each flow belong to the same constant set, the

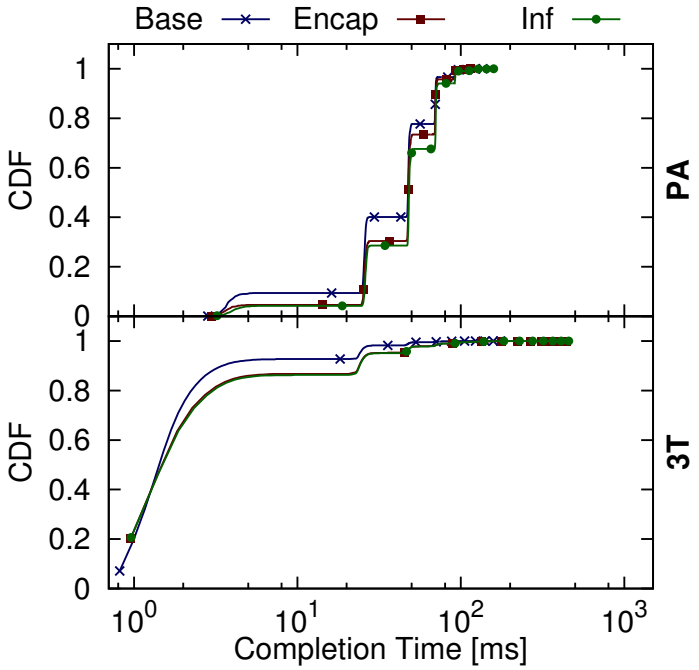


Figure 3.10.: Cumulative distributions of completion times for TCP NewReno, without congestion management, without background flows.

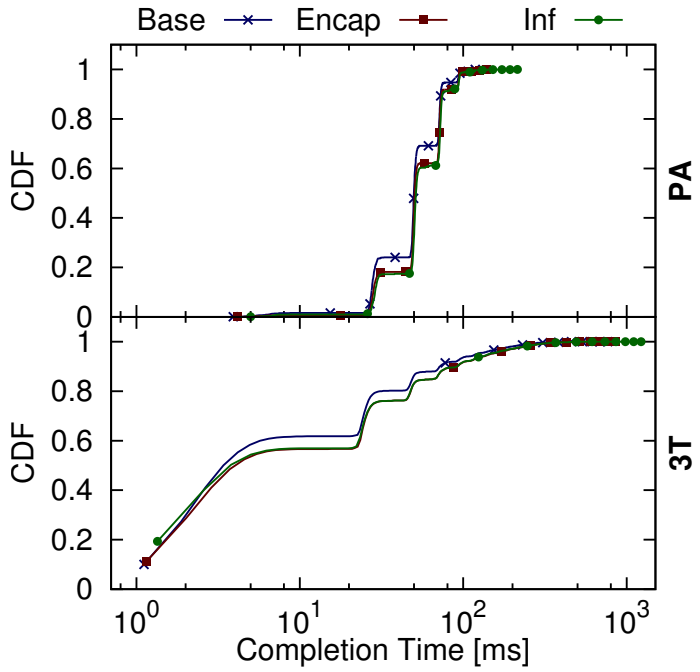
discovery overhead is low compared to the encapsulation overhead. The distributions have a stair-like appearance because of the discrete values of the RTOs. The distributions' tails coincide because for some flows the repeated packet loss or loss of initial SYN segment raises the RTO into 100s of milliseconds.

We conclude that in the absence of congestion management the overlay network introduces up to 20% of additional delays. The RED and QCN control loops might need redesign to take into account the overlay network – subject of future research.

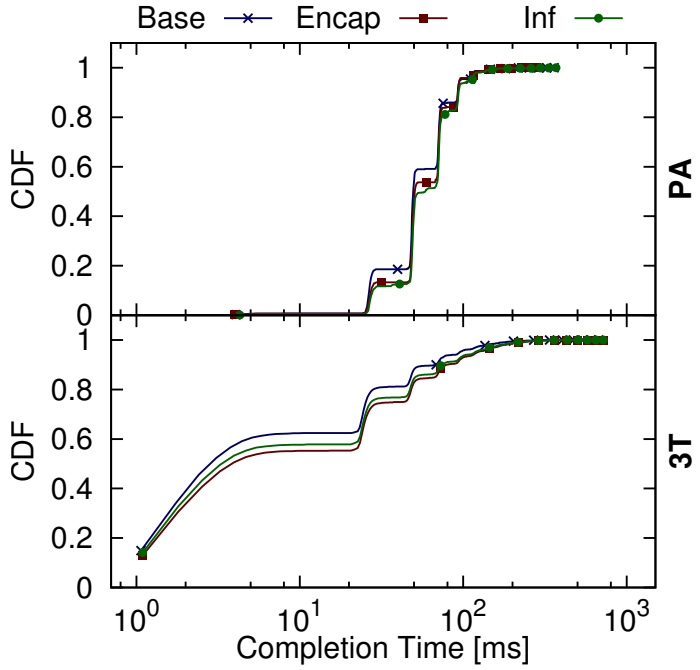
3.6.2. Virtual Switch Cache Design

The previous assumption of ideal infinite cache is unrealistic. Next we consider finite size caches of sizes from 16 to 256 entries. The lower bound is chosen because a VM running PA has at least 16 concurrent flows. The total amount of VMs in the network is 4096, i.e., 256 servers with 16 VMs each. However it is highly unlikely that a server communicates simultaneously with all the other VMs. We observe that across all runs the 256-entries cache yields results identical to the infinite cache. For larger topologies these values should be scaled accordingly.

Furthermore, due to the finite cache size, old entries have to be evicted. As a second parameter in the cache design space we consider the following 5 eviction policies: First In First Out (FIFO), evict the oldest cache entry; Random (RND), evict a random entry; Least Used (LU), evict the entry which the smallest number of hits;

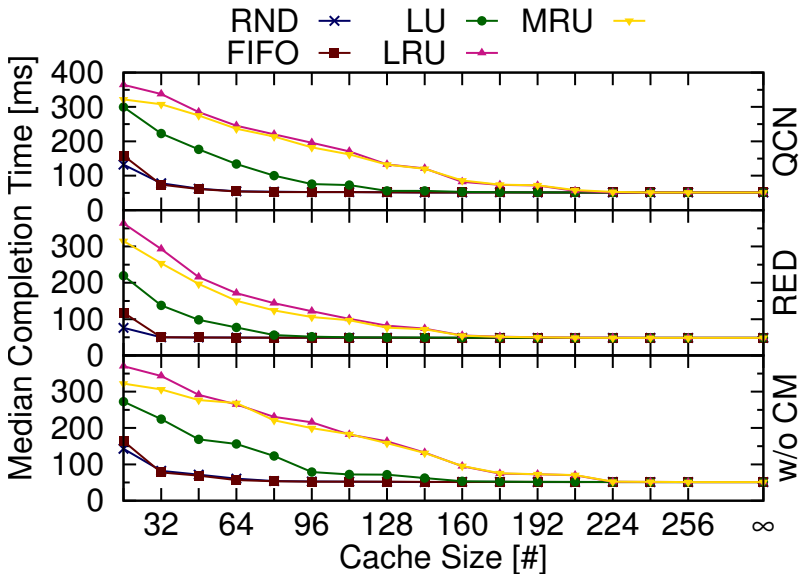


(a) TCP background.

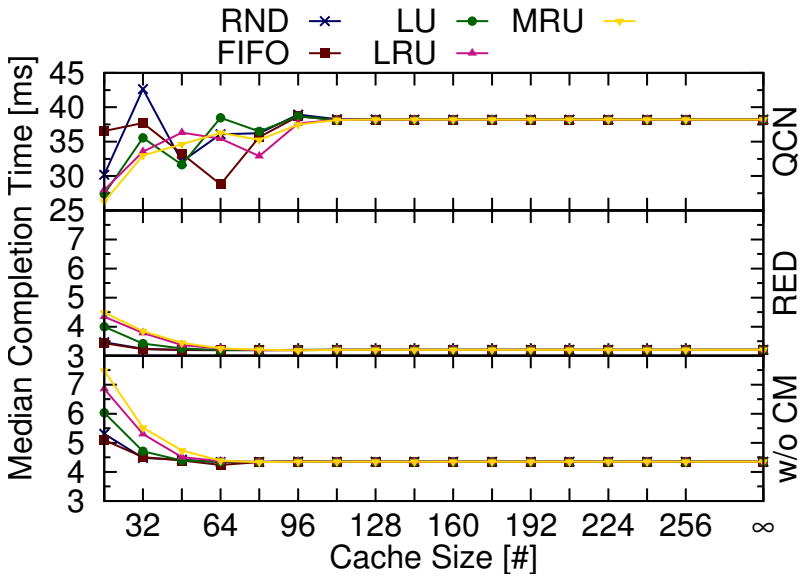


(b) UDP background.

Figure 3.11.: Cumulative distributions of completion times for TCP NewReno, without congestion management.



(a) PA, TCP NewReno, TCP background.



(b) 3T, TCP NewReno, TCP background.

Figure 3.12.: Cache eviction policy and cache size impact on median completion time.

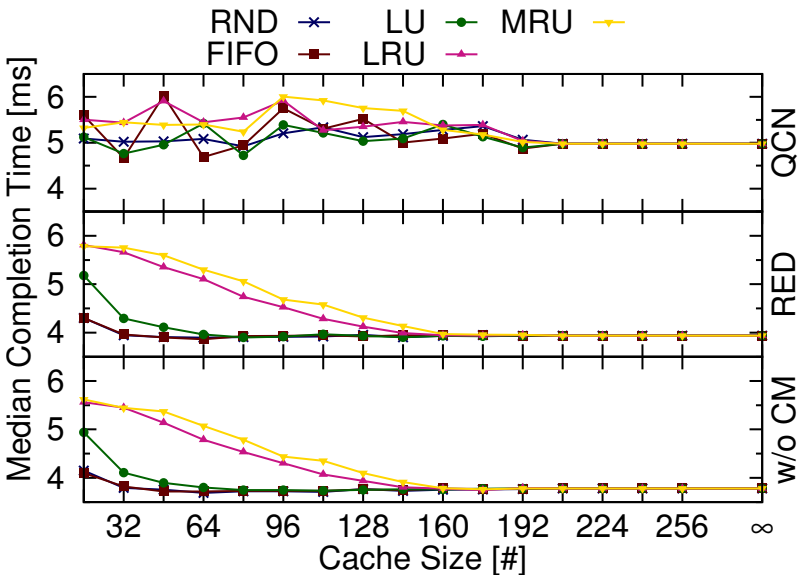


Figure 3.13.: 3T, TCP Vegas, UDP background. Cache eviction policy and cache size impact on median completion time.

Least Recently Used (LRU), evict the entry not used for the longest time; Most Recently Used (MRU), evict the latest used entry. Adding these two parameters to the setups described at the onset of Section 3.6 yields over 4000 simulation runs. Here we select only the most relevant results.

Figure 3.12a shows the PA performance with NewReno and TCP background flows. All other TCP versions and background flow types produce similar results. FIFO and Random policies *consistently* yield the best results, while LU and LRU are the worst because the entry selected for eviction is generally the one corresponding to the reply that has to be sent back to the MLA, HLA or client. The Random heuristic is better because it has higher chances of evicting a background UDP flow. FIFO is efficient because it evicts the dead flows, whereas MRU has the tendency to evict live flows. Systematically across all configurations, the completion time decreases with the increase of the cache size. Figure 3.14 shows the miss ratios for the same configuration. Qualitatively comparing the plot shapes we conclude that the virtual switch misses are responsible for the deterioration of completion times.

The results for 3T are shown in Figure 3.12b. Cache size is less influential for the 3T since a 64-entries cache performs as well as an infinite cache. Here a VM contacts only one or two other VMs at a time. On the other hand, a PA's MLA or HLA can contact up to 16 VMs in parallel. Therefore the completion time converges faster for 3T than for PA. With UDP background flows, the pressure on the cache increases as shown in Figure 3.13. When enabling QCN (Figure 3.12(b)(c)) we observe the counter-intuitive effect that a smaller cache performs better than a larger one. The completion time is influenced by two factors: the intensity of the background flows and the cache misses. When the cache is small it produces more misses, but it also

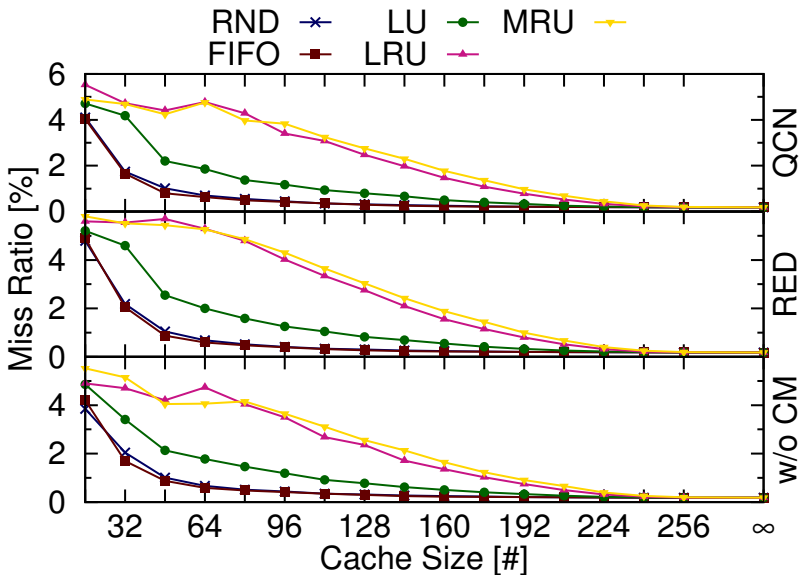


Figure 3.14.: PA, TCP NewReno, TCP background. Aggregated miss ratio of caches from all virtual switches. Observe the similarity with Figure 3.12a.

reduces the background load. When the cache is large there are only compulsory misses, but the background load increases. For example in Figure 3.12b with QCN and FIFO heuristic the best execution times are obtained for a 64-entries cache. More insights about the QCN performance are given in Section 3.6.5.2.

3.6.3. Controller Design

The centralized controller is replicated for redundancy into four replicas (C1 to C4), each attached to one of the core switches shown in Figure 3.1. We assume that all four controllers have a *coherent* view of the network maintained by the means of a lightweight protocol, trivial in the absence of migration. Each virtual switch holds the address of each controller. For load balancing, each request is sent to a randomly selected controller.

Figure 3.15 shows the average load per controller for PA over TCP Vegas without background flows. This setup generates the largest average load of 3.83MReq/s for RED with Random eviction and a 16-entries cache. This corresponds to an average load of 451MB/s apparently far below the ideal 625MB/s limit. This is indeed half of the line speed of 1250MB/s when each 64B request or reply receives 18B Ethernet headers while the switch internal fabric operates in 64B buffer units. Hence each controller request or reply requires two internal buffers. Despite the low average load, the controller links are saturated for tens of milliseconds, making the controller into a major bottleneck that elicits further attention.

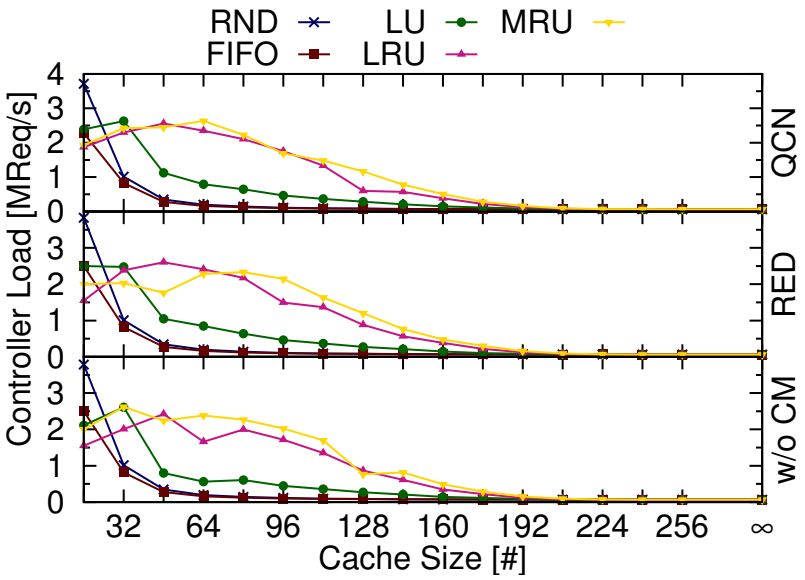


Figure 3.15.: PA, TCP Vegas, No background. Average controller load over the entire application run.

3.6.4. TCP Version Selection

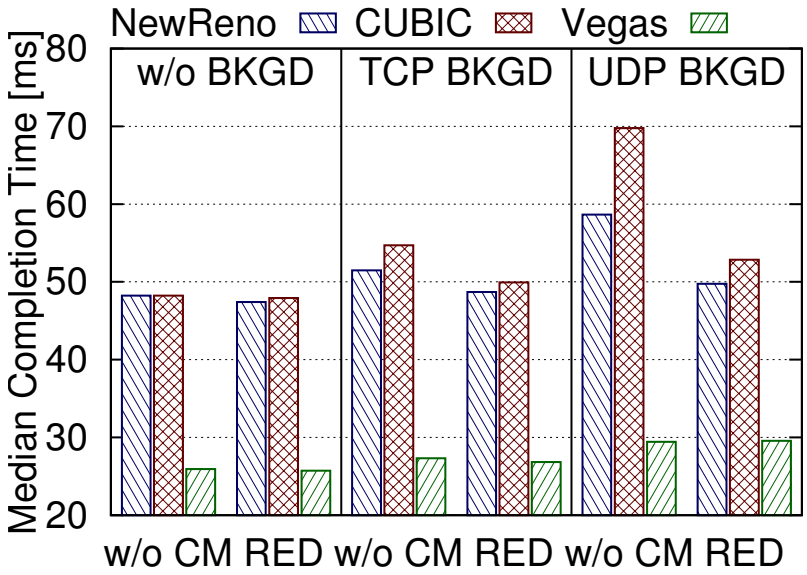
Here we compare the selected three TCP versions. In Figure 3.16 we plot the corresponding median completion times for an infinite cache.

3.6.4.1. Vegas

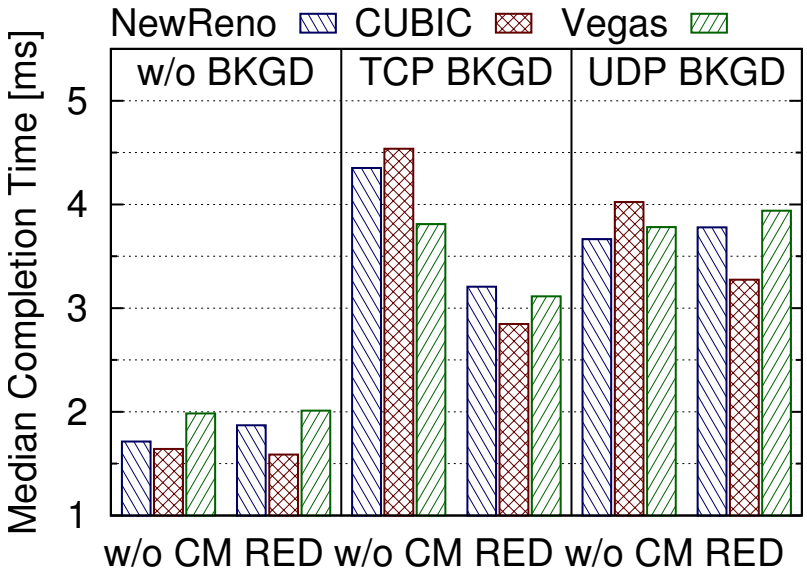
Vegas [30] adjusts the congestion window based on the measured delays. For the PA workload (see Figure 3.16a) Vegas produces the shortest completion times, up to 49.7% better than NewReno with UDP background flows. This is because Vegas has the lowest drop rate of all the three TCP versions. RED brings no additional benefits. Yet for the 3T workload in Figure 3.16b the differences between Vegas and NewReno are smaller, at most 12% improvement with TCP background flows. Also, with 3T, Vegas does not always outperform NewReno: e.g., without background flows. RED is beneficial only in the presence of backlog and drops, which Vegas avoids.

3.6.4.2. CUBIC

Across *all* the PA runs CUBIC [60] performs worse than NewReno (Figure 3.16a). Its aggressive increase of the congestion window aggravates the TCP incast congestion. With the 3T workload (Figure 3.16b) the rapid increase of the congestion window is beneficial only in the absence of background traffic. With background flows, CUBIC increases the loss ratios, hence longer execution times. Finally, here RED helps CUBIC to reduce its congestion windows.



(a) PA application.



(b) 3T application.

Figure 3.16.: The impact of TCP version on median completion times. The overlay network is active with infinite cache in each virtual switch.

3.6.5. Congestion Management Effectiveness

3.6.5.1. RED

Intuitively one of the main factors affecting the flow completion time is the number of packet drops. Drops occur during the TCP incast episodes in the PA aggregation phase. Because the bursty replies cannot fit into the switch buffers, some segments are tail-dropped. Sometimes the TCP fast-retransmit mechanism promptly recovers the drop, but often this is nullified by the small replies (20KB) fitting into 14 segments. The last segments, however, have higher drop probability due to their arrival when the buffer is already filled by the previous segments. Dropping these segments does not allow the fast-retransmit to receive the required number of duplicate ACKs (3) to activate and, therefore, the sender has to wait for the RTO timer.

Another extreme example is when the initial SYN is lost. The RTO estimator is then initialized with the (high) default value, thus heavily punishing the flow completion time. RED helps to avoid drops by keeping the queue occupancies low, hence in most of the runs it leads to shorter execution times. Counter-example: NewReno, 3T in the absence of background flows (Figure 3.16b). A possible problem that the overlay network can cause with RED is the *doubling* in size of the control segments, i.e., ACKs. These segments cannot be ECN marked but still they occupy buffer space contributing to congestion.

3.6.5.2. QCN

QCN avoids losses by reducing the injection rate of the flows that take more than their fair share of bandwidth, and implicitly hog too much buffer space. We observe that the 3T queries with active background flows are penalized by the QCN activation (compare QCN median completion times from Table 3.5 with those without CM). There are two main reasons.

First, the TCP traffic is inherently bursty, because the TCP sources segment the data to be transmitted and then inject as many consecutive packets as necessary to fill the congestion window. This leads to sudden increases of the buffer occupancies in switches. In contrast to RED, which operates on a low-pass smoothed queue length, QCN samples the instantaneous queue length. The TCP bursts produce spikes in the queue occupancy signaled via the QCN feedback. This triggers superfluous congestion notification messages that instantiate unnecessary rate limiters.

Second, the activation of QCN rate limiters leads to filling the transmission queues of the network adapters. This in turn leads to buffers filling and drops in the virtual switch upstream of the QCN rate limiters. We conclude that under these circumstances the standard-tuned QCN interacts poorly with our overlay. This issue is the subject of future research.

3.7. Saturation Results

3.7.1. Evaluation Metrics

Resource saturation is a well-known problem. Rather than addressing it by itself, here we aim at studying how network resources saturate by using the novel notion of elasticity derived from the mathematical concept of point elasticity. Informally, elasticity denotes the sensitivity of a dependent variable to changes in one or more other variables (parameters). An elastic supply should be sensitive to small changes in load. For example, if the traffic load increases, additional bandwidth should be provisioned. Supply elasticity with respect to load seems intuitively desirable in a datacenter. On the other hand, an inelastic variable should remain insensitive to changes in other variables (parameters). For example, the performance experienced by a tenant should remain oblivious to the other tenants sharing the same physical infrastructure.

Depending on the operator I-/P-/S-aaS model, the datacenter resources can be allocated to tenants, either physically, by adding or moving new machines and switches, or virtually, by provisioning and migrating VMs and virtual networks (VNs). The latter replicates the virtual resources – server, switch, adapter, link – to create the illusion of new additional servers and networks. As long as the existent physical resources are under-utilized, their virtualization increases the supply elasticity. The supply of VMs and/or VNs increases with the tenant load, ideally along a linear curve. However, the linear dynamic range is practically limited by several factors: workload type, traffic patterns, as well as the capacities of the physical resources, and increasingly, their virtualization technologies. For example, a current quad-core CPU may support 8-16, but not yet 256 VMs.

The most vexing issue is: Which metrics are are suitable to best quantify the performance – and ultimately, the elasticity of the resources? We propose the following set of five simple, but descriptive, metrics:

1. Aggregate throughput, $Tput$, expressed in HTTP queries per second – as operator metric.
2. Query completion times, T_c, representative as primary tenant performance metric.
3. Packet loss ratio, as a metric for the network service quality.
4. Power [66], $P = \frac{Tput}{T_c}$, a metric revealing the throughput-delay tradeoffs between operator and tenants.
5. Global efficiency, $W = \sum_t U_t(T_c)$, sum of all tenant utilities U_t as a function of T_c such that $U_t \in [0, 1]$.

We also argue that the first two metrics are intrinsic to the SLAs between operator and tenants.

3.7.2. Traffic Scenario

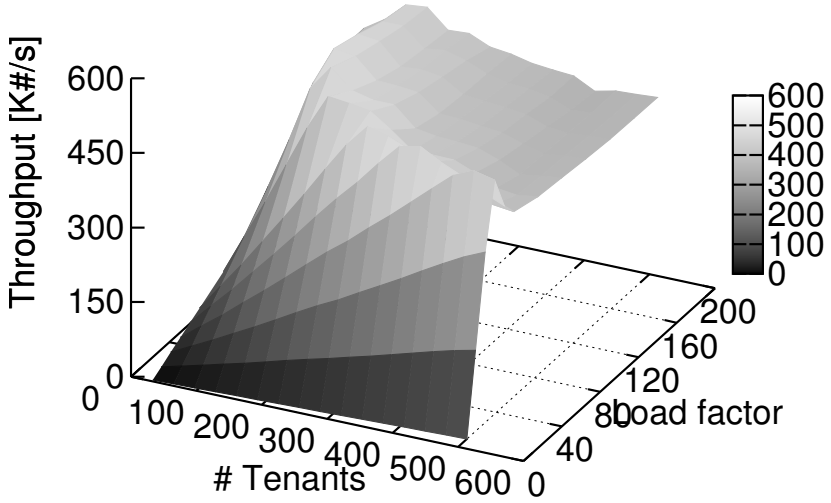
In the following, we focus on a simple and concrete, albeit limited, example of the above investigating a homogenous set of tenants running 3-tier workloads in a generic virtualized datacenter. Although extremely interesting, for space reasons, alternate scenarios such as different workloads, e.g. map-reduce, and/or heterogeneous sets of tenants have to be left for future work.

We perform our investigation as follows. (i) We vary the consolidation factor, i.e., the number of tenants, from 32 to 512, each deploying the 3-tier workload described previously. As all tenants share the same physical network, its load increases with the number of tenants. (ii) Furthermore, we vary the load factor, starting from the reference load (1x), then progressively increasing it up to 200x – corresponding to more external clients per tenant. In all simulations, a tenant serves precisely 1000 HTTP queries. When all tenants have finished, the simulation is stopped, and statistics are gathered. For each query, we measure the completion time, accounting for all physical and virtual network-induced delays, including the end-node protocol stacks. We assume that all servers have infinite CPU resources, and that a VM serves each request in zero delay. Even if a bit unrealistic, this assumption helps isolate the communication bottlenecks from the end-node processing variability as well as the VM scheduling side-effects, which are not of interest here.

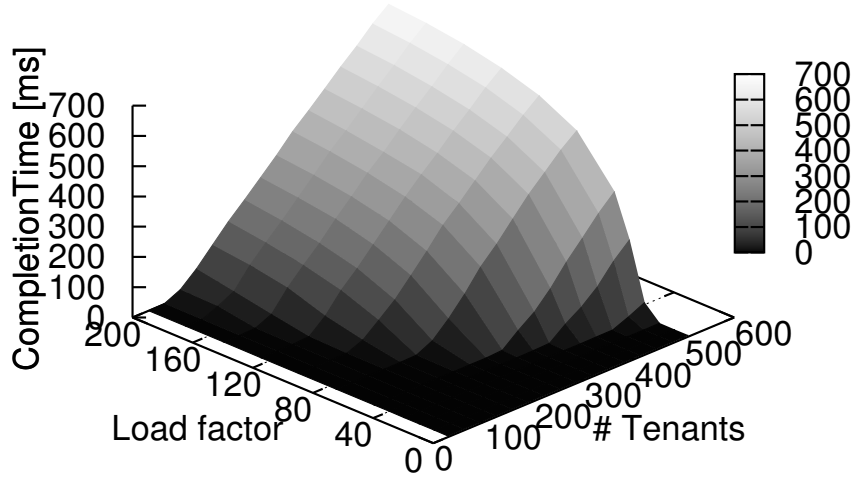
3.7.3. Aggregate Throughput and Query Completion Time

The datacenter *Tput* is calculated by dividing the total number of HTTP requests by the time needed to serve 99% of them to completion. We use the 99th percentile for being robuster to outliers. The results are plotted in Figure 3.17a. Increasing the number of tenants is beneficial to the aggregate throughput until a peak is reached at around 500K requests/s. The peak is reached earlier when inter-arrivals are shortened, corresponding to higher load factors per tenant. After the peak, the datacenter network saturates. The number of tenants grows linearly, but the completion times grow with higher slopes (Figure 3.17b), and therefore the aggregate throughput decreases (saturation). The drop in throughput is caused by the increase in packet loss, which further overloads the fabric because of retransmits. All flows contain at most 15 segments. This is far too short for the TCP control loop to react properly.

For low to medium load factors ($\lessapprox$ 80x), the measured throughput monotonically increases with a higher slope than the query completion time (delay). As seen in Figure 3.17a, the saturation peak is variable with both the consolidation and the load factor. In the linear region below saturation, increasing the consolidation factor, e.g., by adding new tenants, does not influence the median completion times beyond the set threshold. Thus, in the linear region, each new tenant positively contributes to the OVN aggregate throughput, whereas beyond the saturation point each new



(a) Aggregate $Tput$.



(b) Median completion time.

Figure 3.17.: Established performance metrics.

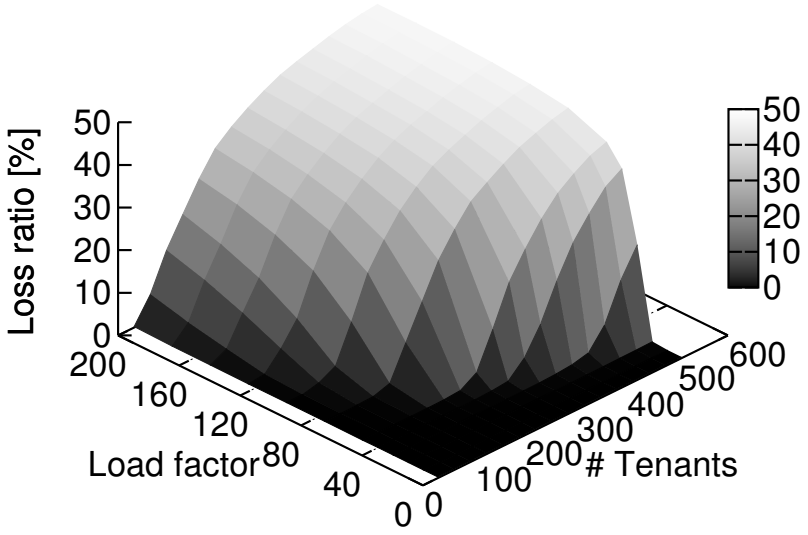


Figure 3.18.: Packet loss ratios.

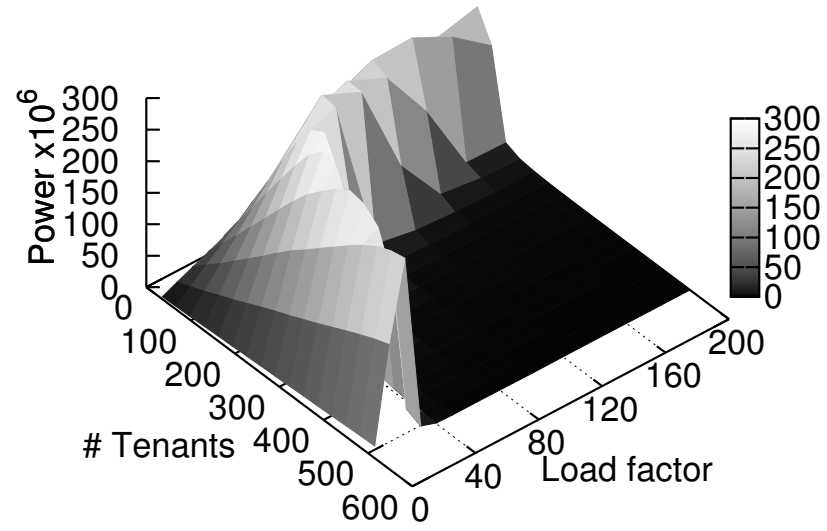
tenant will diminish it and increase the latency, with a relatively smooth roll-off. The datacenter fabric is elastic if and only if its performance is inelastic. This is true in the linear operation region of the datacenter network.

3.7.4. Packet Loss Ratios

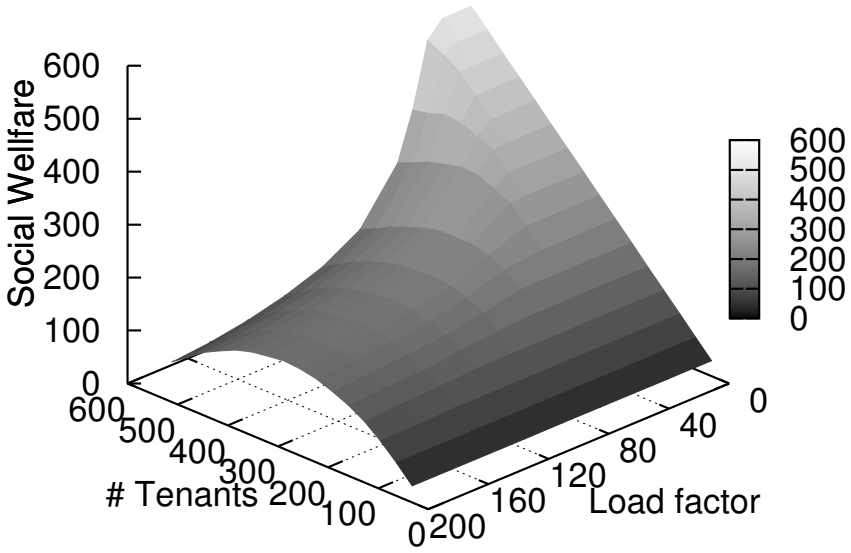
Packet loss ratios, plotted in Figure 3.18, are required for completeness because the modeled commodity 10GigE infrastructure does not employ link level flow control (i.e. Priority Flow Control), and are obtained by dividing the total number of bytes sent by all VMs to the total number of bytes dropped by all the switches. After the throughput peak is reached, the network saturates, as well as the queues' occupancies at the physical switches. The percentage of losses grows accordingly, leading to longer execution times.

3.7.5. Network Power

The network power is the ratio of throughput to completion time. It reveals the throughput-delay tradeoffs between the datacenter operator and its tenants. While the operator maximizes the datacenter throughput by provisioning additional VMs, the query completion time, i.e., the main tenant performance metric, is adversely affected. Neither of the two individual metrics in Figure 3.17 conveys the undesirable performance elasticity. This, however, is clearly revealed by the power metric plotted in Figure 3.19a. We observe a more pronounced peak to surface around



(a) Power.



(b) Global efficiency.

Figure 3.19.: Adapted performance metrics.

the saturation point, followed by a sudden drop to nearly zero. In contrast, this behavior is not distinguishable in the aggregate throughput, hence the expressivity of power.

3.7.6. Global Efficiency

We define the datacenter efficiency as a tenant-oriented metric that sums the utility of all active tenants. Specifically, as utility per tenant U_t we build a synthetic linear function $U_t(T_c) = max(0,\ 1 - \alpha\ T_c)$ of the median query completion time T_c, which we derived based on data from various search engine usage models, e.g. [109]. The tenant utility is maximized (1.0) when the completion time (ideally) converges to zero; then it decreases linearly and remains zero (coarse approximation of a non-linear decay) for completion times larger than $\frac{1}{\alpha} = 800ms$. The global efficiency under these conditions is shown in Figure 3.19b. At low load factors ($\lessapprox$ 40x) the efficiency increases monotonically. This validates the intuition that additional new tenants do not influence the completion times past the accepted threshold. Before saturation, a new tenant contributes to the datacenter efficiency, but beyond saturation diminishes it. At high loads, new tenants saturate the network earlier, hence the global efficiency peaks around 320 tenants. In addition to the elasticity information derived from power, now we can clearly distinguish – from the saddle-shaped maximum surface – that the optimal consolidation factor across multiple load factors is ≈ 320. However, this value is highly dependent of the choice of tenant utility function, which remains an open issue of operational research for cloud and datacenters.

3.8. Related Work

Several performance evaluations were recently published, addressing different aspects of datacenter networking. Overlay networks were proposed in [55, 110, 81, 88, 37, 24]. In [118], the impact of server virtualization on the network performance in datacenter is studied. In [72], the datacenter traffic was measured and characterized in a large operational cluster. Unlike the above studies, we perform here a cross-layer study measuring the influence of overlays on applications performance. Our commercial workload traffic generator is based on [72, 16, 26, 27]. QCN is defined in [11] and further analyzed in [83] and [70]. Its main challenges, i.e., unfairness and lack of RTT adaptivity [16] have been addressed by E^{2}CM [59], AF-QCN [70], and FQCN [123]. A first comparative evaluation of Short-Fat TCP for CEE datacenter networks is done in [38]. Performance reviews of modern TCP variants for long-fat networks are available in [77, 32]. Regarding PA traffic patterns, the TCP Incast problem has been analyzed in [33, 117], where a $10 - 1000\times$ retransmission timeout reduction and high resolution timers have been proposed. Another TCP Incast solution is DCTCP [16], using a modified RED/ECN and a multibit feedback estimator

that filters the incoming single-bit ECN stream. Also related is [44] which analyzes the TCP Incast problem in a QCN-enabled lossy network. More practically, the TCP Incast is studied in [111] using Stream Control Transmission Protocol (SCTP) and ECN.

3.9. Discussion

We summarize the results of our evaluation by answering the questions set forth in Section 3.1.4.

(**Q1**) What is the influence of TCP parameters on the application performance in an OVN environment?

The first observation is that the *Default* TCP parameters were unsuitable for the simulated datacenter networks producing up to 87% longer flow completion times when compared with the *Tuned* TCP parameters. Furthermore, the activation of the overlay virtual network did not change the figure *Tuned* behaving consistently better than *Default*.

(**Q2**) Does a DOVE-like OVN impact the performance of our two workloads? If yes, how much? What are the expected performance bounds over overlays?

Overlays diminish the performance of the two selected workloads due to encapsulation and discovery overhead. In Section 3.6 we showed that the increase in the completion time of the HTTP queries ranges from 1.5% up to 18.2%. However, we argue that even the worst case provides an acceptable trade-off in return for the overlay's benefits in terms of manageability and security. The primary performance gating factor is the cache size and eviction policy, as shown in Section 3.6.2. The optimal cache size was shown to be dependent on the amount of concurrent flows initiated by each VM. Therefore, the 3T requires smaller caches than the PA application. The Random and FIFO eviction policies were proven to be the best strategies, even for modest caches. The secondary performance gating factor is the overlay tunnel efficiency. As shown in Table 3.6 and 3.5 the impact ranges between 0.6% and 32.5%.

At a given moment in time, each source VM has a set of destination VMs to which it communicates. Our results show that for scenarios where this set is small and constant over time, the overlay performance impact is lower than for scenarios where this set is large and variable. Additionally for short flows, specific to 3T and PA, the overlay impact is larger than for long transfers in which the discovery overhead is negligible relative to the entire flow duration.

Finally, RED improves the query completion time up to 27% – thus a low cost addition with a positive impact. Because of the higher network load and different traffic pattern, the RED-induced improvements are not as large as previously shown in [28]. RED has no perceptible influence on Vegas, whereas, with NewReno

and CUBIC, RED reduces the loss rate, thus lowering the completion times. In its default configuration tested here, QCN currently interacts negatively with the overlay. By activating its rate limiters, QCN shifts the congestion point and increases the loss ratio in the upstream virtual switch. This provides an interesting research opportunity.

(**Q3**) What are the best metrics to quantify the saturation of network resources in overlay virtual networks?

We selected and adapted five descriptive metrics to our method: aggregate throughput, median query completion time, packet loss ratio, datacenter network power, and its global efficiency. With these, we investigated the performance of homogenous sets of tenants running 3-tier workloads in a realistic virtualized datacenter network. As shown in Section 3.7, the datacenter has a linear region of operation – elastic scalability with inelastic performance. Herein the load growth, induced by the consolidation factor, the load factor, or both, contributes positively to the aggregate throughput, without negatively influencing the completion times. On the other hand, outside the linear region, the datacenter performance becomes elastic with respect to load, i.e., inelastic scalability. Thus we empirically found the boundaries of the linear operation region.

In addressing these issues we hope to provide insights and guidance for the datacenter and overlay architects. To the best of our knowledge we have contributed the following: (1) We provided the first completion time cross-layer evaluation of partition/aggregate *and* 3-tier applications in a realistically virtualized datacenter network, using an *actual* TCP stack running over a *detailed* L2 CEE fabric model. (2) We measured the influence of the OVN design parameters on the system level performance.

4. Zero-loss Overlay Virtual Network

In Chapter 2 we showed that the new features of Converged Enhanced Ethernet (CEE) can reduce the completion time of various commercial and scientific applications. Next, in Chapter 3, we described overlay virtual networks, an emerging solution aiming to overcome the obstacles to network virtualization. As shown in Chapter 3 the overlay networks introduce additional protocols between the traditional TCP stack and the physical layer. In this chapter we show that the performance of applications in virtualized networks is harmed by the non-deterministic packet drops in these new layers. We introduce the novel zero-loss Overlay Virtual Network (zOVN) that extends the CEE features described in Chapter 2 into the overlay virtual networks studied in Chapter 3.

Datacenter networking is currently dominated by two major trends. One aims toward lossless, flat layer-2 fabrics based on Converged Enhanced Ethernet or InfiniBand, with benefits in efficiency and performance. The other targets flexibility based on Software Defined Networking, which enables Overlay Virtual Networking. Although clearly complementary, these trends also exhibit some conflicts: In contrast to physical fabrics, which avoid packet drops by means of flow control, practically all current virtual networks are lossy. In this chapter we quantify these losses for several common combinations of hypervisors and virtual switches, and show their detrimental effect on application performance.

Next, we propose a zero-loss Overlay Virtual Network (zOVN) designed to reduce the query and flow completion time of latency-sensitive datacenter applications. We describe its architecture and detail the design of its key component, the zVALE lossless virtual switch. As proof of concept, we implemented a zOVN prototype and benchmark it with Partition-Aggregate in two testbeds, achieving an up to 15-fold reduction of the mean completion time with three widespread TCP versions. For larger-scale validation and deeper introspection into zOVN, we developed an OMNeT++ model for accurate cross-layer simulations of a virtualized datacenter, which confirm the validity of our results.

4.1. Introduction

In recent years, profound changes have occurred in datacenter networking that are likely to impact the performance of latency-sensitive workloads, collectively referred to as on-line and data-intensive [115]. Particularly relevant are the rise of Overlay

Virtual Networking (OVN) – remarkable application of Software-Defined Networking (SDN) – and, simultaneously, the shift to lossless layer-2 fabrics based on Converged Enhanced Ethernet (CEE) or InfiniBand. So far, the trends in virtualization and the commoditization of high-performance-computing-like lossless[1] fabrics have been decoupled, each making independent inroads into the datacenter.

While the research community increasingly focuses on the performance of horizontally-distributed data-intensive applications [33, 16, 17, 71, 115, 119, 122], and recently also on virtualization overlays for multitenant datacenters [88, 118, 37], we argue that the combination of virtualization and such workloads merits closer scrutiny [28]. Our main objective is to analyze the impact of the absence versus presence of flow control on workload performance in a virtualized network. As our study specifically focuses on latency-sensitive, data-intensive workloads, the performance metric of interest is flow completion time (FCT) [46]. As a representative workload model, we selected Partition-Aggregate [16, 119].

4.1.1. Network Virtualization

As server virtualization allows dynamic and automatic creation, deletion, and migration of virtual machines (VMs), the datacenter network must support these functions without imposing restrictions, such as IP subnet and state requirements. In addition to VM mobility and ease of management, complete traffic isolation is desirable for improved security, which can be achieved by layer-2 and -3 virtualization. Rather than treating the virtual network as a dumb extension of the physical network, these requirements can be effectively met by creating SDN-based overlays such as VXLAN [81] and DOVE [37]. An exemplary architectural exposition of modern virtual overlays is NetLord [88], which covers the key motivations and design principles.

SDN as a concept decouples the control and data planes, introducing programmability and presenting applications with an abstraction of the underlying physical network. Scalable and flexible "soft" networks can thus be designed to adapt to changing workloads and to datacenter tenants and operators needs. In a nutshell, SDN trades some degree of performance to simplify network control and management, to automate virtualization services, and to provide a platform upon which new network functionalities can be built. In doing so, it leverages both the OpenFlow [82, 95] and the IETF network virtualization overlay [110, 81] standards.

Based on the adoption rate of virtualization in datacenters, the underlying assumption is that virtual networks (VN) will be deployed in practically most, if not all, multitenant datacenters, providing a fully virtualized Cloud platform by default.

[1] In this chapter we use *lossless* and *zero-loss* in the sense of avoiding packet drops due to congestion. Packets might still be discarded because of CRC errors in the physical links. These, however, are extremely rare events under normal conditions (typical bit error rates are 10^{-12} or less) and recovered by TCP. In Chapter 5 we will show how the overlay virtual network can be further extended to also recover these errors.

For the remainder of this chapter, we presume that VN overlay is an intrinsic part of the extended datacenter network infrastructure. Thus we envision a fully virtualized datacenter in which "bare-metal" workloads become the exception, even for mission-critical applications.

However, current hypervisors, virtual switches (vSwitches) and virtual network interface cards (vNICs) critically differ from their modern physical counterparts. In fact, they have a propensity to liberally drop packets even under minor congestive transients. These losses can be considerable and non-deterministic, as will be presented in Section 4.2.3. Consequently, current non-flow-controlled virtual networks will significantly cancel out the investments of upgrading datacenter networks with flow-controlled CEE and InfiniBand fabrics. We argue that this lossy legacy unnecessarily hinders both the application performance and the progress of future datacenters.

4.1.2. Lossless Fabrics

The recent standardization of 802 Data Center Bridging for 10-100 Gbps CEE triggered the commoditization of high-performance lossless fabrics. First generation 10G products are already on the market, and CEE fabrics at 40G, or even 100G, have been announced by several vendors.

Traditionally, Ethernet did not guarantee losslessness: packets were dropped whenever a buffer reached its maximum capacity. This behavior does not match the modern semantics of datacenter applications, including High-Performance Computing (HPC) environments [41], storage (Fibre Channel over Ethernet [10]), or Remote Direct Memory Access (RDMA) over Ethernet [36].

CEE upgrades Ethernet with two new mechanisms of interest here: A link-level flow control, i.e., Priority Flow Control (PFC) [13], and an end-to-end congestion management known as Quantized Congestion Notification (QCN). PFC divides the controlled traffic into eight priority classes based on the 802.1p Class of Service field. Within each priority PFC acts as the prior 802.3x PAUSE, except that a paused priority will not affect the others. Hence, a 10-100G link is not fully stopped whenever a particularly aggressive flow exceeds its allotted buffer share. Despite the marked improvement over the original PAUSE, a side-effect of PFC still remains the potential global throughput collapse, which differs from the lossy case. The buffer of a flow-controlled blocked receiver may recursively block buffers upstream, spreading the initial congestion into a saturation tree [97]. To address these head-of-line blocking issues, QCN was defined and extensively simulated prior to releasing PFC. For a comprehensive description of the aforementioned CEE protocols the reader is directed to Chapter 2.

4.1.3. Contributions and Structure

The contributions of this chapter are as follows:

1. We identify and characterize the problem of packet drops in virtual networks. We show that virtual networks are affected by considerable and non-deterministic losses that harm performance of latency sensitive applications.
2. We implement the first zero-loss Overlay Virtual Network (zOVN) to address the problem of packet drops in converged multitenant datacenters.
3. We quantitatively verify how zOVN improves the *standard* TCP performance for data-intensive applications. Testing Partition-Aggregate on top of zOVN, we achieved up to 15-fold reductions in flow completion times using two distinct testbeds with 1G and 10G Ethernet respectively, and three standard TCPs.
4. Finally, we investigate the scalability of zOVN by means of accurate full system cross-layer simulations.

The remainder of this chapter is structured as follows: In Section 4.2 we present the main issues of current virtual networks. In Section 4.3 we explore the design space of virtual overlays. We provide the details of our zOVN prototype in Section 4.4 and evaluate its performance in Section 4.5. We analyze the results in Section 4.6 and we summarize the related work in Section 4.7. Finally we conclude the chapter in Section 4.8.

4.2. Virtual Networks Challenges

The two deficiencies of current virtual networks are latency penalties and excessive packet dropping.

4.2.1. Latency

A virtual link does not present a well-defined channel capacity. Neither arrivals nor departures can be strictly bounded. The virtual link service time remains a stochastic process depending on the processor design, kernel interrupts, and process scheduling. This negatively affects jitter, burstiness, and quality-of-service. Hence, virtual networks without dedicated real-time CPU support remain a hard networking problem. In addition, virtual networks introduce new protocols spanning layer-2 to 4 and touch every flow or, in extreme cases, even every packet [88, 37]. The result is a heavier stack, with encapsulation-induced delays and overheads possibly leading to fragmentation and inefficient offload processing.

However, the more critical performance aspect is the impact on latency-sensitive datacenter applications. Latency and flow-completion time have been recently established as crucial for horizontally-distributed workloads such as Partition - Aggregate, typically classified as soft real-time. The 200ms end-user deadline [16, 119, 62]

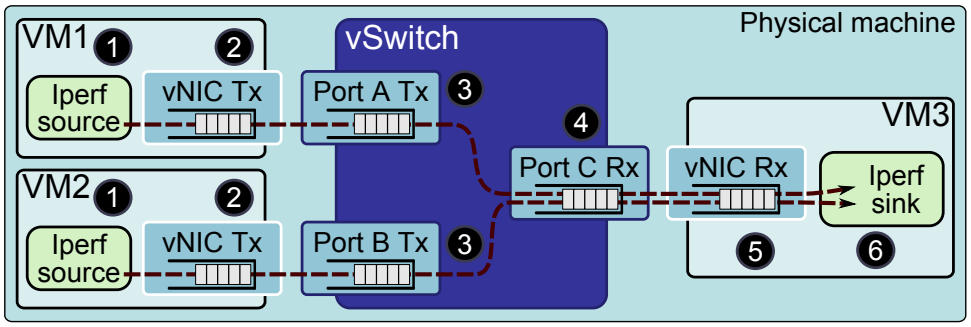


Figure 4.1.: Experimental setup for virtual network loss measurements.

translates into constraints of few 10s of milliseconds for the lower-level workers. Although the additional VN-induced delay may be negligible in a basic ping test [88], its impact on more realistic Partition-Aggregate workloads can lead to an increase in the mean flow completion time of up to 82% [28]. This raises concerns about potentially unacceptable VN performance degradations for such critical latency-sensitive applications in a virtualized multitenant environment.

4.2.2. Losslessness

Ideally a VN should preserve the lossless abstraction assumed by *converged* datacenter applications such as Fibre Channel over Ethernet [10], RDMA over Ethernet [36] or HPC environments [41]. Yet currently *all* the commercial and open-source VNs that we have tested are *lossy*. As losslessness is a critical qualitative feature for the future of converged datacenter networking, CEE spared no effort to ensure zero-loss operation by using two complementary flow and congestion control protocols, namely, PFC and QCN. The same holds for InfiniBand, with its link level credit-based flow control and its FECN/BECN-based end-to-end Congestion Control Annex. In comparison, despite the possibility of relatively simpler and lower-cost flow control implementations, current VNs still resort to packet drop during congestion. This not only degrades datacenter performance, but also fails to correctly terminate modern flow-controlled fabrics, canceling out the investments in a lossless physical network. As an alternative, we demonstrate how a zero-loss Overlay Virtual Network (zOVN) can meet both the desired losslessness *and* the performance requirements.

4.2.3. Loss measurements

To support the above claims, we assess the extent of packet drops using commonly available virtualization solutions. We perform the experiment shown in Figure 4.1 in which VM1 and VM2 act as sources and send their traffic towards VM3, which acts as sink, creating a common congestion scenario. We evaluate (i) where and how

	Hypervisor	vNIC	vSwitch
C1	Qemu/KVM	virtio	Linux Bridge
C2	Qemu/KVM	virtio	Open vSwitch
C3	Qemu/KVM	virtio	VALE
C4	H2	N2	S4
C5	H2	e1000	S4
C6	Qemu/KVM	e1000	Linux Bridge
C7	Qemu/KVM	e1000	Open vSwitch

Table 4.1.: Configurations for loss measurements.

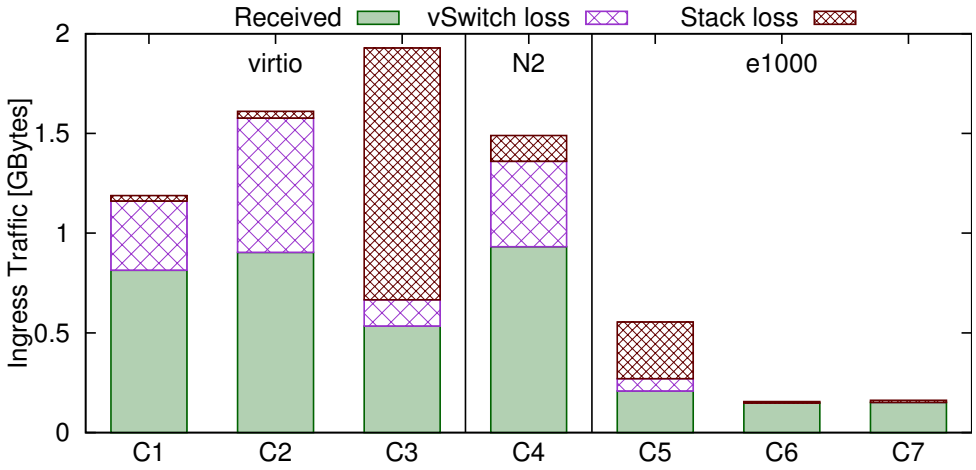


Figure 4.2.: Causes of packet losses. Configurations C1-C7 defined in Table 4.1.

frequently losses occur, and (ii) the maximum bandwidth that a virtual switch can sustain without dropping packets.

We considered the combinations of hypervisors, vNICs, and virtual switches shown in Table 4.1. Qemu/KVM is an open-source hypervisor, whereas H2 is a commercial x86 hypervisor. They were used with two types of vNICs: virtio [105] and N2 are virtualization optimized vNICs designed for Qemu and H2, respectively, whereas e1000 fully emulates the common Intel[2] e1000 adapter. In combination with Qemu, we used three virtual switches: Linux Bridge [4], Open vSwitch [7] and VALE [102]. The first two are stable products used in various production deployments whereas VALE is currently a prototype. The combination Qemu-e1000-VALE was omitted as it was affected by an implementation bug that allows internal queues to grow indefinitely, resulting in substantially diverging results between runs. With H2 we used its own internal virtual switch S4. All configurations have been tested on a Lenovo T60p Laptop (part of Testbed 1 detailed in Figure 4.7). Across all experiments, iperf [3] injects 1514B frames of UDP traffic. We determine the losses and bandwidths using the six measurement points shown in Figure 4.1: (1) and (6) are inside the application itself, (2) and (5) are on the TX and RX side of each vNIC, whereas (3) and (4) are at the virtual switch ports.

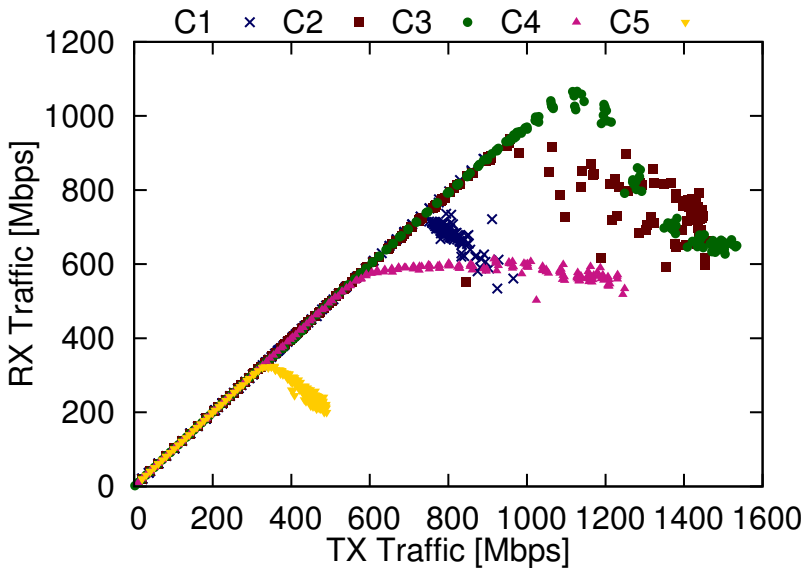
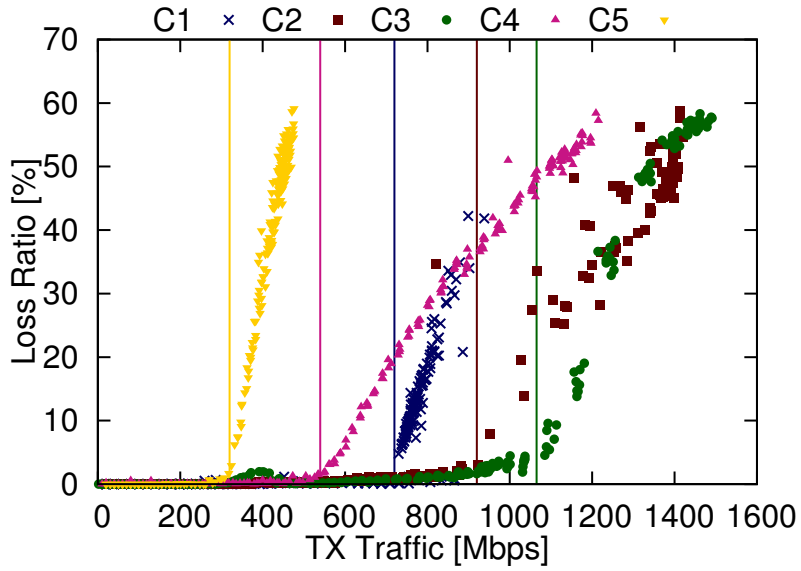


Figure 4.3.: vSwitch forwarding performance.

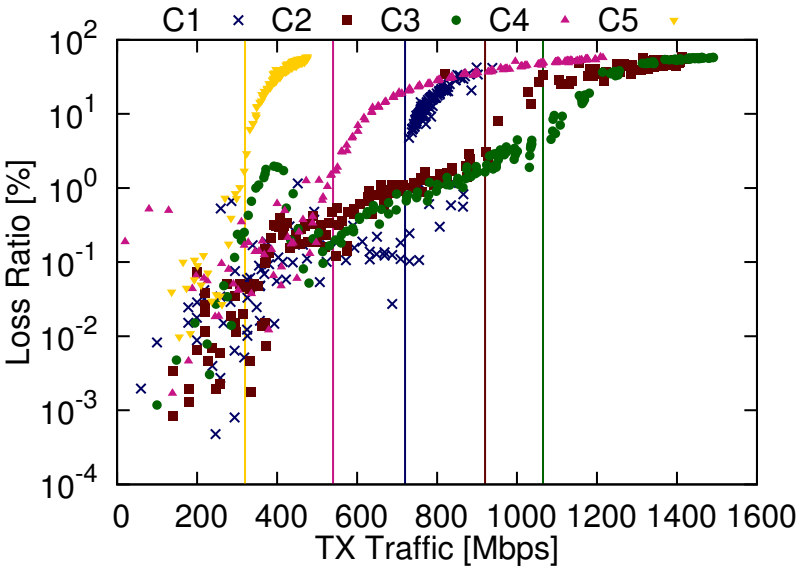
Experiment 1: Both generators injected traffic at full speed for 10s, with the last packet being marked. We computed the number of lost packets as the difference between the number of packets transmitted and received at the other end. We investigate (i) *vSwitch losses*, i.e., packets received by the vSwitch input ports (3) and never forwarded to the vSwitch output port (4), and (ii) *receive stack losses*, i.e., packets received by the destination vNIC (5) and never forwarded to the sink (6). The TX path is backpressured up to the vSwitch, hence no losses were observed between other measurement points. A more accurate analysis of the possible loss points is presented in Section 4.4. With VALE and S4, we could not access the points (3) and (4). Hereby the difference between the sender vNIC and the receiver vNIC counters (points (2) and (5), respectively) was accounted as virtual switch losses. The results are plotted in Figure 4.2.

Depending on configuration, the total traffic forwarded during the 10s window varied widely. In virtualized networks performance is bounded by the computational resources assigned to each block by the host operating system. Compute-intensive configurations score lower throughputs, inducing less losses in the vSwitch. An example is given by the e1000-based configurations that emulate a fake hardware to "deceive" the guest driver. The virtualization-optimized vNICs – i.e., virtio and N2 – achieved higher rates, thus causing overflows in the virtual switch. The performance optimized VALE switch shifted the bottleneck further along the path, into the destination VM stack. All these results are evidence of the lack of flow control between the virtual network devices, and confirm our initial conjecture.

Experiment 2: To analyze the maximum sustainable bandwidth for the virtual switches, we varied the target injection rate at each generator in increments of 5 Mb/s, starting from 5 Mb/s. The aggregated virtual switch input traffic is the



(a) Packet loss ratio.



(b) Packet loss ratio (log scale).

Figure 4.4.: Experimental loss results. The losses are measured between points 1 and 6 from Figure 4.1.

sum, i.e., twice the injection rate. Figure 4.3 and Figure 4.4a plot, respectively, the RX rate and loss ratio as a function of the total injection rate. Both were calculated at application level (points (1) and (6)). All configurations exhibit saturation behaviors. The RX rate first increases linearly with the TX rate, up to a saturation peak. Beyond this, with the exception of C4, we observe a drop indicating a lossy congestive collapse, rather than the desired steady saturation plateau. The overloaded system wastes resources to generate more packets, instead of dedicating sufficient resources to the virtual switch and destination VM to actually forward and consume the packets. Although the saturation point varied considerably across configurations, loss rates well in excess of 50% were observed for all configurations (Figure 4.4a). Even far below the saturation load, marked by vertical lines, we measured losses in the virtual network (Figure 4.4b) that were significantly above the loss rates expected in its physical counterpart, i.e., up to 10^{-2} instead of 10^{-8} for MTU-sized frames with a typical bit-error rate of 10^{-12}.

The "noise" in Figure 4.4b confirms our intuitive hypothesis about large non-causal performance variability in virtual networks. In fact, the service rate of each virtual link depends critically on the CPU, load, process scheduling, and the computational intensity of the virtual network code. Suboptimal and load oblivious scheduling causes frequent losses, e.g., by scheduling a sender prior to a backlogged receiver. Lossless virtual switches would be of great interest, not only in terms of efficiency but also for performance predictability. The next sections will present how flow control can be implemented in virtualized datacenter networks.

4.3. zOVN Design

In this section we outline the core principles that guided the design of our lossless virtual network.

4.3.1. Objectives

A converged virtualized network infrastructure must simultaneously satisfy the requirements from the domains being converged. As mentioned above, losslessness is a *functional* requirement of various HPC, storage and IO applications, whereas online data-intensive workloads impose *performance* requirements of 200 ms user-level response times.

We base our lossless virtual datacenter stack on CEE-compatible flow control. Transport-wise, we anchor zOVN's design on the established TCP stack combined with lossless overlays as proposed here. Our objectives are :

1) Reconcile the flow completion time application *performance* with datacenter *efficiency* and ease of management. This proves that network virtualization and

horizontally-distributed latency-sensitive applications are not mutually exclusive. This may remove an obstacle for virtual network deployment in performance-oriented datacenters.

2) Prove that *commodity* solutions can be adapted for sizable performance gains. As shown in Section 4.5, a 15-fold flow completion time reduction is also attainable without a clean-slate deconstruction of the existing fabrics and stacks. One can achieve *comparable* performance *gains* with CEE fabrics and standard TCP stacks. Considering the total costs of ownership, this evolutionary reconstruction approach is likely to outperform other, possibly technically superior, alternatives in terms of cost/performance ratios.

3) Expose packet loss as a costly and avertable singularity for modern datacenters, and, conversely, *losslessness* as a key enabler in multitenant datacenters for both (i) the query and flow completion time performance of horizontally-distributed latency-sensitive workloads, and (ii) the convergence of loss-sensitive storage and HPC applications. This basic HPC principle has already been proved by decades of experiences in large-scale deployments. As faster InfiniBand and CEE fabrics are widely available at decreasing prices, datacenters could also now benefit from prior HPC investments in lossless networks.

4) Design and implement a proof-of-concept zero-loss virtual network prototype to experimentally validate the above design principles in a controllable hardware and software environment.

5) Finally, extend and validate at scale the experimental prototype with a detailed cross-layer simulation model.

4.3.2. End-to-end Argument

The wide availability of lossless fabrics and the thrust of SDN/OpenFlow have prompted us to reconsider the end-to-end and "dumb network" arguments in the context of datacenters. The end-to-end principle [106] can be traced back to the inception of packet networks [25]. Briefly stated, application-specific functions are better implemented in the end nodes than in the intermediate nodes: for example, error detection and correction should reside in NICs and operating system stacks and not in switches and routers. While one of the most enduring design principles, this can also restrict the system level performance in end-to-end delay, flow completion time and throughput [29].

In datacenters, the delay of latency-sensitive flows is impacted not only by network congestion, but also by the end-node protocol stacks [101]. Historically, for low-latency communications, both Arpanet and Internet adopted "raw" transports - unreliable, yet light and fast - instead of TCP-like stacks. Similarly, InfiniBand employs an Unreliable Datagram protocol for faster and more scalable "light" communications. Also HPC protocols have traditionally used low-latency end-node stacks

based on the assumption of a lossless network with very low bit-error rates. Given the increasing relevance of latency-sensitive datacenter applications, current solutions [16, 17, 119, 115] adopted an intriguing option: decouple flow control from the fabric. Here we show that coupling flow control with the fabric positively impacts the workload performance.

4.3.3. Overlay Virtual Network Design Space

The simplest virtual network would start with a large flat layer-2 network for each tenant. However, this approach does not scale within the practical constraints of current datacenter network technologies. The increasing number of VMs has led to a MAC address explosion, whereby switches need increasingly larger forwarding tables. Also, dynamic VM management stresses the broadcast domains [88]. Moreover, today's limit of 4K Ethernet VLANs is insufficient for multitenant datacenters unless Q-in-Q/MAC-in-MAC encapsulation is used. Finally, the datacenter network must support dynamic and automatic provisioning and migration of VMs and virtual disks without layer-2 or -3 addressing constraints. The emerging solution to full network virtualization are the overlay virtual networks. A number of overlays have recently been proposed [55, 110, 81, 88, 24, 37]. Their key architectural abstraction lies in the separation of virtual networking from the underlying physical infrastructure. Overlays enable an arbitrary deployment of VMs within a datacenter, independent of the underlying layout and configuration of the physical network, without changing or reconfiguring the existing hardware.

Current overlays are predominantly built using layer-2 to -4 encapsulation in UDP, whereby the virtual switches intercept the VM traffic, perform the en-/de-capsulation, and tunnel the traffic over the physical network. Each VM has an associated network state residing in the adjacent switch. Upon VM migration, the virtual switches update their forwarding tables to reflect the new location. Using encapsulation over IP [88, 81, 24, 37], the VM locations are neither limited by the layer-2 broadcast domains, nor by VLAN exhaustion. Instead, full IP functionality is preserved, including QoS and load balancing. Furthermore overlays are independent of location, domains and the physical networking capabilities. Thus these virtual switches are similar to traditional hypervisor switches, but now with additional functionality as overlay nodes. Inherently an overlay network trades some of the bare-metal performance for manageability, flexibility and security.

Performance-wise, such overlays influence datacenter's efficiency and scalability. First, on the data plane: they use encapsulation to build tunnels between virtual switches. Current encapsulation solutions, such as VXLAN [81] and NVGRE [110], solve the original VLAN limitation while reducing the configuration and management overhead. Second, on the management plane: configuration, distribution, and learning protocols are necessary to create tunnels at each virtual switch. To create a tunnel, the overlay switch must map the destination address to its physical location

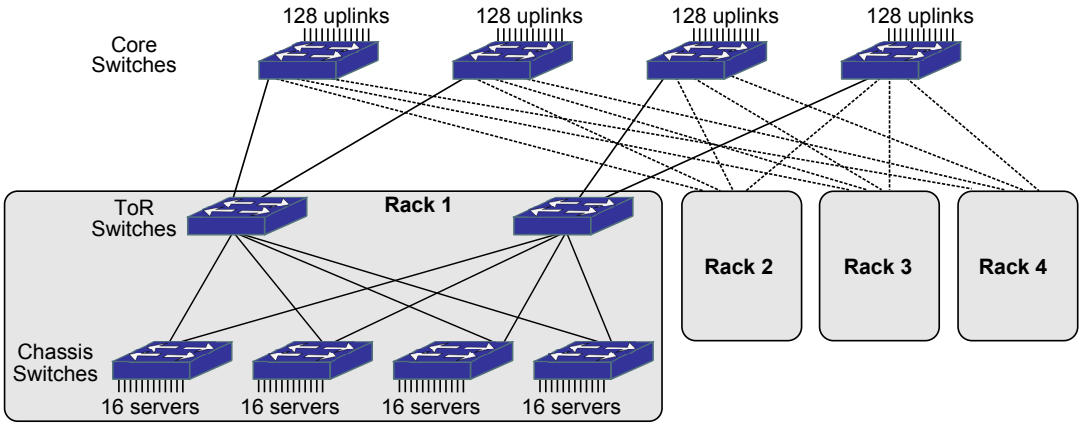


Figure 4.5.: Flat layer-2 fabric with 256 servers.

using either the learning or the centralized approach. The *learning* approach, used by VXLAN [81], floods packets with unknown destinations. The *centralized* approach relies on the virtual switches to retrieve the information required for encapsulation. In NetLord [88], this information is learnt by the switches as they communicate with each other and from a central configuration repository. In DOVE [24, 37], this configuration information is retrieved from a centralized database. Both the central configuration repository in NetLord and the centralized database in DOVE must be highly available and persistent. This poses a challenge for multi-million node datacenters, thus indicating a future third option of a distributed repository approach, presuming the entailing coherency issues can be solved efficiently. For now, the learning and centralized approaches are simpler to design and manage. Notably, the centralized method also inherently prevents flooding, the main drawback of the learning approach. For zOVN we have adopted and extended DOVE's centralized approach with a custom encapsulation header.

4.4. zOVN Implementation

In this section we describe the details of the implementation of our proposed lossless overlay network (zOVN). We assume a collection of virtualized servers, each running a set of virtual machines. The servers are interconnected through a flat layer-2 fabric (an example is shown in Figure 4.5). The physical network has per-priority flow control, allowing the network administrator to configure one or more priorities as lossless. The physical per-priority flow control is extended into the virtual domain by our proposed zOVN hypervisor software.

Without loss of generality, to simplify the description, we assume that a single lossless priority is used. In a real setup, different priority classes can be configured to segregate loss tolerant traffic, from mission-critical latency-sensitive traffic that benefits from losslessness, as shown in the next sections.

4.4.1. Path of a Packet in zOVN

The data packets travel between processes (applications) running inside VMs. Along the way, packets are moved from queue to queue within different software and hardware components. Here we describe the details of this queuing system, with emphasis on the flow control mechanism between each queue pair. The packet path trace is shown in Figure 4.6.

After processing the packets in the VM's guest kernel, they are transferred to the hypervisor through a vNIC. The hypervisor forwards them to the virtual switch, which provides the communication between VMs and the physical adapter. Packets destined to remote VMs are taken over by a bridge with OVN tunneling functionality that encapsulates and moves them into the physical adapter queues. After traversing the physical network, they are delivered to the destination server, where they are received by the remote bridge, which terminates the OVN tunnel by decapsulating and moving them into the destination's virtual switch input queues. The virtual switch forwards the decapsulated packets to the local hypervisor, which in turn forwards them to the guest OS. After processing in the guest kernel, the received packets are eventually delivered to the destination application. Based on a careful analysis of the end-to-end path, we identified and fixed the points of potential loss, labeled in white on black in Figure 4.6, i.e., the vSwitch and the reception path in the guest kernel.

4.4.1.1. Transmission Path

On the transmit side, packets are generated by the user-space processes. As shown in Figure 4.6, the process issues a `send` system call that copies a packet from user space into the guest kernel space. Next, packets are stored in an `sk_buff` data structure and enqueued in the transmit (TX) buffer of the socket opened by the application. The application knows whether the TX buffer is full from the return value of the system call, making this a lossless operation.

Packets from the socket TX buffer are enqueued in the *Qdisc* associated with the virtual interface. The Qdisc stores a list of pointers to the packets belonging to each socket. These pointers are sorted according to the selected discipline, FIFO by default. To avoid losses at this step, we increase the length of the Qdisc to match the sum of all socket TX queues. This change requires negligible extra memory. The Qdisc tries to send the packets by enqueuing them into the adapter TX queue. If the TX queue reaches a threshold – typically one MTU below maximum – the Qdisc is stopped and the transmission is paused, thus avoiding losses on the TX path of the kernel. When the TX queue drops below the threshold, the Qdisc is restarted and new packets can be enqueued in the TX queue of the virtual adapter. Hence, the transmission path in the guest OS remains lossless as long as the Qdisc length is properly sized.

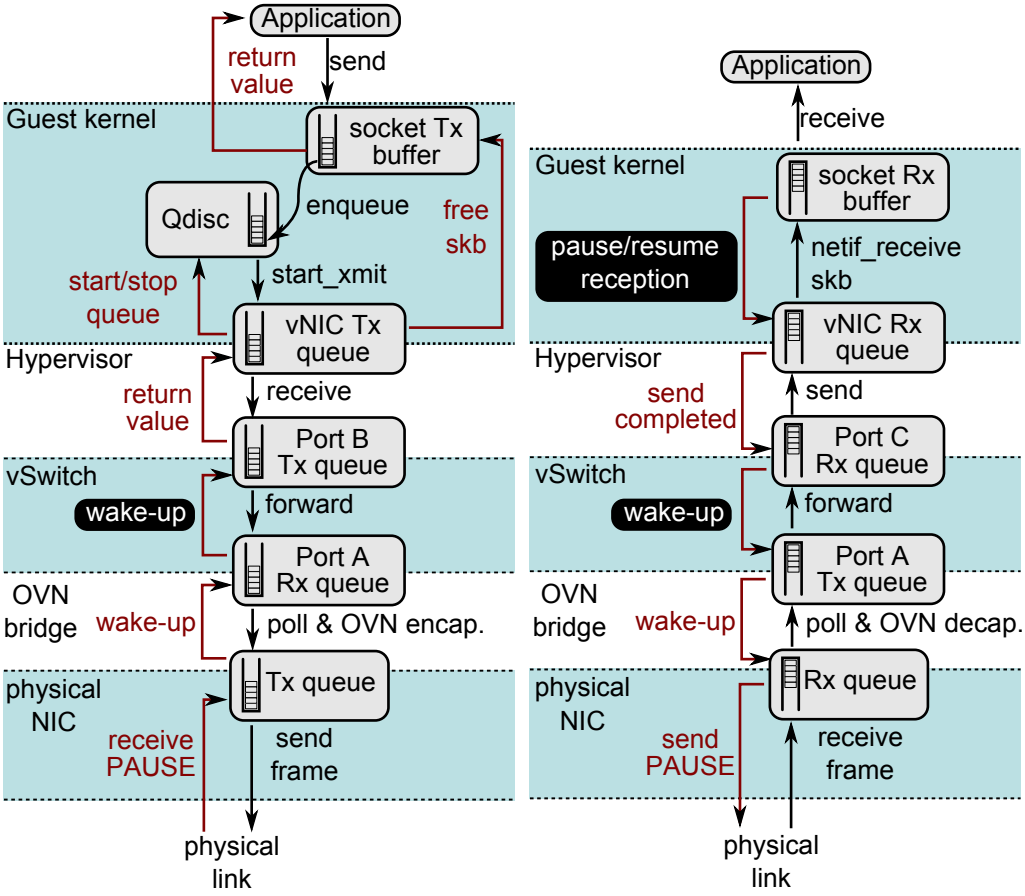


Figure 4.6.: Life of a packet in a virtualized network.

Our architecture is based on virtio technology [105], hence the virtual adapter queues are shared between the guest kernel and the underlying hypervisor software running in the host user space. The virtio adapter informs the hypervisor when new packets are enqueued in the TX queue. The hypervisor software is based on Qemu [8] and is responsible for dequeuing packets from the TX queue of the virtual adapter and copying them to the TX queue of the zOVN virtual switch.

The Qemu networking code contains two components: virtual network devices and network backends. We use the virtio network device coupled to a Netmap [101] backend. We took the Netmap backend code of the VALE [102] virtual switch and ported it to the latest version of Qemu with the necessary bug fixes, mainly related to concurrent access to the Netmap rings. We use a lossless coupling between the device and the backend, avoiding – via configuration flags – the lossy Qemu VLANs. Packets arrive at the vSwitch TX queue of the port to which the VM is attached. The vSwitch forwards packets from the TX queues of the input ports to the RX queues of the output ports using a forwarding (FIB) table that contains only the MAC addresses of the locally connected VMs. If the destination is found to be local, the respective packets are moved to the corresponding RX queue; else they

are enqueued in the RX port corresponding to the physical interface. From here, packets are consumed by a bridge that encapsulates and enqueues them in the TX queue of the physical adapter. Then the lossless CEE physical network delivers the packets to the destination server's physical RX queue.

As shown in Section 4.2.3, none of the current virtual switches implement flow control, as also confirmed by our discussions with some of the virtualization vendors. Therefore we have redesigned the VALE vSwitch to add internal flow control and to make the TX path fully lossless, as described in Section 4.4.2.

4.4.1.2. Reception Path

The incoming packets are consumed and decapsulated by the OVN tunneling bridge from the RX queue of the physical NIC. Next, they are enqueued in the TX queue of the virtual switch that forwards them to the RX queue corresponding to the destination VM. This forwarding is again lossless, see Section 4.4.2. The packets are consumed by the Qemu hypervisor, which copies them into the virtio virtual device. The virtual device RX queue is shared between the hypervisor and the guest kernel. The hypervisor notifies the guest when a packet has been received and the guest OS receives an interrupt. This interrupt is handled according to the Linux[2] NAPI framework. A *softirq* is raised, which triggers packet consumption from the RX queue. The packet is transferred to the `netif_receive_skb` function that performs IP routing and filtering. If the packet is destined to the local stack, it is enqueued in the destination socket RX buffer based on the port number. If the destination socket is full, then the packet is discarded. With TCP sockets this should never happen because TCP has end-to-end flow control that limits the number of injected packets to the advertised window of the receiver. UDP sockets, however, require additional care. We modified the Linux kernel such that when the destination socket RX queue occupancy reaches a threshold – i.e., one MTU below maximum – the softirq is canceled and reception is paused. Once the process consumes data from the socket, reception is resumed. This ensures full lossless operation for both TCP and UDP sockets.

4.4.2. zVALE: Lossless virtual Switch

As stated before, our lossless vSwitch is derived from VALE [102], which is based on the Netmap architecture [101]. It has one port for each active VM, plus one additional port for the physical interface. Each port has an input (TX) queue for the packets produced by the VMs or received from the physical link, and an output (RX) queue for the packets to be consumed by VMs or sent out over the physical link. The lossy state-of-the-art implementation forwards packets from input to output queues as fast as they arrive. If an output queue is full, packets are locally discarded.

Algorithm 4.1: Lossless Virtual Switch Operation

```
globals: N input queues I_j, N output queues O_k
on send(input queue I_j, frame F)
    if input queue I_j full then
        sleep
    else
        I_j.enqueue(F)
        start forwarder(I_j)
    end
end
on receive(output queue O_k, frame F)
    if output queue O_k empty then
        for j ← 1..N do
            start forwarder(I_j)
        end
    end
    if output queue O_k empty then
        sleep
    else
        F ← O_k.dequeue()
    end
end
on forwarder(input queue I_j)
    foreach frame F in input queue I_j do
        output port k ← forwarding table lookup(F.dstMAC)
        if not output queue O_k full then
            I_j.remove(F)
            O_k.enqueue(F)
            wake_up receiver(O_k) and sender(I_j)
        end
    end
end
```

To make such a software switch lossless, we designed and implemented the pseudocode shown in Algorithm 4.1. Each sender (producer) is connected to an input queue I_j, and each receiver (consumer) is connected to an output queue O_k. After a packet has been produced, the sender checks whether the associated input queue is full. If the queue is full, the sender goes to *sleep* until a free buffer becomes available, else the sender enqueues the packet in the input queue and then starts a forwarding process to try to *push* packets from the input to the output queues. The forwarder checks each output queue for available space. If a queue has room, the forwarder transfers the packets to the output queue and *wakes* up the corresponding consumers that might be waiting for new packets. On the receiver side, the associated output queue is checked; if not empty, a packet is consumed from it, else the forwarding

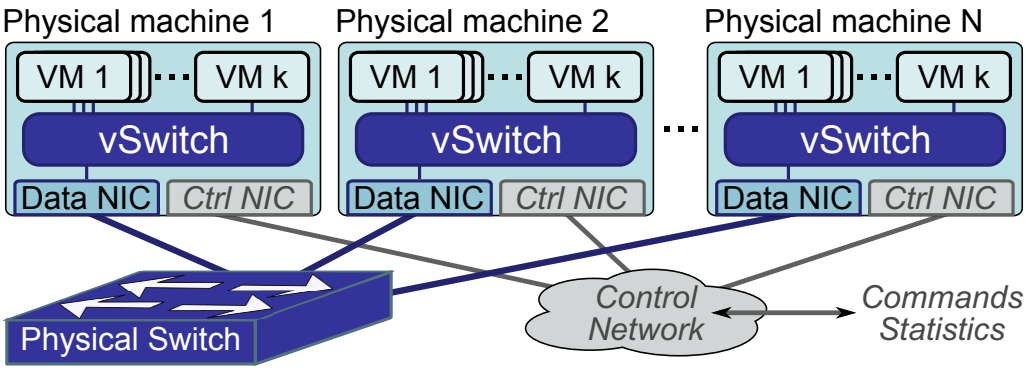


Figure 4.7.: Real implementation testbeds.

	Testbed 1	**Testbed 2**
System Type	Lenovo T60p Laptops	IBM System x3550 M4 Rack Servers
CPU	1x Intel Core 2 T7600	2x Intel Xeon E5-2690
Total cores	2	16
Clock speed [GHz]	2.33	2.90
Memory [GB]	3	96
Physical machines	8	4
VMs/machine	4	16
Data network	1G Ethernet	10G CEE
Physical switch	HP 1810-8G	IBM RackSwitch G8264
Control network	wireless	1G wired
Linux kernel	3.0.3 64-bit	3.0.3 64-bit

Table 4.2.: Experimental testbed configurations.

process is started to *pull* packets from the input queues to this output queue. If data is actually pulled, it is consumed; else the receiver sleeps until woken up by the sender.

The vSwitch is designed to operate in a dual push/pull mode. When the sender is faster (than the receiver), it will sleep most of the time waiting for free buffers, while the receiver will wake it up only when it consumes data. When the receiver is faster (than the sender), it will sleep most of the time, while the sender will wake it up only when new data becomes available. The overhead of lossless operation is thus reduced to a minimum.

4.5. Evaluation

In this section we evaluate our proposed lossless vSwitch architecture, applying the Partition-Aggregate (PA) workload described in Section 4.5.1. We run this workload

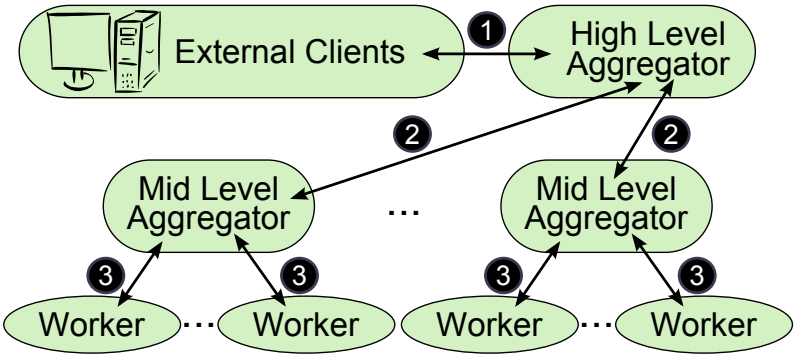


Figure 4.8.: Partition-Aggregate (PA) application.

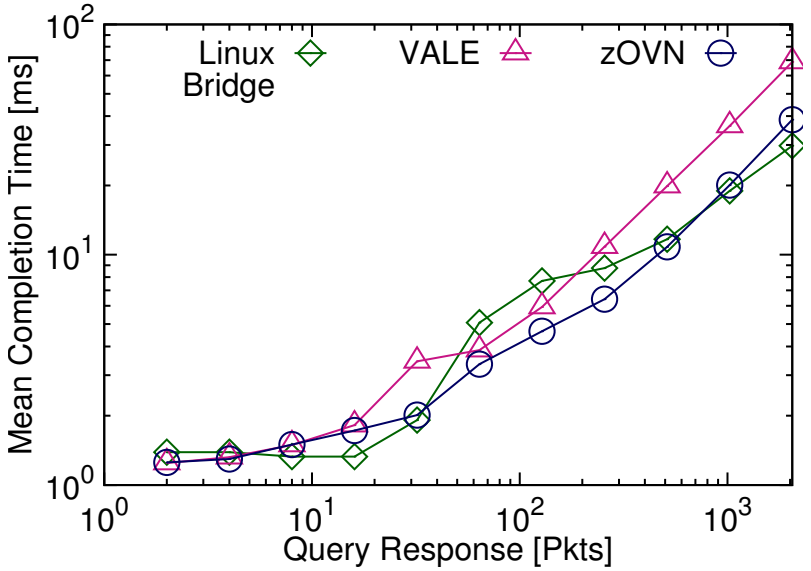
both in two lab-scale experiments with 32 VMs and in a larger-scale simulation using an OMNeT++ model of a 256-server network.

4.5.1. Partition-Aggregate Workload

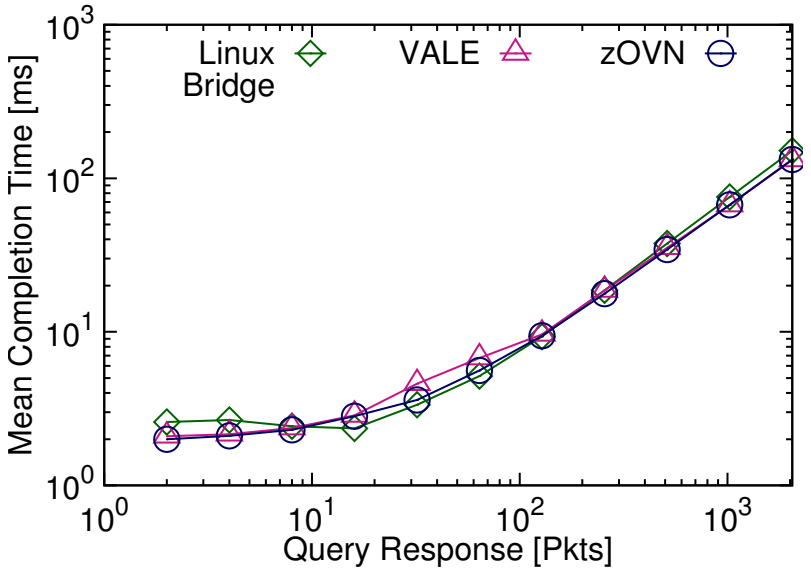
A generic 3-tier PA application is presented in [16, 119] and illustrated in Figure 4.8. At the top tier, a high-level aggregator (HLA) receives HTTP queries from external clients (1). Upon reception of such a request, the HLA contacts randomly selected Mid-Level Aggregators (MLA) and sends them a subquery (2). The MLAs further split the subquery across their workers, one in each server in the same chassis (3). Eventually, each worker replies to the MLA by returning a response. The MLA collects the partial results from workers. When *all* results have been received, the MLA sends its aggregated response to the HLA. The query is completed when the HLA receives the aggregated response from each MLA. The key metric of interest is the flow (or query) completion time, measured from arrival of the external query until query completion at the HLA. In the prototype experiments, similar with the experiments described in [16, 119], we use a reduced two-tier PA workload, in which the MLAs have been omitted, and the HLAs contact the workers directly. In the simulations, on the other hand, we use the full configuration. In both cases, the flows are sent over TCP. The connections between the various components are kept open during the runs to allow TCP to find the optimal congestion window sizes and to avoid slow start.

4.5.2. Microbenchmarks

First, we deployed our prototype implementation on two Lenovo M91p-7034 desktops (Intel i5-2400 @ 3.10GHz CPU, 8GB memory, Linux 3.0.3 64-bit kernel both for host and guests). The machines were connected through a 1 Gbps 3com 3GSU05 consumer-level Ethernet switch supporting IEEE 802.3x. The host kernel was patched with the Netmap [101] extensions and our zOVN switch and bridge. The guest kernel was patched with our lossless UDP socket extension.



(a) 1 server



(b) 2 servers

Figure 4.9.: Microbenchmarks: 6 VMs PA.

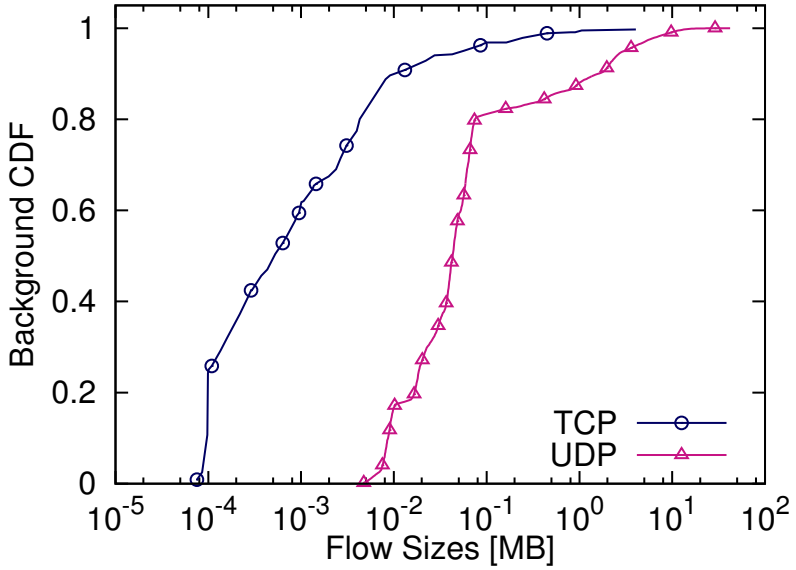
We ran PA queries with a single aggregator and five workers. In Figure 4.9 we report the mean query completion time. In Figure 4.9a the aggregators and the workers resided in VMs on the same server (1-server setup), whereas in Figure 4.9b the aggregator was on a different server than the workers (2-server setup). We varied the size of the workers response to the aggregator from 2 to 2048 MTUs. To achieve statistical confidence, each run consisted of 10K repetitions. We compared the Linux Bridge [4] with the lossy VALE implementation [102] and our proposed lossless zOVN. On the 2-server setup, the Netmap-based solutions outperformed the Linux Bridge, but only for small response sizes (up to 30% for 2 MTUs). For medium-sized flows, the Linux Bridge was better (e.g., 8% performance degradation for 64 MTUs when using zOVN). For large response sizes, the three implementations exhibited similar response times. The physical link has a constant service rate, so that TCP was able to find the proper congestion window to avoid most losses. On the desktop machines, the vSwitch could support up to 1.45 Gbps of traffic without losses, compared with the 256 Mbps for the laptop machines. However, the maximum bandwidth through the vSwitch was limited to the 1 Gbps of the physical link, which was the bottleneck in this case. Accordingly, we measured loss ratios of less than 0.02%. Enabling losslessness on such a configuration brings no additional benefits. However, this result validates the efficiency of our implementation.

In the 1-server setup, the zOVN switch was consistently better than the lossy VALE switch across all runs. The Linux Bridge exhibited performance variabilities (up to +19% improvement for the 16 MTU responses over zOVN, but as much as –65% degradation over zOVN for 128 MTU responses). The architecture of the Linux Bridge requires one extra copy for each packet sent or received. This extra overhead slows down the workers reducing the pressure on the vSwitch, thereby reducing packet losses. In the 2-server scenario, the extra overhead was hidden by the physical link bottleneck.

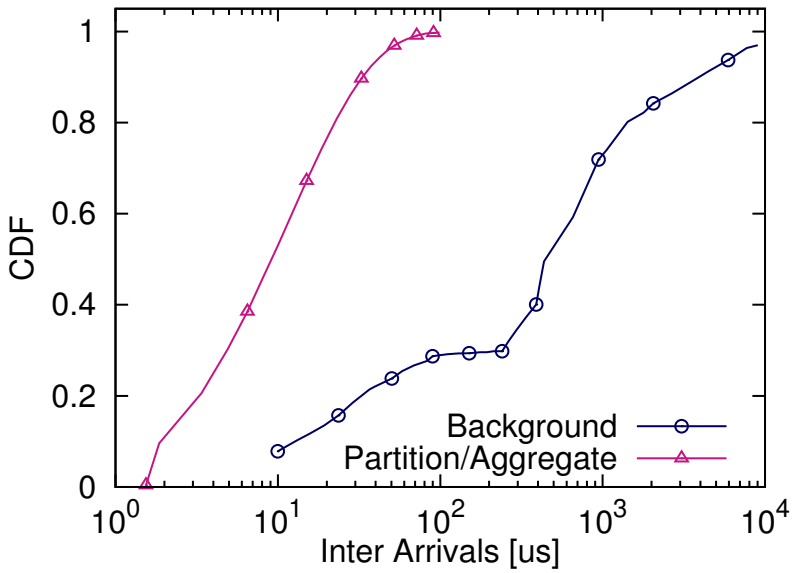
4.5.3. Lab-Scale Experiments

Next, we deployed zOVN over the two testbeds described in Figure 4.7. We ran a PA workload using 32 VMs with the same methodology and flow sizes as in the previous paragraph. In addition, we varied the TCP version between NewReno, Vegas and Cubic. As shown in Figure 4.7, each physical machine has two network interfaces. The PA traffic that is subject to measurements flows through an isolated data network. The workers, aggregators and background traffic generators are started and killed through a separate control network, which is also used to configure the data network before each run and to gather the statistics at the end without interfering with the experiments.

Testbed 1: Laptops. In Figure 4.11a and 4.11b, we report the mean completion time and performance gain of zero-loss (Z) over lossy (L). The zero-loss configuration has flow control enabled both in the physical and the virtual network, whereas the

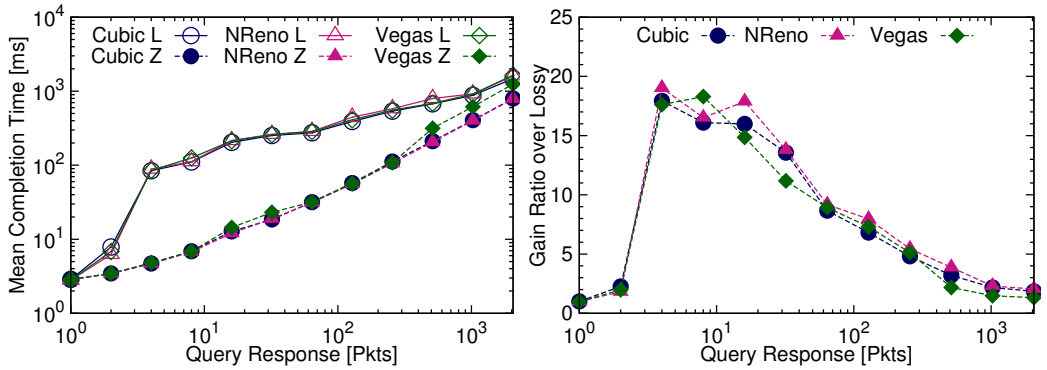


(a) Background flow size.

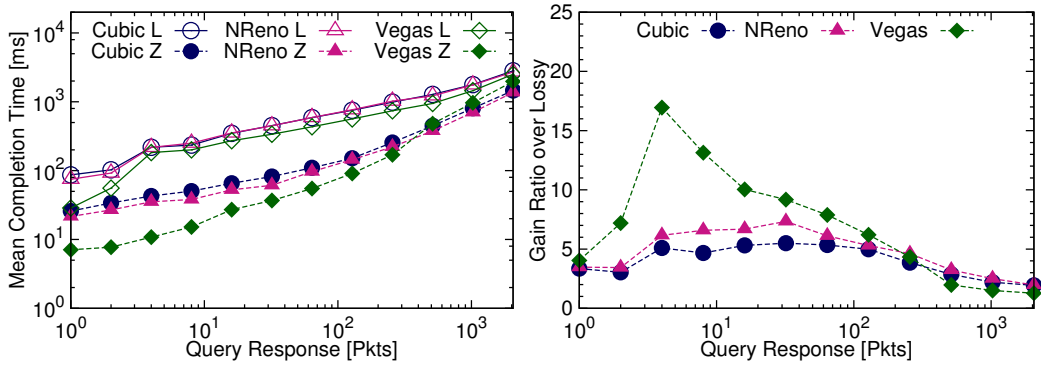


(b) Inter-arrival times.

Figure 4.10.: Flow size and inter-arrival distribution.



(a) w/o bkgd. (b) w/o bkgd.

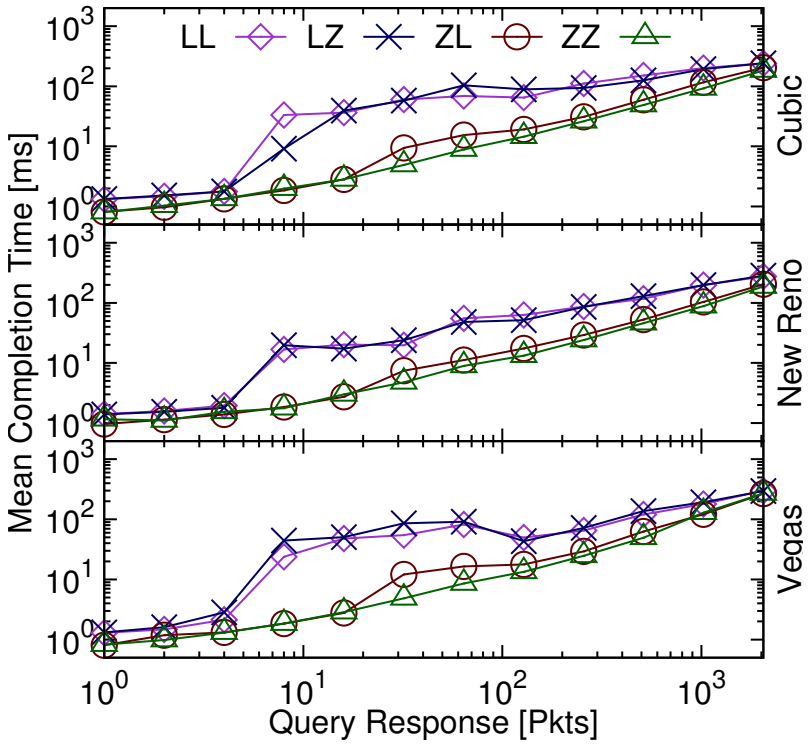


(c) with bkgd. (d) with bkgd.

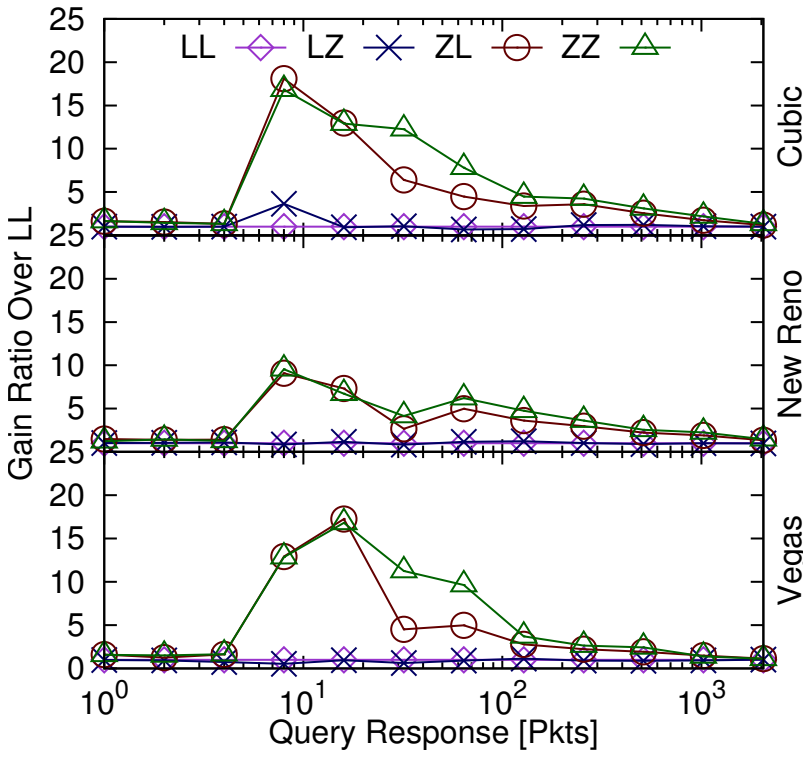
Figure 4.11.: Testbed 1 results: 32 VMs PA running on 8 laptops.

lossy configuration has no flow control in any of the two networks. The mean flow completion time was reduced by a factor of up to 19.1×. The highest benefit was achieved for flow sizes between 6 KB and 48 KB (4 and 32 packets). For very small flows, the total size of all worker responses was too small to cause any buffer overflow. For long flows, the losses were recovered through fast-retransmit and selective acknowledgments. All TCP versions performed about equally.

In Figure 4.11c and 4.11d, we report the same metrics, but with background traffic. In this scenario, each VM hosts an additional traffic generator producing background flows. The generator chooses a random uniformly distributed destination, then it sends to it a TCP flow with the length drawn from the distribution in Figure 4.10a. Afterward, the generator sleeps according to the background flow inter-arrival distribution shown in Figure 4.10b. Both the PA and the background flows use the same TCP version. The gain is smaller than in the previous scenario, because the background flows also benefit from losslessness obtaining a higher throughput. In particular, the congestion window of NewReno and Cubic are kept open due to the absence of losses. On the other hand, the latency sensitive Vegas injects background traffic at a lower rate, thus the completion times are shorter.

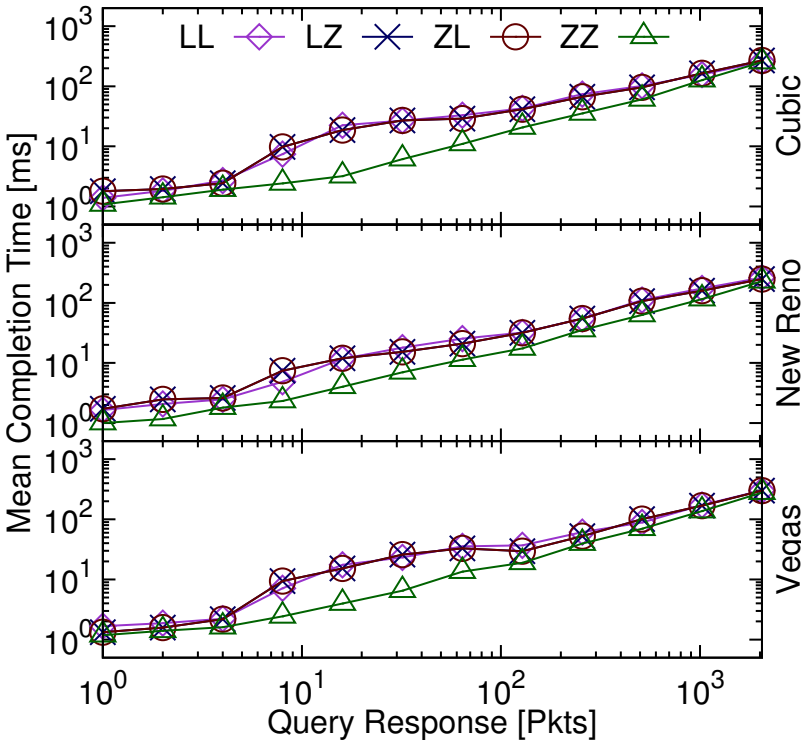


(a)

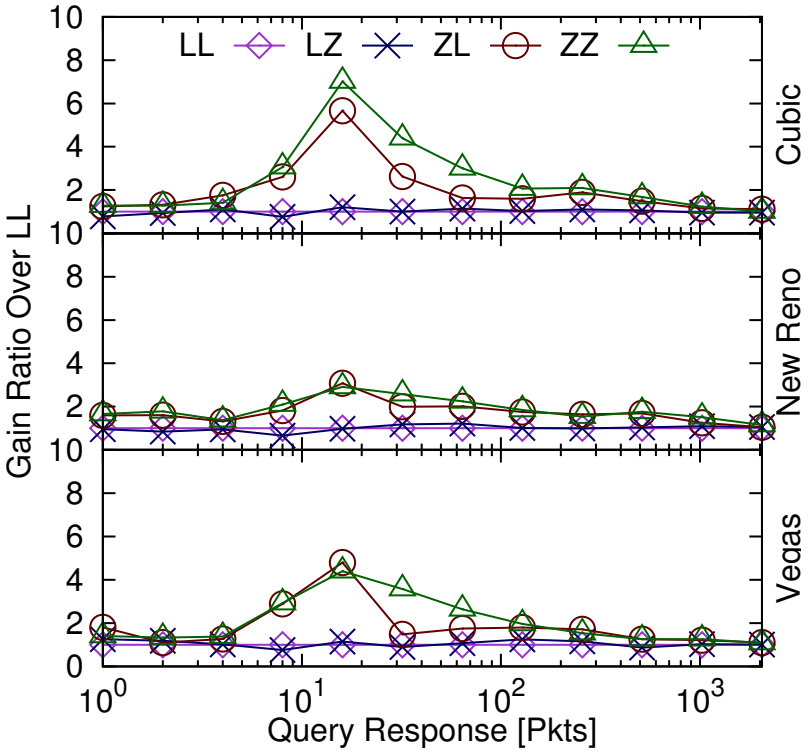


(b)

Figure 4.12.: Testbed 2 results: Without background flows, 32 VMs PA running on 4 rack servers.



(a)



(b)

Figure 4.13.: Testbed 2 results: With background flows, 32 VMs PA running on 4 rack servers.

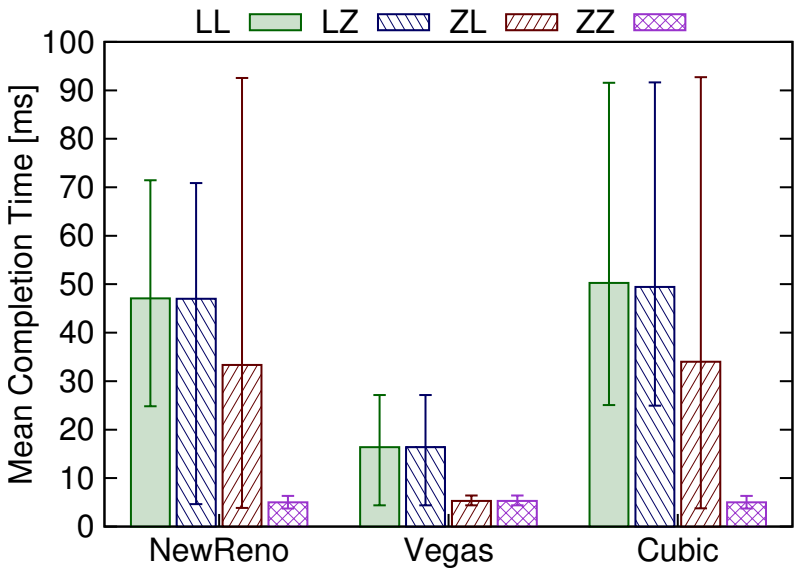
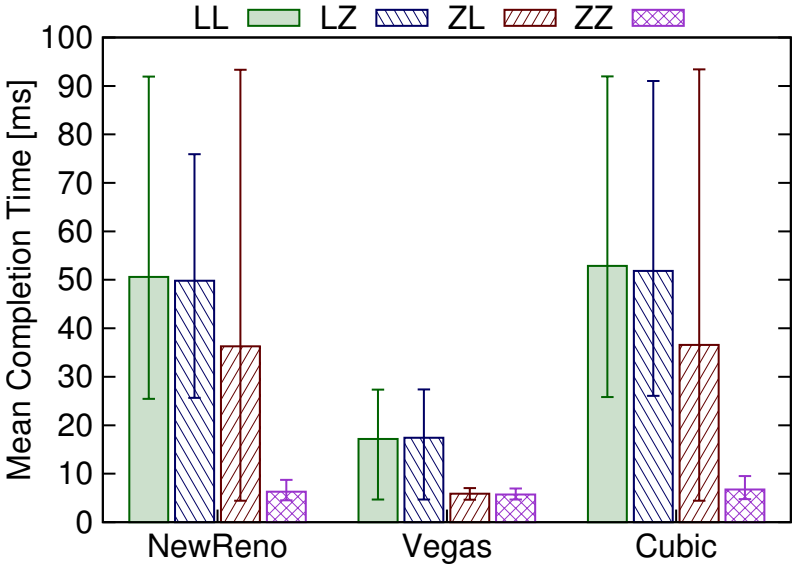


Figure 4.14.: Simulation results: Without background flows. 768 VMs PA with 256 servers.

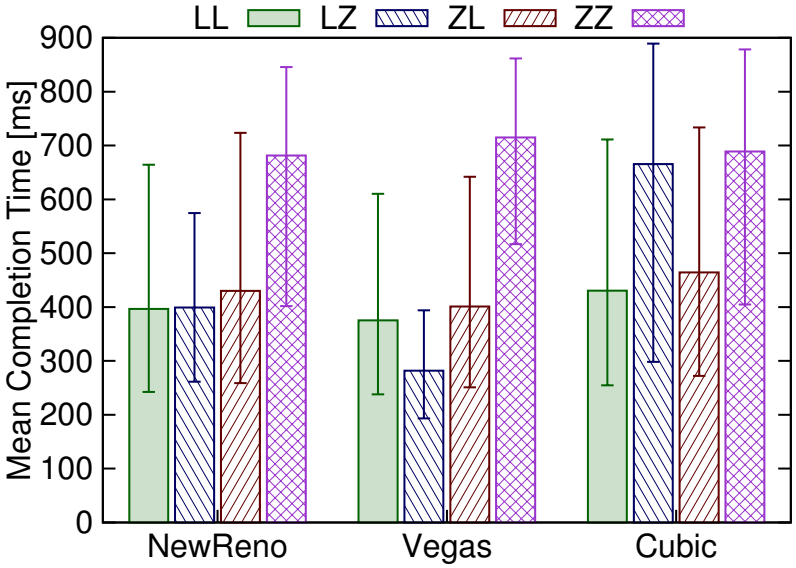
Testbed 2: Rack Servers. We repeat the above experiments on 4 rack servers with a 10G CEE network. Each server hosts 16 VMs: 8 for PA traffic and 8 VMs for generating background traffic. We studied four flow control configurations: no flow control (LL), flow control activated in the physical network (LZ), flow control activated in the virtual network (ZL), and flow control activated in both (ZZ). The mean completion times and gains over LL are reported in Figure 4.12a and 4.12b. The mean completion times are reduced by a factor up to 15.95×, similar to the laptop experiments. Although the server CPUs have more resources than the laptop CPUs, they have to handle more VMs and more traffic from a 10× faster network. Activating flow control only in the physical network (LZ) showed no major benefit in this scenario, where the primary bottleneck is in the vSwitches. Also, enabling flow control only in the vSwitch (ZL) shifted the drop point from the virtual to the physical domain. Finally, in Figure 4.13a and 4.13b, we repeated the experiments with background traffic, confirming the findings from Testbed 1.

4.5.4. Simulation Experiments

To finalize our validation, we implemented a model of the zOVN system on top of the OMNeT++ network simulator. The simulator models a 10G CEE fabric at frame level with generic input-buffered output-queued switches. As the TCP models implemented in OMNeT++, as well as those from NS2/3, are highly simplified, we ported the TCP stack from a FreeBSD v9 kernel into this simulator with only minimal changes, most of them related to memory management. As we focus on the network, we did not model the endnode CPUs, assuming that the endnodes can process the segments as fast as they arrive, and that the applications can reply



(a) TCP bkgd



(b) UDP bkgd

Figure 4.15.: Simulation results: 768 VMs PA with 256 servers.

immediately. The stack adds only a *fixed* delay to each segment, calibrated from our prior hardware experiments. Even if idealized, these assumptions are consistent with our network-centric methodology. The simulator also incorporates a thin UDP layer used for background flows performing simple segmentation and encapsulation of the application data.

The zOVN model performs switching and bridging in the same way as in the testbed experiment. However, here we chose a different encapsulation size of 54B, reflecting a VXLAN-type encapsulation: 18B outer Ethernet header + 20B outer IP header + 8B UDP header + 8B VXLAN header. To avoid fragmentation, we decreased the MTU value accordingly from 1500B to 1446B. Modern CEE hardware is able to increase its physical MTUs, thus preserving the default settings.

The simulated network topology is shown in Figure 4.5. It consists of 256 servers, distributed in 16 chassis, and interconnected through a three-layer fat tree. Clients attached to the up-links inject HTTP queries that are served by the VMs residing on each virtualized server. The queries were generated according to the inter-arrival times shown in Figure 4.10b. Each server hosts 3 VMs, one HLA, one MLA and one worker. The client query reaches a randomly chosen HLA that in turns chooses 16 MLAs, one in each chassis. Each MLA contacts all worker VMs from the same chassis. The messages exchanged between the HLA, MLAs and workers have a fixed size of 20KB.

Figure 4.15 compares the mean completion times and the 5- and 95-percentiles for different flow control configurations under no, light, and heavy background traffic. We studied the four flow control configurations introduced above (LL, LZ, ZL, and ZZ) and the same three TCP versions as before. Enabling flow control in only one network (either physical or virtual) is not beneficial, because packet losses are merely shifted from one domain to the other. However, the effects were not altogether identical, because the virtual flow control still benefited inter-VM communications on the same host. Therefore, enabling only the virtual flow control (ZL) still led to a performance improvement, although smaller than in the ZZ case. Enabling both flow controls (ZZ) achieved significant gains, similar to those observed in the testbed: a reduction in FCT of up to 10.1× with Cubic, and no background flows. When adding light background traffic, we observed similar gain decreases. However, a new insight is that in the presence of heavy UDP background traffic, enabling flow control will harm performance. In this case, the uncooperative background UDP packets did no longer get dropped and, consequently, hogged link capacity and harmed the foreground PA workload traffic. These results confirmed the need to segregate the traffic into PFC priorities with true resource separation and scheduling. It may also suggest the need for a layer-2 congestion management loop as in [38].

With background traffic, Vegas outperformed NewReno and Cubic, confirming the results obtained on the testbed setups. In the case without background traffic Vegas was again better. Nonetheless, on the testbeds, all TCP versions produced similar results. The difference here is due to the more complex communication pattern with

more hops, as more flows share the same path. This causes longer queues, especially in the core switches. The longer delays are detected by Vegas, which will reduce its congestion window, thus obtaining shorter completion times.

4.6. Results Analysis

Here we review the main takeaways from the results presented in this chapter. Using zOVN's experimental platform, we demonstrated both absence of packet drops – in support of converged storage and HPC applications – *and* improved flow completion time (FCT) performance. Thus, we have achieved our primary objective of reconciling performance with losslessness for overlay virtual networks.

Is lossless flow control more relevant for physical or for virtual networks? Having tested all four combinations of lossy and lossless physical and virtual flow control both in our testbed and in simulations, we found that contiguous end-to-end flow control, hop-by-hop within each domain, yields the largest reductions in FCT: PA over zOVN with 32 virtual workers distributed across four physical rack servers achieved up to 15-fold peak speedup. Relevant to on-line and data-intensive workloads in general, the highest speedups recorded are for flows between 6 and 50 KB. Unexpectedly, if a suboptimal choice between flow control in either the physical or the virtual network must still be made, the latter is better for FCT performance, as demonstrated by the results for ZL vs. LZ in Figure 4.15. As noted initially, this situation entails a paradoxical twist: Although CEE and InfiniBand fabrics have already implemented the costlier (buffers, logic, and signaling) hardware flow control, this remains practically non-existent in today's virtual networks - despite much lower implementation efforts.

Are our modest experimental platforms relevant for hundreds of blade-based racks and top-of-rack switches with 40-100 Gbps uplinks? While the definitive answer would entail a multi-million dollar datacenter setup, we are confident in the relevance of our admittedly limited prototype platforms. Thin and embedded low-power CPUs as used in microservers as well as fully virtualized, and hence loaded, "fat" CPUs are likely to exhibit qualitatively similar behaviors as these measured on our two testbeds.

During zOVN experiments we consistently observed how the loss ratio is influenced by the CPU/network speed ratio. On the transmit side, a fast Intel Xeon[2] CPU can easily overload a slower 1G network, producing more losses in the vSwitch than a slower CPU (Intel Core 2) with the same 1G NIC does. On the other hand, on the receive side, a fast 10G network coupled with a loaded Intel Xeon CPU produces more drops than the 1G network with the same CPU does. As TX is network-limited, a fast network is beneficial on the TX side – but hurts performance on the RX side – whereas a fast CPU is beneficial on the RX side – processor-limited – while it hurts the TX side. In conclusion, a different CPU/network speed ratio is

not a viable substitute for a correct implementation of flow control in the virtual network.

4.7. Related Work

In recent years, the TCP incast and flow completion time performance of Partition-Aggregate applications has been extensively analyzed. For example, [33, 117] suggest a 10-1000× retransmission timeout reduction. Other proposals achieve sizable flow completion time reductions for typical datacenter workloads using new single-path [16, 17, 119, 115] or multi-path [122, 62, 107, 15] transports. These are coupled with deadline-aware or agnostic schedulers and per-flow queuing. Related to our work and to [58, 38], DeTail [122] identifies packet loss in physical networks as one of the three main issues. The authors enable flow control, i.e., PFC, and introduce a new multi-path congestion management scheme targeted against flash hotspots typical of Partition-Aggregate workloads. They also employ explicit congestion notification (ECN) against persistent congestion. DeTail uses a modified version of NewReno to reduce flow completion time by 50% at the 99.9-percentile, but does not address virtual overlays.

pFabric [18] re-evaluates the end-to-end argument. It introduces a "deconstructed" light transport stack resident in the end node and re-designed specifically for latency-sensitive datacenter applications. Furthermore, a greedy scheduler implements a deadline-aware global scheduling and a simplified retransmission scheme recovers losses. By replacing both the TCP stack *and* the standard datacenter fabric, this scheme achieves near-ideal performance for short flows. Open issues are the scalability to datacenter-scale port counts, costs of replacing commodity fabrics and TCP version, fairness, and compatibility with the *lossless* converged datacenter applications.

DCTCP [16] uses a modified ECN feedback loop with a multibit feedback estimator filtering the incoming ECN stream. This compensates the stiff active queue management in the congestion point detector with a smooth congestion window reduction function reminiscent of QCN's rate decrease. DCTCP reduces the flow completion time by 29%, however, as a deadline-agnostic TCP it misses about 7% of the deadlines. D3 [119] is a deadline-aware first-come first-reserved non-TCP transport. Its performance comes at the cost of priority inversions for about 33% of the requests [115] and a new protocol stack. PDQ [62] introduces a multi-path preemptive scheduling layer for meeting flow deadlines using a FIFO taildrop similar to D3. By allocating resources to the most critical flows first, PDQ improves on D3, RCP and TCP by circa 30%. As it is not TCP, its fairness remains to be studied. D2TCP [115] improves on D3 and DCTCP, with which it shares common features in the ECN filter, by penalizing the window size with a gamma factor. Thus, it provides iterative feedback to near-deadline flows and prevents congestive collapse. This deadline-aware TCP-friendly proposal yields 75% and 50% fewer deadline misses

than DCTCP and D3, respectively. Hedera and MP-TPC [15, 61, 100] propose multi-path TCP versions optimized for load balancing and persistent congestion. However, short flows with fewer than 10 packets or FCT-sensitive applications do not benefit, despite the complexity of introducing new sub-sequence numbers in the multi-path TCP loop.

4.8. Discussion

Fabric-level per-lane flow control to prevent packet loss due to contention and transient congestion has long been the signature feature of high-end networks and HPC interconnects. The recent introduction of CEE priority flow control has now made it a commodity. In spite of the advances at layer-2, we have shown that present virtual overlays lag behind. Congestion, whether inherent in the traffic pattern or as an artifact of transient CPU overloads, is still handled here by dropping packets, thus breaking convergence requirements, degrading performance, and wasting CPU and network resources.

In this chapter we provided first evidence that, for latency-sensitive virtualized datacenter applications, packet loss is a costly singularity in terms of performance. To remedy this situation, we have identified the origins of packet drops across the entire virtualized communication stack, and then designed and implemented a fully lossless virtual network prototype.

Based on the experimental results using our prototype implementations and also larger-scale simulations, we have demonstrated average FCT improvements of one order of magnitude. Additional takeaways are that (i) packet loss in virtualized datacenters is even costlier than previously studied in physical networking; (ii) FCT performance of Partition-Aggregate workloads is greatly improved by losslessness in the virtualized network; (iii) commodity CEE fabrics and standard TCP stacks still have untapped performance benefits. Furthermore, zOVN can be orthogonally composed with other schemes for functional or performance enhancements on layers 2 to 5.

Next, in Chapter 5, we will further extend zOVN with an optimized hypervisor-based transport layer and new mechanisms to improved security and reliability even in the presence of aggressive traffic sources.

5. zFabric: Virtualized Transport for Converged Enhanced Ethernet

In Chapter 4 we proposed a zero-loss Overlay Virtual Network (zOVN) and the associated lossless virtual switch. We showed that zOVN achieves up to one order of magnitude reductions of the flow completion times for latency sensitive applications. This is acomplished by extending the link-level flow control of Converged Enhanced Ethernet (CEE), described in Chapter 2, into the virtual domain. In this chapter we deconstruct the existing TCP stack from the VMs kernel and consolitate its functions into zFabric, a new hypervisor build around the lossless virtual switch introduced in Chapter 4.

Lossless CEE is a crucial step in embracing storage, cluster, and high-performance computing fabrics under a converged network. However, the adoption of CEE in virtualized datacenters is hindered by the lack of hypervisor software that addresses the major issues of losslessness, i.e., head-of-line blocking and saturation trees.

Our objective is to design a hypervisor that prevents misconfigured or malicious virtual machines (VMs) from filling the lossless cluster with stalled packets, thus compromising tenant isolation. Furthermore, we observe that current hypervisors perform compulsory isolation, management, and mobility functions, but introduce new bottlenecks on the datapath. By taking advantage of the lossless fabric, we deconstruct the existing virtualized networking stack into its core functions and consolidate them into zFabric, an efficient hypervisor that meets our aforementioned goals. In addition, zFabric allows us to optimize the performance of TCP.

To demonstrate zFabric's benefits, we evaluate a prototype implementation on a datacenter testbed. Besides resolving HOL-blocking, zFabric improves throughputs for long flows by up to *56%*, lowers CPU utilization by up to *63%*, and shortens completion times by up to *7x* for partition-aggregate queries when compared with current virtualized TCP stacks.

5.1. Introduction

Driven by new datacenter workloads, standards and technologies –e.g. Software Defined Networking (SDN), flattened datacenter fabrics, and Converged Enhanced Ethernet (CEE)– wired networking undergoes a silent, yet disruptive transition period. Concurring to a 'perfect storm' in the datacenter, virtualization and SDN

technologies introduce new protocols [108, 68, 88, 37], inserted between the TCP stack and the physical network. The minimal functions implemented by most hypervisors include forwarding, filtering and multiplexing of virtual machine (VM) traffic. On top of these, hypervisors must also perform congestion control to enforce fairness between VMs [68, 108, 22, 56, 99, 23, 74]. Furthermore, hypervisors must isolate the agile virtual networks from the physical infrastructure [88, 37, 81, 110].

We argue that these layers, although compulsory, create new bottlenecks. Indeed, thus far the SDN community has focused on the *control* plane, –i.e., management, security, VM mobility– and less on the *datapath* performance, whether from a workload or individual flow's perspective.

A challenging opportunity in the datacenter is the rise of *lossless* Converged Enhanced Ethernet. CEE was driven by the desire to reduce costs through the convergence of cluster, storage and high-performance computing networks. In addition, lossless CEE networks convey performance benefits for cluster traffic, for example by eliminating TCP incast throughput collapse and by reducing the flow-completion time of latency-critical applications [98, 40].

Previous work [68, 108, 22, 56, 99, 23, 74] outlined that, in the case of lossy networks, hypervisor level mechanisms are needed to deal with malicious or misconfigured VMs and applications that use TCP-unfriendly protocols. We will show that analogous mechanisms are needed for sharing lossless networks. Currently, lossless Ethernet lacks such a hypervisor software capable of avoiding saturation trees and head-of-line blocking caused by misbehaving or buggy VMs. Effectively, this slows down the adoption of CEE in virtualized environments, despite the fact that CEE hardware is available from most commercial vendors.

Our objective is to design and implement a hypervisor that will prevent misbehaving or malicious flows, VMs or tenants from filling the lossless cluster with stalled packets, thus compromising tenant isolation. Our design is guided by the following key observations: *(i)* the hypervisor-based flow and congestion control partly overlaps with functions of the VM's transport; and *(ii)* in a lossless Ethernet, drops are rare events, hence the original reliability functions of the transport can be accelerated by the new Ethernet.

5.1.1. Contributions and Structure

The contributions of this chapter are as follows:

1. We introduce a slim hypervisor stack, named *zFabric*, optimized for lossless Ethernet. It avoids HOL blocking –and the ensuing interference between VMs– by managing the buffers between each vNIC communication pair through a VM-to-VM credit-based scheme. For the reliable delivery of both user data and credit messages, zFabric implements a thin reliability scheme on top of the lossless CEE hardware. The so created zFabric channels are shown in Figure 5.1, whereas the main differences between the standard vs. the proposed

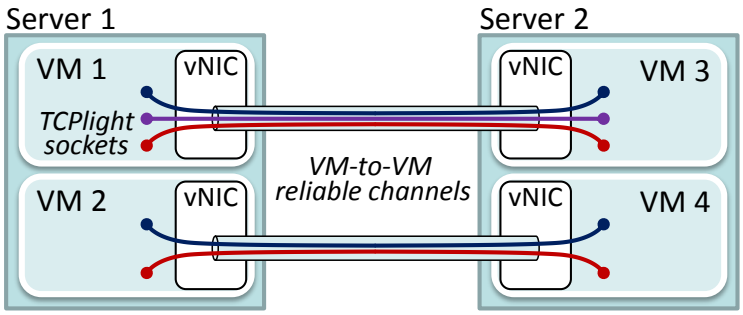


Figure 5.1.: zFabric channels. Segments of TCPlight flows are send through end-to-end reliable and flow-controlled VM-to-VM channels managed by the hypervisor.

stack are shown in Figure 5.2. A deployment of zFabric requires no changes to the applications and to the CEE hardware.

2. We propose *TCPlight*, a slim replacement for the TCP sockets. Although zFabric works with any user transport, optimal results are obtained with the newly introduced lightweight *TCPlight* socket, which is responsible for connection handling and data segmentation.

3. We build a working zFabric prototype and evaluate it using long throughput-bounded transfers and short latency sensitive flows. Section 5.4. zFabric: *(i)* increases the throughput of long VM-to-VM flows by up to *56%*; *(ii)* effectively solves CEE's HOL-blocking problem; *(iii)* enforces TCP-friendly *fairness* independent of the user transport type; *(iv)* achieves *7x - 14x* shorter completion times for partition-aggregate workloads; *(v)* while also lowering the CPU utilization up to *63%*.

The rest of this chapter is structured as follows: In Section 5.2 we detail our background and motivation. In Section 5.3 we present the architecture of our proposal, obtained by deconstructing the VM and hypervisor transports, and redistributing their core functions. We evaluate our zFabric implementation in Section 5.4. We discuss some practical deployment issues in Section 5.5 and present the related work in Section 5.6. Finally we conclude the chapter in Section 5.7.

5.2. Background and Motivation

5.2.1. Virtualized Stacks and Sharing the Datacenter with Noisy Neighbors

Literature [68, 108, 22, 56, 99, 74, 23, 88, 37, 110, 81] suggests that the current virtualized datacenter stacks are increasingly heavy, often sacrificing datapath performance in favor of advanced functionality on the control path. First, virtualization inserts additional networking layers in the hypervisor for routing, forwarding and

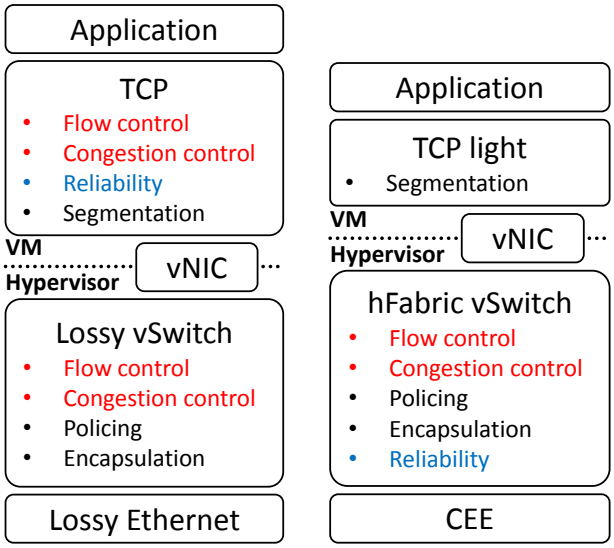
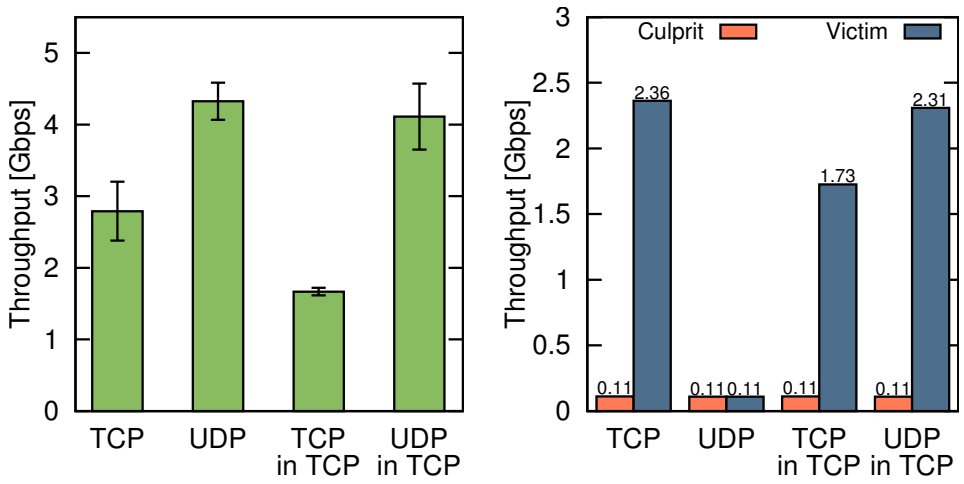


Figure 5.2.: **(a)** Current architecture: 'Fat' transport layer within the VM, with duplicated transport functions in the hypervisor. **(b)** Proposed zFabric architecture: 'Light' transport layer within the VM; all key transport functions moved to the hypervisor.

multiplexing of the VM traffic. Second, the same layers are often enriched with additional policy enforcing features. Revealing of the latency consequences, a ping test between two distinct physical servers in our testbed yields an average RTT of $27\mu s$. However, the average RTT between two VMs collocated on the *same* two physical machines is $221\mu s$. This order of magnitude increase in RTT, caused by the virtualization and the VM scheduling within the host OS, severely impacts the TCP performance for short flows.

As another manifestation of the 'heavy' virtualized stacks, Figure 5.3a compares the average throughputs of a 5GB flow using TCP and UDP between two VMs hosted on different physical servers connected by 10 Gbps. The experiment is repeated 30 times; the same connection is reused for all repetitions to avoid slow-start, while the hypervisors perform only basic packet forwarding. UDP reaches a 57% higher throughput than TCP. We attribute TCP's throughput penalty: *(i)* to the overhead of TCP acknowledgements, which pass through all the hypervisor layers, thus consuming CPU cycles that could be otherwise used for forwarding data segments; and *(ii)* to its more complex per packet processing as compared to UDP. Even if the typical path of a segment in TCP may be short, each segment has to undergo more checks to evaluate the large amount of possible TCP options and features.

Therefore, the more aggressive tenants of a virtualized datacenter (aka, 'noisy neighbors') have incentives to abandon TCP in favor of customized lighter protocols, derived from UDP [108, 68]. These typically TCP-unfriendly protocols can hog unfair bandwidth shares and harm TCP-based applications. To counteract the problems of misbehaving or malicious flows, VMs or tenants, recent solutions



(a) Single flow throughput.

(b) Flow interference.

Figure 5.3.: Throughput and head-of-line blocking issues (PFC enabled).

[68, 108, 22, 56, 99, 23, 74] stack up additional layers that perform flow and congestion control within the hypervisor. Therefore, we observe the rise of a *hypervisor transport* that replicates some functions of TCP, further increasing the per-packet processing overhead, while rendering the TCP-unfriendly transports fairer.

Furthermore, within the control plane of a virtualized datacenter, the hypervisor is responsible for the isolation of address spaces of different tenants, using encapsulation – see Overlay Virtual Networks, such as VXLAN [81], NetLord [88], and DOVE [37]. The increasing complexity of such operations performed on every flow, or even packet, significantly contributes to datapath performance penalties.

TCP offload techniques available in the host OS are not readily accessible from the guest OS. Allowing the guest OS to directly communicate with the hardware would break the security guarantees. The lack of standardization of these techniques precludes the migration of VMs between servers with different network adapters.

A summary of the current network stack architecture, as envisioned in several prior works [68, 108, 22, 56, 99, 74, 23, 88, 37, 110, 81], is shown in Figure 5.2a. The takeaway from this section is that the TCP stack, which was originally created and optimized to run directly over the network hardware, now runs over a new stack of layers responsible for virtualization, isolation, and encapsulation. Our main contribution is an alternative slimmer and more efficient –higher throughput, lower latency– software stack for lossless virtualized fabrics.

5.2.2. Emerging Lossless Fabrics

Traditionally Ethernet was lossy. Frames were dropped whenever a receive buffer had reached its capacity, under the generally accepted end-to-end assumption [106]

that an upper layer protocol such as TCP will take the corrective steps to recover. Such a lossy network does not properly meet the semantics of the converged datacenter applications such as Fibre Channel over Ethernet (FCoE) [10] or Remote Direct Memory Access (RDMA) over Ethernet [36].

This mismatch has been recently corrected in Converged Enhanced Ethernet (CEE), that segregates Ethernet frames into eight different hardware priorities. Each priority may be configured as either lossy or lossless. Within a lossless priority, Priority Flow Control (PFC) acts as the earlier 802.3x PAUSE, preventing buffer overflows in a hop-by-hop manner – except that a paused priority does not affect other priorities.

Prior work has demonstrated that besides enabling network convergence, lossless Ethernet clusters can improve the performance of soft real-time, scale-out applications, that harness big-data. In particular, lossless fabrics avoid TCP incast throughput collapse, and can reduce the completion times by up to an order of magnitude [98, 40].

Despite the potential improvements, PFC introduces head-of-line (HOL) blocking. While two priorities do not interfere, flows of the same priority can HOL-block each other. Obviously, the 8 priority levels of PFC cannot separate and isolate the potentially millions of active flows. For example, consider two flows that share a congested link in a lossless cluster, as in the setup from Figure 5.8a. The first flow, i.e., 'culprit', targets a busy destination that can only receive packets at a fraction of the link speed. The second flow, i.e., 'victim', targets an uncongested destination.

Figure 5.3b shows the measured throughputs. Using TCP, the culprit flow does not impact the victim flow: TCP adapts the transmission rate of the culprit flow to the slow receiver. UDP, however, lacks any flow or congestion control mechanism. Unable to proceed towards the blocked destination, the packets of the culprit flow monopolize the shared buffer space in the upstream switches. The net result is that the throughput of the victimized flow drops to the level of the culprit. Using a similar strategy, a malicious tenant could easily fill the lossless cluster with stalled packets, thus compromising bandwidth sharing and tenant isolation. Given the fact that the cluster administrator has no control over the code running within the VMs, our objective is to avoid such unfair situations by adding a *hypervisor transport for lossless cluster fabrics*.

5.2.3. TCP Tunnels

We aim to design an optimized hypervisor stack for virtualization of the emerging lossless fabrics. This stack should mitigate or eliminate HOL blocking and the ensuing interferences among tenants.

We first consider TCP tunnels [75]. The hypervisor captures all VM traffic and, based on its destination, encapsulates it into TCP flows. The destination hypervisor terminates the tunnel and delivers it to the targeted VM. Note that all the inter-VM flows will be using this tunnel, irrespective of their original transport.

In Figure 5.3b, we observe that encapsulating UDP flows into TCP tunnels avoids HOL blocking, but induces a throughput degradation as seen in Figure 5.3a. The throughput degradation is mild for UDP flows: i.e., 6.3%. However, for TCP flows, the degradation exceeds 40%, since the TCP layer is duplicated within the hypervisor. Furthermore, our results in Section 5.4 show that TCP tunnels deteriorate the completion times of latency-critical flows.

We observe that UDP in TCP obtains a 47% higher throughput than TCP alone. Both configurations run the same TCP protocol, but located at different levels of the virtualized stack. With TCP in the guest VM the ACKs must pass through all virtualization layers and therefore suffer from the VM scheduling delays revealed by the previous ping test in Section 5.2.1.

This result motivates us to move transport functions from VM to the hypervisor, where they can be handled more efficiently – e.g., by shortening the path of the control messages.

5.3. zFabric Architecture

We now present the architecture of the proposed lossless communication stack for virtualized datacenters. First, we will decompose the VM *and* hypervisor transports, and distribute their core functions between the upper and lower layer. Effectively, we will obtain a slim and efficient hypervisor networking stack, *zFabric*, that avoids HOL blocking in lossless Ethernet, and *TCPlight*, a lightweight TCP for lossless virtualized datacenters.

We shall assume a datacenter as a collection of virtualized physical servers, each hosting a set of virtual machines. The proposed zFabric is currently included in the hypervisor and intended for communications between the VMs on the same host as well as VMs located on different hosts interconnected by a physical network. Furthermore, we assume the interconnection network to be a flat 10Gbps layer-2 CEE fabric with PFC enabled on some, not necessarily all, priorities. Hence, with zFabric we target communications between VMs belonging to the same PFC domain. As we discuss, owing to the losslessness of the underlying fabric, zFabric imposes significantly less overhead compared to similar solutions [68, 108, 22, 56, 99, 23, 74] that focused on lossy networks. The latter solutions can be used for 'remote' flows, which leave this PFC domain, or for flows and services that run on lossy network priorities.

zFabric comprises two key mechanisms: *zCredit*, for flow and congestion control, which as explained in Section 5.2.1 is needed at the hypervisor level to enforce fairness and isolation between the VMs; and *zBridge*, which guarantees reliable host-to-host delivery, i.e., unique reception of in-order packets.

The reasons why we implement a reliable delivery within the hypervisor, and not rely on reliable user-level transports, are twofold. First, zCredit and some of the

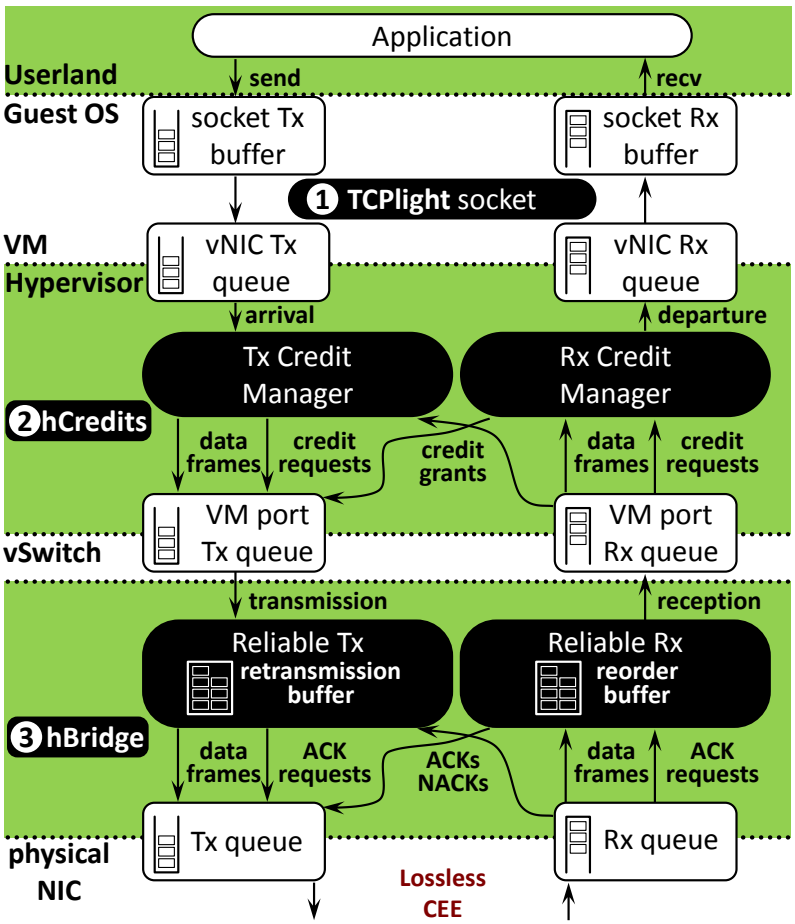


Figure 5.4.: The zFabric lossless virtualized communication stack. The proposed components are: **(1)** lightweight TCP socket (*TCPlight*), **(2)** credit-based flow control between vNICs (*zCredit*), and, **(3)** reliable overlay virtual network bridge (*zBridge*).

services that run in the VMs assume guaranteed delivery of messages. Although a network that employs PFC does not drop packets, we need to protect zCredit control messages (and user data) from occasional in-flight errors: zBridge implements a slim automatic repeat request scheme that minimizes the acknowledgement traffic to correct these errors. Second, by processing the acknowledgements in the zBridge, i.e., at the hypervisor's interface with the physical network, we reduce the acknowledgements processing overhead. Observe that zBridge is used only by flows that leave the local server – we ignore losses due to memory corruption or software bugs.

Although zFabric works with any user transport, it provides the opportunity to strip the VM's transport layer of the now redundant functions. Hence, our overall proposal encompasses a 'deconstructed' TCP socket, TCPlight, that runs in the VM on top of the zFabric hypervisor. TCPlight is left solely with the main functions of connection handling, data addressing/multiplexing, and data segmentation.

5.3.1. Packet Path Overview

Data packets travel between processes (applications) running inside the VMs. Along the datapath, packets are moved from queue to queue by different software and hardware entities. Next we describe this queuing system. Without loss of generality, we focus on the (i) lossless stack, whereby we (ii) assume a single active priority. Figure 5.4 shows in detail the packet path corresponding to this scenario.

The application starts by opening a TCPlight socket with the same interface and guarantees as a traditional TCP socket. The vNIC Tx queue forwards its packets to the zCredit layer. The zCredit transmission manager will enforce the SDN-required control policies, e.g., are source and destination VMs allowed to communicate. If yes, the appropriate amount of credits is requested from the destination VM hypervisor. When these credits are eventually granted, the data packets are forwarded to the vSwitch, which provides connectivity between the local VMs and also towards the physical layer, with the guarantees of no-drop and no misordering. Whenever the destination VM is local, the packets are directly forwarded.

Packets for remote VMs are taken over by the zBridge layer, which performs two functions. First, it acts as an overlay tunnel endpoint, performing encapsulation and decapsulation [88, 37]. This is needed to 'hide' the VMs from the physical fabric and to facilitate their automatic migration, creation and deletion, without configuration changes in the tables of the physical switches/routers. Second, it extends the lossless Ethernet fabric with a (thin) reliability layer. The encapsulation header includes a sequence number to identify losses and correct misordering events.

The encapsulated packets are enqueued in the physical NIC transmission queues and traverse the flow-controlled physical Ethernet network until delivered to the ingress port of the destination server. From there, they are processed by the zBridge that decapsulates them, performs reordering, if needed, and delivers them in-order to the zCredit unit. The latter updates the credit counters and delivers the packets to the vNIC Rx queue. From the vNIC, they are processed by the TCPlight socket and delivered to the application. With respect to the unmodified virtualized stack, our proposal requires only one extra memory copy operation.

5.3.2. Lightweight Socket: TCPlight

The transmission path in the Linux kernel has built-in flow control. If the user properly configures the size of the *Qdisc* associated with the virtual interface, no packets will be dropped. However, the reception path is exposed to packet drops. When a packet is enqueued by the hypervisor in the vNIC Rx queue, the guest OS receives an interrupt. This schedules the *NET Rx softirq* [5], which consumes packets from the vNIC queue and starts processing the packet headers. Packets destined to the local stack are enqueued in the destination socket RX buffer, based on their protocol and port number. Standard sockets are not flow-controlled, hence if the

application is slow in consuming data, the destination RX socket buffer can fill up and eventually segments are dropped within the stack. Thus we patched the Linux kernel such that when the destination RX socket occupancy reaches a threshold – i.e., one MTU below maximum size– the *softirq* is canceled and reception is paused. Once the process consumes data from the socket, reception is resumed. Head-of-line blocking within the kernel can be avoided using a garbage collector periodically removing stalling packets of crashed applications not implemented in our prototype.

The TCPlight socket API is identical with TCP. Hence, adapting existing applications from TCP to TCPlight socket is trivial, or automatic if, for example a wrapper library is used to intercept the *socket* system calls. The TCPlight sockets are primarily optimized for efficiency, low complexity and low latency operation made possible by shifting most transport functionalities to the hypervisor. We argue that TCPlight sockets are the only ones needed by future virtualized applications where the hypervisor provides the transport functionalities.

5.3.3. Congestion Management: zCredit

The zCredit layer within the zFabric is responsible for multiplexing the VM-to-VM traffic. The main objectives are: *(i)* to avoid HOL-blocking and saturation trees collapse, specific to lossless fabrics; and *(ii)* to fairly multiplex VM-to-VM traffic, enforcing TCP-friendly fairness across all VMs, irrespective of the protocols used by the applications.

Our mechanism cannot rely on loss for congestion signaling, as flow control prevents losses due to congestion. Instead, zCredit resolves congestion in an admission-oriented, request-grant scheme. Thus, zCredit also tolerates a cold startup delay of one RTT for the request-grant to complete. Observe that this latency overhead is compensated by the lack of slow start. Once granted, an uncontested flow can have its transmission window fully open.

When PFC is enabled, unconsumed packets can potentially block, causing saturation trees. For example, a crashed VM and does not consume packets from its associated vNIC RX queue. This queue fills up and the stalled packets consume the vSwitch buffer space and eventually spread to the buffers of the physical network. To avoid such a congestion propagation, any packet injected must be guaranteed to be consumed. However, with current hypervisors, a VM can inject an unbounded number of packets, either by opening multiple parallel TCP connections or by using TCP-unfriendly protocols such as UDP.

The zCredit design guarantees packet consumption at the receiver and a bounded number of in-flight packets. Before a packet is injected, buffer space is reserved at the receiving vNIC, by acquiring credits from the receiver. Effectively, if a destination is slow, a new packet will be injected towards it only after an old packet departs – a self-clocking property similar to that of TCP. In this case the self-clocking applies to the aggregate flow heading to a particular destination.

Algorithm 5.1: **Credit Management** – Transmission

```
globals: tx_occupancy, cdt_received, cdt_requested
on arrival(frame)
    tx_occupancy++
    wake_up forwarder
end
on forwarder
    if cdt_received > 0 then
        cdt_received--
        tx_occupancy--
        forward frame
    else
        if tx_occupancy > cdt_requested then
            delta ← tx_occupancy - cdt_requested
            request delta
            cdt_requested += delta
        else sleep
    end
end
on credit grant(num_credits)
    cdt_received += num_credits
    cdt_requested -= num_credits
    wake_up forwarder
end
```

As shown in Figure 5.4, zCredit operates at the hypervisor's vNIC backend level. The transmission side implements Algorithm 5.1. When a packet arrives from the sender VM, the transmitter schedules a forwarder to send the packet out. The forwarder checks if it has enough credits for the targeted VM. If not, it sends out a *credit request* and sleeps until a *credit grant* is received. When the grant is received, new packets can be injected in the network.

The receive side runs Algorithm 5.2. We assume that each vNIC can hold up to *RXQsize* packets in its receive queue. This is the fundamental parameter for the credit congestion management scheme. The VM-to-VM credit protocol makes sure that this queue never fills and never has to exert backpressure. Its size should be at least one RTT worth of packets, but for safety it can be much larger. On the other hand, the smaller the value is, the fewer packets will be in-flight towards a congested server, targeting any of its VMs. This opens the possibility to prevent filling up the (physical) switch-to-server output queue by configuring a low *RXQsize*. In our experiments we used *RXQsize*=512.

Upon a credit request, zCredit receive side will check if new credits can be granted. If yes, a credit grant is sent, else the request is enqueued in a waiting list associated with the targeted vNIC. To schedule fairly between multiple source VMs targeting the same destination VM, each vNIC maintains a separate waiting list per source VM. A work-conserving weighted-round-robin scheduler selects among them. As

Algorithm 5.2: **Credit Management** – Reception

```
globals: total_granted, waitQ
on departure(frame)
    total_granted--
    source ← waitQ.WRR()
    if source then
        cdt_req ← waitQ.dequeue(source)
        grant min(cdt_req, RXQsize - total_granted)
    end
end
on credit request(source,cdt_req)
    if total_granted < RXQsize then
        delta ← min(cdt_req, RXQsize - total_granted)
        grant delta
        total_granted += delta
    else waitQ.enqueue(source,cdt_req)
end
```

the data packets are consumed by the destination VM, new Rx vNIC buffer space is freed and credit requests from the associated waiting lists can be served.

This zCredit mechanism, running in the hypervisor, beyond the tenant's reach, strictly bounds the number of packets in-flight towards each vNIC, serializing the competing source VMs, according to the desired quality-of-service (QoS).

In our proof of concept implementation, credit requests and grants are standalone frames with a particular EtherType to allow the vNIC to filter them from regular data frames. The algorithms were implemented in QEMU-KVM [8] and Virtio vNIC [105].

Dealing with internal congestion. The zCredit mechanism eliminates the saturation trees that are rooted at the receiving vNICs. Furthermore, as mentioned above, by configuring a low *RXQsize*, zCredit can also resolve congestion at the switch-to-server ports of the physical network. In addition, zCredit helps to mitigate the effects of internal saturation trees: if the network is congested, both data *and* credit messages will be proportionally *paced* down, thus slowing the flow injection rates. However, zCredit is an end-point mechanism and cannot completely resolve saturation trees rooted at arbitrary points in the network. Nevertheless, similar to EyeQ [68], our solution benefits from the fact that most modern datacenter fabrics are based on full-bisection fat-trees. As shown in EyeQ, but also in previous papers on multi-stage fabrics [92, 35], in non-blocking topologies with fine-grained multi-path routing –e.g. packet-spraying or ECMP–, dealing with end-point congestion virtually eliminates saturation trees throughout the fabric.

Algorithm 5.3: **hBridge** – Reliability

```
globals: inflight, timer
on transmission(frame)
    if inflight >= BUF then sleep
    else
        mark every P frame with ACK request
        send frame
        inflight++
        reset timer
    end
end
on timer
    send ACK request
end
on ACK(N frames)
    inflight -= N
    if inflight = 0 then kill timer
end
on NACK(N frames)
    resend N frames
end
```

```
on reception(frame)
    if inorder then forward frame
    else send NACK for missing frames
end
on ACK request
    send ACK/NACK for received/missing frames
end
```

5.3.4. Reliability: zBridge

This layer provides reliable server-to-server channels over the physical network and is optimized for flow-controlled Ethernet. The activation of PFC does not guarantee by itself the reliable in-order delivery of frames. First, the frames can be lost due to in-flight corruption. Second, Ethernet does not guarantee in-order delivery. Improper configuration of link bundling schemes, transient routing errors, optimizations in the switch micro-architecture, etc., can all lead to out-of-order deliveries.

The reliability protocol is optimized to operate over flow-controlled Ethernet networks where frame corruptions and losses are rare events. Therefore, according to the design principle "make the common case fast, and the uncommon case correct", we keep the overhead of a correct transmission –the most likely outcome– as low as possible.

Unlike TCP, which aggregates reliability, flow control, and congestion management, we implemented the reliability as a distinct layer. This solution has three advantages. First, the reliability layer is redundant for local traffic, between VMs hosted on the same machine, since packets remain within the same system memory and

the vSwitch already guarantees reliable in-order delivery. A possible VM placement strategy could minimize the latency by collocating VM's that exchange large amounts of data. Second, we expect that in the future this feature will be accelerated in commodity hardware even for Ethernet. For example InfiniBand already provides guaranteed in order delivery in hardware; here the zBridge can be safely disabled. Third, all packets must be encapsulated before they reach the physical network [88, 37, 81, 110]. Here we propose to also use these encapsulation headers to provide reliability.

The reliability protocol is shown in Figure 5.4 and its operation is described in Algorithm 5.3. The scheme establishes initial sequence numbers through a 3-way handshake identical to TCP's. Each VM-to-VM flow has its own sequence number to allow equal-cost multi-path (ECMP) routing between servers. Each data frame is encapsulated in a reliability header containing a sequence number, an acknowledgement number, a checksum, and a flag field signaling the type of frame: i.e., *Data*, *Ack*, *Nack*, or *Ack Request*. The sender stores a copy of every sent frame in a retransmission buffer of fixed size BUF. To avoid throughput penalties, this buffer should be larger than the maximum bandwidth-delay product within the datacenter network. We avoid per frame *Acks*. The receiver sends back *Acks* only on explicit requests of the sender, through an *Ack Request*. The receiver answers to the *Ack requests* by *Ack*-ing or *Nack*-ing any frame that was received or missing. The *Nacks* carry a range of contiguous sequence numbers of the packets that have to be retransmitted.

The sender asks periodically for *Acks*, in order to free up the retransmission buffer. The *Ack request* are sent every P frames to avoid filling the buffer or after a timeout to avoid stalling the connection, whichever condition happens first. The receiver verifies the checksums and the sequence numbers of each data frame coming from the physical NIC. If the frames are not corrupted and in-order, they are decapsulated and handed to the vSwitch, which forwards them to the upper layers.

We can control the overhead of the acknowledgements by setting the P parameter and timeout. The only condition on the choice of the timeout is to be large enough to avoid injecting duplicate frames. In our experiments the timeout is set three orders of magnitude higher than the base RTT, and we send an *Ack request* every P=1024 frames. Lost *Nacks* and *Ack requests* are recovered through the same timeout.

For systems where a vNIC-to-vNIC pair uses a single routing path in each direction, we can perform the following optimization, to accelerate the retransmission of corrupted packets. When the receiver detects a gap in the sequence numbers, it transmits a *Nack* back to the sender, which retransmits the lost frame(s) from its retransmission buffer.

In a lossy network, TCP interprets any packet drop as an indication of congestion, as hardware failures are rare events. Hence, reducing the rate in response to packet drops in a lossy network is almost always a good choice. On the other hand, in a lossless datacenter network the opposite is true, i.e., drops are caused exclusively by

System Type	IBM System x3550 M4 Rack Servers
CPU	2x Intel Xeon E5-2690
Total cores	16
Clock speed [GHz]	2.90
Memory [GB]	96
Physical machines	6
VMs/machine	16
Data network	10G CEE
Physical switch	IBM RackSwitch G8264
Control network	1G Ethernet
Linux kernel	3.0.3 64-bit

Table 5.1.: Testbed configuration parameters.

hardware failures. Hence it is not necessary for the zBridge to reduce the injection rate upon detecting a loss.

To summarize, any application running on top of the zBridge layer will have the illusion of a reliable Ethernet network with low overhead.

5.4. Evaluation

In this section we evaluate the proposed TCPlight and zFabric architecture, outlined in the previous section, using a prototype implementation running on our hardware testbed.

5.4.1. Methodology and Testbed

This work involved changes in the kernel of the guest OS where the TCPlight sockets reside, in the host OS where the vSwitch runs and in the hypervisor software where the zCredit mechanism is implemented. Analytical models or simulations are not currently appropriate for evaluating the full system performance that is influenced by the OS scheduling, buffering, and CPU architecture. Therefore we rely on prototypes running on actual hardware.

We use two racks, each populated with Intel-based servers and a top-of-the-rack switch. The network is 10Gbps CEE, hence commodity lossless Ethernet. This network is used for all the test traffic subject to our measurements. In parallel, we use another 1Gbps control network to configure the data network and servers, to gather the statistics, and to start and kill the traffic generators - all without interfering with the data network. Figure 5.5 shows the testbed topology, whereas Table 5.1 lists the configuration details.

We compare several protocol combinations. We vary two components of the stack: the VM transport, used by the applications running within the guest OS, and the hypervisor transport. For the VM transport we use either the unmodified *TCP* socket or our *TCPlight* simplified socket. For the hypervisor transport we choose

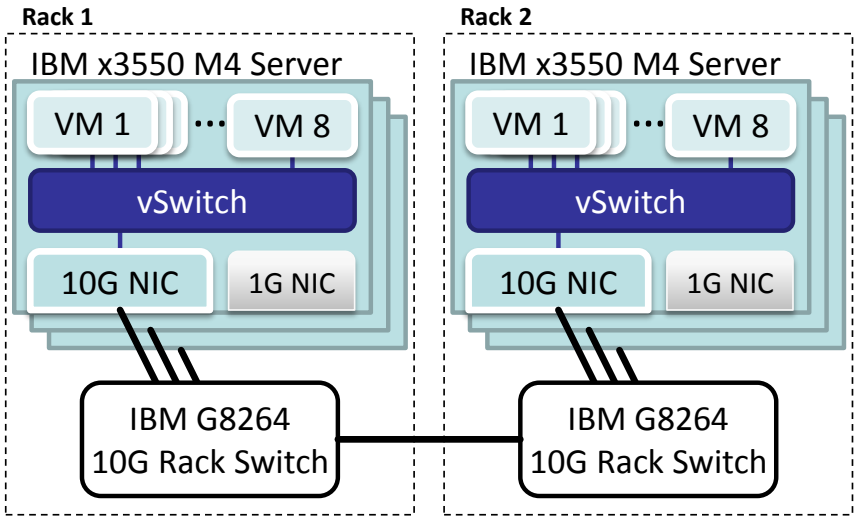


Figure 5.5.: Testbed setup.

between: *Lossy*, *PFC*, *TCP* and *zFabric*. *Lossy* stands for the traditional virtualized network stack, with a lossy virtual switch that performs no flow control and runs over a physical network without flow control (PFC disabled). In the *Lossy* setup both the vSwitch and the physical network can drop packets. *PFC* enables priority flow control in the physical network, paired with a lossless vSwitch. Both *Lossy* and *PFC* lack hypervisor flow control, making them both unsafe and allowing malicious or noisy tenants to negatively impact the well-behaved tenants. The *TCP* hypervisor configuration tunnels all the traffic between VMs through TCP connections over a lossless network. Finally, *zFabric* runs our proposed zCredit flow and congestion control layer and the zBridge reliability layer.

From a performance perspective, the overhead of *Lossy* and *PFC* is zero, while *TCP* is the heaviest hypervisor-based flow control. The solutions proposed for lossy networks in [68, 108, 22, 56, 99, 23, 74] are expected to have an overhead situated between the two extreme configurations that are our baselines. The proposed *TCP-light* socket only works when paired with *PFC*, *TCP* or *zFabric* hypervisors. On the other hand the unmodified *TCP* socket can be paired with any of the four hypervisors. The TCP congestion management algorithm in Linux is by default CUBIC [60].

Performance Metrics: For each experiment we measure throughputs and completion times at application level. All the data transfers are VM-to-VM. To gain additional insights into the HOL-blocking behaviors, we also measure the average throughput at $1s$ granularity. Finally, parsing `/proc/stat` in the host OS, before and after each experiment, on all servers involved, we compute the total *CPU utilization*, measured in seconds, and representing the total time the processor spent working, i.e., not idle.

On the servers and VMs involved we used a customized Debian-based distribution where we removed all the background services that could bias our CPU statistics. The benchmark application always executes the same code for all experiments thus

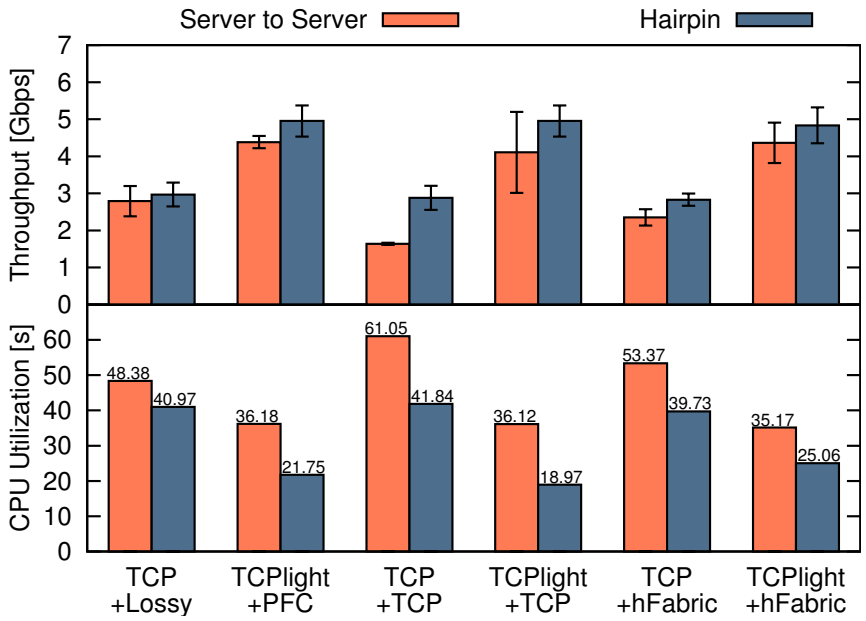


Figure 5.6.: Single flow experiment.

the same amount of work. Despite these precautions we cannot attribute the entire CPU utilization to the transport layer, meaning that the absolute value per se carries no useful information. A profiling of multiple kernels running in parallel to extract only the part of the CPU utilization that is caused by the transport is out of the scope of this work. Yet the relative variation of the total CPU utilization can be safely attributed to the only component that has changed between experiments, i.e., the transport layers of the VMs and hypervisors.

5.4.2. Single Flow Throughput

We begin with a single 5GB transfer between two VMs, repeated 30 times for improved confidence. The TCP sockets are reused between experiments. In this way only the first repetition can be potentially affected by TCP's slow start. We distinguish two VM placements: either on distinct servers, i.e., *server to server* traffic, or collocated, i.e., *hairpin* traffic.

Figure 5.6 shows both the average throughputs and CPU utilizations. The CPU utilizations are computed only once over all repetitions to reduce the CPU noise due to statistics gathering. *TCP+zFabric* shows a 15% throughput deterioration when compared to *TCP+Lossy*. However, *TCP+TCP* degrades the throughput even more – i.e., 41% with respect to *TCP+Lossy*. For hairpin traffic, *TCP+zFabric* produces only a 5% deterioration. *TCPlight+zFabric,* compared to *TCP+Lossy,* shows a *1.56x* throughput improvement for server-to-server traffic, and a *1.63x* CPU utilization reduction for hairpin traffic.

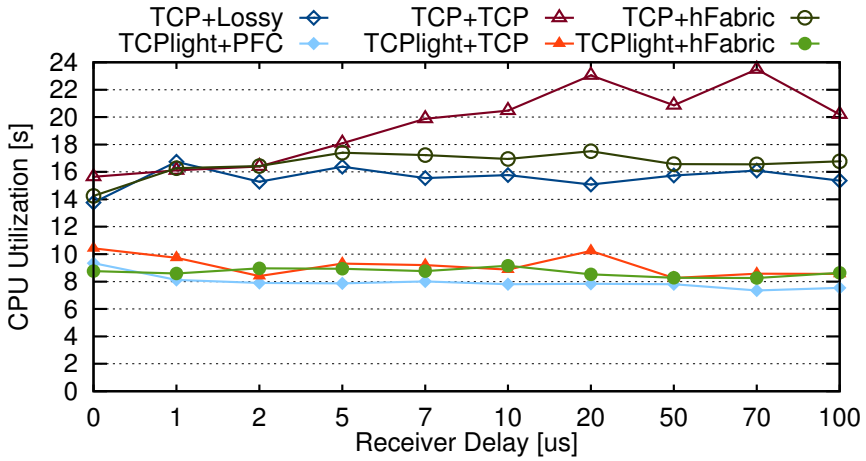
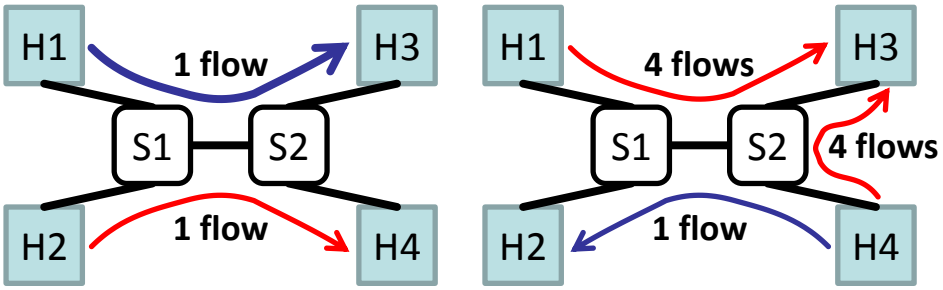


Figure 5.7.: Single flow CPU utilization against varying receiver delay to emulate decreasing throughputs as we go from left to right.

Next we consider 2GB transfers between two VMs located on two distinct servers. To emulate different application transfer speeds, the receiver sleeps for a configurable time every 100 received packets. Hence, varying the sleep time will modify the average throughput and the transfer completion time. We perform the experiment for the 6 combinations under scrutiny, and sleep times between 0 and 100μs. We repeat each experiment 20 times, reusing the sockets, and we measure the CPU utilization for the entire 20×2GB transfer. The flows achieve variable throughputs between 100Mbps and 4Gbps.

Figure 5.7 plots the CPU utilization against the receiver sleep time. Besides *TCP+TCP,* all other combinations show that the number of CPU cycles needed to perform a transfer does not depend on the speed of the transfer, being an *invariant* of the protocol itself.

The results are divided into two groups. The stacks using TCP in the VMs are heavier, consuming up to *2x* more CPU cycles than their TCPlight counterparts. This is caused in part by the per-segment acknowledgements that TCP uses, which add overhead in a virtualized network – these packets must traverse various software layers before reaching the VM's TCP. This is confirmed by the fact that *TCP+Lossy* requires *1.7x* more CPU cycles than *TCPlight+TCP*. On the other hand the TCP tunnels add an overhead of 45% to the VM's running unmodified TCP. The proposed zFabric adds a maximum 15% overhead –larger for TCP and smaller for TCPlight– when compared with the versions without hypervisor transports. On average, *TCPlight+zFabric* requires *1.8x* less CPU cycles than *TCP+Lossy* demonstrating the efficiency of our proposal.



(a) 2 flows. (b) 9 flows.

Figure 5.8.: HOL-blocking traffic scenarios. **(a)** Destination H4 is busy receiving packets at a fraction of the link speed. Ideally flow H1-H3 should not be affected by flow H2-H4. **(b)** 8 flows target the destination H3 that is overloaded. Ideally flow H4-H2 should not be affected by the congestion on link S2-H3.

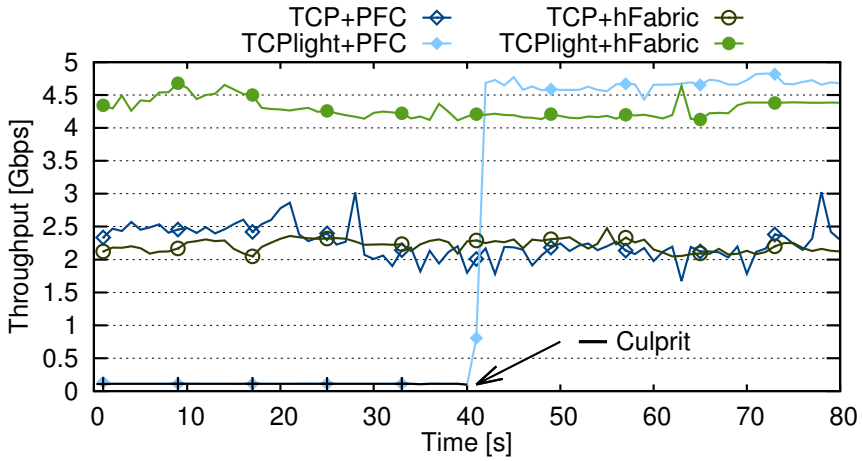


Figure 5.9.: Throughput evolution under the HOL-blocking scenario presented in Figure 5.8a.

5.4.3. Head-of-Line Blocking

We evaluate the resilience to HOL-blocking. This occurs when flows share one or more links while targeting congested and uncongested destinations. As before, we call these flows *culprit* and *victim* flows, respectively. If congestion management is absent, backlogged packets belonging to the culprit flows can hog the shared buffers, preventing the progress of victim flows.

We distinguish two types of congestion. (i) *Output-generated* hotspot: Packets are not consumed fast enough by the destination VM. E.g., a remote file transfer written to a slow medium at the destination. Here the bottleneck is the processing capacity at the destination. (ii) *Input-generated* hotspot: Traffic from multiple inputs target the same overwhelmed destination. An example is the Aggregate phase of a Partition-Aggregate job when many worker flows target the aggregator.

2 Flows (Output-Generated) We consider the scenario shown in Figure 5.8a.

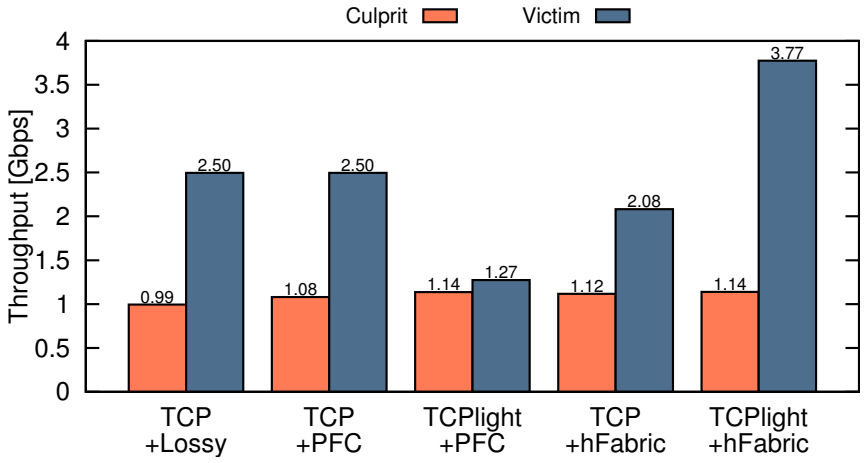


Figure 5.10.: Throughput under the HOL-blocking scenario presented in Figure 5.8b.

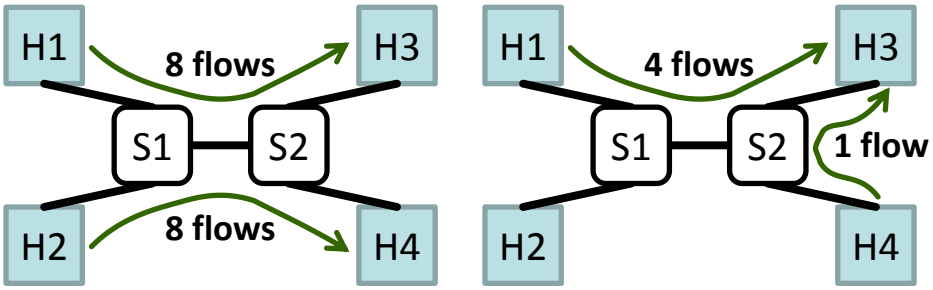
The destination of the H2-H4 flow (culprit) is slow and receives packets at maximum 100Mbps. The H1-H3 flow (victim) should not be obstructed by the packets of the culprit flow.

Figure 5.9 depicts the throughputs of both flows under different protocol stacks. All standard TCP based protocols and *TCPlight+zFabric* avoid the HOL-blocking. We observe that with *TCPlight+PFC*, which lacks hypervisor congestion management, the throughput of the victim flow drops to the level of the culprit flow –i.e., around 100Mbps– due to HOL-blocking. This is why we need the zCredit congestion management, which successfully resolves the blocking.

9 Flows (Input-Generated) Next we consider a more complex scenario depicted in Figure 5.8b. 8 flows target the destination H3: 4 originating from H1 and 4 from H4. The 4 flows from H4 to H3 share the link with a victim flow from H4 to H2. Ideally, the first 4 flows H4-H3 should get half of the bandwidth of the link H4-S2, whereas the other half should be available to the victim flow H4-H2. Due to HOL-blocking the victim throughput goes down to $\frac{1}{8}$ of the link bandwidth, the same level as the other 8 flows in the scenario. Figure 5.10 plots the average throughput of the culprit flows and victim flow. *TCPlight+PFC* suffers from HOL-blocking, whereas the credit-based congestion management solves the issue. Furthermore, *TCPlight+zFabric* produces a higher throughput than the standard TCP-based setups because of lower CPU utilization.

5.4.4. Fairness

We measure the fairness in link bandwidth sharing using Jain's fairness index [66]. We consider scenarios with TCP flows only, TCPlight flows only, and a mix of TCP and TCPlight flows, all with and without the zFabric hypervisor transport. The results are shown in Figure 5.12.



(a) 16 flows. (b) 5 flows.

Figure 5.11.: Fairness traffic scenarios. **(a)** Each flow should receive $\frac{1}{16}$ of the link bandwidth. **(b)** Each flow should receive $\frac{1}{5}$ of the link bandwidth.

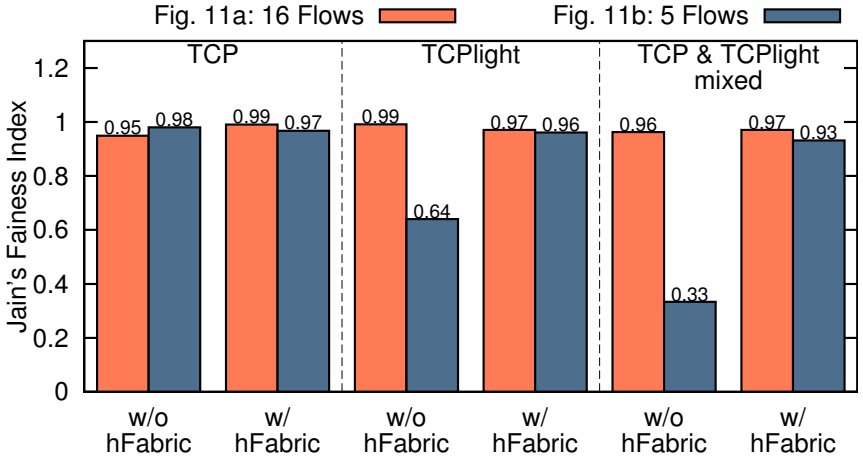


Figure 5.12.: Jain's fairness index for the scenarios shown in Figure 5.11.

16 Flows We look at the fairness of link sharing between 16 flows sharing a bottleneck as shown in Figure 5.11a. In this scenario both TCP and TCPlight reach an almost perfect fair share –i.e., index equal to 1– of the common link S1-S2. Our zFabric does not harm the fairness neither for TCP (left) nor for TCPlight (middle). Furthermore, it exhibits a good fairness index when TCPlight flows share the link with standard TCP flows (right), hence demonstrating TCP friendliness.

Parking-lot fairness Next we consider the parking-lot traffic scenario from Figure 5.11b. The flow H4-H3 shares the link S2-H3 with 4 others originating from H1. The switch S2 enforces fairness between its input ports hence the traffic from H4 might receive almost half of the bandwidth of the link S2-H3 while the other 4 flows from H1 must share the remaining bandwidth. This unfair allocation is seen in Figure 5.12 (middle) where TCPlight's fairness index drops to 0.64. Enabling the zFabric restores a fair allocation. Examining the TCP friendliness (right), the allocation between the competing 4 H1-H3 TCP flows and the H4-H3 TCPlight flow is unfair. As can be seen, enabling congestion management restores fairness.

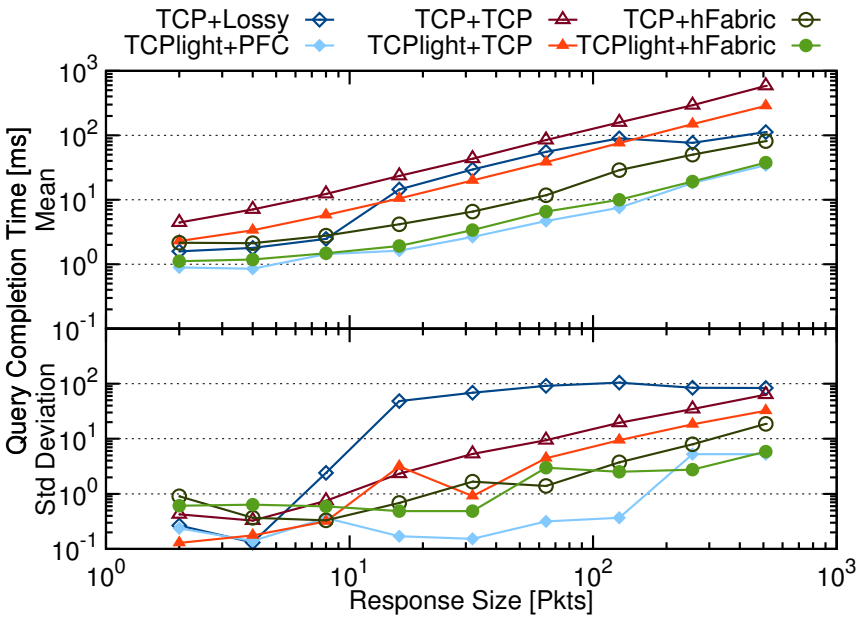


Figure 5.13.: Partition-Aggregate performance.

5.4.5. Partition-Aggregate Traffic

In this section, we use a two-tier Partition-Aggregate workload similar to [16, 119]. An aggregator receives a query from a client and distributes it to a set of workers, which then will reply back to the aggregator. When the workers send the data to the aggregator, packets from multiple flows will meet in the switch buffer causing TCP-incast in a lossy network harming the flow completion time [33, 44].

We vary the response size between 2 and 512 MTU-sized packets. The metric of interest is the query completion time measured from the arrival of the external query until the reception of all the workers' responses. We run this workload using 32 VMs distributed over 4 physical machines in our testbed.

Figure 5.13 shows the mean completion times and standard deviations measured over 10K repetitions of the same query. Figure 5.14 shows the performance gains over *TCP+TCP* using as metric the completion time. Replacing the TCP tunnels with zFabric tunnels delivers a reduction of up to *7x* of the mean completion time for 64 packets responses. Next, when running over zFabric, replacing TCP with TCPlight in the VM brings an *additional 2x* improvement – i.e., *14x* peak.

By running only TCPlight without any hypervisor transport, one obtains even shorter completion times. However, as shown in the previous experiments, the lack of a hypervisor flow control makes the fabric operation unsafe and unreliable. The proposed zFabric scheme also cuts the tail of flow completion times compared to the other configurations, as shown by smaller standard deviation values. Not shown are the CPU utilizations. We measured them and observed similar improvements as in the previous long flow scenarios.

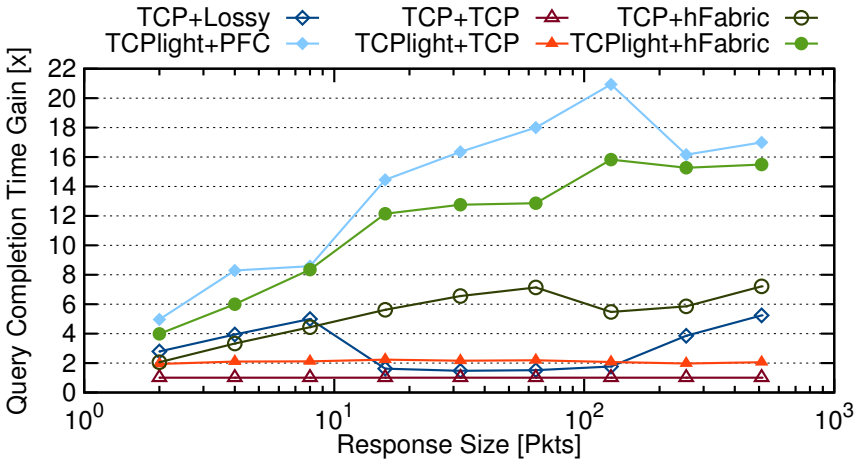


Figure 5.14.: Partition-Aggregate performance gains over the TCP+TCP configuration.

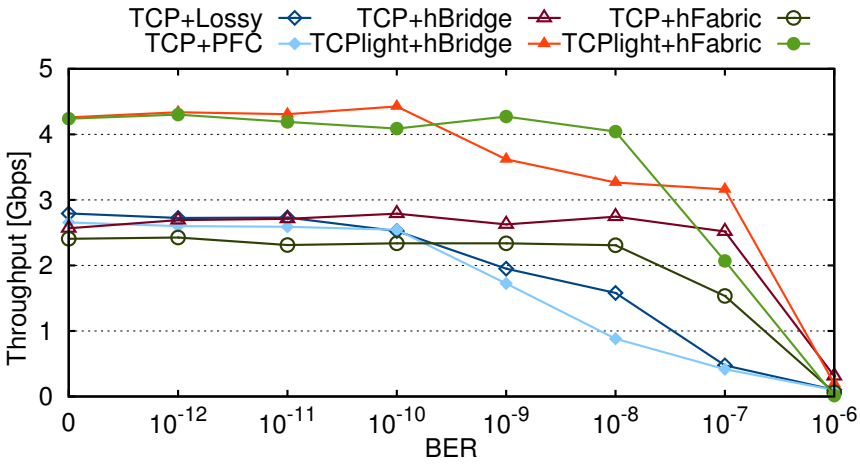


Figure 5.15.: Performance robustness under increasing link bit error rates.

5.4.6. Link Error Rate Influence

Finally we evaluate the behavior of zFabric under different bit error rates (BER). The physical links that we used in our testbed have a declared BER $< 10^{-12}$. To consider BERs ranging from 10^{-12} to 10^{-6}, we artificially emulate drops of corrupted frames by randomly discarding frames in the physical NIC driver at the receiving side. The scenario consists of a single 250GB flow running between two VMs located on different servers. The large enough size of the transfer guarantees that even for low error probabilities some frames will be discarded.

Figure 5.15 depicts the average throughput for different BER values. Corrupted frame losses on high reliability links with BERs between 10^{-12} and 10^{-10} have no impact on throughput, whereas BER values above 10^{-9} increasingly impact the performance. Especially TCP is affected, either when running over a lossy or over flow-controlled physical network. As with wireless and satellite channels, here the

vNIC	eth0	eth1
VM Transport	TCPlite/TCP/UDP	TCP/UDP
Hypervisor Stack	hFabric	lossy stack
PFC priority	lossless	lossy

Table 5.2.: Example deployment configuration.

TCP incorrectly interprets the drops as congestion signals reducing the injection rate, whereas in this scenario the drops are exclusively caused by an unreliable link. This tests prove the correctness of zFabric's reliability protocol.

5.5. Deployment Considerations

Here we address some of the practical implementation and deployment issues. These are likely to confront those willing to replicate our results or further develop the zFabric concepts.

What is zFabric's scope of application? The proposed credit-based hypervisor transport was designed and optimized specifically for intra-datacenter transfers using lossless Ethernet-based networks.

How would the zFabric be practically deployed? How invasive is it? In the first step, a practical zFabric deployment requires an update of the hypervisor software in all involved end-nodes. While arguably invasive, this is the minimal requirement for any hypervisor and SDN improvement. In the second step, the OS images of all the VMs involved can be patched with the TCPlight code. This step, however, is optional because all the legacy TCP stacks also run over zFabric, albeit non-optimally as we showed in Section 5.4. Essentially, no changes are required to the application structure in order to perform the migration from TCP to TCPlight sockets. Also no changes are needed to the existing network infrastructure, with the only constraint of PFC-compliant switches and network adapters, available from practically all commercial 10GigE vendors as of 2014.

What scope and size of deployment? We have tested zFabric in a limited 2-rack testbed. Although a large scale deployment in a large commercial datacenter may not be straightforward (see step 1 above), one could start by deploying it within small- to medium-scale performance optimized clusters, before committing to production datacenters.

Communication with remote hosts? zFabric does not preclude the use of lossy hypervisor stacks. E.g., each VM could be configured with two vNICs, one attached to the zFabric and the other to the default stack; here the zFabric should be natively mapped to a lossless PFC priority, whereas the default stack to a lossy priority. Table 5.2 shows possible protocols that can be used with each interface in this scenario. The second interface can be used for communications with any remote host on the Internet, using standard protocols.

5.6. Related Work

New hypervisor extensions were recently proposed to enforce the fair utilization of the virtualized network. Nearest related to our approach is EyeQ [68], which uses per VM rate limiters and end-to-end flow control between sources and destinations. Seawall [108] encapsulates the VM traffic within TCP-like tunnels. Oktopus [22] and SecondNet [56] use fixed bandwidth allocations between different tenants. NetShare [74] uses the QoS features available in routers, in association with a bandwidth allocator. FairCloud [99] explores different bandwidth allocation policies. The implications of the increased inter-tenants traffic within a virtualized datacenter are investigated in [23]; accordingly, the Hadrian network-sharing framework is proposed. Generally, these proposals increase the weight of the hypervisor stack, mostly by inserting new layers responsible for QoS and bandwidth allocation between TCP and the physical network. Instead, for zFabric we adopted the more radical –yet still dirty-slate– approach of relocating (most of the) TCP stack functions from VM to hypervisor. Our resulting transport can thus be orthogonally combined with any of the above proposals into a more efficient stack. Notably, the prior art above does not addresses the problems specific to the emerging lossless fabrics.

The datacenter research community increasingly focuses on the performance of horizontally-distributed online data-intensive workloads [33, 16, 17, 71, 115, 119, 122], while the industry promotes SDNs and virtualization, e.g., overlay virtual networks were proposed in [88, 118, 37, 81, 110, 24]. Network resource sharing issues for virtualized datacenters are studied in [90], whereas [67] studies a multi-datacenter setup. PVTCP [34] exposes the problems of the TCP stack running in a virtualized environment. The network virtualization performance in a large scale deployment is evaluated in [118].

A retransmission timeout reduction to alleviate TCP incast is proposed in [33, 117]. Other proposals achieve flow completion time reductions using new single-path [16, 17, 119, 115, 62] or multi-path [122, 62, 107, 15, 100, 61] transports. DeTail [122] enables Ethernet's PFC to prevent packet loss. pFabric [18, 19] re-evaluates the end-to-end argument and introduces a clean-slate transport stack resident in the end node, re-designed specifically for latency-sensitive datacenter applications. Our zFabric proposal, running over lossless fabrics, offers outstanding reductions of flow completion times in virtualized environments.

5.7. Discussion

We have argued that the current virtualized datacenter stacks are necessarily heavy, often sacrificing datapath performance in favor of advanced functionality on the control path. Independently, the recent introduction of Priority Flow Control has made lossless Ethernet fabrics a commodity. This opens new practical opportunities for simplification of the networking stack.

In this chapter we have contributed *zFabric*, a new hypervisor stack, optimized for intra-datacenter transfers over lossless Ethernet. By judiciously deconstructing and reallocating the transport functionalities, the zFabric manages the buffers between each virtual NIC pair through a VM-to-VM credit-based scheme. zFabric implements a slim reliability scheme, providing reliable server-to-server channels on top of the lossless CEE hardware. Although zFabric works with any traditional transport, optimal results have been shown here with the newly proposed lightweight *TCPlight* socket.

We have evaluated zFabric on a hardware testbed and shown significant improvements: *(i)* increases VM-to-VM throughput up to *56%*; *(ii)* solves lossless Ethernet's HOL-blocking; *(iii)* enforces TCP-friendly *fairness* independent of the VM transport type; *(iv) 7x - 14x* shorter flow completion times for partition-aggregate; *(v)* all the above while also lowering the CPU utilization up to *63%*. A practical deployment can be incremental and requires changes only to the hypervisor software – and optionally to the guest OS for further improvements. As limitations, the zFabric scheme is expected to operate within the domains of a performance optimized cluster. We plan to extend it to larger production-sized datacenters.

6. Conclusions

In this thesis we showed that it is possible to reduce the flow completion times of latency sensitive applications by avoiding packet drops in the virtualized networking stack. Furthermore, we proved that it is possible to simplify the heavy networking stack by moving functionality from the TCP stack in the guest OS to the hypervisor, and by exploiting the hardware link-level flow control from CEE. We summarize the work by answering our initial question from Chapter 1.

(Q1) What is the influence of CEE protocols on the completion time of TCP based applications?

In Chapter 2, we showed that PFC significantly improves TCP performance across all tested configurations and workloads, hence our recommendation to enable PFC whenever possible. The commercial workload completion time improved by 27% on average, and up to 91%. Scientific workloads showed higher gains by enabling PFC: 45% on average, and up to 92%.

Standard QCN, obtained mixed results, partly caused by its lack of adaptivity and fairness. On the other hand, we showed that the QCN rate limiters can be combined with VLAN-based routing and is possible to improve both performance and stability beyond the current state of the art routing in datacenters. Our evaluations showed that for HPC benchmarks, our proposed R^3C^2 routing scheme can be up to 98% faster than random routing, on average 40%.

(Q2) How do latency sensitive applications perform in a virtualized environment? What are the main performance gating factors of overlay virtual networks?

In Chapter 3 we confirmed that overlays diminish the performance of Partition-Aggregate and 3-Tier workloads. We showed that the increase in the completion time of the HTTP queries ranges from 1.5% up to 18.2%. The main performance gating factors are the encapsulation and discovery overhead. While the encapsulation overhead produces a fixed decrease of throughput, the discovery overhead is variable and influenced by the controller design and by the size of the cache used in every virtual switch.

(Q3) What is the cause of packet drops in virtualized networks? What is the performance penalty of the packet drops? How can they be avoided?

In Chapter 4 we showed that packet drops in the network stack are caused by the lack of flow control between different queues from the data path. We provided evidence that, for latency-sensitive virtualized datacenter applications, packet loss is a costly singularity in terms of performance. To remedy this situation, we have identified the origins of packet drops across the entire virtualized communication stack, and then designed and implemented a fully lossless virtual network prototype. Based on the experimental results, obtained using our prototype implementations and larger-scale simulations, we have demonstrated average flow completion time improvements of one order of magnitude.

(Q4) Can we design a lighter virtualized stack that improves performance of socket-based application, running on top of CEE hardware?

In Chapter 5 we showed that is possible to design a lighter stack by judiciously deconstructing and reallocating the transport functionalities. We introduced *zFabric*, a new hypervisor stack, optimized for intra-datacenter transfers over lossless Ethernet. We have evaluated zFabric on a hardware testbed and shown significant improvements: *(i)* increases VM-to-VM throughput up to *56%*; *(ii)* solves lossless Ethernet's HOL-blocking; *(iii)* enforces TCP-friendly fairness independent of the VM transport type; *(iv)* *7x - 14x* shorter flow completion times for partition-aggregate; *(v)* all the above while also lowering the CPU utilization up to *63%*.

Bibliography

[1] Arista 7500 Series Data Center Switch. http://www.aristanetworks.com/media/system/pdf/Datasheets/7500_Datasheet.pdf.

[2] Intel Many Integrated Core Architecture. http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html.

[3] Iperf. http://iperf.sourceforge.net.

[4] Linux Bridge. http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge.

[5] Linux NAPI. http://www.linuxfoundation.org/collaborate/workgroups/networking/napi.

[6] ns-3 Network Simulator. http://www.nsnam.org.

[7] Open vSwitch. http://openvswitch.org.

[8] QEMU-KVM. http://www.linux-kvm.org.

[9] RUBiS: Rice University Bidding System. http://rubis.ow2.org.

[10] Fabric convergence with lossless Ethernet and Fibre Channel over Ethernet (FCoE). http://www.bladenetwork.net/userfiles/file/PDFs/WP_Fabric_Convergence.pdf, 2008.

[11] 802.1Qau - Virtual Bridged Local Area Networks - Amendment: Congestion Notification, 2010.

[12] P802.1Qaz/D2.5 - Virtual Bridged Local Area Networks - Amendment: Enhanced Transmission Selection for Bandwidth Sharing Between Traffic Classes, 2011.

[13] P802.1Qbb/D2.3 - Virtual Bridged Local Area Networks - Amendment: Priority-based Flow Control, 2011.

[14] Al-Fares, M., Loukissas, A., and Vahdat, A. A Scalable, Commodity Data Center Network Architecture. In *Proc. ACM SIGCOMM 2008* (Seattle, WA, August 2008).

[15] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. NSDI 2010* (San Jose, CA, April 2010).

[16] Alizadeh, M., Greenberg, A., Maltz, D. A., et al. DCTCP: Efficient Packet Transport for the Commoditized Data Center. In *Proc. ACM SIGCOMM 2010* (New Delhi, India, August 2010).

[17] Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., Vahdat, A., and Yasuda, M. Less is More: Trading a little Bandwidth for Ultra-Low Latency in the Data Center. In *Proc. NSDI 2012* (San Jose, CA, April 2012).

[18] Alizadeh, M., Yang, S., Katti, S., McKeown, N., et al. Deconstructing Datacenter Packet Transport. In *Proc. HotNets 2012* (Redmond, WA, 2012).

[19] Alizadeh, M., Yang, S., Sharif, M., Katti, S., McKeown, N., Prabhakar, B., , and Shenker, S. pFabric: Minimal Near-Optimal Datacenter Transport. In *Proc. ACM SIGCOMM 2013* (Hong Kong, China, August 2013).

[20] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California at Berkeley, Berkeley, CA, Feb 2009.

[21] Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. The NAS Parallel Benchmarks. NASA Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA, March 1994.

[22] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. Towards Predictable Datacenter Networks. In *Proc. ACM SIGCOMM 2011* (Toronto, Canada, August 2011).

[23] Ballani, H., Jang, K., Karagiannis, T., Kim, C., Gunawardena, D., and O'Shea, G. Chatty Tenants and the Cloud Network Sharing Problem. In *Proc. NSDI 2013* (Lombard, IL, April 2013).

[24] Barabash, K., Cohen, R., Hadas, D., Jain, V., et al. A Case for Overlays in DCN Virtualization. In *Proc. DCCAVES'11* (San Francisco, CA, 2011).

[25] Baran, P. On Distributed Communications Networks. *IEEE Transactions on Communications 12*, 1 (March 1964), 1–9.

[26] Benson, T., Akella, A., and Maltz, D. A. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. Internet Measurement Conference (IMC 2010)* (Melbourne, Australia, November 2010).

[27] Benson, T., Anand, A., Akella, A., and Zhang, M. Understanding Data Center Traffic Characteristics. In *Proc. ACM SIGCOMM Workshop for*

Research on Enterprise Networks (WREN 2009) (Barcelona, Spain, August 2009).

[28] Birke, R., Crisan, D., Barabash, K., Levin, A., DeCusatis, C., Minkenberg, C., and Gusat, M. Partition/Aggregate in Commodity 10G Ethernet Software-Defined Networking. In *Proc. HPSR 2012* (Belgrade, Serbia, June 2012).

[29] Blumenthal, M. S., and Clark, D. D. Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave New World. *ACM Transactions on Internet Technology 1*, 1 (August 2001), 70–109.

[30] Brakmo, L., O'Malley, S., and Peterson, L. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proc. ACM SIGCOMM 1994* (London, UK, August 1994).

[31] Briscoe, B. Tunnelling of Explicit Congestion Notification. RFC 6040, IETF, November 2010.

[32] Bullot, H., Cottrell, R. L., and Hughes-Jones, R. Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks. *Journal of Grid Computing 1*, 4 (December 2003), 345–359.

[33] Chen, Y., Griffith, R., Liu, J., Katz, R. H., and Joseph, A. D. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *Proc. WREN 2009* (Barcelona, Spain, August 2009).

[34] Cheng, L., Wang, C.-L., and Lau, F. C. M. PVTCP: Towards Practical and Effective Congestion Control in Virtualized Datacenters. In *Proc. IEEE International Conference on Network Protocols (ICNP)* (Gottingen, Germany, October 2013).

[35] Chrysos, N., and Katevenis, M. Scheduling in Non-Blocking, Buffered, Three-Stage Switching Fabrics. In *Proc. 25th Conference on Computer Communications (INFOCOM 2006)* (Barcelona, Spain, April 2006).

[36] Cohen, D., Talpey, T., Kanevsky, A., et al. Remote Direct Memory Access over the Converged Enhanced Ethernet Fabric: Evaluating the Options. In *Proc. HOTI 2009* (New York, NY, August 2009).

[37] Cohen, R., Barabash, K., Rochwerger, B., Schour, L., Crisan, D., Birke, R., Minkenberg, C., Gusat, M., et al. An Intent-based Approach for Network Virtualization. In *Proc. IFIP/IEEE IM 2013* (Ghent, Belgium, 2013).

[38] Crisan, D., Anghel, A. S., Birke, R., Minkenberg, C., and Gusat, M. Short and Fat: TCP Performance in CEE Datacenter Networks. In *Proc. HOTI 2011* (Santa Clara, CA, August 2011).

[39] Crisan, D., Birke, R., Chrysos, N., and Gusat, M. How Elastic is Your Virtualized Datacenter Fabric? In *Proc. INA-OCMC 2013* (Berlin, Germany, January 2013).

[40] Crisan, D., Birke, R., Cressier, G., Minkenberg, C., and Gusat, M. Got Loss? Get zOVN! In *Proc. ACM SIGCOMM 2013* (Hong Kong, China, August 2013).

[41] Dally, W., and Towles, B. *Principles and Practices of Interconnection Networks, Chapter 13*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2003.

[42] Dean, J., and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. 6th Symposium on Operating System Design and Implementation (OSDI 2004)* (San Francisco, CA, December 2004).

[43] Denzel, W. E., Li, J., Walker, P., and Jin, Y. A Framework for End-to-end Simulation of High performance Computing Systems. In *Proc. 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008)* (Marseille, France, March 2008).

[44] Devkota, P., and Reddy, A. L. N. Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers. In *Proc. 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2010)* (Miami Beach, FL, August 2010).

[45] Duato, J., Yalamanchili, S., and Ni, L. *Interconnection Networks. An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2004.

[46] Dukkipati, N., and McKeown, N. Why Flow-Completion Time is the Right Metric for Congestion Control. *ACM SIGCOMM CCR 36*, 1 (January 2006), 59–62.

[47] Flich, J., Malumbres, M. P., López, P., and Duato, J. Improving Routing Performance in Myrinet Networks. In *Proc. 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)* (Cancun, Mexico, May 2000).

[48] Floyd, S., Henderson, T., and Gurtov, A. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, IETF, April 2004.

[49] Floyd, S., and Jacobson, V. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking 1*, 4 (August 1993), 397–413.

[50] Franco, D., Garcés, I., and Luque, E. A New Method to Make Communication Latency Uniform: Distributed Routing Balancing. In *Proc. International Conference on Supercomputing* (Rhodes, Greece, June 1999).

[51] Geoffray, P., and Hoefler, T. Adaptive Routing Strategies for Modern High Performance Networks. In *Proc. 16th Symposium on High Performance Interconnects (HOTI 2008)* (Stanford, CA, August 2008).

[52] Gilabert, F., Gómez, M. E., López, P., and Duato, J. On the Influence of the Selection Function on the Performance of Fat-Trees. In *Proc. 12th International Euro-Par Conference* (Dresden, Germany, August 2006).

[53] Gómez, C., Gilabert, F., Gómez, M. E., López, P., and Duato, J. Deterministic versus Adaptive Routing in Fat-Trees. In *Proc. 21st International Parallel and Distributed Processing Symposium* (Long Beach, CA, March 2007).

[54] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., and Pat, P. VL2: A Scalable and Flexible Data Center Network. In *Proc. ACM SIGCOMM 2009* (Barcelona, Spain, August 2009).

[55] Grover, H., Rao, D., Farinacci, D., and Moreno, V. Overlay Transport Virtualization. Internet draft, IETF, July 2011.

[56] Guo, C., Lu, G., Wang, H., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y. SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. In *Proc. 6th International Conference on emerging Networking EXperiments and Technologies (CoNEXT 2010)* (Philadelphia, PA, November 2010).

[57] Gusat, M., Birke, R., and Minkenberg, C. Delay-based Cloud Congestion Control. In *Proc. IEEE GLOBECOM 2009 Global Communications Conference* (Honolulu, HI, December 2009).

[58] Gusat, M., Crisan, D., Minkenberg, C., and DeCusatis, C. R3C2: Reactive Route and Rate Control for CEE. In *Proc. HOTI 2010* (Mountain View, CA, August 2010).

[59] Gusat, M., Minkenberg, C., and Luijten, R. Extended Ethernet Congestion Management (E2CM): Per Path ECM - A Hybrid Proposal. http://ieee802.org/1/files/public/docs2007/au-sim-IBM-ZRL-E2CM-proposal-r1.09.pdf, March 2007.

[60] Ha, S., Rhee, I., and Xu, L. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review 42*, 5 (July 2008), 64–74.

[61] Han, H., Shakkottai, S., Hollot, C. V., Srikant, R., and Towsley, D. Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet. *IEEE/ACM Transactions on Networking 14*, 6 (December 2006), 1260–1271.

[62] Hong, C.-Y., Caesar, M., and Godfrey, P. B. Finishing Flows Quickly with Preemptive Scheduling. In *Proc. ACM SIGCOMM 2012* (Helsinky, Finland, 2012).

[63] Hopps, C. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, IETF, November 2000.

[64] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *Proc. European Conference on Computer Systems (EuroSys 2007)* (Lisbon, Portugal, March 2007).

[65] Jacobson, V. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM 1988* (Stanford, CA, August 1988).

[66] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons Inc., New York, NY, 1991.

[67] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. B4: Experience with a Globally-Deployed Software Defined WAN. In *Proc. ACM SIGCOMM 2013* (Hong Kong, China, August 2013).

[68] Jeyakumar, V., Alizadeh, M., Mazieres, D., Prabhakar, B., Kim, C., and Greenberg, A. EyeQ: Practical Network Performance Isolation at the Edge. In *Proc. NSDI 2013* (Lombard, IL, April 2013).

[69] Johnson, G., Kerbyson, D. J., and Lang, M. Optimization of InfiniBand for Scientific Applications. In *Proc. 22nd International Parallel and Distributed Processing Symposium* (Miami, FL, April 2008).

[70] Kabbani, A., Alizadeh, M., Yasuda, M., Pan, R., and Prabhakar, B. AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenanted Data Centers. In *Proc. 18th Symposium on High-Performance Interconnects (HOTI 2010)* (Mountain View, CA, August 2010).

[71] Kandula, S., Katabi, D., Sinha, S., and Berger, A. Dynamic Load Balancing Without Packet Reordering. *ACM SIGCOMM Computer Communication Review 37*, 2 (April 2007), 53–62.

[72] Kandula, S., Sengupta, S., Greenberg, A., Patel, P., and Chaiken, R. The Nature of Datacenter Traffic: Measurements & Analysis. In *Proc. Internet Measurement Conference (IMC 2009)* (Chicago, IL, November 2009).

[73] Kudoh, T., Tezuka, H., Matsuda, M., Kodama, Y., Tatebe, O., and Sekiguchi, S. VLAN-based Routing: Multi-path L2 Ethernet Network for HPC Clusters. In *Proc. 2004 IEEE International Conference on Cluster Computing (Cluster 2004)* (San Diego, CA, September 2004).

[74] Lam, V. T., Radhakrishnan, S., Pan, R., Vahdat, A., and Varghese, G. NetShare and Stochastic NetShare: Predictable Bandwidth Allocation for Data Centers. *ACM SIGCOMM Computer Communication Review 42*, 3 (July 2012), 6–11.

[75] Lee, B., Balan, R., Jacob, L., Seah, W., and Ananda, A. Avoiding congestion collapse on the Internet using TCP tunnels. *Computer Networks 39* (2002), 207–219.

[76] Leiserson, C., Abuhamdeh, Z. S., Douglas, D. C., Feynman, C. R., Ganmukhi, M. N., Hill, J. V., Hillis, W. D., Kuszmaul, B. C., Pierre, M. A. S., Wells, D. S., Wong, M. C., Yang, S., and Zak, R. The Network Architecture of the Connection Machine CM-5. In *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures* (San Diego, CA, June 1992).

[77] Li, Y.-T., Leith, D., and Shorten, R. N. Experimental Evaluation of TCP Protocols for High-Speed Networks. *IEEE/ACM Transactions on Networking 15*, 5 (October 2007), 1109–1122.

[78] Lin, X.-Y., Chung, Y.-C., and Huang, T.-Y. A Multiple LID Routing Scheme for Fat-Tree-Based InfiniBand Networks. In *Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)* (Santa Fe, NM, April 2004).

[79] Lu, Y., Pan, R., Prabhakar, B., Bergamasco, D., Alaria, V., and Baldini, A. Congestion Control in Networks with No Congestion Drops. In *Proc. 44th Annual Allerton Conference on Communication, Control, and Computing* (Monticello, IL, September 2006).

[80] Lugones, D., Franco, D., and Luque, E. Dynamic and Distributed Multipath Routing Policy For High-Speed Cluster Networks. In *Proc. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2009)* (Shanghai, China, May 2009).

[81] Mahalingam, M., Dutt, D., Duda, K., et al. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. Internet draft, IETF, August 2011.

[82] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., et al. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (April 2008), 69–74.

[83] Minkenberg, C., and Gusat, M. Congestion Management for 10G Ethernet. In *Proc. 2nd Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip (INA-OCMC 2008)* (Goteborg, Sweden, January 2008).

[84] Minkenberg, C., Gusat, M., and Rodriguez, G. Adaptive Routing in Data Center Bridges. In *Proc. 17th Symposium on High-Performance Interconnects (HOTI 2009)* (New York, NY, August 2009).

[85] Minkenberg, C., and Rodriguez, G. Trace-driven Co-simulation of High-Performance Computing Systems using OMNeT++. In *Proc. 2nd SIMUTools International Workshop on OMNeT++* (Rome, Italy, March 2009).

[86] Minkenberg, C., Scicchitano, A., and Gusat, M. Adaptive routing for Convergence Enhanced Ethernet. In *Proc. 2009 International Workshop on High-Performance Switching and Routing (HPSR 2009)* (Paris, France, June 2009).

[87] Miura, S., Boku, T., Sato, M., Takahashi, D., and Okamoto, T. Low-cost High-bandwidth Tree Network for PC Clusters based on Tagged-VLAN Technology. In *Proc. 8th International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN 2005)* (Las Vegas, NV, December 2005).

[88] Mudigonda, J., Yalagandula, P., Mogul, J. C., et al. NetLord: A Scalable Multi-Tenant Network Architecture for Virtualized Datacenters. In *Proc. ACM SIGCOMM 2011* (Toronto, Canada, 2011).

[89] Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *Proc. ACM SIGCOMM 2009* (Barcelona, Spain, August 2009).

[90] Novakovic, D., Vasic, N., Novakovic, S., Kostic, D., and Bianchini, R. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *Proc. USENIX ATC 2013* (San Jose, CA, June 2013).

[91] Öhring, S. R., Ibel, M., Das, S. K., and Kumar, M. On Generalized Fat Trees. In *Proc. 9th International Parallel Processing Symposium (IPPS 1995)* (Santa Barbara, CA, April 1995).

[92] Pappu, P., Parwatikar, J., Turner, J., and Wong, K. Distributed Queueing in Scalable High Performance Routers. In *Proc. 22rd Conference on Computer Communications (INFOCOM 2003)* (San Francisco, CA, April 2003).

[93] Petrini, F., and Vanneschi, M. A Comparison of Wormhole-Routed Interconnection Networks. In *Proc. 3rd International Conference on Computer Science and Informatics* (Research Triangle Park, NC, March 1997).

[94] Petrini, F., and Vanneschi, M. k-ary n-trees: High Performance Networks for Massively Parallel Architectures. In *Proc. 11th International Parallel Processing Symposium (IPPS 1997)* (Geneva, Switzerland, April 1997).

[95] Pfaff, B., Lantz, B., Heller, B., Barker, C., et al. OpenFlow Switch Specification Version 1.1.0. Specification, Stanford University, February 2011.

[96] Pfister, G., and Kumar, V. The Onset of Hotspot Contention. In *Proc. International Conference in Parallel Processing (ICPP 1986)* (University Park, PA, August 1986).

[97] Pfister, G., and Norton, V. Hot Spot Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computers C-34*, 10 (October 1985), 943–948.

[98] Phanishayee, A., Krevat, E., et al. Measurement and Analysis of TCP Throughput Collapse in Cluster-Based Storage Systems. In *Proc. 6th USENIX Conference on File and Storage Technologies (FAST 2008)* (San Jose, CA, February 2008).

[99] Popa, L., Kumar, G., Chowdhury, M., Krishnamurthy, A., Ratnasamy, S., and Stoica, I. FairCloud: Sharing the Network in Cloud Computing. In *Proc. ACM SIGCOMM 2012* (Helsinky, Finland, 2012).

[100] Raiciu, C., Barre, S., and Pluntke, C. Improving Datacenter Performance and Robustness with Multipath TCP. In *Proc. ACM SIGCOMM 2011* (Toronto, Canada, August 2011).

[101] Rizzo, L. netmap: A Novel Framework for Fast Packet I/O. In *Proc. USENIX ATC 2012* (Boston, MA, 2012).

[102] Rizzo, L., and Lettieri, G. VALE, a Switched Ethernet for Virtual Machines. In *Proc. CoNEXT 2012* (Nice, France, December 2012).

[103] Rodriguez, G., Beivide, R., Minkenberg, C., Labarta, J., and Valero, M. Exploring Pattern-aware Routing in Generalized Fat Tree Networks. In *Proc. 23rd International Conference on Supercomputing (ICS 2009)* (Yorktown Heights, NY, June 2009).

[104] Rodriguez, G., Minkenberg, C., Beivide, R., Luijten, R. P., Labarta, J., and Valero, M. Oblivious Routing Schemes in Extended Generalized Fat Tree Networks. In *Proc. 2009 Workshop on High Performance Interconnects for Distributed Computing (HPI-DC 2009)* (New Orleans, LA, August 2009).

[105] Russell, R. virtio: Towards a De-Facto Standard For Virtual I/O Devices. *ACM SIGOPS Operating System Review 42*, 5 (July 2008), 95–103.

[106] Saltzer, J. H., Reed, D. P., and Clark, D. D. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems 2*, 4 (November 1984), 277–288.

[107] Scharf, M., and Banniza, T. MCTCP: A Multipath Transport Shim Layer. In *Proc. IEEE GLOBECOM 2011* (Houston, TX, December 2011).

[108] Shieh, A., Kandula, S., Greenberg, A., Kim, C., and Saha, B. Sharing the Data Center Network. In *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2011)* (Boston, MA, April 2011).

[109] Shurman, E., and Brutlag, J. The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search. http://velocityconf.com/velocity2009/public/schedule/detail/8523, June 2009.

[110] Sridharan, M., Duda, K., Ganga, I., Greenberg, A., et al. NVGRE: Network Virtualization using Generic Routing Encapsulation. Internet draft, IETF, September 2011.

[111] Stewart, R. R., Tuxen, M., and Neville-Neil, G. V. An Investigation into Data Center Congestion with ECN. In *Proc. 2011 Technical BSD Conference (BSDCan 2011)* (Ottawa, Canada, May 2011).

[112] Tan, K., Song, J., Zhang, Q., and Sridharan, M. A Compound TCP Approach for High-speed and Long Distance Networks. In *Proc. 25th Conference on Computer Communications (INFOCOM 2006)* (Barcelona, Spain, April 2006).

[113] Thaler, D., and Hopps, C. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991, IETF, November 2000.

[114] Valiant, L. G., and Brebner, G. J. Universal Schemes for Parallel Communication. In *Proc. 13th annual ACM Symposium on Theory of Computing* (Milwaukee, WI, May 1981).

[115] Vamanan, B., Hasan, J., and Vijaykumar, T. N. Deadline-Aware Datacenter TCP (D2TCP). In *Proc. ACM SIGCOMM 2012* (Helsinki, Finland, 2012).

[116] Varga, A. The OMNeT++ Discrete Event Simulation System. In *Proc. European Simulation Multiconference (ESM 2001)* (Prague, Czech Republic, June 2001).

[117] Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E., Andersen, D. G., Ganger, G. R., Gibson, G. A., and Mueller, B. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In *Proc. ACM SIGCOMM 2009* (Barcelona, Spain, 2009).

[118] Wang, G., and Ng, T. S. E. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *Proc. INFOCOM 2010* (San Diego, CA, March 2010).

[119] Wilson, C., Ballani, H., Karagiannis, T., and Rowstron, A. Better Never than Late: Meeting Deadlines in Datacenter Networks. In *Proc. ACM SIGCOMM 2011* (Toronto, Canada, August 2011).

[120] Xu, L., Harfoush, K., and Rhee, I. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *Proc. 23rd Conference on Computer Communications (INFOCOM 2004)* (Hong Kong, China, March 2004).

[121] Zahavi, E., Johnson, G., Kerbyson, D. J., and Lang, M. Optimized InfiniBand Fat-tree Routing for Shift All-to-All Communication Patterns. In *Proc. International Supercomputing Conference (ISC 2007)* (Dresden, Germany, November 2007).

[122] Zats, D., Das, T., Mohan, P., Borthakur, D., and Katz, R. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *Proc. ACM SIGCOMM 2012* (Helsinky, Finland, August 2012).

[123] Zhang, Y., and Ansari, N. On mitigating TCP Incast in Data Center Networks. In *Proc. 30th Conference on Computer Communications (INFOCOM 2011)* (Shanghai, China, April 2011).

List of Publications

The following list contains the publications which constitute the basis of this thesis. The corresponding chapter of the thesis is given in brackets.

Daniel Crisan, Andreea Anghel, Robert Birke, Cyriel Minkenberg, Mitch Gusat. *Short and Fat: TCP Performance in CEE Datacenter Networks.* In Proceedings of the 19th Annual Symposium on High-Performance Interconnects (HOTI 2011), Santa Clara, CA, August 2011, pages 43–50. (Chapter 2)

Mitch Gusat, Daniel Crisan, Cyriel Minkenberg, Casimer DeCusatis. *R3C2: Reactive Route & Rate Control for CEE.* In Proceedings of the 18th Annual Symposium on High-Performance Interconnects (HOTI 2010), Mountain View, CA, August 2010, pages 50–57. (Chapter 2)

Daniel Crisan, Robert Birke, Katherine Barabash, Rami Cohen, Mitch Gusat. *Datacenter Applications in Virtualized Networks: A Cross-layer Performance Study.* In IEEE Journal on Selected Areas in Communications (JSAC), January 2014, Volume 32, Issue 1, pages 77–87. (Chapter 3)

Daniel Crisan, Robert Birke, Nikolaos Chrysos, Mitch Gusat. *How Elastic is Your Virtualized Datacenter Fabric?* In Proceedings of 7th International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip (INA-OCMC 2013), Berlin, Germany, January 2013, pages 17–20. (Chapter 3)

Robert Birke, Daniel Crisan, Katherine Barabash, Anna Levin, Casimer DeCusatis, Cyriel Minkenberg, Mitch Gusat. *Partition/Aggregate in Commodity 10G Ethernet Software-Defined Networking.* In Proceedings of the 13th International Conference on High Performance Switching and Routing (HPSR 2012), Belgrade, Serbia, June 2012, pages 7–14. (Chapter 3)

Daniel Crisan, Robert Birke, Gilles Cressier, Cyriel Minkenberg, Mitch Gusat. *Got Loss? Get zOVN!* In Proceedings of ACM Conference on Data Communication (SIGCOMM 2013), Hong Kong, China, August 2013, pages 423–434. (Chapter 4)

Daniel Crisan, Robert Birke, Nikolaos Chrysos, Cyriel Minkenberg, Mitch Gusat. *zFabric: How to Virtualize Lossless Ethernet?* to appear In Proceedings of IEEE International Conference on Cluster Computing (Cluster 2014), Madrid, Spain, September 2014, pages 75–83. (Chapter 5)

The following list contains publications that are *not* covered in this thesis.

Rami Cohen, Katherine Barabash, Benny Rochwerger, Liran Schour, Daniel Crisan, Robert Birke, Cyriel Minkenberg, Mitchell Gusat, Renato Recio, Vinit Jain. *An Intent-based Approach for Network Virtualization.* In Proceedings of 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, May 2013, pages 42–50.

Daniel Crisan, Robert Birke, Mitch Gusat, Cyriel Minkenberg. *Lossless Virtual Networks.* Demonstration presented at the 32nd IEEE International Conference on Computer Communications (INFOCOM 2013), Turin, Italy, pages 624–625.

Fredy Neeser, Nikolaos Chrysos, Rolf Clauberg, Daniel Crisan, Mitch Gusat, Cyriel Minkenberg, Kenneth Valk, Claude Basso. *Occupancy Sampling for Terabit CEE Switches.* In Proceedings of the 20th Annual Symposium on High-Performance Interconnects (HOTI 2012), Santa Clara, CA, August 2012, pages 64–71.

Andreea Anghel, Robert Birke, Daniel Crisan, Mitch Gusat. *Cross-Layer Flow and Congestion Control for Datacenter Networks.* In Proceedings of the 3rd Workshop on Data Center - Converged and Virtual Ethernet Switching (DC CAVES), San Francisco, CA, September 2010, pages 44–62.

Daniel Crisan, Mitch Gusat, Cyriel Minkenberg. *Comparative Evaluation of CEE-based Switch Adaptive Routing.* In Proceedings of the 2nd Workshop on Data Center - Converged and Virtual Ethernet Switching (DC CAVES), Amsterdam, Netherlands, September 2010.

Curriculum Vitæ

Personal Details

Name	**Daniel CRISAN**
Date of Birth	September 3rd, 1982
Place of Birth	Ramnicu Valcea, Romania
Citizenship	Romanian

Education

2010–2014	**ETH Zurich**, PhD Student, Department of Information Technology and Electrical Engineering (D-ITET)
2008–2010	**EPF Lausanne,** Master of Science in Computer Science
2005–2008	**Ecole Polytechnique,** France, Graduate Engineer
2001–2005	**"Politehnica" University of Bucharest**, Romania, Automatic Control and Computers Faculty, Graduate Enginner
1997–2001	**"Mircea cel Batran" High School**, Ramnicu Valcea, Romania

Professional Experience

April 2014–present	**Google Switzerland**, Software Engineer, Site Reliability Engineering
April 2010–March 2014	**IBM Research**, Zurich Research Laboratory, Pre-Doc, System Fabrics group
Sept. 2009–March 2010	**IBM Research**, Zurich Research Laboratory, Master Thesis Student
April–August 2008	**Google Switzerland**, Internship
April–September 2005	**Freescale Semiconductor**, Internship