DISS. ETH NO. 27737

LEVERAGING WORKLOAD KNOWLEDGE TO DESIGN DATA CENTER NETWORKS

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

VOJISLAV DUKIC
Master of Science in Computer Science, FTN Novi Sad

born on 05.07.1991.

accepted on the recommendation of

Prof. Dr. Ankit Singla (ETH Zurich), examiner
Prof. Dr. Ana Klimovic (ETH Zurich), co-examiner
Prof. Dr. Gustavo Alonso (ETH Zurich), co-examiner
Dr. Paolo Costa (MSR Cambridge), co-examiner

2021

ABSTRACT

Data center networks are at the heart of cloud infrastructure. They allow cloud workloads to scale, be flexible, and meet the needs of modern businesses. Since the demand for higher bandwidth, lower latency, and minimal resource cost, is growing, cloud providers must continuously improve the efficiency of their infrastructure. The key to achieving optimal performance of data center networks is to understand the communication needs and behavior of modern cloud workloads.

Intuitively, as cloud operators collect more knowledge about their tenants and applications, *i.e.,* obtain *workload specifications* like required bandwidth, tail latency requirements, time to first byte, etc., they can use that knowledge to precisely provision the physical network infrastructure and deploy sophisticated control algorithms that maximize performance, increase utilization, and reduce the cost of cloud resources. However, obtaining and leveraging workload specifications is challenging in practice. On the one hand, users do not have clear incentives in terms of performance and cost benefits to invest the effort and help obtain the specifications of their workloads. On the other hand, cloud operators cannot provide those benefits without having a substantial number of workload specifications. Thus, many proposed systems that depend on application-specific knowledge remain in the domain of academic research, despite their significant performance advantages.

To break this vicious circle, we propose a set of methods, theoretical results, and systems that enhance the process of obtaining and using workload specifications in the data center network environment. Moreover, we demonstrate how to explore and utilize the space of various workload specifications. We start the exploration from coarse-grained insights from the past execution of cloud workloads and demonstrate how they can be used to reduce the cost of physical network infrastructure. Our system, *Iris*, leverages historical knowledge to reduce the overall cost of one of the most expensive parts of cloud networks – Data Center Interconnect (DCI) – by an order of magnitude compared to equivalent workload-agnostic solutions.

Furthermore, we analyze how to obtain fine-grained workload specifications that describe the future behavior of cloud applications and use them to enhance network efficiency. Thanks to our system, *Flux*, we automatically infer advance specifications using machine learning methods. Then, we show how to leverage these specification estimates to deploy sophisticated network control and scheduling mechanisms that achieve an order of magnitude improvement in terms of flow completion time and queue occupancy compared to the systems deployed in the cloud today.

Finally, we provide a set of rules and guidelines that cloud providers need to satisfy in order to motivate tenants to collaborate in the process of obtaining and utilizing workload specifications, and ultimately, make these specification-dependant systems practical in the modern cloud environment.

ZUSAMMENFASSUNG

Rechenzentren bilden das Kernstück der Cloud-Infrastruktur. Sie ermöglichen die Cloud-Workloads zu skalieren, flexibel zu sein und die Anforderungen moderner Unternehmen zu erfüllen. Da die Nachfrage nach höherer Bandbreite, geringerer Latenz und minimalen Ressourcenkosten wächst, müssen Cloud-Anbieter die Effizienz ihrer Infrastruktur kontinuierlich verbessern. Der Schlüssel zur Erzielung einer optimalen Leistung von Rechenzentren besteht darin, die Kommunikationsanforderungen und das Verhalten moderner Cloud-Workloads zu verstehen.

Wenn Cloud-Betreiber mehr Wissen über ihre Kunden und Anwendungen sammeln, d.h. Workload-Spezifikationen erhalten, können sie dieses Wissen intuitiv nutzen, um die physische Netzwerkinfrastruktur präzise bereitzustellen und ausgefeilte Steuerungsalgorithmen bereitzustellen, die die Leistung maximieren, die Auslastung erhöhen und die Cloud-Kosten senken. In der Praxis ist es jedoch eine Herausforderung, Spezifikationen für den Workload zu erhalten und zu nutzen. Einerseits haben Kunden keine klaren Anreize in Bezug auf Leistung und Kostenvorteile um den Aufwand rechtzufertigen eine Spezifikation ihrer Workloads zu erstellen. Auf der anderen Seite können Cloud-Betreiber diese Vorteile nicht bieten, ohne über eine erhebliche Anzahl von Workload-Spezifikationen zu verfügen. Daher bleiben viele vorgeschlagene Systeme, die von anwendungsspezifischem Wissen abhängen, trotz ihrer erheblichen Leistungsvorteile im Bereich der akademischen Forschung.

Um dieses Zirkelverhalten zu durchbrechen, schlagen wir eine Reihe von Methoden, theoretischen Ergebnissen und Systemen vor, die den Prozess des Abrufs und der Verwendung von Workload-Spezifikationen in der Netzwerkumgebung des Rechenzentrums verbessern. Darüber hinaus zeigen wir, wie verschiedene Typen von Workload-Spezifikationen erkundet und genutzen werden können. Wir beginnen die Untersuchung mit grobkörnigen Einsichten über das Verhalten von Cloud Workload in der Vergangenheit an und zeigen, wie sie verwendet werden können, um die Kosten der physischen Netzwerkinfrastruktur zu senken. Unser System, Iris, nutzt historische Spezifikationen, um die Gesamtkosten eines der teuersten Teile von Cloud-Netzwerken - Data Center Interconnect (DCI) - um eine Grössenordnung im Vergleich zu äquivalenten spezifikationsunabhängigen Lösungen zu senken.

Darüber hinaus analysieren wir die Verwendung feinkörniger Workload-Spezifikationen, die das zukünftige Verhalten von Cloud-Anwendungen beschreiben. Dank unseres Systems Flux erhalten wir automatisch Vorabspezifikationen mit Methoden des maschinellen Lernens. Anschliessend zeigen wir, wie diese Spezifikationsschätzungen genutzt werden können, um ausgefeilte Netzwerksteuerungs- und Planungsmechanismen bereitzustellen, mit denen sich die Ablaufzeit und die Warteschlangenbelegung im Vergleich zu den heute in der Cloud bereitgestellten Systemen um eine Grössenordnung verbessern lassen.

Schliesslich bieten wir eine Reihe von Regeln und Einschränkungen, die Cloud-Anbieter erfüllen müssen, um Kunden zur Zusammenarbeit beim Abrufen und Verwenden von Workload-Spezifikationen zu motivieren und diese spezifikationsabhängigen Systeme letztendlich in der modernen Cloud-Umgebung praktikabel zu machen.

Мом оцу

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Ankit Singla for all the great advice, support, and feedback provided throughout my doctoral studies. I want to thank Ce Zhang and Sangeetha Abdu Jyothi for supporting me in the early stage of my PhD journey. I would also like to express my gratitude to Gustavo Alonso and Ana Klimovic for being great collaborators and for being on my thesis committee.

I would like to thank my collaborators from Microsoft Research Cambridge, Christos Gkantsidis, Thomas Karagiannis, and Francesca Parmigiani, for guiding me through the world of regional cloud networks. Also, I thank Paolo Costa for supporting me throughout my internship at MSR and for being on my thesis committee.

Furthermore, I want to thank my colleagues and friends in the System Group that made it a pleasure to be part of the group: Debopam, Lukas, Melissa, Simon, Bojan, Max, Catalina, Elham, Nora, Cedric, Thomas, Rodrigo, Kaan, Fabio, Luka, Claude, Gerd, Darko, Jana.

Over the course of my studies, I particularly enjoyed working with my younger colleagues. Thus, I would like to thank Jonas, Timon, Valentino, Jan, Daniel, Damien, Hanjing, Elizaveta, Kenan, and Amory for all their work and effort on their bachelor's and master's thesis projects.

I want to thank my parents Duško and Ljiljana, and my sister Dušica for always supporting me in life. Finally, thanks and love to my beautiful wife Milena and my daughter Dunja, who supported me and endured the best and worst moments with me.

CONTENTS

1 INTRODUCTION 1

1.1 Cloud environment . . . 3

1.2 Making cloud networks more efficient . . . 4

1.3 Workload specifications . . . 6

1.3.1 Workload specification space . . . 6

1.3.2 Workload specification in the cloud environment . . . 8

1.4 Contributions . . . 10

1.5 Dissertation outline . . . 13

1.6 Publications . . . 15

2 DATA CENTER INTERCONNECT DESIGN 17

2.1 Introduction to DCI design . . . 19

2.2 The DCI network design problem . . . 20

2.2.1 Dimension #1: Latency . . . 25

2.2.2 Dimension #2: Siting flexibility . . . 26

2.2.3 Dimension #3: Implementation ease . . . 29

2.2.4 Dimension #4: Cost . . . 30

2.2.5 Analysis summary . . . 32

2.3 DCI design goals and constraints . . . 32

2.3.1 Operational constraints . . . 33

2.3.2 Technology-rooted constraints . . . 34

2.3.3 Component costs and operational costs . . . 37

2.4 Summary . . . 38

3 A NEW GENERATION OF DCI 39

3.1 Cost comparison: a motivating example . . . 42

3.2 Topology & capacity provisioning . . . 43

3.3 Electrical packet-switched network . . . 45

3.4 Pure wavelength-switching . . . 45

3.5 *Iris* - optical fiber-switched network . . . 46

3.5.1 Amplifier and cut-through link placement 48
3.5.2 *Iris* physical implementation 51
3.5.3 Dynamic reconfiguration 53
3.6 Hybrid DCI design 55
3.7 Evaluation 59
3.7.1 Cost analysis on real fiber maps 59
3.7.2 Physical layer feasibility 61
3.7.3 Impact of network reconfiguration 64
3.8 Related systems 66
3.9 *Iris* limitations 67
3.10 Summary 68
4 ADVANCE KNOWLEDGE OF FLOW SIZES 71
4.1 Background 72
4.2 Knowing flow sizes - easy or difficult? 74
4.3 Flow size estimation: design space 75
4.4 Exact sizes provided by application 76
4.5 Flow aging 77
4.6 TCP buffer occupancy 78
4.7 Monitoring system calls 80
4.8 Learning from past traces 81
4.9 Summary 81
5 LEARNING FLOW SIZES 83
5.1 Introduction to learning flow sizes 84
5.1.1 Workloads 84
5.1.2 Machine learning models 86
5.2 Opening the black box 88
5.2.1 The treachery of time 88
5.2.2 Why these features? 89
5.2.3 Model accuracy 90
5.3 Flow-size-based network scheduling 93
5.4 Evaluation 95
5.4.1 Flow-level scheduling 95
5.4.2 Coflow scheduling 98
5.5 Design & Implementation 99

5.5.1 Design overview 99
5.5.2 *Flux* implementation 101
5.6 Deployment environment 105
5.6.1 Where does *Flux* operate? 105
5.6.2 Hardware implementation 109
5.6.3 More complex environments 110
5.7 *Flux* limitations 111
5.8 Summary 112
6 WORKLOAD SPECIFICATION IN PRACTICE 113
6.1 Known vs.unknown in flow scheduling 114
6.2 Known vs.unknown in coflow scheduling 116
6.3 Guaranteed performance improvement 118
6.4 Toward practical performance guarantees 121
6.5 Limitations 122
6.6 Summary 124
7 CONCLUSION 127
7.1 Summary 127
7.2 Research outlook 130
7.2.1 Future of general workload specifications 130
7.2.2 Integration with user-level metrics 132

BIBLIOGRAPHY 133
A APPENDIX 149
A.1 Physical layer experimental details 149

1

INTRODUCTION

Data center networks are at the core of modern cloud infrastructure. The growing demand for more and faster data processing has pressured cloud operators to continuously innovate and improve the performance of their networks. These efforts have resulted in advances in physical network equipment, network control algorithms, and infrastructure management.

However, data center networks still struggle to meet the needs of modern businesses. Existing and emerging applications like disaggregated cloud [1], [2], autonomous driving [3], [4], cloud gaming [5]–[7], or virtual reality [8], [9], all have bandwidth and latency requirements that cannot be fully satisfied by today's cloud networks.

Besides high performance, cloud providers must ensure the lowest possible cost of their infrastructure. Competitive cloud computing market and user needs force major cloud providers to invest in new technologies that lower the cost of network hardware components, simplify network management, and reduce network operating costs. Another factor that drastically increases the communication cost is that cloud networks do not operate at the maximum utilization. To support the congestion-free operation of the network, cloud operators rarely stress their communication infrastructure beyond 50% of its maximum capacity [10]–[12].

Providing high-performance, cheap, and reliable communication in the modern cloud is still an open problem, and there is significant room for improvement. Making cheaper and faster hardware [13]–[15], optimizing network topology [16]–[18], and improving routing and congestion avoidance [19]–[21], are all active research directions that try to meet the demanding communication requirements of modern cloud applications.

Although very different in implementation, many techniques for improving the efficiency of cloud networks have something in common – implicitly or explicitly, they rely on having knowledge about the workloads that are running within data centers. Information about how much data applications are going to send, where, and when, can help cloud operators specialize their systems for particular application needs, and at the same time, precisely provision network capacity, synchronize applications and network flows to avoid congestion, or reconfigure the network to suit application needs.

Intuitively, we can assume that:

The more we specialize our data centers for their target workloads, the higher their performance and efficiency will be.

Thesis goal: The main goal of this thesis is to explore how to specialize physical data center network design as well as network control and scheduling algorithms by leveraging knowledge about cloud applications. We are interested in obtaining more knowledge about cloud workloads, *i.e.,* obtaining workload specifications and characteristics, and using those specifications to improve the efficiency of cloud network infrastructure. This entails an exploration of what workload specifications exist, how to obtain them, and how to exploit them in various aspects of data center network design.

1.1 Cloud environment

The cloud environment allows cloud providers to have excellent visibility into application behavior, learn about application needs, and deploy new systems that leverage that knowledge to improve network efficiency. Cloud infrastructure provides great flexibility and supports custom changes to almost any part of the network stack, which is impossible in more constrained networks like the Internet.

Making a global change on the Internet requires an agreement between multiple organizations (Autonomous Systems) that constitute the network. These organizations must agree on protocols, interfaces, and data transfer cost. However, this is challenging in practice because the interests of these organizations are often conflicting. Autonomous systems are driven by the business needs of their owners, and in many cases influenced by the policies of the local governments, which makes the process of deploying various types of changes slow and impractical.

On the other hand, data center networks are ruled and operated by a single organization. This allows cloud operators to have full visibility and control over their infrastructure and eliminate many business-incentive-related challenges that exist on the Internet. This centralized control and flexibility bring freedom to develop and quickly deploy techniques that require sophisticated changes at any point in the network stack.

Let us illustrate this contrast between the Internet and cloud networks with an example. Traditionally, TCP uses packet loss as a signal for network congestion. This approach is relatively simple because it does not require any in-network hardware support. Instead, source/designation hosts can use a set of timers and heuristics to detect packet loss. Although appealing because of its implementation simplicity, this approach has a negative performance impact – it takes too long to detect congestion. An old idea to mitigate this problem is to equip switches with Explicit Congestion Notification (ECN) capability so that they can immediately notify senders and receivers about congestion within the network [22]. Although ECN has been standardized since 2001 [23] and requires only a tiny change to the network hardware, it is not entirely supported on the Internet to this day [24].

What took decades to be implemented on the Internet requires months within the cloud. Since a new transport protocol that was tailored for the cloud, DCTCP [19], required ECN support, cloud providers were eager to deploy suitable hardware configuration in all of their data centers immediately. In return, DCTCP provided substantial improvements in performance thanks to efficient congestion avoidance.

Having a flexible environment is only a precondition for achieving high network efficiency. Next, we explore the opportunities to leverage that flexibility to maximize network performance.

1.2 Making cloud networks more efficient

Building and operating cloud networks is expensive. They contribute to around 15% of the total cost of cloud infrastructure [25], which, at the current scale of cloud computing, translates to tens of billions of dollars that ultimately cloud users need to cover. Furthermore, violations of the network Service Level Agreement (SLA) due to poor performance and unreliable hardware create a substantial expense both for cloud providers and cloud users. It has been show that the increase in network latency [26], [27] and the number of network outages [28], [29] do serious damage to modern businesses.

The factors and challenges that determine the performance and cost of the data center networks are fundamentally not different from those that appear in other types of networks, for instance: how to detect and handle partial and complete hardware failures, how to pick the proper network topology, how to balance the network traffic and choose the optimal route for each data flow, or how to handle congestion when multiple flows collide on the same path. Although all of those problems have solutions that operators developed for other types of networks, their performance can be drastically improved within data centers because of the flexibility that the cloud environment provides across the entire network stack.

At a high level, we can define two classes of techniques for building more efficient networks.

- *Improving the cost and performance of physical network infrastructure and devices used to build the network.* Building the right physical network topology to provide sufficient performance to various cloud applications has a critical impact on the overall efficiency of cloud infrastructure. Besides performance, important factors to consider when building a physical network are management complexity, network expandability, and component cost. Previous work has focused on various static [16]–[18] and dynamic [30]–[33] topologies that try to strike the right balance between cost, performance, and complexity.

  Another approach to increasing physical infrastructure performance is to rely on advancements in the domain of network hardware. New achievements in the domain of optical transceivers [14], [34] and switching fabric [35], [36] make networks cheaper, faster, and more reliable.

- *Improving data, control, and management plane algorithms on top of physical infrastructure.* Once the physical layer is established, there is substantial room for improvement of performance and efficiency by using smart traffic control algorithms. Full control over the network allows cloud providers to deploy unconventional and sophisticated approaches to congestion control, routing, packet forwarding and prioritization [19], [37]–[39].

  These techniques usually operate at much finer granularity compared to physical layer optimizations. They focus on the needs of individual applications and act at the level of network flows [40], flowlets [41], or coflows [42], [43].

Note that many of these optimizations at both layers have something in common – they are made possible by particular insights about application behavior. Knowing the application goals and how applications behave under different circumstances can help us make smarter decisions about network provisioning, routing, or traffic engineering. we call this knowledge about a particular application *workload specification*.

Next, we describe the main characteristics of workload specifications and how to deploy systems that leverage those specifications to improve network efficiency in the modern cloud environment.

1.3 Workload specifications

Knowing how much traffic an application is going to send, when, and where, *i.e.,* having a network *workload specification*, allows cloud providers to apply sophisticated optimization techniques to maximize performance and reduce the cost. The more knowledge we obtain about the running workload, the better we can integrate applications with the network infrastructure and ultimately bring them closer to optimal performance.

For instance, knowing the *size of each network flow* can be used to prioritize latency-sensitive traffic over more robust background traffic [12], [44]. That same information is also helpful to perform bandwidth allocation and altogether avoid the need for congestion control mechanisms [38], [45]. Further, understanding at what point in time an application is going to use the network is critical for dynamic networks that require reconfiguration depending on the changing traffic patterns [30], [46], [47]. By having this time information, dynamic networks can perform costly reconfiguration operations in advance and reduce the impact on application performance.

1.3.1 Workload specification space

The requirements from workload specifications vastly vary depending on the systems and processes that utilize them. For physical data center network design and provisioning, it is essential to have insights across many cloud workloads over a long period of time. Information like average bandwidth requirements, maximum aggregated traffic at any given time in the data center, frequency of substantial traffic changes, or general application sensitivity to increase in latency all have a massive impact on how we deploy physical infrastructure in data centers.

These general statistics about workload behavior help design large-scale general-purpose physical systems. However, making intelligent decisions about controlling and managing physical infrastructure requires more detailed workload characteristics as well as predicting *future* behavior of the workload. For instance, knowing the bandwidth requirement for a particular application in the future, who the application communicates with, and how sensitive the application is to congestion significantly impacts how we do routing, congestion control, load balancing, or prioritize network flows.

Based on these observations, we can define the space of all workload specifications with two dimensions: *granularity* and *time*. Granularity determines what part of an application (or set of applications) is described with a workload specification. Coarse-grained specifications describe the behavior of many applications together. They aggregate information of thousands of programs across the entire data center and provide insights like average or maximum bandwidth requirements, distribution of network flow sizes, frequency of traffic pattern changes. Further, specifications with finer granularity concentrate on the behavior of a single application, while most fine-grained specifications describe critical features of individual network flows or even isolated network packets.

On the other hand, the *time* dimension ranges from historical insights about past application behavior to providing estimates about future events.

Note that both *time* and *granularity* are not completely discrete dimensions. Instead, there is a continuum of workload specifications. They range from course-grained specifications that describe past behavior to those that provide fine-grained insights into the future.

1.3.2 Workload specification in the cloud environment

There are multiple challenges cloud providers need to address before they deploy specification-dependent systems and optimizations within their infrastructure.

How to obtain workload specifications? One obvious approach would be to ask cloud tenants to provide specifications for their applications to the cloud explicitly. For example, the application could provide its specification to the cloud through a predefined API call. Although appealing, this approach has two significant flaws. First, today's applications are not designed to understand and express their specification automatically, which means that using such API would require substantial manual changes to many applications. Second, and even more important, cloud users often do not even understand the requirements of their applications. Sometimes, it is just fundamentally impossible to know the exact characteristics because the performance and behavior depend on external factors that application owners do not control, *e.g.,* number of concurrent users on the website.

Obtaining workload specifications requires a lot of effort from both cloud tenants and cloud operators. Thus, they have to work together towards building new concepts, frameworks, and APIs that automatically capture and describe application behavior. This thesis explores this challenge in the context of cloud networks and proposes new approaches to obtaining workload specifications (§4, §5).

How to design a system that leverages those specifications to improve efficiency? Typically, systems that rely on having workload specifications require changes across multiple layers of the cloud stack. Besides obtaining the specification, cloud providers have to decide how to store and utilize them. Some systems are distributed and can utilize the specifications at places where they are made (usually endhost machines) [37], while more sophisticated scheduling and control algorithms require coordination and collecting many specifications at a centralized location to process them [38]. As a result, these centralized approaches compile a list of actions that should be applied to improve performance and efficiency. These actions are then distributed back to network switches, middleboxes, and end-host machines. To close the feedback loop, many of these components must monitor the effects of applied actions, which further increases the complexity of workload specification-based systems. In this thesis, we provide examples of such systems in §5.

How to guarantee performance improvements? Note that due to fundamental limitations, specifications for some applications are not obtainable in practice. Thus, cloud providers must support both workloads with *known* and *unknown* specifications. This has important implications for deploying specification-based control systems in the cloud environment. Since obtaining and leveraging knowledge requires substantial effort from cloud tenants, providers must make sure that moving a workload from unknown to known category will be justified by significant performance improvement. However, this is not easy to achieve for many systems and algorithms, as we will discuss in this thesis (§6).

How to utilize a workload specification that is incorrect or imprecise? In many situations, the workload specification cannot be provided accurately either due to limitations of the workload, *e.g.,* the workload depends on enduser actions, or because operators use heuristics and machine learning techniques to obtain the specification. Thus, specification-based control systems must be robust and deploy mechanisms that estimate the accuracy of individual specifications. On top of that, an additional challenge to those systems in practice is malicious user behavior. Namely, cloud tenants can intentionally provide wrong specifications or tweak their applications to gain certain benefits in terms of performance or cost. For example, if short network flows have higher priority, malicious tenants may report all their flows to be short and gain a substantial performance boost. Thus, it is essential to deploy defense mechanisms that will assure fairness and remove incentives for misbehavior. We discuss these challenges in more details in §6.

Next, we describe how we explore the workload specification space and answer these critical questions in the context of this thesis.

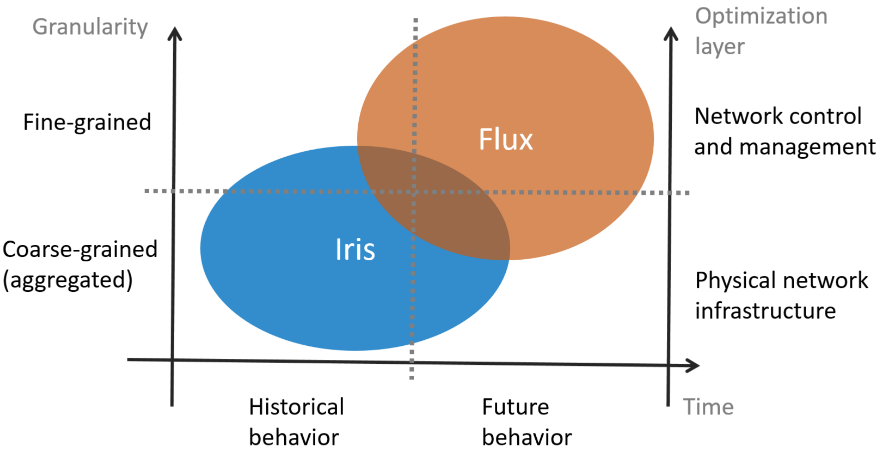

Granularity
Fine-grained
Coarse-grained (aggregated)
Flux
Iris
Optimization layer
Network control and management
Physical network infrastructure
Historical behavior
Future behavior
Time

Fig. 1.1: We explore workload specification space by designing and analyzing two systems: *Iris* and *Flux*.

1.4 Contributions

This dissertation focuses on improving the efficiency of cloud networks by building systems that rely on network-specific workload specifications. In particular, we focus on answering three high-level research questions:

- *How to obtain network-specific workload specification in the cloud environment?*
- *How to create highly-efficient systems that leverage those specifications*
- *How to increase the use of workload specifications in today's cloud?*

We answer those three questions by creating systems and algorithms that operate throughout the entire workload specification space: starting from course-grained historical workload specifications to fine-grained application-specific specifications that describe the future behavior of the workload. Also, we contribute to both designing new physical infrastructure and improving the efficiency of control algorithms on top of that infrastructure. Fig. 1.1

visualizes our contribution in workload specification space. We navigate through that space by designing and analyzing two systems.

First, we propose *Iris* [48][1], a novel all-optical network design that leverages course-grained workload specifications aggregated over long periods of time to improve the efficiency of data center networks. In particular, *Iris* makes an observation about traffic stability between data centers thanks to historic workload specifications. That insight then allowed us to leverage cheap, *high-switching-latency* equipment to create a novel optical fiber-switching network architecture. Our design lowers the cost of one of the most expensive components of today's data center network infrastructure - regional cloud networks and Data Center Interconnect (DCI) infrastructure. Namely, the cost of today's DCI is so high that it forces cloud providers to compromise on network latency and management flexibility in order to keep the cost within reasonable bounds. However, *Iris* reduces the cost by 7× on average compared to today's solutions allowing cloud providers to design and build a new generation of DCI that offers the best possible performance to cloud applications.

Second, we present *Flux* [49], a system that operates at the other extreme of the workload specification space. *Flux* automatically predicts fine-grained workload specifications about future network-related behavior of individual applications and uses them to improve network flow and coflow scheduling efficiency. Predictions made by *Flux* allow deploying fine-grained bandwidth allocation, avoiding congestion and enabling near-zero queuing to maximize network utilization and minimize latency. Furthermore, the predictions also allow smart packet prioritization that mitigates the head-of-line blocking problem and improves the flow completion time by 11× compared to workload-specification-agnostic systems. *Flux* achieves that while not requiring cloud tenants to make any modifications of their applications. Also, *Flux* does not have any access to the application source code or the client's internal infrastructure. Instead, it obtains advance knowledge by collecting resource

---

[1] Results related to DCI design and *Iris* that are covered in this thesis are a result of a collaboration with Microsoft Research Cambridge, and they were the subject of my internship at the MSR Cambridge Lab.

utilization statistics already available to cloud providers and applying machine learning techniques to make predictions about future behavior.

As we will show, obtaining workload specifications requires substantial effort from both cloud providers and/or cloud users. To make specification-dependant systems practical, it is essential to create clear incentives for everyone to invest time and effort into obtaining the best possible workload specifications. Thus, as a part of this thesis, we analyzed and proposed a set of techniques that *formally guarantee* that the system efficiency only improves when more knowledge about workloads is added to the system. This guarantee is critical for all systems that depend on application knowledge and workload specifications. Suppose additional effort invested in obtaining that knowledge could result in performance degradation. In that case, the systems we proposed, together with other specification-dependent techniques, have little chance to see widespread adoption in the modern cloud environment.

Here we provide a more detailed list of contributions of this dissertation:

- We propose *Iris*, an all-optical fiber-switching network architecture that leverages historic workload specifications to lower the cost and complexity of modern DCI design. We show that our architecture can provide a factor of 7× cost savings compared to equivalent state-of-the-art solutions while having minimal impact on latency, throughput, and performance.

- We design *Flux*, a framework for estimating flow sizes in advance by looking at the past resource utilization of the application, history of communication, and system-level parameters.

- Obtaining workload specifications on time, with low latency, can be critical for providing performance benefits. On the example of *Flux*, we show how to achieve microsecond-scale latency in obtaining estimates of future workload behavior using traditional hardware and hardware accelerators.

- We evaluate the utility of inferred (often imprecise) flow sizes across multiple scheduling techniques, finding significant benefits compared to today's scheduling algorithms that not to leverage workload specifications, *e.g.,* the improvement of 11× in terms of mean flow completion time.

- We analyze systems and algorithms that operate in environments with partially available workload specifications. We show that the impact of increasing the amount of knowledge about workload behavior in these systems does not always lead to performance improvements. Surprisingly, sometimes more knowledge degrades performance.

- We prove that there is a class of scheduling algorithms that can *formally guarantee* better performance given more knowledge about the system. Besides this formal performance guarantee, we list other requirements from the specification-dependant systems that need to be satisfied before major cloud providers adopt and deploy them.

1.5 Dissertation outline

This dissertation is organized in 7 chapters.

- *Chapter 2: Data Center Interconnect design* - Before we present a system that improves network efficiency by leveraging historical workload specification, we have to provide sufficient background to the challenge that system is solving. Thus, we introduce the problem of designing Data Center Interconnect (DCI) infrastructure. We analyze the design goals as well as the technological and operational constraints that influence the process of building a DCI deployment. We show how the design choices influence network latency, data center sitting flexibility, implementation ease, and the overall infrastructure cost.

- *Chapter 3: A new generation of DCI* - in this chapter we present *Iris*, a new all-optical fiber-switching approach to building physical

DCI infrastructure that was enabled by insights made using historical workload specification. We also present other DCI implementation options based on electrical and wavelength switching. We evaluate their key properties and comparing them to *Iris*.

- *Chapter 4: Advance knowledge of flow sizes* - moving up in the network stack and specification granularity, instead of improving physical infrastructure, here we focus on approaches that improve the network control logic by exploiting fine-grained workload specifications about future program behavior.

- *Chapter 5: Learning flow sizes* - we describe *Flux*, a new machine learning-based system for obtaining more knowledge about future network demand of individual applications and evaluate how this information can be used in existing and new scheduling algorithms to improve network efficiency.

- *Chapter 6: Workload specification in practice* - we discuss constraints that must be met before specification-dependent systems are deployed in practice. For instance, somewhat counterintuitively, we show that adding more knowledge about a particular system does not guarantee higher efficiency of that systems. Instead, it can degrade system's performance. This property cloud completely discourage cloud providers and users to invest effort into obtaining workload specifications. Thus, we focus on a class of control algorithms that *formally guarantee* that additional knowledge can only improve efficiency of the system.

- *Chapter 7: Conclusion* - we summarize the key results of this thesis and provide directions for future research.

1.6 Publications

Part of the work in this thesis has already been published and is listed here for reference:

[1] V. Dukic, S. A. Jyothi, B. Karlas, M. Owaida, C. Zhang, and A. Singla, "Is advance knowledge of flow sizes a plausible assumption?", in *USENIX NSDI*, 2019.

[2] V. Dukic, G. Khanna, C. Gkantsidis, T. Karagiannis, F. Parmigiani, A. Singla, M. Filer, J. L. Cox, A. Ptasznik, N. Harland, W. Saunders, and C. Belady, "Beyond the mega-data center: Networking multi-data center regions", in *ACM SIGCOMM*, 2020.

Throughput my doctoral studies I worked on other research topics that are not directly covered in this thesis. That work has been published in the following publications:

[3] V. Dukic and A. Singla, "Happiness index: Right-sizing the cloud's tenant-provider interface", *USENIX HotCloud*, 2019.

[4] V. Dukic, R. Bruno, A. Singla, and G. Alonso, "Photons: Lambdas on a diet", *ACM SoCC*, 2020.

2

DATA CENTER INTERCONNECT DESIGN

We start exploring the workload specification space by demonstrating how insights obtained using coursed-grained historic workload specifications influence the design decisions in deploying one of the most expensive parts of modern cloud networks - Data Center Interconnect.

To support the growing demand for cloud resources today, major cloud providers must build multiple data centers (DCs) within a relatively small metro area around large cities. Those data centers, typically less than 20 of them, interconnected with a high-bandwidth, low-latency network comprise a cloud region. This fast network that cloud providers build and operate to support only their needs and workloads is called Data Center Interconnect (DCI).

In this chapter, we first outline the design space of DCI topologies, ranging from fully centralized ones with all DCs connected to one hub (in practice two for reliability) to distributed ones that either eschew such hubs entirely or reduce dependence on them by building closer or direct connectivity between some subsets of DCs. We show that DCI design involves more nuance than just the clichéd centralized-distributed dichotomy may suggest, fleshing out its complexity by: (a) analyzing data from several of Microsoft Azure's regions; and (b) performing testbed experiments that demonstrate the physical-layer constraints.

Our analysis shows that distributed topologies provide much lower DC-DC latency than DC-hub-DC connectivity: compared to a centralized topology, latency reduces for at least 60% of DC-DC paths, and in more than 20% of cases we analyzed, the latency is $> 2\times$ lower.

The latency advantage is of high and growing value. By looking at the needs and specifications of today's applications, we observe that customers are increasingly asking for lower latency service level agreements. For instance, latency-sensitive applications like synchronous replication are going mainstream at the region level [50]. While the latency advantage of direct DC-DC connectivity is unsurprising, we also show that distributed topologies increase flexibility in choosing DC sites. In the analyzed regions, the area in which new DCs could be located increases by 2-5× with distributed topologies.

Unfortunately, as they would be implemented today, with electrical packet switching, distributed topologies perform poorly compared to centralized approaches across two key metrics: cost and complexity. The centralized approach is much more cost-effective, by as much as 7× in the settings we studied, and is significantly easier to manage, requiring a much smaller number of components. Thus, cloud providers need a distributed DCI network design that can reduce the cost and complexity, *i.e.,* the number of ports, while not sacrificing any network performance.

This chapter focuses on providing background on DCI design and defining constraints that play an essential role in the DCI design space. In the following chapter (§3), we propose a concrete DCI architecture, *Iris*, that leverages coarse-grained historic workload specifications and achieves low cost and complexity.

Outline This chapter is organized as follows: in Section 2.1 we provide a short history of cloud development that led to the need to build efficient DCI today. Section 2.2 precisely defines the problem of DCI design and provides the main insights from today's cloud application specifications, while Section 2.3 lists operational and technology-rooted constraints that shape the final DCI network design.

2.1 Introduction to DCI design

Cloud computing's growth has forced commensurate scaling of data center (DC) infrastructure. Until recently, such scaling meant building "mega"-DCs with hundreds of thousands of servers across the world, and interconnecting them into a wide-area backbone.

However, a different scaling strategy has quickly become standard industry practice. Instead of serving each broad geographic area from just one or two mega-DCs, in many geographies, large cloud providers have transitioned to using a collection (typically 2-20) of smaller DCs within tens of kilometers of each other, referred to as a "region". This shift away from mega-DCs is driven by two pressures: (a) the difficulty of siting and provisioning large facilities in or near dense metro areas due to limited resources such as land, power and connectivity; and (b) application desire for fault tolerance in the face of losing one or two large facilities to catastrophes like flooding and earthquakes. These fundamentals have forced all of the largest DC operators, including Amazon [51], Facebook [52], Google [53], and Microsoft [54], to increasingly rely on such regions.

Large volumes of traffic flow between DCs in a region, thus requiring a high-capacity network typically referred to as a regional Data-Center Interconnect (DCI). The growth of the DCI has led to it incurring significant costs for cloud providers, as, for example, seen by the explosive increase in the total number of 100G ports deployed: there are two orders of magnitude more regional DC-to-DC ports than WAN-facing ports [54]. High capacity notwithstanding, superficially, the design of such DCIs appears trivial:

- The number of DCs to interconnect is small.
- Each DC has a known available capacity.
- DCs are only a few tens of kilometers apart at most.

Yet, as we shall show, DCI design is challenging due to several operational, cost, and technological constraints that are different from those for both intra-DC networks, and DC-WANs used for inter-region connectivity. These constraints lead to complex decisions on both the network's topology, and how this topology is realized with appropriate switching technology.

2.2 The DCI network design problem

A regional DCI connects 5-20 DC sites within tens of kilometers. The problem of network design in this setting requires 4 **inputs**:

- **DC site locations**. Our focus is the network; DC siting is itself an interesting problem but requires separate treatment as many of the involved factors are non-networking, *e.g.,* the particular buildings available, their cost, connectivity to not just network providers, but power and ground transit infrastructure, etc.

- **DC capacities**. Based on each DC's size and other business factors, we know each DC's network capacity, *i.e.,* how much traffic a DC can maximally send or receive to other DCs in the region. For convenience, we translate the Gbps capacity into a number of fibers, *e.g.,* capacity B Gbps translates to ${}^{B}/_{C} \cdot \lambda$ fibers, where λ is the number of wavelengths per fiber, and C the bandwidth per wavelength in Gbps. In this example, $P = {}^{B}/_{C}$ is the number of electrical ports, *i.e.,* transceivers, required at each DC.

- **Fiber map**. The region's available fiber is known in terms of fiber ducts between two types of nodes: DCs and "fiber huts", which are intermediate nodes housing switching and other equipment like amplifiers. Where convenient, huts can co-exist with DCs. For our purposes, fiber ducts are unconstrained in the fiber available to lease: each fiber duct contains hundreds of individual fibers, with typically only a fraction of those lit. This is standard industry practice to amortize the cost of constructing a duct.

- **Component failure model** Network devices fail over time. Failures like fiber cuts or transceiver/switch malfunctioning can have a severe impact on cloud application performance. Since even the slightest degradation in network performance has a substantial influence on distributed workloads [26], [55], cloud providers must make sure that failures are resolved in the order of several seconds. In this thesis, we work with a simplified failure model where we provision additional capacity to support up to F_{max} simultaneous fiber duct cuts, given that the cuts of the entire duct are one of the most damaging failure types.

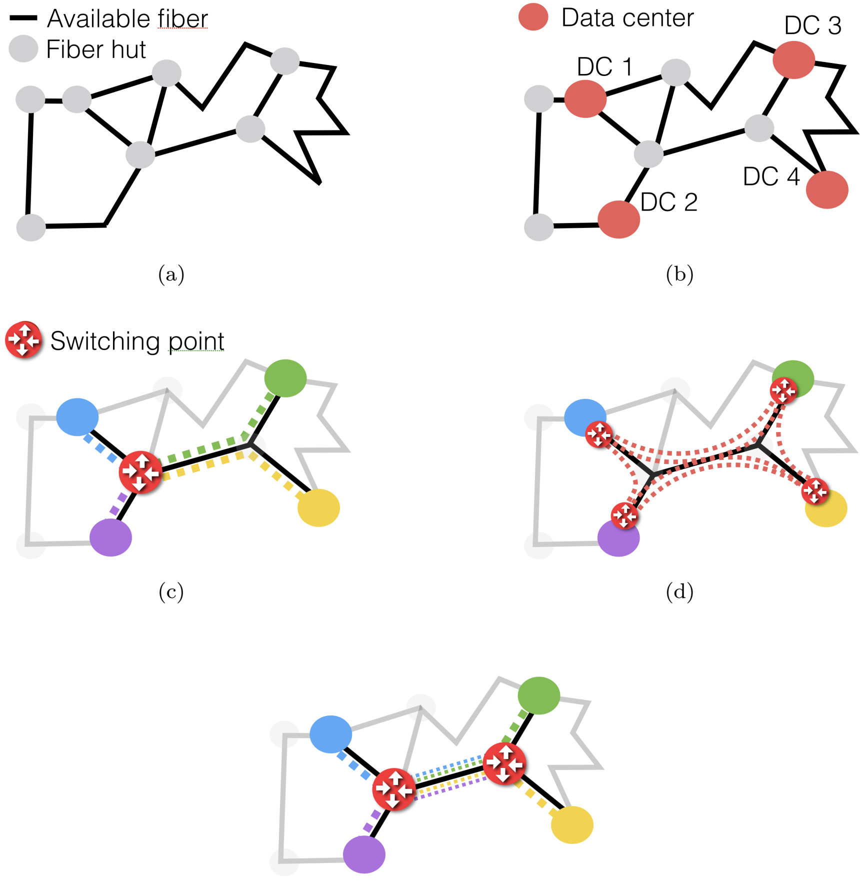
The above inputs are *outside* the network designer's control. DC sites, capacities, and the maximum number of failures allowed are set by operational needs. Expanding the fiber map is possible in some regions but is typically avoided: it is time-consuming, has a high up-front cost, and is unlikely to improve routes, especially in dense metro areas that already have plentiful fiber and are space-constrained against further expansion.

A simple example of DCI input specification is shown in Fig.2.1. The region's fiber map, including *all* available fiber huts and ducts, is demonstrated in Fig.2.1(a), and the 4 DCs the operator has built or plans to build in this region are shown in Fig.2.1(b). For this running example, we will assume that all DCs have the same capacity of f fibers each.

Given the DC sites, capacities, and fiber map, we must decide on the following **outputs**:

- **Topology**. Which DC-DC connections are *direct*, *i.e.,* without needing intermediate routing at other DCs or huts? This decision dictates the subset of the fiber map that is used, *i.e.,* which huts and ducts are needed.

- **Capacity**. What number of fibers are leased in each fiber duct?



(a) (b) (c) (d) (e)

Fig. 2.1: DCI design example: (a) The fiber map containing all available fiber ducts and huts. (b) The region has 4 DCs for which DCI connectivity is to be determined. (c) The centralized approach uses a hub to which all DCs connect; in practice, 2 hubs are used for resilience, but only one is shown for clarity. (d) An extreme version of the distributed approach, with all pairs of DCs connected directly to each other. (e) A sparser distributed approach, with two pairs of DCs – each pair connects to a hub, and the two hubs connect to each other.

- **Switching**. how is switching (*e.g.,* electrically vs. optically) implemented at the DCs and huts?

Loosely, one can think of the topology and capacity decisions as provisioning problems, answers to which depend on the design **goals**: Do we insist on shortest path connectivity, or are longer paths acceptable? Do we provision non-blocking connectivity between all DCs, or is an oversubscribed fabric acceptable? How much failure resilience do we need in terms of fail-over paths?

On the other hand, switching is more tied to implementation: What equipment is used at DCs and fiber huts, and how is it interconnected such that it correctly instantiates the topology and capacity decisions? The industry's standard method of switching is to deploy electrical switches. The data travels on each fiber in optical wavelengths, and at given switching points, it leaves the optical domain, such that switches can reroute data as necessary.

However, there is a complex interplay between topology and capacity and switching: the switching technology can place **constraints** on the topology. For instance, an uninterrupted run of fiber, without amplification or termination at a DC or hut, referred to as a "fiber span", cannot be longer than a particular length.

While we will make the goals and constraints more precise in §2.3, the above context suffices to examine the design space and trade-offs for DCI networks in terms of two broad approaches.

The centralized approach uses a hub-and-spoke topology: DCs in a region all connect to a centralized hub. In the example in Fig.2.1(c), one of the huts is used as a hub, and no other huts are used. There are no direct DC-DC connections, with all connectivity going through the hub. For a non-blocking interconnect, the fiber ducts connected directly at the four DCs will carry f fiber-pairs to connect each DC's full capacity to the hub, where sufficient switching hardware must be provisioned. The remaining central duct carries the $2 \cdot f$ fiber-pairs from the two DCs on the right.

For simplicity, we illustrate and discuss only one hub in our example, but for failure resilience, two hubs are used, and each DC connects to both. The hubs provide a “big switch” abstraction, whereby all DC-pairs are connected in a non-blocking fashion to each other. This approach is presently used in Microsoft Azure [54].

The distributed approach directly connects DCs to each other. An extreme version of this approach would build all pairs of DC-DC connections, *i.e.,* $O(n^2)$ for n DCs, like in Fig.2.1(d). In this example, for non-blocking connectivity, $3 \cdot f$ fiber-pairs are needed at the four fiber ducts that originate at the DCs (one fiber-pair each for the other three DCs), with $12 \cdot f$ fiber-pairs on the central duct. We also highlight here the aforementioned interplay with switching: due to technology constraints, it may not be possible to instantiate this design as is, *e.g.,* because some of the DC-pairs that we want to connect directly are too far to be connected over an uninterrupted fiber span, and need amplification at a hut in between.

More generally, one can build a variety of sparser distributed networks, with some DC-DC pairs eschewing direct connectivity in favor of transit through other DCs or huts. An example of this is shown in Fig.2.1(e), where two pairs of DCs connect to hubs, with the hubs connecting to each other. In this case, for non-blocking connectivity, f fiber pairs are needed on the 4 DC-incident fiber ducts, and $2 \cdot f$ fiber-pairs on the central duct. From public resources [51], it appears Amazon AWS broadly uses this approach.

Note: In the above discussion, we highlighted the amount of fiber used primarily to clarify how different connectivity models can be instantiated atop a given fiber map. However, the impact of the design choices is much deeper than just the quantity of fiber used. Different solutions achieve vastly different outcomes in the following design dimensions: performance, reliability, operational flexibility, and cost, as we discuss next.

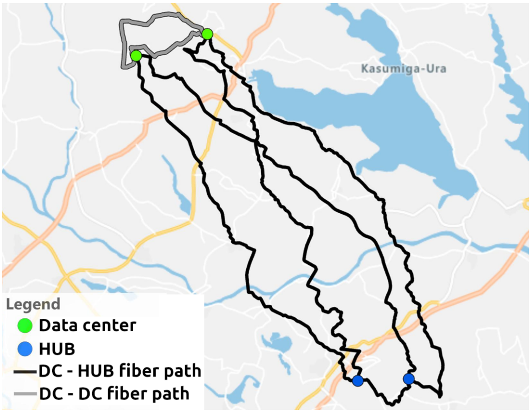


FIG. 2.2: DC-hub-DC paths can sometimes be much longer than DC-DC ones.

2.2.1 Dimension #1: Latency

An obvious distinction in the centralized and distributed models is the propagation latency they provide between DCs — the distributed approach, provided the right DC-DC links are provisioned, can substantially lower latency by eschewing transit through a hub. Fig.2.2 demonstrates this contrast in the Tokyo region.[1] The two hubs are located South of two of the DCs in the region. The DC-hub connections are 53 km to 60 km in terms of fiber distance, resulting in a maximum DC-DC roundtrip latency of 1.2 ms. In contrast, a direct DC-DC connection of 19 km would achieve a 0.2 ms latency, a 6× latency reduction.

Fig.2.3 investigates this latency inflation by using Microsoft Azure's DC locations across 22 regions. In some cases, direct DC-DC paths can reduce roundtrip propagation latency by several times, similar to the example in Fig.2.2; in more than 20% of cases, the reduction is more than 2×[2]. As not

---

[1] Example regions and fiber maps used throughout the paper use mock-up drawings that resemble but do not represent Microsoft Azure's network maps.

[2] Inter-connecting DCs within Availability Zones [54] may alleviate some of this latency inflation of centralized topologies similar to semi-distributed topologies as in Fig.2.1(e).

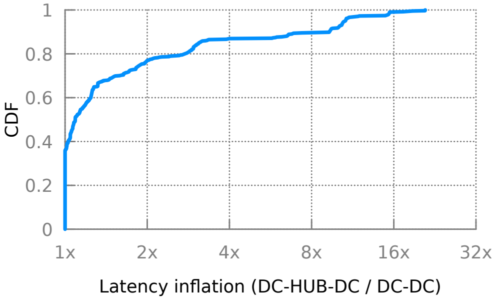


FIG. 2.3: Latency inflation of paths via a hub compared to direct ones.

all DCs are connected to one another in these regions, we estimate DC-DC latency using an industry rule of thumb: multiplying the geo-distance by 2× [56], [57].

The astute reader will notice from Fig.2.2 that part of the reason the DC-hub-DC paths are much longer is that both hubs are close to each other – if they were more spread out in the region, in many cases, at least one hub-path could be much shorter. Unfortunately, the hub placement is not this flexible, as we discuss next.

2.2.2 Dimension #2: Siting flexibility

Bounding DC-DC latency requires constraining the locations of DCs and hubs. The maximum latency allowed between any two DCs is typically specified in Regional Service-Level Agreements (SLAs) that implicitly define the maximum DC-DC fiber distance — Azure limits fiber-distance to 120 km for any DC pair [54]. Analyzing data from Microsoft Azure's regions shows that the resulting siting constraints are much more rigid for the centralized design than the distributed one, making the latter preferable for maximizing deployment flexibility.

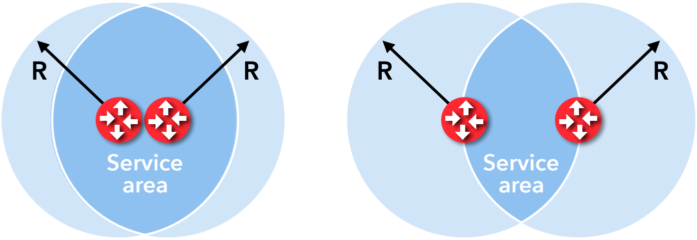


FIG. 2.4: Reliability vs.flexibility in the centralized approach. The circles are for intuition; in practice, we must consider real fiber distances.

For the centralized approach, the 120 km limit restricts each DC-hub connection to at most 60 km of fiber. Thus, once the hubs are placed, a service area for placing DCs is determined as the intersection of their 60 km-radii, as shown in Fig.2.4. Comparing the left and right parts of Fig.2.4, we see that placing hubs close to each other would maximize the permissible service area (intersection). But this comes at the cost of latency and reliability: (a) if hubs are placed close to each other, DC-hub-DC paths can be longer; and (b) if one hub is lost to a catastrophic event, the other is more likely to be also affected if it is nearby. Thus, in practice, operators using a centralized DCI approach must trade-off latency and reliability if they want greater DC siting flexibility.

In contrast, by eschewing hubs, the distributed approach simplifies DC siting and alleviates the difficult flexibility-reliability trade-off. We show in Fig.2.5 this contrast visually for 4 regions, in the form of permissible area for siting one new DC given existing DCs or hubs. The top and bottom rows of the figure are for the same regions, except in the top row, the hubs are placed nearby (within 4 km to 7 km of each other), while in the bottom row, they are farther apart (20 km to 24 km). For the centralized approach, the service area is smaller when the hubs are closer. The service area for the distributed approach remains the same across the top and bottom rows as it does not use or depend on hubs. In each case, the distributed approach allows a much higher flexibility in picking DC sites. This analysis uses real fiber maps and distances and the same criteria as cloud operation teams follow for DC and hub placement.

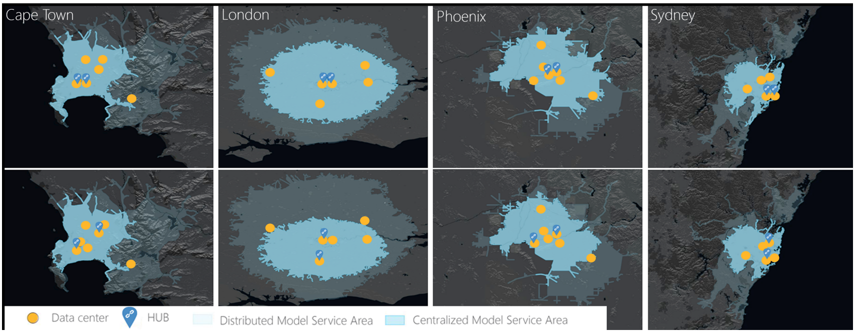

Cape Town
London
Phoenix
Sydney
Data center
HUB
Distributed Model Service Area
Centralized Model Service Area

FIG. 2.5: The distributed approach expands available area for building new DCs. These maps are for hypothetical regions, but with DC and hub placement using real criteria as analyzed by Microsoft Azure's deployment team. The top row shows results with hubs within 4–7 km, and the bottom within 20–24 km. The maximum allowed fiber distance for all DC-DC communication is 120 km for both models. In the distributed model, DCs can be placed in the extended shaded area, which is out of reach in the centralized model.

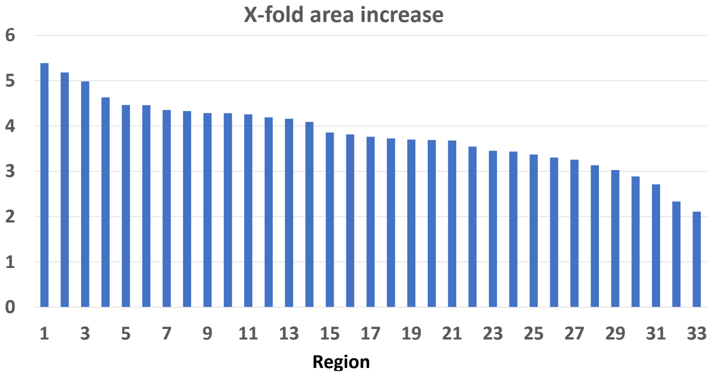


FIG. 2.6: Across existing regions (different bars) service area increases by 2-5× with a distributed approach compared to a centralized one.

Using similar analysis, Fig.2.6 shows that the permissible siting area for one new DC (given existing sites) would increase by 2–5× across 33 existing regions with the distributed approach compared to the centralized one. Even though each additional DC that is built constrains future sites in the distributed approach, it is still much more flexible than the centralized one — the number of DCs in the regions used for this analysis ranges from 5–15 existing DCs, with regions with more DCs showing (as expected) smaller, but still sizable (at least 2×), benefits with the distributed approach.

The size of the service area greatly impacts deployment costs and the availability of critical resources like space, especially in busy metro areas. Even a small increase in the service area can provide significant flexibility for a provider and reduce capital costs.[3]

2.2.3 Dimension #3: Implementation ease

The implementation of the centralized approach is simple, effectively breaking up a mega-DC into multiple sites — the uppermost (core) switching tier of what would have otherwise been a mega-DC resides at the hubs, such that connections between this and lower topology tiers are now externalized fiber connections traversing a few tens of kilometers. Operationally, the first step is picking the sites for the hubs and provisioning them, anticipating the needed switching capacity. Then over time, the DCs are built such that each DC is within a threshold fiber distance from each hub — as all DC-DC connectivity traverses a hub, this constraint ensures that DC-DC distances (latencies) are bounded per the SLA. The big-switch abstraction further eases management and provisioning; each DC connects all its capacity to the central switching fabric, where a non-blocking network connects it to other DCs. This approach can be easily replicated across regions irrespective of the underlying fiber layout.

---

[3] Land scarcity has even motivated building vertical DCs [58].

A distributed approach requires greater design effort in planning which DC-DC connections are made and at what capacity, such that appropriate infrastructure can be provisioned at each DC. Operationally, the first DCs can be built relatively unconstrained, but later DCs must be within a fiber distance threshold of each existing DC. Once it is determined which physical DC-DC links will be built, one must decide on routing such that each DC-DC pair has a path, direct or otherwise, with enough capacity. Given the physical links and routing, DC-DC link capacity can thus be determined and implemented at the physical fiber layer. For traffic from DC A to C transiting through DC B, the A-B fiber carries both direct A-B traffic and A-C traffic. The A-C traffic is switched using electrical switches installed at B, requiring conversion from the optical domain to electrical, followed by electrical switching, followed by conversion to optics again. Thus, capacity provisioning must account for transit capacity appropriately. Further, small DC facilities are typically severely constrained in terms of available power and space resources, and supporting connectivity to multiple other DCs may not be feasible. Thus, care needs to be taken as to which DCs can be inter-connected beyond just fiber capacity.

Thus, for provisioning, the centralized approach is a natural extension of today's Clos networks, while the distributed approach needs additional design effort. Further, expanding a region to add more DCs or capacity at existing DCs also poses different challenges for the two approaches. Centralized DCIs require the hubs to have enough space and power for the *maximum* predicted region scale; accommodating unanticipated growth in a region is thus difficult. The distributed approach requires similar provisioning at multiple (smaller) switching points when a region is expanded.

2.2.4 Dimension #4: Cost

While we defer a complete cost analysis to §2.3.3, we can use regional network port counts to flesh out the design space coarsely.

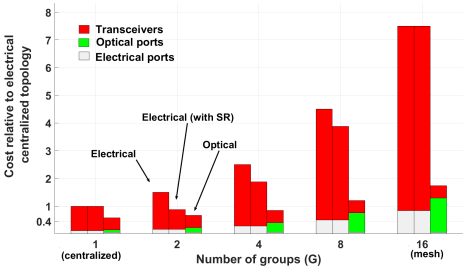


FIG. 2.7: Relative port cost breakdown for electrical and optical networks as topologies become more distributed. Total cost is estimated based only on per-port cost. "Electrical with SR" uses cheaper short-reach transceivers for connecting DCs within a group.

To understand the cost implications of supporting distributed topologies, we look at a simple model of N DCs of capacity P, in terms of physical DCI ports. Here, a DCI port reflects an electrical switch port of some bandwidth dedicated to the DCI network at a particular DC. We further assume that the N DCs are organized in G groups. To simplify, we consider all G groups to be balanced in size and that all DCs in a group are interconnected using a group-local hub. Further, we assume all-pairs direct connectivity across groups. This simple model allows us to move gradually from centralized towards distributed topologies: $G = N$ represents a fully distributed topology with all DC-DC pairs directly connected, while $G = 1$ represents the centralized topology.

For $G = 1$ and a capacity of P ports per DC, the total number of ports required in the topology is equal to $2 \cdot N \cdot P$, *i.e.*, double the total capacity of all DCs, as $N \cdot P$ ports are required at the hub. For $G > 1$, the number of ports needed to connect DCs within a group is $2 \cdot P \cdot N/G$. Each group hub needs to support $P \cdot N/G$ capacity downstream and $(G-1) \cdot N/G \cdot P$ ports upstream to other groups, for a total of $N \cdot P$ ports. This means that the capacity of the hub is essentially independent of the size of the group N/G; each group hub needs to support the same capacity irrespective of how distributed or centralized the topology is. In total, the topology requires $(G+1) \cdot N \cdot P$ ports.

This is shown in Fig.2.7 using an example region of 16 DCs. The figure further breaks down cost contributions from different hardware components: (a) electrical switch ports and (b) DCI transceivers, based on realistic cloud-provider prices where a transceiver costs roughly 10× an electrical port. The figure shows that in such a region, the relative cost of supporting a fully meshed distributed topology is roughly 7× the cost of the centralized topology. The semi-distributed topologies are also more expensive than a centralized one, even when we account for group-internal connectivity using cheaper short-reach optical transceivers, which is optimistic, as the required hub-DC distances to be able to use such transceivers (≤ 2 km) will not always be achievable. The results highlight that the biggest contributor to the cost is the optical transceivers. The third column shows the cost of an optical DCI network, assuming we could replace transceivers with optical reconfigurable ports, the approach we advocate.

2.2.5 Analysis summary

Our analysis reveals clear pros and cons for each approach: the distributed approach has clear advantages in latency and siting flexibility, but entails greater complexity and cost. Thus, to make the distributed part of the design spectrum more accessible by lowering these cost and complexity barriers, we propose *Iris*.

2.3 DCI design goals and constraints

Practically realizing a Data Center Interconnect deployment requires addressing a large set of constraints: *operational constraints* that derive from application requirements and workload specifications, and *technology constraints* imposed by the physical characteristics of the optical equipment used.

2.3.1 Operational constraints

OC1. **Latency SLA** — To provide application placement flexibility, DCI has to offer very low latency between data centers, so that parts of a distributed application that are deployed in different data centers within the same region, still do not observe any performance degradation. It has been shown statistically that many cloud applications today suffer under latency increase higher than 1ms [26], [59], [60]. Thus, while designing a DCI, we have to provide strict latency SLAs and bound the physical distance between data centers. For existing SLAs and industry practices, this translates to a maximum DC-DC fiber distance of 120 km (§2.2.2).

OC2. **Any traffic matrix** — Each DC's aggregate network capacity is known based on the number of machines in each one of them, application needs, and other business factors. To provide the desired flexibility as well as high performance, the DCI should accommodate any traffic demands that are not bounded by DC capacity, as in the hose model [61]. DCI links are typically symmetric, so we do not distinguish between ingress and egress capacities, assuming symmetric demands without loss of generality.

OC3. **Shortest path** — Given that many cloud applications can vastly benefit even from the smallest improvements in latency, traffic between DCs must always use the shortest available physical path.

OC4. **Failure resilience** — Based on reliability goals, an operator specifies a number of fiber cuts that must be tolerated, *i.e.,* for any number of cuts up to the specified tolerance, *OC1-OC3* should continue to hold. A fiber cut here means a *fiber duct destruction*, *i.e.,* all capacity for all fibers traversing the duct are lost. For our description, we use a tolerance of 2 cuts, in line with operational practice.

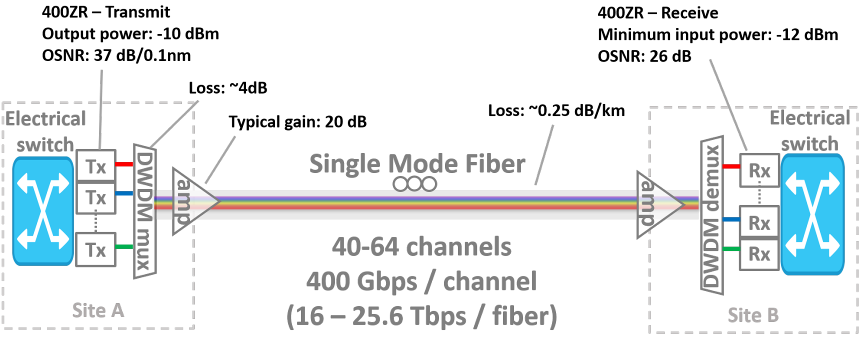


Fig. 2.8: Typical DCI optical link, components and 400ZR specifications.

2.3.2 Technology-rooted constraints

The constraints imposed by the physical characteristic of communication devices, and in particular optical communication components, has the critical impact on how we build and deploy DCI networks. Today's DCI is electrically-switched. That means that the network comprise point-to-point static optical links between any two sites (DC-DC or DC-Hub). Fig. 2.8 shows a typical example. We consider transceivers that plug directly into DC electrical switches (Tx,Rx in Fig. 2.8) [54], and in particular, the 400ZR transceivers (400 G, 16 QAM) [14], which have been standardized for DCI and are expected to be deployed soon across most providers. Dense Wavelength Division Multiplexing (DWDM) is used to combine $40 - 64$ optical signals at different wavelengths (colors), one per transceiver, covering the C-band.

On the receive side, optical signals need to respect the minimum optical power and optical signal-to-noise ratio (OSNR) thresholds given by transceiver specifications. The received *optical power* is dictated by the sending transceiver's transmit output power minus losses due to optical components in the link, such as the fiber and mux/demux elements. *OSNR* is affected by noise introduced by elements like amplifiers. Fig. 2.8 includes the details of expected 400ZR OSNR and power values, as well as typical

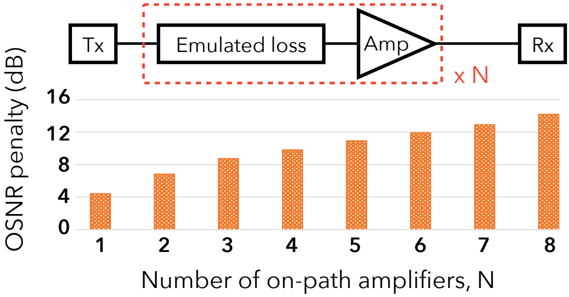


Fig. 2.9: OSNR penalty vs. amplifiers. The experimental setup (top) uses attenuators between amplifiers to match the amplifiers' gain.

losses for elements on point-to-point links. Any DCI architecture would need to respect these thresholds, which, in turn, lead to the following constraints.

TC1. **Optical link distance** — Optical amplifiers on both side of the link compensate for power losses, and have a typical gain of 20 dB. Thus, assuming a typical fiber loss of 0.25 dB/km [54], the receiving amplifier (Fig. 2.8) can compensate loss for a maximum DC-DC link distance of 80 km, absent in-line amplification.

TC2. **End-to-end amplifier count** — Additional in-line amplifiers between sites can increase reach (*e.g.,* up to 120 km) and/or allow for extra on-path optical components to enable reconfigurability. Unfortunately, amplifiers add noise, degrading the amplified signal's OSNR [62]. To quantify this, we measure the OSNR of transmitted signals at the output of multiple amplifiers in our testbed (Fig. 2.9). The first amplifier adds an OSNR penalty to the unamplified signal equal to the amplifier's specified noise figure (~4.5 dB). Beyond this, each doubling of the number of amplifiers on the line degrades OSNR by ~3 dB. The observed penalty agrees with theoretical models that examine the impact of cascaded amplifiers on OSNR [63]. With 400ZR, between sites, we can tolerate up to 11 dB OSNR penalty (Fig. 2.8). Allowing an additional couple of dBs for various transmission impairments and amplifier gain ripples,this translates to an amplifier budget of 9 dB, or a maximum amplifier-count of 3 end-to-end (Fig. 2.9). Thus, at most one extra in-line amplifier can be added in any reconfigurable physical layer design with maximum distance of 120 km.

TC3. **Power management** — When the optical network is (occasionally) reconfigured, the fiber spans part of a path can change. In turn, some optical amplifiers see their input power change, *e.g.,* if the input fiber span is now shorter, their input signal sees lower loss, and requires less amplification. Absent an adjustment in the amplifier's gain or proper management of the input power, the signal OSNR would be degraded. Unfortunately, adjusting the gain of amplifiers region-wide in a synchronized fashion would be severely limiting, as it can take several seconds for optical signals to stabilize [64]. Thus, appropriate management of input power to the amplifiers is mandatory in any architecture where the same amplifier compensates losses across different paths over time.

TC4. **Number of optical reconfiguration elements** — Components that allow optical reconfiguration also cause optical power loss, the degree of which depends on the components used. Reconfiguration can be achieved at two granularities: (a) at the fiber level, with all traffic from one fiber shifted to another, using optical space switches (OSSes) with up to a few hundred ports [65], [66]; and (b) at the wavelength level, shifting individual wavelengths across fibers, using a Wavelength Selective Switch (WSS) with at most a few tens of inputs. Large-scale wavelength-level switching requires combining individual components (de/mux and OSSes) into what is called an Optical Cross-Connect (OXC) [67], [68].

For a maximum distance of 120 km with one extra amplifier (*i.e.,* 40 dB total budget), after accounting for a fiber loss of 0.25 dB/km, we have 10 dB available for optical reconfiguration elements. OXCs and OSSes have typical losses of 9 dB and 1.5 dB, respectively. This translates to at most one OXC or 6 OSSes end-to-end.

2.3.3 Component costs and operational costs

Besides the above constraints, cost is also crucial in DCI design, especially given that each DCI deployment costs tens of millions of dollars and major providers have tens of regions.

Transceivers are the most crucial cost factor, given their large volume: each electrical port needs one. A DC-DC connection that carries λ wavelengths requires $2 \cdot \lambda$ transceivers. The transceivers used in our analysis are DWDM switch-pluggable transceivers like the 400ZR, or today's 100G equivalent [54] designed to cover DCI distances, *e.g.,* up to 120 km. Prices for such DCI transceivers are not public, with only volume-based prices offered to cloud providers. As a coarse reference point, vendors are estimating such DCI transceivers at roughly \$10/Gbps [69]; this implies an approximate cost of $\sim \$1,300$ per year after accounting for 3-year amortization. Note that while traditional long-haul coherent transceivers designed to cover thousands of kilometers may be used in DCI, their cost is several times the one of custom-designed DCI transceivers [69], and thus are not considered further in our analysis.

Fiber in regional networks is typically inexpensive because already laid out fiber ducts are abundant in metro areas. The caveat is that fiber cannot be arbitrarily added to minimize distances (§2.2). Fiber-pairs are priced per span, independent of distance, with lease price varying significantly across regions. A ballpark amortized figure is $\sim \$3,600$ per year [70]. Recall that a single fiber carries data from $40 - 64$ transceivers, which results in the total cost of deploying one 64 wavelength fiber being around $170,000$. Also note here that the cost of one fiber is lower than the cost of only 3 transceivers. This will be the essential observation in creating our new DCI architecture (§3).

OSS ports cost an order of magnitude less than one transceiver, *e.g.,* 100-200 dollars per (unidirectional) port [71].

OXC ports are slightly more expensive than OSS ports, due to the need for de/muxes, but still much cheaper than transceivers.

Amplifiers are equivalent in cost to a few transceivers. However, since each of them amplifies all the wavelengths in a given fiber, their contribution to the cost is not substantial.

Operational costs. While our quantitative analysis only accounts for component costs, we briefly comment on operational costs of two types: (a) network management; and (b) power and equipment space. Precisely appraising management costs is inherently hard, especially for novel, non-operational architectures like the one proposed in this thesis. Indeed, we expect that there will be some initial ramp-up cost for developing tooling to manage a novel architecture, but once done, steady-state management cost should be similar to or lower than today's designs across the entire spectrum of centralized to distributed DCI networks, on account of the reduction in the number of ports to be managed in our fully-optical design. Costs like power and space, on the other hand, are expected to be significantly lower with our solution: most of the optical devices used are passive, requiring orders of magnitude less power than an electrical fabric. In terms of space, optical switches with hundreds of ports are just a few rack-units in size [65], in comparison to the rack-size electrical switches needed at this scale.

2.4 Summary

In this chapter, we explored the trade-off between centralized and distributed topologies. We demonstrated that despite centralized architectures being simpler and more cost-effective implemented with today's technology, the demand for lower latency and larger siting flexibility moves cloud providers towards distributed approaches. Thus, in the following chapter, we present a new architecture, *Iris*, that lowers the cost of distributed DCI topologies and makes them practically feasible while satisfying all operational and technology-rooted constraints described in this chapter.

3

A NEW GENERATION OF DCI

As we have shown, distributed data center interconnect design has many desirable properties. It lowers the latency, which is essential for many cloud workloads, and increases data center siting flexibility, which has a critical impact allowing cloud providers to build new cloud regions in densely populated areas faster so that they can follow the ever-growing need for scaling modern applications and workloads. However, the cost and complexity of the distributed DCI design prohibit modern providers from deploying fully distributed DCI topologies today.

To lower the cost and complexity barriers in DCI network design, we propose *Iris*, which uses an all-optical circuit-switched network core. Thanks to the observation based on coarse-grained historical workload specifications that the traffic is stable at the DCI layer, *Iris* is able to leverage low-cost high-latency fiber-switching equipment and eliminates the need for in-network transceivers. Compared to today's electrical DCI networks, *Iris* simplifies network structure by reducing the total number of in-network ports. The resulting reductions in cost and complexity benefit networks on the entire spectrum from fully centralized to fully distributed, but are much larger for larger-scale regions and more distributed network designs. Thus, *Iris* makes more of the design space practicable, unlocking the latency and siting flexibility advantages of distributed networks while lowering their cost and complexity. Note that *Iris* substantially reduces, but does not completely ameliorate the complexity of distributed design, which, with *any* architecture, necessitates the management of in-network equipment across multiple sites, instead of just two hubs. But if the pressure by cloud tenants and their workloads for low latency persists, a shift towards distributed designs may be inevitable.

Iris exploits two key observations: (a) DCI cost is dominated by the specialized electrical-optical transceivers needed for covering DCI distances, and (b) regional fiber is abundant and cheap relative to transceiver cost. *Iris* design thus makes an extremely favorable cost trade: some additional fiber in exchange for vastly reducing the number of transceivers. To exploit this cost structure, *Iris*'s all-optical approach gives up the finer switching granularity of packet switching in favor of coarser optical switching.

Although simple in terms of the number and type of components, *Iris* introduces an additional complexity dimension that is not present in electrical DCI architectures due to their fine-grained switching capabilities - *Iris* requires dynamic component reconfiguration under changes in traffic patterns. Even though these network reconfigurations may become challenging in a general case and cause partial network downtime, we show that they will not have significant negative impact on the performance of today's cloud applications. By looking at the long history of execution of modern cloud applications and their communication patterns, we know that aggregated traffic between data centers is relatively stable over time, which gives *Iris* enough time to reconfigure the network. However, to truly eliminate the danger of traffic changes for future cloud workloads, *Iris* should not only look at the history of workload specifications in the cloud but also be integrated with *Flux* (§5), a system that can automatically obtain fine-grained workload specifications about the future behavior of applications and communication patterns, so that the network can prepare and execute the reconfiguration ahead of time.

While optical switching is well-studied for both intra-DC and DC-WAN networks, the constraints of regional DCIs provide unique challenges and opportunities. Unlike intra-DC optics [72]–[75], fast reconfigurability is not necessary as the traffic is slow-changing; the challenges rather stem from the physical layer, which needs to ensure that the budgets of optical devices for power and signal quality are respected across a wide range of distances, and through a varying number of optical switches. On the other hand, while optical DC-WAN networking accounts for even more stringent physical-layer constraints due to the long and diverse distances, the solutions there typically involve optimizing spectral efficiency and switching at the wavelength granularity, *e.g.,* OWAN [76]. For DCI networking, we find that this is more complex than necessary. Instead, *Iris* only switches capacity at fiber granularity using relatively simple fiber switches, thus requiring minimal support from the physical layer. We find that wavelength switching is *more* expensive for DCIs, making *Iris* the preferable solution in both cost and complexity.

Using testbed experiments, augmented with large-scale simulations, we evaluate the benefits and feasibility of *Iris*. We find that *Iris*: (a) can be implemented using off-the-shelf hardware; (b) involves limited reconfiguration that does not hurt application-layer performance; (c) enables the latency and location-flexibility advantages of the distributed approach; (d) allows the distributed approach to be implemented at a cost within 1.1× of a traditional centralized approach, and in fact, *cheaper* than it in more than 98% of the settings examined; and (e) reduces network complexity by reducing the total number of ports, electrical or optical, that need to be managed.

Outline In this chapter, we first provide a high-level motivating example that shows the benefits of using an all-optical design compared to simple electrical switching (§3.1). Section §3.2 describes how to provision DCI topology irrespective of switching technology being used in the process. Sections §3.3, §3.4, §3.5 show how a particular topology can be implemented using electrical switching, wavelength switching, or *Iris* respectively, while §3.6 introduces a hybrid design that combines *Iris* with the wavelength switching design to reduce the fiber overhead at the cost of slightly increasing the network complexity. Section §3.7 provides experimental details related to the complexity, cost, and physical feasibility of our approach. Section §3.8 gives an overview of related work in other domains that exploited similar design ideas to *Iris*, while §3.9 discusses some of the limitations of our work and how to further mitigate those limitations.

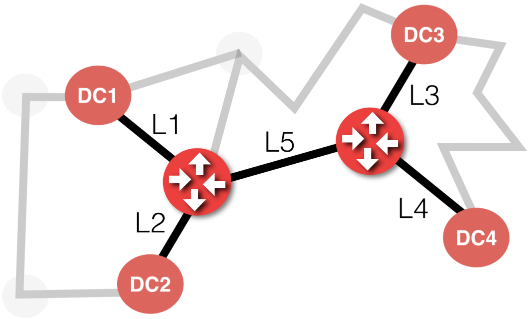


Fig. 3.1: An example fiber map with data center placement. Assuming shortest-paths the dark highlighted links are only used.

3.1 Cost comparison: a motivating example

To motivate our all-optical design strategy, we use a small, toy DCI design example, with a fixed topology implemented both ways, *i.e.*, using either a traditional electrical approach, or *Iris*'s all-optical approach. The topology used is the same semi-distributed one in Fig. 2.1(e), but is redrawn with labeling in Fig. 3.1. DC1 and DC2 connect to one hub and DC3 and DC4 to another. Each of the 4 DCs has a capacity of 160 Tbps. With 400 Gbps for each of 40 wavelengths, this translates to $f = 10$ fiber-pairs.

For the electrical design, L1-L4 each carry 10 fiber-pairs, so each DC's full capacity is connected to its hub. L5 carries 20 fiber-pairs, such that the network is non-blocking. The total number of fiber-pairs is thus $F_E = 60$, and the number of transceivers is $T_E = 2 \cdot F_E \cdot \lambda = 4800$, as each fiber terminates in a transceiver.

With *Iris*, transceivers are needed only at the DCs, *i.e.*, $T_O = 4 \cdot 10 \cdot \lambda = 1600$ transceivers. However, for optical switching in the network, *Iris* uses additional fiber and OSS ports. §3.5 details how this is done, but in this specific example, L1-L4 need 3 additional fiber-pairs, and L5 needs 6 additional fiber-pairs. The total number of fiber-pairs thus increases to $F_O = 78$. Each fiber-pair terminates at OSS ports at both ends, so 312 OSS ports are needed in total.

Using the prices described in §2.3.3, the electrical design costs 2.7× more than the optical one.[1] This difference is rooted in the fact that transceivers are the overwhelming expense: an OSS port costs an order of magnitude less than a transceiver, and while one fiber's cost is a few times that of a transceiver, the absolute number of fibers needed is nearly two orders of magnitude smaller. Thus, using some extra fiber and OSS ports to reduce the number of transceivers is a very profitable trade.

Iris's advantage is greater for larger regions, and for more distributed topologies. Thus, *Iris* enables cost-effective networking for larger-scale regions with the favorable characteristics of distributed topologies. Our detailed analysis (§3.7) using real fiber maps and cloud-provider component costs shows that *Iris* would be 7× cheaper in the median than an electrical switching implementation.

3.2 Topology & capacity provisioning

As previously discussed, planning a regional DCI network entails using the region's fiber map and data center locations and capacities, to decide on the topology, fiber capacity of each connection, and the use of switching to implement the topology and capacity decisions.

We first jointly address topology and capacity, as these derive primarily from operational constraints, and are *largely* the same regardless of switching. For optical switching, meeting the technological constraints sometimes requires revisiting topology and capacity decisions; we discuss such cases separately in §3.5.

---

[1] As other costs are much smaller, accounting for only fiber and transceivers arrives at nearly the same number, *i.e.,* $(1300T_E + 3600F_E)/(1300T_O + 3600F_O) = 2.73$.

We use a natural graph abstraction: DCs and huts are nodes of graph G, and the available fiber forms edges between them. Fiber edges longer than 80 km can be excluded right away: regardless of electrical / optical switching, longer *point-to-point* connections are not possible (*TC1*). Our task then is to decide which subset of edges are used, and at what capacity. Algorithm 1 achieves this by computing which links lie on shortest paths (*OC1* and *OC3*) in *any* failure scenario (*OC4*) by exhaustively enumerating the latter.

Algorithm 1: Topology & capacity planning.

```
$\mathsf{G}_{init} \leftarrow$ fiber map
$\forall$ edge $\mathsf{e} \in \mathsf{G}_{init}$: $\mathsf{capacity}_e \leftarrow 0$
foreach failure scenario do
    $\mathsf{G} \leftarrow \mathsf{G}_{init} \setminus$ failed fiber ducts
    $\mathsf{SP} \leftarrow$ {shortest paths in $\mathsf{G}$ $\forall$ pairs}
    foreach edge $\mathsf{e} \in \mathsf{G}$ do
        $\mathrm{sp}_e \leftarrow \{\mathsf{sp} \in \mathsf{SP} \mid \mathsf{sp}$ uses $\mathsf{e}\}$
        $\mathrm{G}_e \leftarrow$ construct flow graph for $\mathsf{e}$ using $\mathrm{sp}_e$
        $\mathsf{capacity}_e \leftarrow \mathtt{max}(\mathsf{capacity}_e,$ max flow of $\mathrm{G}_e)$
```

Determining which edges are used is trivial, but assigning their capacities is not. Since each DC-pair uses only its (typically unique) shortest path, one may naively assume that to support the hose traffic model (*OC2*), the capacity of each edge is simply the sum of demands for DC-pairs traversing it, where a DC-pair's demand is the minimum of the two DCs' capacities. However, this leads to needless over-provisioning: *e.g.,* a DC, say A, may be part of multiple DC-pairs, say A-B and A-C, traversing an edge over shortest paths; this naive approach would double-count A's capacity for this edge. A precise solution to capacity provisioning requires a max-flow computation across an appropriately constructed "flow graph". We adapt this from prior work [77], and thus omit the details.

Algorithm 1 yields not only edge capacities, but also which fiber huts are used: if a hut has no edges of non-zero capacity, it is unused. Thus, it fully determines the network's topology and capacity. Note that if shortest paths are unique, as is typically true across real fiber maps, Algorithm 1 yields the unique (and hence optimal) solution for topology and capacity planning: only one set of chosen huts and edges meets the constraint of achieving shortest paths under all failure scenarios (*OC3* and *OC4*). For settings with multiple shortest paths, or when the shortest path constraint is relaxed, this is only a heuristic that still meets all constraints, but does not necessarily provision the minimal infrastructure.

We next discuss three granularities for switching, the last decision needed to fully describe DCI planning, drawing out the reasoning for picking *Iris* as the choice for optical fiber switching.

3.3 Electrical packet-switched network

Given the topology and capacity provisioning, an electrical packet-switched (EPS) fabric is simple to build: just deploy enough switching capacity at the DCs and huts using standard Clos networking techniques. Each fiber is terminated with $2 \times \lambda$ transceivers, λ on each side. Transceivers are used to send out traffic in the form of an optical signal to traverse long distances, but also to convert the same signal back to the electrical domain for fine-grained packet switching at DCs or huts. Although this conversion happens with little to no impact on network performance, As noted in §3.1, the key impairment of the electrical approach is its cost: it requires a large number of electrical ports and transceivers, directly proportional to the number of wavelengths per fiber terminated at each fiber hut.

3.4 Pure wavelength-switching

Instead of relying on fine-grained electrical packet switching to build DCI networks, cloud providers could leverage wavelength switching using an optical cross connect (OXC) architecture [67], [68]. To eliminate the need for expensive transceivers, and switch traffic purely in the optical domain, cloud providers could just replace every electrical port with a significantly cheaper OXC port, and keep the rest of the architecture almost unchanged. Wouldn't such a design be *obviously* superior? Surprisingly, the answer is no. We analyzed this precisely, and here we only summarize the reasoning behind this result:

- With at most one OXC switch on path (*TC4*) and only one amplifier per path (*TC2*), it is not feasible to benefit from wavelength switching in many settings.
- Wavelength switching adds complexity, requiring the solution of a graph-coloring problem to avoid wavelength collisions.
- Even ignoring the above two issues, and using settings favorable to wavelength switching (*e.g.,* large n=20), at today's prices, the additional components needed for wavelength switching are pricier than the additional fibers our fiber switching design *Iris*, which we describe next.

Although impractical for implementing the entire DCI design with today's technology, wavelength switching can still be used to reduce the fiber overhead created by *Iris*. This hybrid approach where the majority of the traffic is switched using fiber switching technology, and a small part of it still leverages wavelength switching is described in §3.6.

3.5 *Iris* - optical fiber-switched network

To avoid the explosion of electrical ports, *Iris* uses an all-optical network core, *i.e.,* data does not leave the optical domain except at end-points. By leveraging fiber-switching technology, this approach can provide the substantial benefits of a distributed DCI network at cost similar to a centralized one. At each hut, only optical space switches (§2.3.2) are used to direct all wavelengths carried in a fiber from one port to another, thus reducing port requirements to *one per fiber*. This effectively sets up DC-DC optical circuits through the network. However, this requires deploying appropriate optical equipment at intermediate DCs and huts to address three problems:

- Coarse-grained fiber switching needs more network capacity than computed above in §3.2.
- Since DC-DC data streams travel end-end as optical signals, we must deploy amplification as necessary for the now longer distances (*TC1* and *TC2*).
- We must limit the number of optical switches on each end-to-end path (*TC4*).

The latter two problems are self-evident, based on our earlier description of technology constraints in §2.3.2, but the first is a significant challenge of fiber switching, and requires some explanation. The need for additional capacity stems from the coarse granularity: while for EPS fabrics, integer number of wavelengths (as we assume DC capacities are specified in) can be flexibly switched, fiber-switching requires rounding to the fiber-level. Consider a DC that has a capacity equivalent to z fibers, and sends x and y to two DCs, such that $x + y = z$, but y comprises only a fraction of one fiber's capacity, such that $x = z$. Switching at fiber granularity implies that we now need $z + 1$ capacity from the DC. Worst-case scenarios, which we want to tackle per *OC2*, necessitate that for each DC-pair, one additional fiber is necessary to address this issue, increasing fiber cost by $n \cdot (n - 1)$ fibers for a region with n DCs. Note though, that no additional transceivers are needed: transceivers at the DCs can still be multiplexed across the fibers as necessary. Overall, we find that this is a highly favorable trade-off.

For the second problem, amplification, we use a heuristic to ensure that no umamplified segment exceeds our distance constraint (*TC1*), and each path has at most one amplifier (*TC2*). Our heuristic also tries to greedily reduce the number of amplifiers. The intuition is to examine each failure scenario, identify paths that need amplification, score each potential amplifier location in terms of how many paths it would meet constraints for, add amplifiers as needed to the highest-scoring location, and iterate until the constraints are met.

For the third problem, limiting each path's switch-count (*TC4*), we use a similar greedy approach. For each path with > 6 switching points, we add "cut-through links" that replace one or more switch-points for the path with an uninterrupted fiber between the endpoints of the replaced segment, with adequate capacity for that path. We again attempt to minimize such cut-throughs, by finding ones that resolve constraints for multiple paths.

Put together, the above solutions for capacity provisioning, amplification, and cut-through placement, meet all our constraints. Our heuristics use exhaustive enumeration across failure scenarios, and several iterations by making reassessments after placing each amplifier or cut-through, but still execute within a few minutes for even large region sizes with 20 DCs. Given that this process only executes once for network provisioning, this is sufficiently fast, and as we show later, provides significant cost reduction (§3.7).

3.5.1 Amplifier and cut-through link placement

We have to guarantee that transceivers from each data center can reach any other data center in the region. There are two reasons why this could be a potential problem. First, the distance between two sites can be longer than 80 km and the signal requires in-network amplification, and second, there may be too many switching points on the path that reduce the signal power below the power threshold for the receiver.

These problems can be resolved by placing in-network amplifiers (at most one per path) or building "cut-through links" that traverse the switching point without being interrupted (switched), and thus reduce the power loss. Amplifier placement can solve both problems in some situations. Even if the distance is short, but there are many switching points on the path, it may make sense to place amplifiers and increase the signal power instead of building cut-through links that reduce the power loss, because the number of amplifiers needed could be cheaper compared to allocating additional fiber for cut-through links.

Note that there is always *a* configuration that meets all constraints because no links longer that 80 km is allowed in the topology in the first place. This means that a path of 120 km can always be divided into two segments where each of them is not longer than 80 km.

Our goal is to meet all constraints by minimizing the cost. An optimal solution would require exploring every possible combination of failures, amplifier placement and cut-through links. This problem has combinatorial complexity since for each path of h hops, there are 2^h potential cut-through links to be built.

To simplify the process, we place amplifiers using a greedy heuristic described in Algorithm 2. For every failure scenario, we identify all paths that are long and require amplification. For each path, we find all candidate locations where amplifier placement can resolve the power budget constraint. Since one amplifier can amplify only one fiber, the total number of amplifiers needed in a particular location is calculated from the maximum demand on all paths that require amplification at that location, similarly to the maximum capacity calculation in §3.2. We also calculate how many of these long paths suffer from too many hops that reduce the signal power and if amplifier placement at a particular location would resolve that constraint as well. Then, we assign a score to each location based on the total number of constraints resolved versus the total number of amplifiers needed. Finally, we place amplifiers to a location with the maximum score and repeat this process as long as there are paths that require amplification.

After the amplifiers have been placed, there can still be paths that have too many hops that cause the signal power to drop below the acceptable threshold. Thus, we apply a similar heuristic as for the amplifier placement to place cut-through links and avoid fiber switching at every hop. To do that, we introduce the concept of a segment. If a path does not have an amplification point, the segment is equivalent to the path. However, with an amplification point, a path has two segments, one between the source and

Algorithm 2: Algorithm for amplifier placement

```
foreach failure scenario do
    P ← {long paths that require amplification}
    while size(P) > 0 do
        S ← {possible amplifier locations ∀path∈P }
        foreach location ∈ S do
            noa ←# of amplifiers needed at location
            noea ←# of amplifiers already at location
            /* # of amplifiers to be placed                          */
            ntbp ←max(0, noa - noea)
            nop ←# of paths resolved by placing amplifiers at location
            nhop ←# of paths that resolve the n-hop constraint by
              placing an amplifier at location
            location_score ← $\frac{nop + nhop}{ntbp}$
        mloc ←the location with maximum score
        place amplifiers at mloc
        P ←P −{ paths resolved by mloc }
```

the amplifier, and one between the amplifier and the destination. For each segment that has too many hops, we calculate all possible cut-through links that would resolve the power constraint on that segment. Similarly to the previous heuristic, we assign a score to every cut-through candidate based on the number of paths that can utilize the link versus additional fiber needed for that particular link. The cut-through link with the highest score is added to the topology and the process starts again as long as there are segments that have the power budget problem.

The proposed heuristics may not provide an optimal result in terms of cost but they guarantee that all constraints will be met. First, the amplifier placement algorithm assures that there are no long links that require amplification because of distance. Following that, the cut-through placement heuristic guarantees that the distance between source/destination and the amplification point can be bridged with a sufficient power budget.

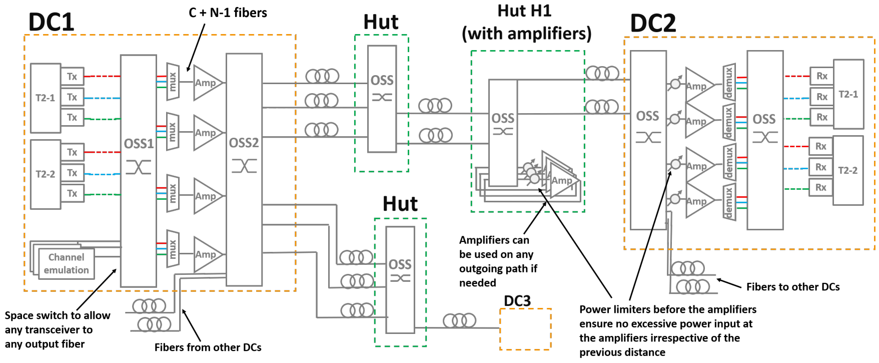

DC1
C + N-1 fibers
Hut
Hut H1
(with amplifiers)
DC2
T2-1
Tx
Tx
Tx
T2-2
Tx
Tx
Tx
OSS1
mux
Amp
OSS2
Channel emulation
OSS
Hut
OSS
OSS
Amp
DC3
OSS
Amp
demux
OSS
Rx
Rx
Rx
T2-1
Rx
Rx
Rx
T2-2
Space switch to allow any transceiver to any output fiber
Fibers from other DCs
Amplifiers can be used on any outgoing path if needed
Power limiters before the amplifiers ensure no excessive power input at the amplifiers irrespective of the previous distance
Fibers to other DCs

Fig. 3.2: *Iris* puts together available commodity components in a manner that respects all the technology constraints, while still meeting our operational goals.

The cost overhead due to additional amplifiers and cut-through links using the described heuristic is 3% on average (8% in the worst case) compared to the total network cost across all test scenarios.

3.5.2 *Iris* physical implementation

We next discuss physical implementation of *Iris*: how different components connect to each other, and how they are managed.

Fig. 3.2 shows a full-system view with the details for 2 of the N DCs drawn out, showing the send/receive parts respectively.

Sending from DC1: DC1's internal Clos fabric sends outgoing traffic to its tier-2 (T2 or core) switches. Internal routing to T2 switches can be achieved using standard mechanisms like ECMP and anycast [78], such that traffic for each external destination arrives at the right T2(s) in a load balanced fashion. Each transceiver at each T2 converts this traffic to a wavelength; Fig. 3.2 shows 3 transceivers / wavelengths for each of the 2 T2s. These wavelengths are mux-ed into fibers (via OSS1), which are then switched towards destination DCs (using OSS2). OSS1's function is allowing any T2-transceiver to be fed into any fiber – thus instead of directly mux-ing wavelengths from T2s, they are first fed into OSS1, whose outputs are then mux-ed. *Iris* uses tunable transceivers at T2s, such that colors can be assigned to each transceiver to make it trivial to pack them into outgoing fibers. After each fiber is packed, it goes through amplification. OSS2 acts like any other switching point; it switches both: (a) DC1's outgoing capacity of C fibers, plus $N-1$ fibers to address the "fractional" capacity; and (b) any fibers this DC transits for other DC-DC traffic (bottom-left in Fig. 3.2).

Intermediate switching: Fiber huts and on-path DCs performing intermediate switching use only an OSS and amplifiers. As noted earlier, amplifiers can be used by any paths passing through. Implementing this in a configurable manner requires using amplifiers in a "loopback" fashion, whereby their input *and* output are both attached to the OSS, such that an arbitrary fiber can be directed through the OSS to the amplifier, fed back into the OSS post-amplification, and then switched to an arbitrary OSS output. (See hut H1 in Fig. 3.2.)

Receiving at DC2: The receive side largely mirrors the send. Besides passing the traffic destined to this DC to demux-es and finally transceivers (after amplification), fibers destined to other DCs can be switched towards them by the OSS.

Amplifier power management: The above implementation, if based on appropriate provisioning (§3.2), suffices to meet all our design constraints except one: amplifier power management (*TC3*, §2.3.2). We use two methods to ensure that amplifier gains do not need careful management. First, we transmit the full C-band spectrum per fiber, *i.e.,* all wavelengths, even if only some carry data. Doing this using transceivers would be expensive; instead we use amplified spontaneous emission (ASE) noise to fill only the unused spectrum that is then combined through muxes with the "live" channels ("Channel emulation" in Fig. 3.2). This ensures uniform gain profiles across fiber segments of equal length regardless of their "live" channels. Second, we operate all amplifiers at a fixed gain irrespective of the fiber length that they compensate. To ensure that no excessive power reaches the following component in the physical link (*i.e.,* the next amplifier), we use a power limiter before each optical amplifier to bound its input optical power. These are one-time design decisions rather than continual online management.

Regional IP routing and WAN traffic Note that irrespective DCI implementation cloud providers choose, regional IP routing and WAN transit remain the same as today. The higher tier of each DC has full regional route visibility, and a few DCs transit WAN traffic. (Note: WAN traffic is a small fraction of regional traffic.).

3.5.3 Dynamic reconfiguration

One potential challenge in deploying our solution is that *Iris* requires reconfiguration of optical components under changes in traffic patterns. That means a subset of links must be drained and turned off for a short amount of time. Although under frequent changes with high amplitude, the downtime could substantially degrade the network performance, these big traffic changes are not that common in today's workloads. The reason for that is that DCI links have capacity much higher than the communication need of any individual application, so particular changes caused by a single application are not immediately visible on the aggregated traffic.

Previous work supports our insights about traffic stability [11], [79]. For instance, the traces collected at Facebook's data centers indicate that the inter-datacenter traffic changes less than 2% on average every second, which would give *Iris* enough time to make smooth reconfiguration without significant performance impact. [11].

Although rapid changes in traffic patterns are not frequent at the layer of DCI, all our key design decisions are still geared towards reducing complexity to make reconfiguration, when it is necessary, a straightforward process:

- fiber switching based on only simple capacity needs

- basic wavelength management separately in each DC
- no online power management for amplifiers or any other optical component

***Iris* controller:** A centralized controller gathers DC-DC traffic demands, and configures the network components appropriately. The small number of sites with only tens of fibers per site, coupled with relatively infrequent reconfigurations, simplify the control problem greatly, especially in comparison to systems using optical reconfiguration in other settings like WAN optimization [76] or data centers [73].

When the controller decides that a reconfiguration is needed, it first drains traffic from paths that need to be torn down. It then reconfigures the OSSes network-wide to enable new paths. The configuration of transceiver wavelengths, and channel emulation is done independently at each DC.

Reconfiguration time: OSSes are the bottleneck here. While tunable transceivers can switch wavelengths in under 1 ms [74], [80], and unused amplifiers can provide gain in under 2 ms [81], [82], the state of the art for OSSes is ∼20 ms [83]. In the future, we expect sub-ms switching for OSSes [84].

Ahead of time reconfiguration: The main insight on traffic stability that made *Iris* possible today is completely based on historic observations about traffic patterns. Thus, the *Iris* controller responsible for network reconfigurations can only react *after* the traffic changes. Although the impact of that *late* reconfiguration is minimal, as we will show in §3.7.3, ideally, cloud providers would like to avoid any potential performance impact and make changes to network configuration ahead of time. Such an *early* reconfiguration system would only be possible if we manage to obtain workload specification about future network behavior of cloud applications. In that context, systems like *Flux* can help us to know the bandwidth requirements in advance, and thus, make early network recofigurations (§5).

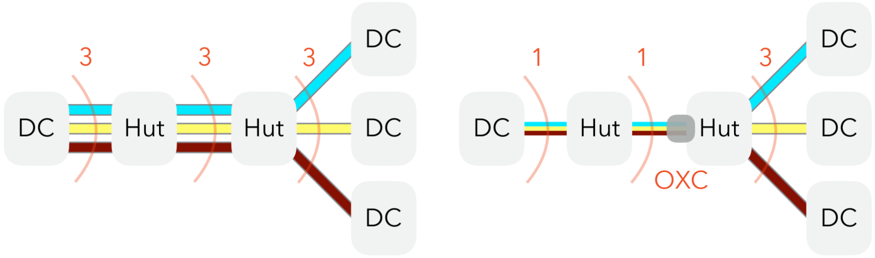


Fig. 3.3: (left) Iris requires having one fiber DC pair that causes the total overhead of $O(n^2)$ fibers; (right) Hybrid design reduces that overhead by combining multiple residual links using wavelength switching.

3.6 Hybrid DCI design

While *Iris* fiber switched network is many times cheaper than an electrical packet-switching fabric, one may wonder if the n^2 fiber overhead of coarse-grained fiber switching can be avoided. To further reduce expense, cloud providers could combine course-grained fiber switching with finer-grained *wavelength* switching technology. In this approach, fiber switching could be responsible for handling most of the traffic, while wavelength switching addresses fractional demands.

While indeed, this hybrid approach can provide savings in terms of fiber compared to a pure fiber-switching network, these savings are entirely negated by the cost of additional equipment required for wavelength switching (see §3.7). It also adds substantial complexity, which would deter deployment. We thus conclude that fiber switching is the most viable switching architecture for regional DCI networks.

Although the hybrid approach is relatively inefficient given today's component costs and the number of data centers, this situation may change in the future. Suppose the price of network metro fiber increases compared to the cost of optical switching components, or the number of connected sites dramatically increases. In that case, the fiber savings provided by the hybrid design may become very valuable.

Hybrid design. To support any traffic matrix, a fiber switched network requires n^2 additional links to carry residual capacity. These links only serve fractional capacity that cannot be accommodated in the base fiber. Intuitively, many of these links could be combined using a finer-grained wavelength switching technology, and thus, reduce the fiber overhead, as shown in the example in Fig. 3.3. Residual capacity to different destinations can be combined at the source data center and carried in one fiber to a particular fiber hut on the shortest path for all combined wavelengths. At the fiber hut, the wavelengths are separated and carried through dedicated fibers to different destinations. The same process applies in the opposite direction as well – residual wavelengths to the same destination can be combined at a fiber hut and carried through one fiber to the common destination.

If we combine x residual fibers into one, we have to guarantee that these x residual fibers combined cannot exceed the capacity of one physical fiber – λ wavelengths.

Observation 1. *Any two residual fibers coming from the same source can be combined into one fiber.*

To show this, we have to define the concept of base capacity. The base capacity is the capacity that has to be provided to satisfy operational constraints defined in §2.3.1, regardless of the technology used for implementation. *Iris* requires the base capacity plus n^2 residual links. The base capacity links are always fully saturated with λ wavelengths. If the demand to a particular destination is less than λ, the traffic is carried through a residual link.

If two residual fibers carry more than λ wavelength, it means there is at least one fiber provisioned among those in the base capacity. Then, the residual capacity to one destination will be transmitted through one fiber from the base capacity and the other one remaining residual fiber. This result enables a simple optimization that should reduce the n^2 fiber overhead to close to $n^2/2$. However, we show we can potentially save even more.

Observation 2. *Any n residual fibers coming from the same source can be combined into $n/4$ fibers.*

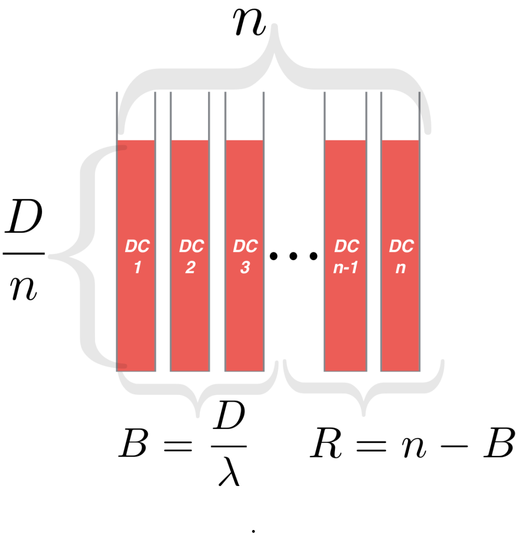


FIG. 3.4: Illustration of the worst-case residual capacity allocation. The total capacity that will be carried over residual links is equivalent to $R \cdot D/n$

Let us assume a data center can reach n destinations (n residual fibers). Assume that the aggregated traffic demand from this data center to all destinations is D wavelengths. Without loss of generality, we assume that $D \leq \lambda \cdot n$, where n is the total number of destinations (for larger D, the difference would be carried through the base capacity). We want to calculate the maximum capacity that *must* be carried through the residual fiber. n residual fibers are shown in Fig. 3.4. By the definition of base capacity, we know that the base capacity provisions at least $B = D/\lambda$ fibers available, and the rest must be transported through the residual fibers. Since there are n destinations, we will need to provision $R = n - D/\lambda$ residual links atop base capacity.

We are looking for a traffic matrix that maximizes the capacity carried over these R links. For any traffic matrix, we take the following approach: we sort all demands to n destinations. The largest B fibers will be scheduled using the base capacity, and the remaining part will go through the R fibers. The total demand in R is maximized if every link carries exactly the same capacity

D/n. Thus, the total capacity carried over residual links is $(n - D/\lambda) \cdot D/n$. This function has the maximum for $D = \lambda \cdot n/2$ and the total worst-case capacity on R fibers is $\lambda \cdot n/4$ wavelengths.

This further means that any n residual links coming from the same source will carry at most $\lambda \cdot n/4$ wavelengths. Since each fiber can carry at most λ wavelengths, this means that given residual capacity can be compressed into:

$$\frac{\lambda \cdot n}{4} \cdot \frac{1}{\lambda} = \frac{n}{4}$$

Note that the theorem holds for residual wavelengths that have the same source, as well as for those that have the same destination. This result allows us to merge any four residual fibers with the same source/destination.

However, two additional challenges prevent us from minimizing the fiber overhead by a factor of 4:

- Optical devices used to pack/unpack wavelengths from the fiber introduce significant signal power loss. Thus, we can afford to have only one wavelength switching device per path.
- Two or more residual fibers can be combined only if they share a subpath from the source / to the destination. For instance, in a distributed network with many direct connections, little fiber can be saved because only a few paths share a subpath.

Note that the devices used for packing and unpacking wavelengths have to be active and dynamic because there are different combinations of residual capacity that these devices have to handle. These combinations change over time, depending on the traffic matrix.

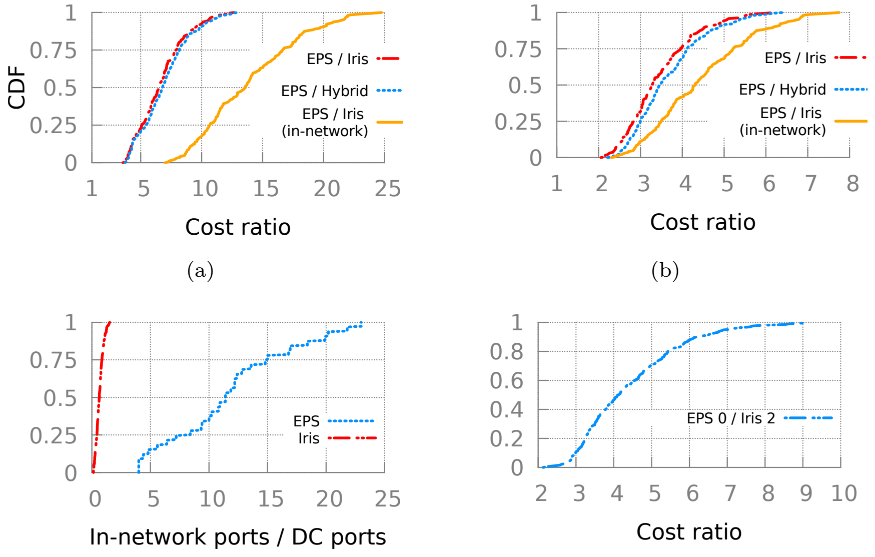
The remaining step in designing a hybrid network is to decide if and where residual fiber links will be aggregated. This problem has similar properties to amplifier placement and cut-through link placement, so we take a similar approach. We compute all possible placements for wavelength switching devices. We give each solution a score based on the potential fiber saving, pick the location with the highest score, place the devices, and repeat this process as long as any fiber saving can be achieved. In our test scenarios, this approach managed to reduce the residual fiber overhead by approximately 50%, which brings some cost reductions, as describe in in the following section. Still, this is not enough to justify the complexity of managing a network with one more type of devices at the current prices. However, we envision that this hybrid design could be the first step toward a more sophisticated solution with less fiber overhead.

3.7 Evaluation

Iris, by design, meets the constraints specified in §2.3. In §2.2, we further demonstrated its latency and flexibility improvements over the centralized approach in real settings. We thus evaluate three aspects that bridge any potential gap between the system's abstract design and its practical realization: (a) cost; (b) physical layer feasibility; and (c) impact of circuit switching under fluctuating traffic.

3.7.1 Cost analysis on real fiber maps

We use 10 real region fiber maps with a randomized placement of $n \in \{5, 10, 15, 20\}$ DCs across each map: the first DC is placed uniformly at random in the service area, and each successive DC is placed randomly (in the more restricted service area given reach from already placed DCs) with the probability of a candidate location being inversely proportional to its distance from the nearest already placed DC. In line with typical values [54], we vary DC capacities in terms of number of fibers, $f \in \{8, 16, 32\}$, where each fiber translates to λ transceivers per fiber, with $\lambda \in \{40, 64\}$. For each of the 240 combinations of these inputs, we calculate the cost of *Iris*, and equivalent EPS (§3.3) and hybrid networks (§3.6). We account for all network



(a) (b) (c) (d)

FIG. 3.5: *Iris* is substantially cheaper: (a) Relative cost of *Iris*, EPS, and hybrid networks across all 240 scenarios. (b) Same as (a) but with DCI transceiver cost assumed (unrealistically optimistically) equal to SR transceivers. (c) EPS uses many more in-network ports, as shown by the ratio of in-network to DC ports across designs. (d) Relative cost of an EPS supporting no failures vs. *Iris*, which handles up to 2 failures.

components with appropriate price amortization (§2.3.3), and the number of ports per hut can be accommodated with today's OSSes.

Fig. 3.5(a) shows the resulting cost comparison in relative terms: in 80% of the examined scenarios, the EPS network is $\geq 5\times$ more expensive than the *Iris* and hybrid networks. Further, the virtually identical costs of the latter two justify *Iris* choice of simpler fiber switching. This analysis includes the transceivers within the DCs, which are fixed across the design space. A sharper contrast between the design choices is revealed when we exclude this fixed cost and only evaluate in-network components. *Iris* cost is then 10× lower for 80% of the scenarios (the "in-network" line in Fig. 3.5(a)).

We also emphasize that these cost differences are not ephemeral. The involved components are all commoditized and high-volume, so we believe this analysis to be fair. Nevertheless, to emphasize the disparity between *Iris* and alternatives, we also examine the potential impact, were DCI transceiver prices to drop (unrealistically) to those of short reach transceivers (presently used for sub-2 km). Fig. 3.5(b) shows that *Iris* would still have a substantial cost advantage. The reason is the number of ports (optical or electrical) needed in different systems: as Fig. 3.5(c) shows, an EPS fabric requires many times more ports in-network than *Iris*.

Finally, per Fig. 3.5(d), *Iris*, which guarantees capacity under up to two failures, is cheaper (by >2× across *all* scenarios) than even an EPS with no guarantees under failures.

3.7.2 Physical layer feasibility

Fig. 3.6 shows our testbed implementation of *Iris*, which uses all the components described in §3.5: Multiple fiber spools 5 km to 50 km in length, that allow us to model any regional distance at a granularity of 5 km; Erbium-doped fiber amplifiers from Ciena; OSSes from Polatis, which also provide per-port power-limiting functionality, arranged to model DC OSSes as well as two fiber huts; WSSes from Finisar used to mux/demux wavelengths; Channel emulator from BKtel to fill unused spectrum; 4 Acacia tunable transceivers (2xAC400, 2xAC200) that can support varying baud-rates, modulation formats, channel grid spacing, etc. These are not switch-pluggable but controlled via evaluation boards, allowing us fine-grained config to emulate the 400ZR specification.

We have also implemented a software controller (in Python, $\approx$ 9K LoC) to control the optical devices through a multitude of interfaces (serial port, HTTPS, and NetConf/REST). Our controller implements APIs for all operations of *Iris* optical layer, namely channel add/drop, reconfiguration of space switches, checking that the devices are in the expected state, etc. Our present testbed evaluation focuses on physical layer feasibility, which then

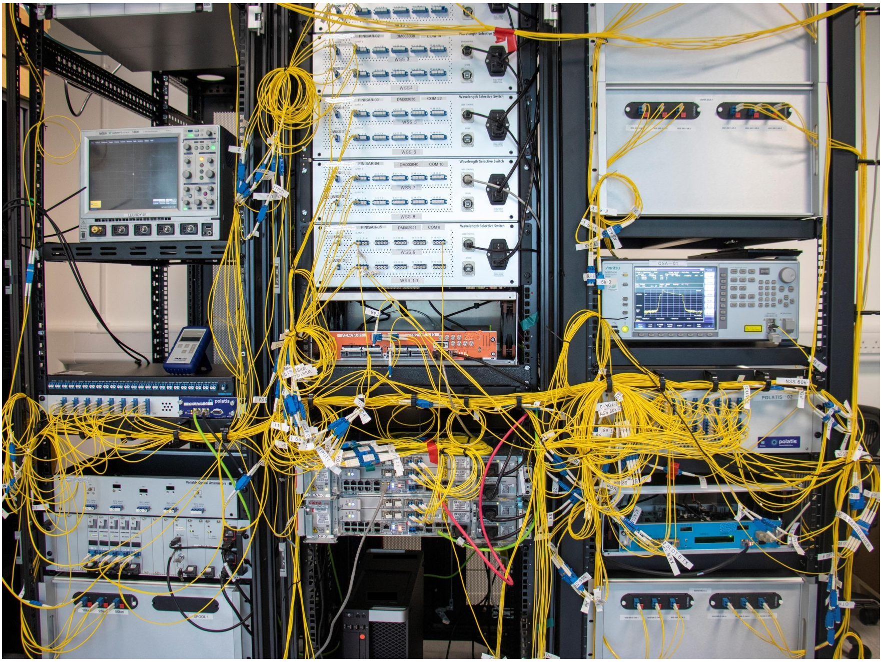


FIG. 3.6: A small *Iris* testbed with all the optical components.

guides our large-scale simulations. Unfortunately, given that our transceivers are controlled through evaluation boards instead of real switches, we cannot run a full control-to-bits evaluation at this time.

Our experimental set-up is shown in Fig. 3.7 and matches the description in §3.5.2. We emulate 3 DCs, one sending traffic to the other two, over two distinct paths of 4 fiber spans in total. We switch the two paths at an intermediate hut. Our high-level description below is targeted at most networking readers, with details for optics experts in Appendix A.1.

We generate 4 optical signals at two different wavelengths together with ASE noise to fill the C-band spectrum in DC1. This traffic is fed into 2 fibers, each carrying the 2 different wavelengths, muxed through the OSS/WSS.

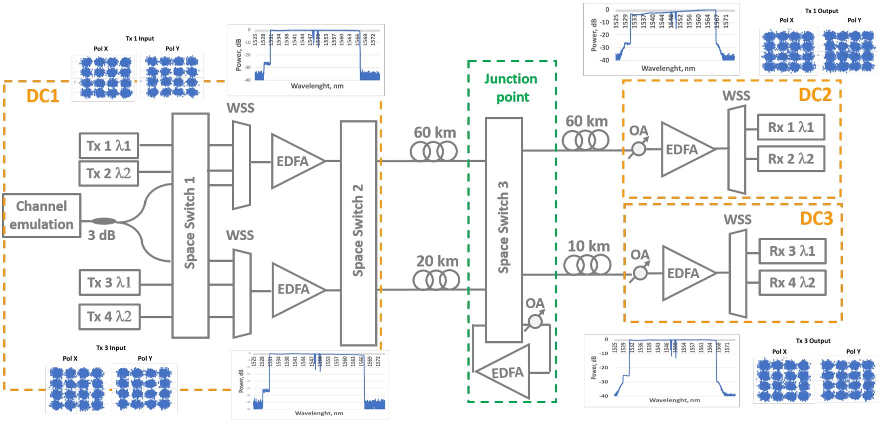


FIG. 3.7: Fiber switching experimental set up. Insets: Examples of fully loaded spectra and constellation diagrams following the expected shape of dual polarization 16-QAM signals as measured at different points in the system (details in Appendinx A.1).

The two fiber spans of 20 and 60 km from DC1 terminate at the hut. The following fiber spans from the hut are 60 km (to DC2) and 10 km (DC3). The experiment periodically swaps which spans are connected, producing two combinations A(60-60, 20-10) and B(20-60, 60-10). For both configurations, the shorter distances do not need amplification, while the hut amplifier is used for the two longer ones. Thus, over time, both DC-DC paths interchangeably utilize the hut amplifier.

This setup tests each piece of our architecture.

Power management. We measure the full spectrum at uniform power at input/output DCs. Our amplifiers work as desired, not causing any power variations after transmissions of varying lengths with occasional in-line amplification.

BER and reconfiguration. Fig. 3.8 shows the maximum bit-error rate (BER) before FEC at two of the receivers as we reconfigure every minute. Results collected over multiple day-long runs are similar. The received pre-FEC BERs are well below the soft decision FEC threshold (2x10^{-2}), translating in

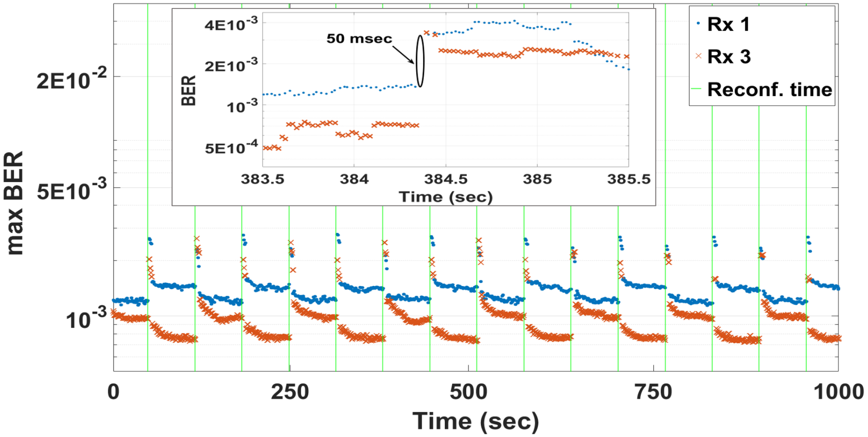


FIG. 3.8: BER over time while reconfiguration occurs (inset).

final BERs below 10^{-15}; this is similar to equivalent *static* optical links. As discussed in §3.5.2, no live traffic will be carried by paths during reconfiguration. We performed similar experiments involving reconfiguration across two independent huts with similarly consistent BERs and maximum switching time of 70 ms.

3.7.3 Impact of network reconfiguration

Iris uses reconfiguration to respond to failures and (slow) changes in DC-DC traffic. To study how this may impact application performance, we perform region-scale flow-based simulations in a custom simulator. The topologies examined reflect the DC connectivity and scale of the regions analyzed in §3.7.1. Note that we drain links before reconfiguration, so transport loss is not a concern. Our experiments thus focus on the impact of capacity reduction during reconfiguration, where a fiber switch takes 70 ms (§3.7.2).

At high traffic aggregation levels as for DC-DC traffic, we expect traffic to be stable over longer time scales. Based on experience, we use heavy-tailed traffic between DCs, with a few pairs exchanging most of the traffic;

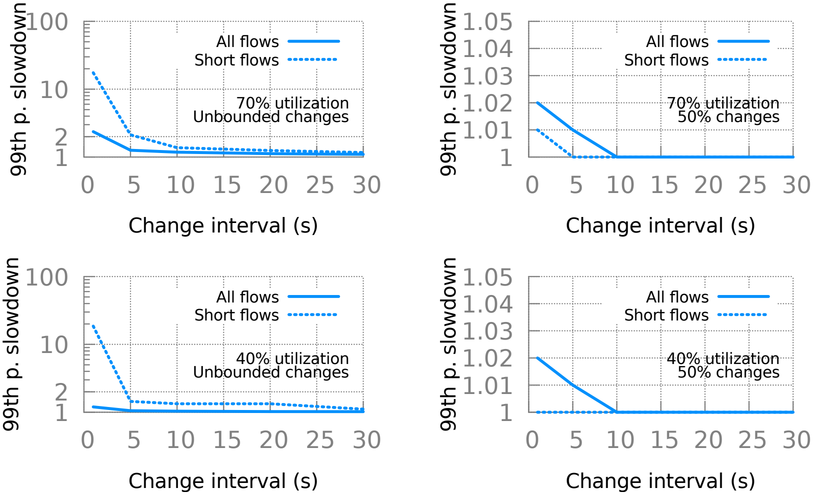


FIG. 3.9: Slowdown under reconfiguration (ratio of 99^{th}-%ile FCT for *Iris* vs. EPS). Even at high utilization (70%) and large changes (>50%), slowdown is minimal for reasonable frequencies of reconfiguration.

unbounded changes in traffic patterns occur when, *e.g.*, a low-traffic DC-DC pair becomes a high-traffic one. Otherwise, we bound the changes to a maximum % value. We study a broad swath of operating conditions: (a) network utilization in $\{10\%, 40\%, 70\%\}$; (b) reconfig frequency of once every 1-30 sec; (c) changes of $\{1\%, 10\%, 50\%, 100\%, \text{unbounded}\}$ in DC-DC traffic; and (d) several distributions for flow sizes [11], [37]. Note that these tests include extremes well beyond those we expect to encounter, *e.g.*, DC-DC aggregate traffic changing by up to 50% every 1 sec. Extreme DC-DC traffic volatility at the granularity of seconds would not be expected. Similarly, we chose to examine flow distributions that reflect intra-DC workloads dominated by short flows. This serves as a stress-test for a circuit-switched network — such flows would be the ones most affected by link reconfiguration, as longer flows are throughput and not latency-sensitive.

We compare the Flow Completion Times (FCTs) for *Iris* to an EPS fabric baseline. Fig. 3.9 highlights the increase in the 99^{th} percentile of FCT across some of our tested parameters, including the most extreme. As the results

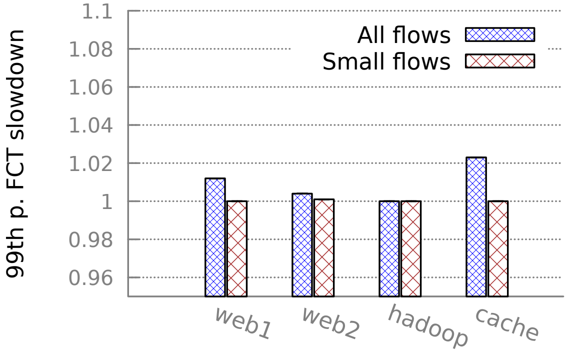


Fig. 3.10: 99^{th}-percentile slowdown at 40% util., 50% traffic changes, and reconfig. every 5 sec for various workloads; web1 is from [37] and the rest from [11]. *Iris*'s slowdown is <2% compared to EPS.

show, with the exception of unbounded intensity changes at high utilization, the effect is minimal, especially for reconfiguration intervals of 10 sec or above. For shorter intervals, there is a maximum slowdown of 2% across all flows at the 99^{th} percentile with *Iris* compared to EPS. This is true across all workloads examined.

Fig. 3.10 similarly shows that this is the case across all tested flow size distributions.

These results are largely expected: the probability of a short flow ($<50KB$) being affected is small given the reconfiguration interval is much larger than short flow completion times; meanwhile, large throughput dependent flows see only a brief, negligible drop in throughput.

3.8 Related systems

For completeness, here we provide a brief overview of related systems and approaches in domains other than data center interconnect architectures.

Cloud WAN networks, like *Iris*, interconnect small numbers of sites. However, long-distance WAN links are much more expensive than regional fiber. This results in WAN proposals like OWAN [76], SWAN [39], and B4 [85], optimizing towards maximum utilization of WAN links. *Iris* design philosophy is the opposite: exploit the cheap, abundant fiber of metro areas to design a simple and cost-effective network. Further, while wavelength switching, as is often used in metro optical networks [86], would improve spectral efficiency, in DCIs we find this to be unnecessarily complex.

Intra-DC networks using optics are also well-studied. Early efforts in this direction used OSSes [75], [87], while newer work is attempting to tackle frequently changing intra-DC traffic at microsecond scale [88], [89]. *Iris*, only needs to address slow-changing aggregate DC-DC traffic, but additionally tackle power and signal quality constraints stemming from the longer link distances. These constraints lead to completely different design choices.

Lastly, we note that DCIs are a hot industry topic [90], especially at the lowest layers, *e.g.,* customizing optical components [91], [92], and defining DCI standards [14]. This work fits within current ways of interconnecting these components, like the centralized and distributed models and their implementations discussed in §2.

3.9 *Iris* limitations

By reducing the cost by an order of magnitude compared to equivalent EPS architectures. However, there is more to be done before *Iris* is ready to be deployed in production data centers. First and foremost, cloud providers and their staff do not have sufficient experience in managing and maintaining all-optical networks. Systems and applications developed for traffic monitoring and failure detection must be adjusted or even completely redesigned to be compatible with the new optical environments. Resolving performance issues and congestion, dealing with hardware gray failures, and reasoning about the network in general will require substantial training for network administrators as well as development of new subsystem and techniques that will allow them to have better control over the optical network.

Another potential problem with *Iris* is that it relies only on the historical workload specifications and insights about DC-to-DC traffic stability. Even though by looking at history of execution of cloud workloads and their communication patterns today, we observe that the traffic is stable at DC-to-DC level, there is no strict guarantee that the traffic remain stable in the future. Massive synchronized μ-second scale bursts could still happen in the future and have negative influence on network performance due to the reconfiguration process. To mitigate this challenge, we should work on systems that obtain workload specifications about future behavior and integrate those predictions with coarse-grained workload specifications to reconfigure the network ahead of time. We describe one such system in the following chapter.

Although all-optical network need more time and development to reach the maturity level of today's EPS architectures, we believe that their simplicity, low cost, and performance will force cloud providers to invest time and effort in deploying them due to highly-competitive cloud market and the constant need for cheaper and more performant cloud resources.

3.10 Summary

Motivated by the growing popularity of multi-data center regions, we study architectures for regional data-center networks and highlight their trade-offs. We find that while distributed networks offer attractive latency and siting flexibility benefits, their implementation with today's de-facto packet-switching design also engenders greater cost and complexity. To simplify DCI network design and lower the barriers for distributed networks, *Iris* introduces an all-optical network core. With *Iris*, data travels between DCs entirely in the optical domain, thus greatly reducing the number of in-network ports. *Iris* simple fiber-level circuit switching only requires a minimal control plane, and off-the-shelf optical equipment, as our testbed implementation demonstrates.

Although dynamic reconfigurations of the *Iris* network are necessary due to changes in traffic patterns, their negative effects are mitigated thanks to the observation about traffic stability at the DC-to-DC level.

4

ADVANCE KNOWLEDGE OF FLOW SIZES

So far, we have been looking at how to leverage historical knowledge about network traffic in modern data centers to improve the efficiency of cloud network infrastructure. Even simple insights like the one the data center traffic does not frequently change between the pairs of data centers helped us achieve an order of magnitude improvement in terms of infrastructure cost.

Intuitively, if we had more knowledge about network traffic, especially information about future network behavior, we could apply more sophisticated scheduling and control algorithms to maximize network performance. This chapter explores the other extreme of the workload specification spectrum – having fine-grained knowledge about future network events. In particular, we are interested in predicting fine-grained characteristics of future network traffic and how that knowledge could be used to improve network efficiency.

Outline Section 4.1 gives a brief overview of fine-grained network workload specifications. Section 4.2 discusses an old dilemma in networking – on whether or not such advance knowledge is even possible. Section 4.3 gives an overview of techniques that can be used for obtaining advance network knowledge in the modern cloud environment, while sections 4.4 to 4.8 provide more details on each of those techniques.

4.1 Background

When users transfer a file, send or receive an email, or access a web page, they want their transaction to complete in the shortest time possible. Such transactions are called network *flows*. A flow is a *finite* and *continuous*[1] sequence of bytes that is exchange between a sender and a receiver.

One of the key goals of cloud network operators is to minimize flow completion time and provide congestion-free experience to cloud applications [93]. To achieve this goal, advance knowledge of future events in a network can be of great help. Such knowledge could potentially benefit many aspects of data center networks, including routing and flow scheduling, circuit switching, packet scheduling in switch queues, and transport protocols. Indeed, past work on each of the above topics has explored this, and in many cases, claimed significant improvements [37], [38], [94]–[96].

Nevertheless, little of this work has achieved deployment. Modern deployments primarily use techniques that do not depend on knowing traffic features in advance, such as shortest path routing with randomization and first-in-first-out queueing. A significant barrier to the adoption of traffic-aware scheduling is that traffic features can be challenging to ascertain in a timely fashion with adequate accuracy in practice.

Four high-level types of fine-grained advance workload specifications can be obtained in the context of network traffic:

- **Flow start time** - knowing when a new network flow will start would allow cloud providers to make changes in the infrastructure in order to prepare it better for the incoming traffic, *e.g.,* reconfigure dynamic links [30], [31], [48]. However, it has been shown that flow start time is challenging to predict with high accuracy due to many sources of noise and interference that exist in the cloud [97], *e.g.,* OS scheduler, access to shared resources, disk latency, etc.

---

[1] Continuous sequence of bytes - no breaks or pauses in time during the transfer.

- **Flow size** - knowing how much data a particular flow carries in advance would enable sophisticated scheduling techniques like prioritization of short flows or fine-grained bandwidth allocation.

- **Flow destination** - knowing where network traffic will go in the future can be important for similar reasons – applying early actions to change network configuration and improve efficiency. However, this information is not particularly useful before we know *when* the flow starts or *how much data* will the following flow carry.

- **Flow importance** - Ideally, cloud providers would like to know how important is a particular flow for achieving application-level goals so that more important flows receive higher priority. To have perfect knowledge of the importance of a specific flow, user intervention is necessary. Users would need to modify their applications and communicate application goals automatically through a pre-defined API. However, this is not necessary for obtaining *some* knowledge about future importance, and previous work has explored how to identify dependencies and importance across flows automatically for distributed cloud workloads [98].

Out of those four, flow size information is the one that has the most applications and systems that are developed while having the assumption of knowing the flow size in advance. Yet, this information is not available in today's cloud. Thus, we focus on the plausibility and utility of obtaining *flow size* information in advance for use in the packet, flow, and coflow scheduling in data centers. Since cloud workloads are commonly repetitive, we first learn the behavior of a workload from past execution traces using machine learning techniques, and then predict flow size based on the current state of the workload.

We thus examine a wide array of possibilities for estimating flow sizes in advance, including modifications to the application-system interface for explicit signaling by the application and more broadly applicable application-agnostic methods, ranging from simple heuristics like reading buffer occupancy and monitoring system calls to sophisticated machine learning. We analyze the complexity, accuracy, and timeliness of different approaches and the utility of the (often imprecise) flow size information gleaned by these methods across various network scheduling techniques.

4.2 Knowing flow sizes - easy or difficult?

Many scheduling techniques for data center networks have been proposed, promising substantial performance gains:

- PDQ [99] and D^3 [100] schedule flows across the fabric, in a “shortest flow first” manner.
- pFabric [37] and EPN [101] schedule packets at switch queues using “least remaining flow” prioritization.
- pHost [94], Homa [96], and FastPass [38] schedule sets of packets across the network.
- Orchestra [102], Varys [42], Sincronia [95], and Baraat [103] schedule coflows (app-level aggregates).
- c-Through [104], Helios [75], and several followup proposals schedule flows along time-varying circuits.

All of these proposals are clairvoyant schedulers, *i.e.,* they assume that the size of a flow is known when it starts. Some of this work has made this assumption explicit:

> *“In many data center applications flow sizes or deadlines are known at initiation time and can be conveyed to the network stack (e.g., through a socket API) ...”*
>
> — Alizadeh et al. [37], 2013

> *“The sender must specify the size of a message when presenting its first byte to the transport ...”*

— Montazeri et al. [96], 2018

There is not, however, consensus on the availability of such information; work in the years intervening the above two proposals has argued the opposite. For instance:

> *"... for many applications, such information is difficult to obtain, and may even be unavailable."*
>
> — Bai et al. [105], 2015

Thus, we explore if the flow size information is easily obtainable in advance. In particular we would like to know what are the techniques for obtaining flow size information, what is the accuracy of each of these techniques, and how valuable those estimates are when in comes to scheduling and improving network efficiency.

4.3 Flow size estimation: design space

Flow size definition Before considering flow size estimation, it is necessary to define a flow. The primary goal of flow size estimation is to improve application performance using network scheduling. Where application messages directly translate to individual TCP connections, using TCP 5-tuples to define flows suffices. However, to avoid the overheads of connection setup and TCP slow start, it is common practice in many data centers to use persistent long-lived TCP connections (*e.g.,* between Web servers and caches) which carry multiple application messages over time. In these settings, it may be more appropriate to consider instead a series of packets between two hosts that are separated from other packet series by a suitable time gap. We note that this is an imprecise method, as system-level variability and workload effects can impair such identification of flows. For instance, multiple small cache responses sent out in quick succession could be mistakenly identified as one flow. This limitation applies to all application-oblivious methods — in some scenarios, mechanisms to identify packets that form an application-level message are inherently bound to be imprecise.

We next describe five approaches for obtaining flow sizes:

- *Flow-size reporting API* - building an interface that allows cloud applications to automatically report their flow size.
- *Flow aging* - using a heuristic that approximates flow size based on the number of bytes that this particular flow has already sent.
- *TCP buffer occupancy* - amount of data currently placed in the sending buffer gives us a lower bound on the remaining flow size.
- *System call monitoring* - applications move data from user space to TCP buffer using particular system calls. The size of those transfers can be used to estimate flow size.
- *Applying machine learning* - An obvious, but the most sophisticated approach is using various ML techniques that predict flow size based on system and application parameters.

Also, the following sections provide intuitive reasons about the efficacy of these techniques in various settings. Experimental evaluations of the quality of estimation and its impact on network scheduling are deferred to §5.4.

4.4 Exact sizes provided by application

Many applications can assess how much data they want to send, such as standard Web servers, RPC libraries, and file servers. Modifying such applications to notify the network of a message's size is thus plausible. This would require a new interface between applications and the network stack, and is clearly doable, but not trivial. The replacement must be interruptible, like `send` in Linux, and it's unclear how best to implement this – what happens when it gets interrupted after sending some bytes? When a new call is made to finish the transfer, how do we decide whether or not this is the same flow? Thus, this may also require introducing some notion of flow identifiers. While this can surely be done, we merely point out that it requires care.

Limitations: As discussed in some depth in prior work [105], there are several scenarios where the application itself is unaware of the final flow size when it starts sending out data, such as for chunked HTTP transfers, streaming applications, and database queries where partial results can be sent out before completion. Also, apart from introducing changes to the host network stack (which are not necessarily prohibitive for private data centers), this approach requires modifying a large number of applications to use the new API. In settings like the public cloud, this may not be feasible.

Still, this approach should not be casually dismissed; a few software packages dominate the vast majority of deployed applications, *e.g.,* a large fraction of Web servers use one of the three most popular server software packages, most data analytics tasks use one of a small number of leading frameworks, etc. Past work (*e.g.,* FlowProphet [106] and HadoopWatch [107]) has in fact explored the use framework-internals for gleaning flow sizes. Thus, this approach could make flow size information accessible to the network for many applications, provided the right APIs are developed.

4.5 Flow aging

A set of application-agnostic techniques have been proposed around the idea of using the number of bytes a flow has already sent as an estimator for its pending data. For instance, Least Attained Service (LAS) [108] gives the highest priority to new flows and then decreases their priority as they send more data. Thus, flow priorities "age" over time. PIAS [105] explores a variant of this approach, coupling it with the use of a small number of discrete queues to fit commodity switches. Aalo [43] applies similar ideas to coflow scheduling.

Limitations: The most significant drawback is that this approach may not benefit scheduling techniques that require absolute flow sizes (as opposed to only relative priorities), such as Sincronia [95][2], FastPass [38] and optical circuit scheduling. Even where applicable, the effectiveness of such methods depends on flow size distribution. For instance, LAS does not work well when there are a large number of flows of similar size. In the limiting case, if all flows are the same size, older flows nearer to completion are deprioritized, which is the opposite of the desired scheduling. More sophisticated methods based on multi-level feedback queues [105] still depend on estimating a *stable* underlying flow size distribution[3]. Further, even in favorable settings, with stable heavy-tailed flow size distributions, the performance of such application-agnostic techniques can be substantially lower than clairvoyant ones. For instance, recent work [96] reports $\sim 2\times$ difference in 99^{th} percentile slowdown between PIAS [105] and pFabric [37]. Similarly, Sincronia [95], the best-known clairvoyant coflow scheduler, claims a $2-8\times$ advantage over Varys, and by extension, over CODA [98], the best-known non-clairvoyant coflow scheduler. (Note however, that the scheduling knowledge involved in CODA is not limited to flow sizes, but also classification of flows into coflows.)

4.6 TCP buffer occupancy

The occupancy of the TCP send buffer at the sending host can provide approximate information on flow sizes. When the buffer occupancy is small, the number of packets in the buffer may be the actual flow size of a small flow. When the buffer is fully occupied, *i.e.,* its draining rate is less than its filling rate, the flow may be categorized as a large flow. Mahout [110] and c-Through [104] used roughly this approach. ADS [111] also suggests (but

---

[2] Sincronia uses only relative priorities in the network, but for assigning these priorities, it computes the bottleneck port using sizes of all flows destined to each port. It is unclear if aging would be effective here.

[3] Alternatively, additional effort must be spent in continuously monitoring and following the changes in the underlying distribution [109].

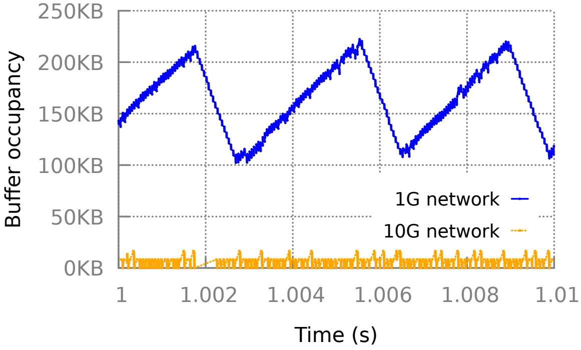


Fig. 4.1: Buffer occupancy while transferring a 1GB static file from the hard disk over 1G and 10G connections. We show a representative 10 ms segment of the trace starting at 1 second.

does not evaluate) a similar mechanism, although it is unclear whether it uses system calls, or buffer occupancy, or both.

Limitations: Buffering reflects flow size only when the sender is network or destination limited. If the bottleneck is elsewhere, the buffer may not be filled even by a large flow. Consider a program that reads a large file from the disk and sends it over the network. The program reads data in chunks of a certain size (*e.g.,* 100 KB) and sends as follows:

```
while(...):
  read(file_desc, read_buffer, 100KB)
  write(socket_desc, read_buffer, 100KB)
```

Today's SSDs achieve a read throughput of ∼6 Gbps, while NIC bandwidths of 10-40 Gbps are common. This disparity implies that the buffer may remain sparsely populated most of the time. To illustrate this behavior, we ran a simple experiment. We transfer a 1GB static file served by a Web server over 1G and 10G connections (Fig.4.1). The file is stored on a regular 7200 RPM hard drive with the maximum read speed of ∼ 1 Gbps. We see that for the faster connection, the buffer is almost empty. For slow connections, the buffer indicates the lower bound of the flow size, but when the buffer occupancy starts decreasing, it is unclear if that means that no additional data will be added. This presents a serious obstacle for flow size estimation.

4.7 Monitoring system calls

The `write/send` system calls from an application to the kernel provide information on the amount of data the application wants to send. The flow size is typically greater or equal to the number of bytes in the first system call of a flow. It is also interesting to notice that many applications have a standard system call length. For instance, Apache Tomcat by default transfers data in chunks of 8 KB. If it wants to send less than 8 KB, it issues a single system call which reflects the exact flow size. For larger flows, multiple calls are, of course, necessary. Other applications behave similarly; MySQL uses chunks of 16 KB, Spark Standalone 100 KB, and YARN 262 KB. Thus, for identifying short flows, this is a reliable approach, and can directly enable algorithms like Homa [96]. Further, recent work from Facebook suggests that a substantial portion of flows is extremely small and most likely transferred over a single system call [112].

To test this approach, we run a simple experiment where we store 100,000 objects of sizes between 500 B to 1 KB using MySQL. Further, we execute three types of queries: fetch an object based on the key, fetch a range of 10 objects using a date index, and fetch 1000 objects (*e.g.,* to perform a join operation or backup). Since results for queries fetching 1 and 10 objects fit into the initial system call, we were, in fact, able to obtain their flow sizes accurately.

Limitations: The flow size information inferred from system calls may correspond to only a part of the flow rather than the whole flow, as in the above example for large queries. Increasing the size of the initial system call could work, but larger system calls require more buffer memory per connection.

Thus, Web servers, databases, and other highly concurrent programs tend to keep system calls small.

4.8 Learning from past traces

We can also apply machine learning to infer flow size information from system traces. Ultimately, data sent out to the network trace causality to some data received read from disk or memory, or generated through computation. Thus, traces of these activities may allow learning network flow sizes. Given that most jobs in data centers today are repetitive, there is a significant opportunity for such learning. For instance, in [113], the authors observed that more than 60% jobs in enterprise clusters are periodic with predictable resource usage patterns. Analysis of publicly available Google data center traces also confirms this finding: most of the resources are consumed by long-term workloads [114]–[116].

Unlike the simpler approaches above, the effectiveness and limitations of learning methods are hard to analyze without a serious attempt at building a learning system. A key challenge here is the short timescale: while past work [117], [118] has explored learning workload characteristics at timescales of minutes and hours, can we learn at the microsecond timescales necessary for flow size estimation? This represents a challenging leap across 8-10 *orders of magnitude* in timescales. We next detail our efforts towards building a learning system for flow size prediction.

4.9 Summary

Many scheduling and network control systems that have been proposed by the academic community promise substantial performance improvements by leveraging advance flow size information. However, none of them achieved a broad adoption in practical deployments, because the flow size information is not available in the cloud environment. For some workloads, having that information is fundamentally impossible, *e.g.,* streaming applications. In contrast, for others, this problem requires careful analysis and substantial investment in terms of time and effort in modifying existing applications or creating new systems that obtain flow size knowledge in advance. We propose one such system in the following chapter.

5

LEARNING FLOW SIZES

As we demonstrated, even simple heuristics effectively estimate flow sizes in situations where flows are relatively small. However, to unlock the full potential of sophisticated network scheduling and control algorithms, we need to provide more accurate information. This can be achieved by deploying machine learning systems to help cloud providers automatically obtain flow-size information.

In this chapter, we present *Flux*, a system that leverages modern machine learning techniques to estimate the size of the upcoming flow by looking at the history of execution of a particular job as well as system-level parameters like resource utilization, TCP buffer occupancy, and system calls.

Besides providing high accuracy of the flow size estimates, we are also interested in obtaining the estimates in a timely, low-overhead manner. Also, we want to understand how flow size information may be used to improve different aspects of network performance and efficiency and how the inaccuracy of flow size estimates impacts the performance of the network.

Outline Section 5.1 describes several machine learning techniques that we tested for obtaining flow size information. Next, section 5.2 gives a more profound intuition and explanation about how and why machine learning techniques work in this setting. Section 5.3 explains how those predictions can be used to improve the efficiency of cloud networks, while in Section 5.4, we evaluate those techniques with our flow size predictions against multiple workloads. Then, Section 5.5 and Section 5.6 discuss the implementation of *Flux* and how to deploy it in the cloud environment to obtain flow size predictions with low latency. Finally, Section 5.7 discusses the limitations of our learning approach.

5.1 Introduction to learning flow sizes

We explore the design of a learning-based approach for flow size estimation, addressing the following questions:

- Which methods can we use for flow size prediction?
- What prediction accuracy is achievable?

Learning task: We would like to learn flow sizes for outgoing flows in advance, using system traces. When a flow f starts, are recent measurements of network, disk, memory I/O, CPU utilization, etc. predictive of f's size?

To answer this question, we first introduced the representative cloud workload we analyzed, and then we focus on machine learning techniques that leverage historical execution traces from those workloads to predict flow size.

5.1.1 Workloads

To explore what inputs are predictive of flow sizes, it is essential to gather job execution traces with as much detail as possible, across many instances of jobs with variable inputs and configurations. Unfortunately, publicly available data center traces do not contain enough information. Facebook's traces [11], by sampling 1 per 30000 packets, provide no visibility at the flow granularity. Google's traces [119] completely omit network data, focusing on CPU, memory utilization, etc. We thus collect traces for (a) a large range of synthetic workloads; and (b) machine learning applications running on our university clusters.

Our traces comprise 5 applications: PageRank, K-Means, and Stochastic Gradient Descent (SGD) implemented on Spark; training deep neural networks using TensorFlow; and a Web workload. The SGD and Tensorflow traces are from instrumented applications running on our university cluster.

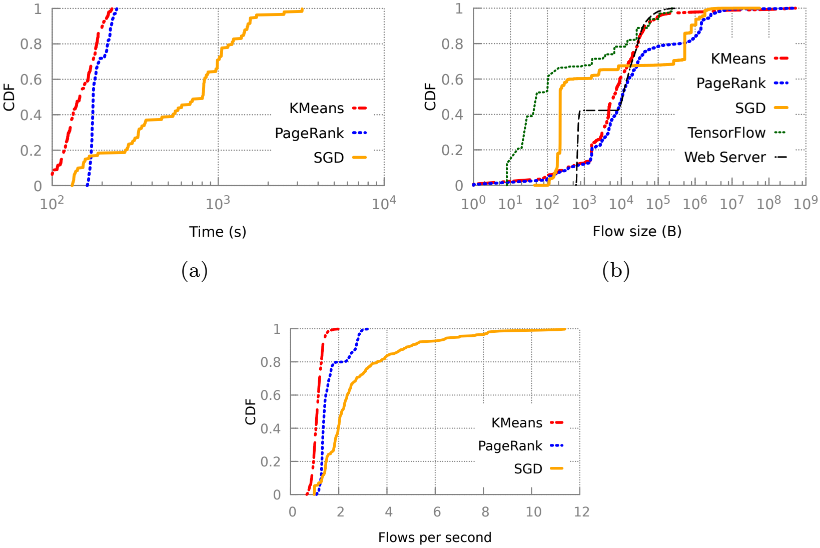
Each of SGD, KMeans, and PageRank runs on a Spark cluster of 8 machines, each machine with 2 CPU sockets (4 cores each) and 24 GB DRAM. For SGD, the input sizes vary from 2-25 GB, with significant variation across the hyperparameters. We also impose large input variations for KMeans and PageRank: for PageRank, we randomly generate new graphs with 1-15 million nodes; and for KMeans, we generate datasets with 20-50 million points, while also varying K. We also vary the number of workers per job from 8 to 64.

The Tensorflow trace consists of one 25 minute long execution of distributed training of AlexNet [120] on the ImageNet dataset on 40 GPU machines.

For the synthetic Web workload, we use Apache Tomcat 7.0 to host a full Wikipedia mirror and fetch random pages.

Fig. 5.1 summarizes these workloads. Fig. 5.1(a) shows the job execution times across different executions for each algorithm. Execution times for KMeans, PageRank, and SGD vary by factors of as much as 2.6×, 1.5×, and 24.5× respectively. Thus, there is substantial variation across executions. The main source of the variations across traces is the change in the size of the input data and the number of iterations. However, for many runs there were more resources in the cluster than required by the job, leading traces to further incorporate the influence of Spark's scheduling decisions.

We also aggregate flow statistics across all jobs and executions for each application to give a sense of the traffic; Fig. 5.1(b) shows the flow size distributions, and Fig. 5.1(c) shows the arrival rates (aggregated across the workers). The TensorFlow workload consists of many short flows with an average arrival rate of 8273 flows/second.



(a) (b)

(c)

Fig. 5.1: Workload diversity: (a) Execution time varies substantially across executions of the same job. We also show the distribution of (b) flow sizes and (c) flow arrival rate across our workloads.

5.1.2 Machine learning models

We evaluate several ML models, but with only modest efforts to optimize these, because our goal is not to identify the best model or hyperparameters, but to show that a variety of methods could work (as we show with results on improvements in scheduling in §5.4) with reasonable effort, modulo the limitations of learning in this context, as discussed in §5.7.

Recurrent Neural Network with LSTM layers: All our traces are time series. Given the natural dependency between data points in the trace (past flows influence the creation of the current flow), we test a network that can keep state and learn these dependencies while processing the trace sequentially. For this purpose, we use an LSTM model [121]. Using Keras [122]

	GBDT	**FFNN**	**LSTM**
Web server	94 | 96	92 | 94	73 | 74
TensorFlow	97 | 97	95 | 95	94 | 94
PageRank	85 | 83	84 | 84	83 | 83
Kmeans	88 | 90	88 | 95	88 | 93
SGD	58 | 79	54 | 72	46 | 0

TABLE 5.1: *Prediction accuracy (shown for validation-set / test-set) across models and workloads in terms of R^2 percentage.*

and Tensorflow [123], we test varied LSTM models with different numbers of layers. The best configuration in our setting uses a single layer with 64 LSTM units.

Gradient Boosting Decision Trees: In many of our traces, simple conditionals reveal information about the flow size. For instance, if the first system call size is below a certain value, that can often reveal the flow's size. Thus, we train GBDT models of different sizes (*i.e.,* numbers of trees) and find that using 50 trees (with maximum depth of 10 per tree) gives fast yet accurate results.

Feed-Forward Neural Network: The dependency between flow size and other system-level features should not strictly depend on the ML model we choose. Thus, we test a standard FFNN model [124] with various configurations for the number of layers and neurons, implemented using Keras [122]. We find that 2 layers (and the ReLU activation function) with 5 neurons each yield the best performance.

Results: We split the traces into 3 fixed sets – training, validation, and test. Table 5.1 compares the three tested models. We use the coefficient of determination (R^2) to measure accuracy. R^2 is very useful because it can be easily compared across different models: $R^2 = 1$ if the model produces perfect predictions, and $R^2 = 0$ if the model makes a prediction of zero value, always predicting the mean. GBDT and FFNN achieve comparable accuracy (Table 5.1), with the high values of R^2 implying highly accurate flow size predictions. For two workloads, LSTM gave inferior results and did not seem to capture the dependencies, particularly across traces where the underlying executions were very different (*e.g.,* test-set for SGD). With greater effort, for instance, specializing the model to these traces, it may be possible to overcome LSTM's apparent deficiency. However, we wanted to use the same training and inference approach across traces.

GBDT's accuracy, fast convergence, and fast inference motivate its choice for *Flux*. The tradeoff is that the model updates in batch mode (not online); this should suffice, unless applications change at sub-second timescales.

5.2 Opening the black box

What explains the high accuracy of our ML approach? We discuss the predictive power of various system-level measurements, and detail refinements that led from poor initial results to these high-accuracy predictions.

5.2.1 The treachery of time

We first tried what we considered a natural model for the data of our interest: time series. To generate time series data, during the execution of each workload, we sampled CPU and memory utilization, and disk and memory I/O, every 20 ms, and recorded headers of all incoming and outgoing packets. We then attempted to predict the next few time-steps for network traffic. However, this gave poor results due to low-level system effects that can have a significant impact on timing.

An alternative representation with a *flow-centric view* treats a job as a series of flows, with several attributes recorded per flow (Table 5.2). This is effective for flow size prediction, as it does not suffer from minor timing variations, and captures the relationship between (for instance) system calls and the volume of outgoing traffic. In addition, the measurements themselves serve as a "clock", one that is more robust to system scheduling artifacts.

Feature	**Description**
Start time, t_f	Start time of f relative to job start time
Flow gap	Time since the end of the previous flow
First Call	Size of the first system call t_f
Network In	Data received until t_f
Network Out	Data sent until t_f
Network In(d)	Data received at flow's dest. d until t_f
Network Out(d)	Data sent by this host to d until t_f
CPU	CPU cycles used until t_f
Disk I/O	Total disk I/O until t_f
Memory I/O	Total memory I/O until t_f
Previous flows	Flow sizes for last k flows

TABLE 5.2: *Features of a flow f. All network, memory, disk and CPU activity is cumulative until this flow's start at t_f.*

5.2.2 Why these features?

New flows are created either by reading data from the disk or memory, processing previously received flows, doing some computation to create data, etc. Thus, features that characterize each of these causal factors could help estimate flow size. For each type of system measurement, we track the total number of operations or bytes from the beginning of program execution. This enables the learning algorithm to know how many operations or bytes were processed between the last and the new flow. In our experiments, when we predict the flow size, we use features from Table 5.2 for last 5 flows. Thus, we try to catch dependencies between consecutive flows as well as resources that have been consumed.

As expected, the most predictive features vary across applications. For instance, some applications do not produce a lot of disk traffic, while others rely completely on the disk. The GBDT model provides a natural way of assessing feature importance: it uses a set of decision trees, with each attribute's contribution measured in terms of "splits", *i.e.,* in what percentage of branch conditions in the decision tree the attribute appears. Fig. 5.2 shows these splits for the aggregated flow-centric traces from a Spark environment. Interestingly, for these traces, we find that similar accuracy can be obtained by a model constrained to *not* use memory, disk, and CPU monitoring, relying only on network data and timing of flows. Web queries, on the other hand, have a different set of critical features where 66% of all splits use disk I/O.

Finding the *best* model and feature-set may require manual work, but effective solutions could be obtained automatically by comparing the accuracy of the largest model to the accuracy of candidate models limited to using only the first model's most salient features.

5.2.3 Model accuracy

The accuracy of predictions obtained using learning depends on three main factors, which we discuss next.

How far the predicted future is: It is critical to make predictions within a time budget. Since most flows in data centers are small, this budget is commensurately small: if the inference takes too long, we might either block on the inference and slow down the flow, or allow packets to flow without

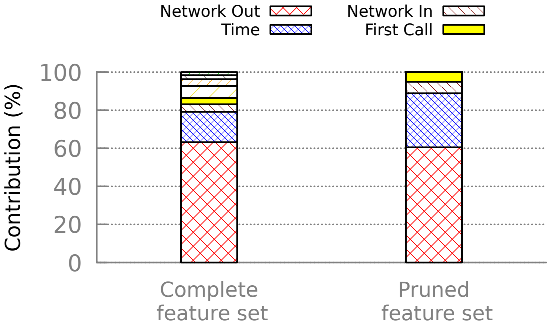


Fig. 5.2: Feature contributions for 2 models for the Spark workloads, measured using GBDT splits. The figure omits labels for the less important features: memory and CPU utilization, and disk and memory I/O. Both models provide the same prediction accuracy. If we exclude any of the top 3 contributors, the accuracy decreases.

having the result of the inference and without tagging them appropriately, resulting in sub-optimal performance.

There are two possibilities for overcoming this issue: (a) when a new flow starts, make a prediction in an extremely small time budget by engineering down the inference time; and (b) when a new flow starts, start inference for the *next* outgoing flow, or even more generally, for some future flow. This choice represents a trade-off: we can either get high-accuracy inference by incorporating the maximum information available for inference, but incurring a data path latency to do so (or use results late, as they become available); or get lower accuracy due to missing some relevant information from needing to predict a farther future.

Fig.5.3 shows the dependence of prediction accuracy on this “future distance”, starting from trying to predict a flow’s size immediately when it starts, through predicting the next several flows. For TensorFlow, predicting several flows into the future is possible with high accuracy, because flows are predictive of future flows. But as expected, for the Web server workload, it is only possible to accurately predict the flow starting now, because two

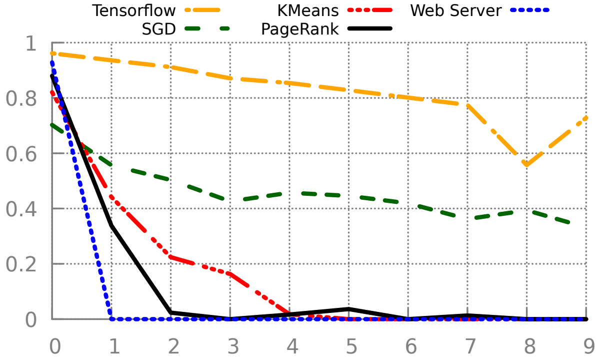


Fig. 5.3: Prediction accuracy declines for more distant flows. $x = 0$ is the current flow, for which packets are starting to be sent out, while $x = 1$ is the next flow after this one, and so on.

consecutive flows share no relationship (because we are requesting random objects from a Wikipedia mirror) – in essence, each "job" is of size one.

Model size: Larger models often yield higher accuracy at the cost of more memory and computation, and consequently, and more crucially, higher latency for inference.

While details of the impact of model accuracy on scheduling performance are deferred to §5.4, we use flow completion times instead of R^2 to compare model sizes.

For an example trace (pFabric scheduling for PageRank), when predicting the current (next) flow's size, the average FCT using a smaller model with 20 trees is worse by 9% (10%) than the larger model with 50 trees. Interestingly, the larger model achieves better results than the smaller one, even when the larger model is impaired by having to predict the *next* flow, while the smaller model predicts the current flow.

Training dataset size: Obviously, the learning approach depends on having seen enough training data, but this "convergence time" varies across workloads. For the Web workload, the model only needs to observe $\sim$ 50 requests to achieve $R^2 > 0.5$. To achieve nearly its maximum prediction accuracy, the model needs to observe $\sim$ 500 requests. For a popular Web server, this is on the order of a few seconds. (Of course, our model for a Web server is extremely simple.

The model for TensorFlow needs to see $\sim$ 3000 flows to reach peak accuracy, but given its flow arrival rate of more than 8200 flows per second, convergence time is sub-second. This is negligible compared to the job duration itself ($\sim$ 25 min). The iterative nature of neural network training, with similar traffic across iterations, allows accurate prediction within a few iterations of monitoring a never-seen-before job. The Spark workloads show the highest variability, and this is reflected in their convergence time. Here we need traces from multiple executions of the same job type to achieve high accuracy; 10 executions suffice for each of our 3 test job types[1]. Fortunately, data processing frameworks are often run repetitively with many instances of the same job [113], [125] since the workloads often involve tasks like making daily reports, code builds, backups or re-indexing data structures.

Note that good results can be achieved even across repeat executions with very different underlying data and run configurations — the job instances over which we train and test exhibit such variations, as discussed in §5.1.1.

5.3 Flow-size-based network scheduling

We explore the utility of advance knowledge in the context of four representative scheduling techniques from past work: pFabric [37], pHost [94], FastPass [38], and Sincronia [95]. Each of these is a clairvoyant scheduler, with advance knowledge of the size of each flow at its start (but not necessarily the flow arrival times).

Here we include brief, simplified background on the representative scheduling techniques:

---

[1] Each execution yields n traces, when n machines execute the job.

pFabric [37]: Each packet is tagged with a priority at the end-host, based on the remaining flow size. Switches then schedule packets in order of least remaining size. This results in near-optimal packet scheduling and can improve average flow completion time (FCT) by as much as 4× for certain workloads, compared to the oblivious FIFO scheme.

pHost [94]: pHost uses distributed scheduling, with the source sending to the destination a “Request To Send” message carrying the number of pending bytes in the flow. The destination clears transmission for the flow with the least bytes. pHost claims an average FCT reduction of 3×.

FastPass [38]: FastPass uses a centralized arbiter to schedule flows. When a host wants to send data, it asks the arbiter to assign it a data transmission time slot and path. The arbiter tries to make a decision based on the traffic demand (flow size) of all active flows. FastPass claims “near-zero queuing” on heavily loaded networks, cutting network RTT by 15×.

Sincronia [95]: Sincronia orders coflows using sizes of individual flows to find network bottlenecks and uses this ordering for priority scheduling. It achieves average coflow completion time (CCT) within 4× of the optimal.

If flow sizes were known *a priori*, such techniques could improve various performance metrics of interest by 3-15× compared to size-unaware ECMP-plus-FIFO scheduling. For some scheduling problems, where only relative flow priorities matter rather than absolute sizes, prior work has developed non-clairvoyant schedulers [43], [98], [105] that also beat the ECMP-FIFO baseline. But as we shall see, their performance improvements are often much more modest than clairvoyant schedulers.

For some problems, non-clairvoyant algorithms are also known, *e.g.,* PIAS [105] for packet scheduling, and Aalo [43] for coflow scheduling. While such techniques outperform FIFO and fair-sharing baselines, there is still a substantial performance gap compared to clairvoyant algorithms. Recent work [96] reports $\sim 2\times$ difference in 99^{th} percentile slowdown between PIAS [105] and pFabric [37]. Similarly, Sincronia [95], the best-known clairvoyant coflow scheduler, claims a $2-8\times$ advantage over Varys, and by extension, over the best-known non-clairvoyant scheduler, CODA [98]. Further, it is unclear if similar non-clairvoyant methods can be developed for scheduling problems such as FastPass [38], where absolute flow sizes are needed, rather than just the relative priorities leveraged by PIAS and Aalo.

5.4 Evaluation

Assessing accuracy of ML-based method in terms of mean error and R^2 is useful, but unsatisfactory — we ultimately want to understand the impact of errors on scheduling that uses the estimates. We thus quantify the performance of both flow-level (FastPass, pFabric, and pHost) and coflow-level (Sincronia) schedulers with varying degrees of inaccuracy in flow sizes. Throughout this evaluation, we use the same traces used for our validation and testing results in §5.1. Also, we account on inference latency, that will be discussed in the following section.

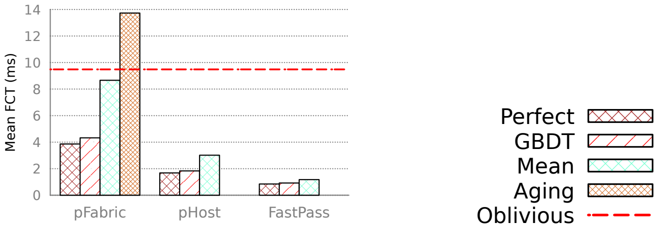
5.4.1 Flow-level scheduling

We use the YAPS simulator [126]. We use the leaf-spine topology used in pFabric [37], with 4 spines, 9 racks, and 144 servers, with all network links being 10 Gbps. To measure the effect of inaccurate predictions on flow completion times (FCT), we replay the network traces collected from our cluster in YAPS. Each experiment uses traces[2] from one of the 5 job types. We run all our tests at 60% network utilization, mirroring the original pFabric and pHost papers.

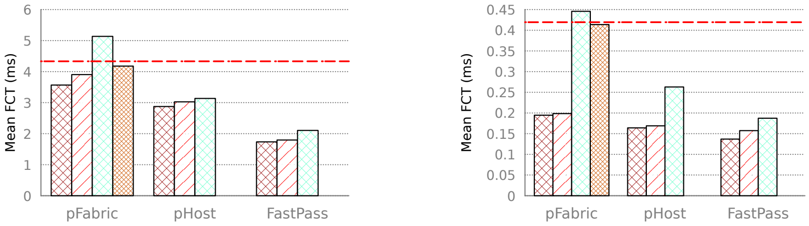
We compare network performance across the following flow estimators: (0) “Perfect”, an ideal predictor with zero error. (1) “Mean”, whereby every flow size is predicted to be the mean. (2) “GBDT”, the gradient-boosting decision

---

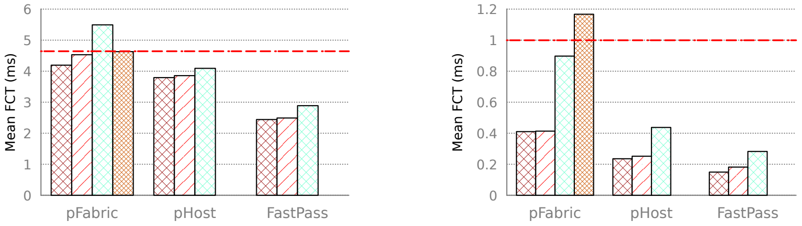
[2] Note that, unfortunately, we cannot provide results for flow size distributions often used in data center research because we do not have the traces to produce the distribution of estimation error for them.



(a) *SGD*

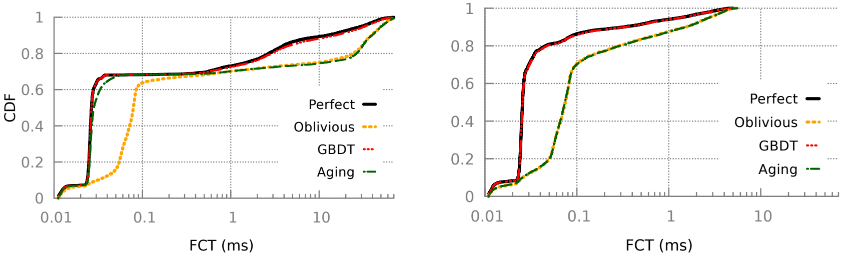


(c) *PageRank* (d) *TensorFlow*



(e) *KMeans* (f) *Web server*

Fig. 5.4: Mean FCT across 4 scheduling techniques, 5 workloads, and several flow size estimators.



(a) *KMeans*

(b) *KMeans*

FIG. 5.5: FCTs for pFabric for the SGD (left) and TensorFlow (right) workloads. Due to the log-scale, small visual differences are significant. On the right plot, Perfect and GBDT are visually indistinguishable, and so are Aging and Oblivious.

tree learning approach with 50 trees. (3) Specifically for pFabric, we also evaluate the 0-knowledge LAS policy – “Aging” (§4.5). Today’s commonly deployed approach – FIFO scheduling at switches and ECMP forwarding – is also evaluated as a baseline (“Oblivious”).

Fig. 5.4 shows the average FCT across all 5 workloads, 3 flow-level scheduling techniques, and these flow estimators. Note that the Aging result is shown only for pFabric, because it can be easily modified to use LAS.

Oblivious often results in mean FCT more than 2× that of Perfect, *e.g.,* compared to FastPass across all workloads, and compared to pFabric in Fig. 5.4(a) and 5.4(d); the largest gap is as large as 11.1×, vs. FastPass in Fig. 5.4(a). GBDT achieves mean FCT close to Perfect across all cases, with the largest gap being 1.21×, vs. FastPass in Fig. 5.4(f). Compared to Oblivious, improvements with GBDT range from 1.1-11.1× across our experiments.

Understanding the performance of these schemes requires a closer look across the entire flow size distribution. Fig. 5.5 (left) shows the distribution for one example – pFabric scheduling over an SGD trace, *i.e.,* details behind the mean FCTs for pFabric in Fig. 5.4(a). Note that the logarithmic x-axis in

Fig. 5.5 visually suppresses significant differences. Aging indeed achieves good results for the short flows for the SGD trace, but for longer flows, which share the same priority for a long time, its performance is worse than Oblivious, resulting in a larger mean FCT (Fig. 5.4(a)). The TensorFlow workload, with most flows being short, presents a difficult scenario for Aging – as noted in §4.5, for such workloads, Aging's behavior is the opposite of desirable (Fig. 5.5 (right)).

In contrast to Aging, GBDT's performance is similar to Perfect across the flow size spectrum for both workloads.

5.4.2 Coflow scheduling

We evaluate Sincronia, a recent proposal that leverages flow size information to provide near-optimal coflow completion time (CCT), with our imprecise flow size estimates.

We generate coflows from our traces by picking r consecutive flows grouped together to create a coflow. For each coflow, r is chosen uniformly at random from $\{1, 2, 3, \ldots, 20\}$. For each of our five traces, we run 200 coflows with Sincronia's offline simulator at 60% network load. We execute experiments for Perfect and GBDT, and record mean CCT. To measure the effect of inaccurate predictions, we define relative performance degradation as *GBDT-CCT / Perfect-CCT*.

Fig. 5.6 shows that the performance degradation for coflow scheduling for PageRank, KMeans and SGD, is substantialy higher than for flow scheduling algorithms. That is because errors in estimates for individual flow sizes compound with coflows. This also explains why workloads with very high accuracy for individual flow sizes, such as Web server and TensorFlow, are only exposed to modest degradation.

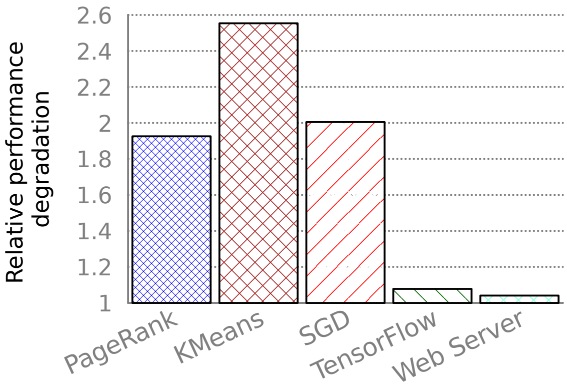


FIG. 5.6: Relative performance degradation for Sincronia expressed as the ratio between mean CCT with imperfect estimates and perfect knowledge.

5.5 Design & Implementation

Being capable to obtain high-accuracy estimates of workload specifications, in this case flow size information, is not sufficient to use those specification in practice. These estimates must be available *on time*, with very short latency, so that the inference latency does not degrade the performance of the entire system and remove the benefits provided by those workload specifications in the first place.

In this section, we provide more details on general *Flux* design, with focus on minimizing the inference latency in different deployment environments.

5.5.1 Design overview

Our goal with *Flux* is to *quickly* and *precisely* estimate flow sizes at the sending host, and tag each flow's packets with this information. Fig. 5.7 shows an

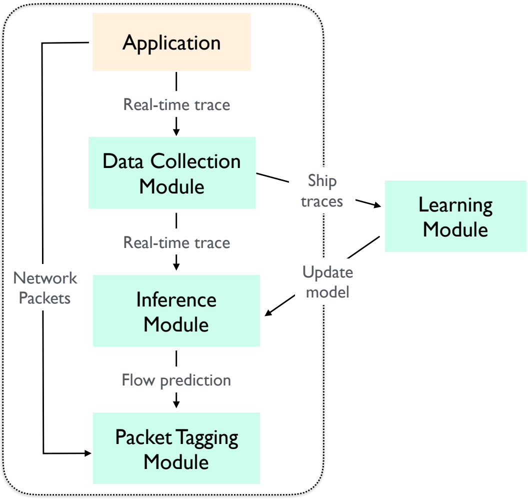


FIG. 5.7: Overview of *Flux*'s design.

overview of *Flux*'s design, along with its data flow. *Flux* is composed of 4 modules:

- Data collection, which gathers system-level traces for information relevant to flow size estimation.
- Learning, which uses these traces to train an inference model for flow size estimation.
- Inference, which uses real-time system traces to estimate flow sizes.
- Packet tagging, which adds the flow estimates to the appropriate packet headers or control messages for use in network scheduling.

The data collection, tagging and inference run on the data path at each host, and must minimize latency. The learning module can be updated in a batched manner, and run independently at a host or in a consolidated manner for multiple (virtual or physical) machines.

5.5.2 *Flux* implementation

We first describe an implementation of *Flux* in a controlled environment, where we have the freedom to make changes across the operating system hosting the applications, and later discuss generalizing this implementation for other environments encountered in data centers.

Data collector: Our results in §5.2.2, reveal that the most predictive features for flow sizes for our workloads are network and disk I/O and flow inter-arrival times. The data collector thus logs these, while still keeping track of general system utilization parameters (CPU, memory).

A simple way to log I/O in a low-overhead manner is to monitor the related system calls. For example, the following two API calls are often used to send or receive data to or from the network:

```
send(dest_id, from_buffer, num_bytes)
num_bytes receive(source_id, to_buffer)
```

Regardless of the OS or the network stack implementation, similar calls with the same or similar parameters exist. When data is to be sent, it is necessary to specify its destination (*dest_id*), where the data is to be sent from (*from_buffer*), and how much data is to be sent (*num_bytes*). Likewise, the receiver must know which sender sent the data (*source_id*), where the data has been received (*to_buffer*), and how many bytes were received (*num_bytes*). An RDMA stack, for instance, uses a different API, but with equivalent information.

For the purpose of our experiments, to capture network I/O, we defined wrappers for the following system calls from the standard *glibc* library: *write, send, sendto, sendfile, sendmsg, read, recv, recvfrom,* and *recvmsg*. We then used *ld_preload* to load our wrappers, thus overriding glibc; this is a standard Linux feature. The wrappers are thin and low-overhead, as they merely log information before calling glibc's implementations.

It is also necessary to identify the application that is currently active as well as the start of its execution to enable the learning module to separately learn characteristics of different applications. Thus, every trace collected is tagged with its identifying *job_id*. In cloud environments, tenant VMs have unique identifiers corresponding to the application or higher-level user subscription. (Analysis of production workloads in past work reveals that a subscription typically maps to a single application [125]). Hence, the needed information is readily available in production environments.

The data collector logs network flows during the whole execution of a job. A flow is identified by its source-destination 4-tuple, and the constraint that if it comprises multiple send calls, the time interval between successive sends must not exceed a *flow gap* threshold. The data collector thus tracks, for each flow, its last send call's timing. If a new send is issued, and the time elapsed since the last call exceeds the flow gap, the data collector considers this send the start of a new flow. This allows us to accommodate data center applications that often establish long-running persistent connections, and reuse them to avoid transport-layer overheads [11].

The data collector sends the collected traces to the learning module for model training. This is a small amount of data batched over time. It also needs to provide inputs to the inference module. When the data collector identifies a new flow, it sends a set of input features to the inference module, to enable flow size estimation.

Packet tagging module: We implemented the packet tagging module as a Linux kernel module that contains a Netfilter. (Netfilter is a low-level packet filtering and manipulation mechanism available for Linux.) For our experiments, we implemented a simple pFabric-style packet tagger, which tags every packet with the remaining amount of data in its flow, based on the inferred flow size and bytes sent. For mechanisms like pHost and FastPass, which implement their own network message protocols, we would need to integrate such tagging in their implementations.

The packet tagging module needs a size estimate for a new flow as soon as possible. There are two possibilities: (a) block on inference, holding packets until they can be tagged; and (b) operate in a non-blocking fashion, but tag packets with a suitable default estimate until an inference is available. The choice between these modes of operation will depend on the scheduling application and inference latency. For instance, for pFabric, non-blocking operation necessitates tagging based on a zero flow size until inference is available — this ensures that later packets will not be prioritized over earlier packets (and cause in-network reordering and transport problems).

In any case, to maximize *Flux*'s usefulness, inference and communication with the inference module must minimize delay.

The packet tagging module also tracks prediction errors for all flows when they finish, and periodically reports them to the learning module. The measured errors allow the learning module to decide if / when to retrain its model. We use simple error measures, like mean square, absolute error, and coefficient of determination (R^2).

Learning & inference modules: The learning module uses training data received from the data collector to build a model for predicting flow sizes for each job. Based on our conclusions in §5.2, we train a GBDT model with 50 trees. For receiving and processing traces from 1000 machines, we estimate this to consume 40 Mbps data and 24 CPU cores assuming an average flow arrival rate of 1000 flows per second per machine. The learning module also receives error data from the packet tagger, and based on this data, decides whether to update or retrain the inference module, in a manner configurable by the operator.

The most latency-critical operation in *Flux* is inference. Based on our comparison on the use of inference for the current flow and the next flow in §5.2.3, we implement inference for the current flow. This necessitates that inference take as little time as possible to provide timely information for packet tagging.

Although the GBDT implementation provided by the XGBoost library [127] has very good performance, it is optimized for batch execution. With data

Number of trees	20	50	100	500
Lines of C code	2k	7k	16k	82k
CPU overhead	0.1%	0.4%	0.6%	4.3%
Model size (KB)	147	292	531	1721

Table 5.3: Inference overhead for different mode sizes.

center round-trip times on the order of 10 μs, our objective is to achieve inference in a fraction of this time.

To efficiently implement GBDT, we use *treelite* [128], which takes an XGBoost model as input, and transforms it into a single C function, which is a long sequence of simple *if-else* statements. The length of the generated C function depends on the number of trees in the XGBoost model, and their maximum depth. Table 5.3 shows the size of functions generated for models with 20, 50, 100, and 500 trees in terms of thousands of lines of C code. This *treelite*-based approach incurs a minor slowdown when the model is updated, but it improves inference performance by an order of magnitude in comparison to the original library. This approach enables us to make inferences in 5 μs in the typical case.

While low latency is crucial for predictions, we must also not incur large compute overheads for inference. Recall that inference is run independently at hosts. If the average flow inter-arrival time is 1 ms at a host (*i.e.,* 1000 new flows per second starting at this host), this is the average frequency at which we make inferences. The resulting average CPU overhead is shown in Table 5.3 as percentage of a single core. Our choice of 50 trees, imposes little CPU overhead (0.4%).

However, there are many other aspects of the cloud environment that can influence the inference latency, as we show next.

5.6 Deployment environment

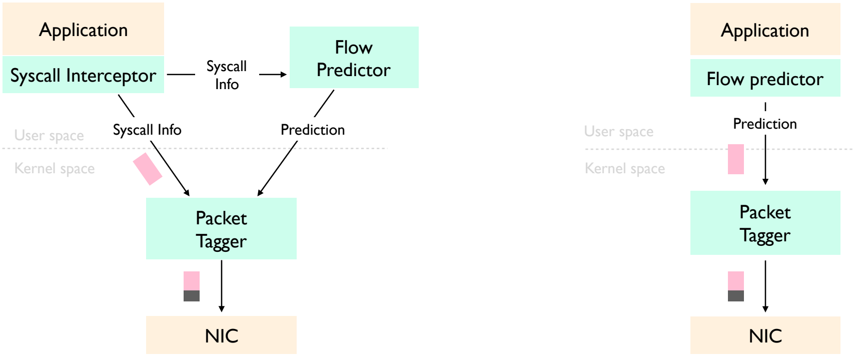
There are a variety of ways in which *Flux* may be embedded into the data center environment, each of which treads the trade-off between *flexibility* and *performance*. We shall explore where *Flux* operates in the application-kernel interface, first in a controlled environment where we can arbitrarily modify the operating system and the application interface to it (§5.6.1); next, if *Flux*'s inference could be implemented in hardware to improve its efficiency (§5.6.2); and finally, how *Flux* may be used in virtualized or containerized environments with and without hardware offload (§5.6.3).

Without loss of generality, in the following, we only consider the pFabric use case, where we need to tag packets with the remaining flow size. For pHost, FastPass, and other use cases, we expect only minor changes to be necessary. We pick this use case as it involves tagging individual packets, the finest granularity of operation.

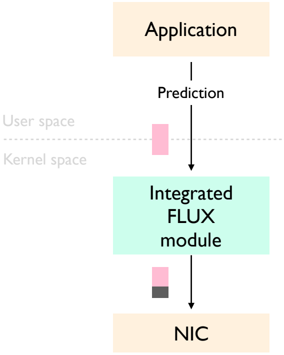
5.6.1 Where does *Flux* operate?

The data collection module must have visibility of I/O calls from the application. Hence, it must be implemented as either a library that intercepts these calls (as described in §5.5), or within the kernel itself. Likewise, the packet tagger must be implemented in the kernel to be able to manipulate packets. The learning module is not on the data path, and may be implemented essentially anywhere, even on a separate machine. The key decision to be made, is: where should we implement the inference module? We explore three possibilities, as outlined in Fig. 5.8.

As a separate process: In this mode of operation, shown in Fig. 5.8(a), *Flux*'s inference module is a standalone process, and receives requests from the syscall interceptor to serve prediction requests whenever a new flow starts. This approach provides high flexibility: deploying a new prediction model or changing an existing one is trivial. Since every model is a single C function, it



(a) *as a process* (b) *as a library*



(c) *as a kernel module*

FIG. 5.8: Flux design options

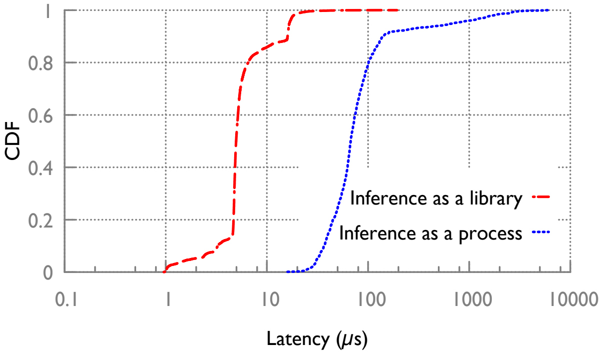


FIG. 5.9: Inference latency with a 50 tree GBDT model implemented in a library vs. in a process

can be compiled to a shared library which is loaded by the inference module process dynamically.

When an application issues a *send*, the data collection module (system call wrapper) records information about the current flow, and sends it to the the inference process, which will send the prediction to the packet tagger. In this case, the prediction path is much longer than the data path, and packets will arrive for tagging before the prediction is available. How much latency would operating in a blocking fashion, waiting for inference, incur?

Fig. 5.9 shows the prediction latency of this approach. The latency is measured across 1000 prediction requests end-to-end, *i.e.*, as the time difference between the arrival of the first packet at the packet tagger and the time when the corresponding prediction arrives at the tagger. The median (95^{th} percentile) latency is $67.4\mu s$ ($720\mu s$) – inevitably, running the inference module as a separate process produces significant latency variation, due to operating system scheduling, context switches, etc.

As an interposed library: To reduce the inter-process communication overhead, the inference module could run in the same library as the data collection module. In this implementation, the *send* is intercepted and then issued to the kernel as soon as the inference finishes. The predicted flow size

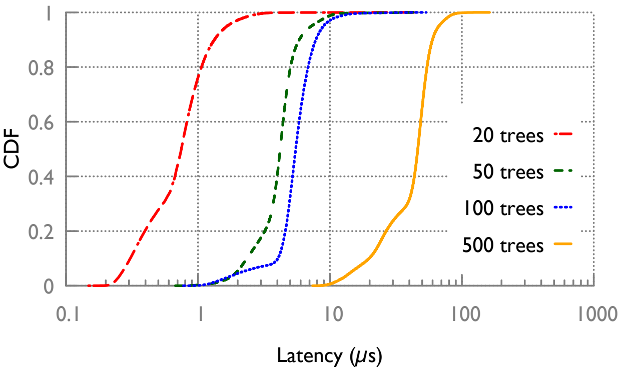


FIG. 5.10: Inference latency as a function of model complexity for GBDT in terms of number of trees in-kernel on a CPU.

is also conveyed to the kernel, where the packet tagger tags packets with this information.

Merging the inference and data collection modules avoids inter-process communication, but this optimization comes at a cost: the system call and the inference have to be executed sequentially, in the same thread, before returning to the application. (Multi-threading in this context could interfere with the logic of the application; in particular, applications which track the number of threads they are using may observe this extra thread.) The application perceives this process as a long system call.

With this approach, one Netlink communication between the kernel and the user space library still remains, which slows down the prediction delivery to the Netfilter. The median latency for inference (measured in the same way as above) is $4.97\mu s$, and the 95th percentile is $16.7\mu s$.

As a kernel module: To eliminate communication overhead, all components except the learning module can be implemented in the kernel space as a single module. This approach provides substantial performance benefits. When an application issues a system call, it is intercepted by the kernel module that runs the inference if a new flow is detected. The prediction is then forwarded to the Netfilter using shared kernel memory. This approach has the least

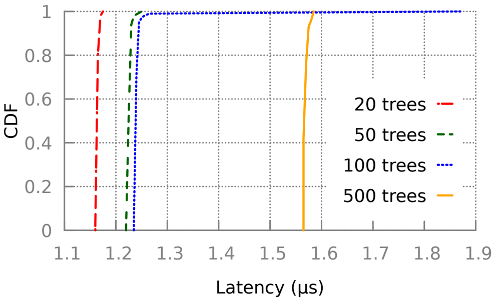


Fig. 5.11: Inference latency using an FPGA.

overhead, but it comes at a cost: different applications may need different inference models, and for each new application, a new inference model must be inserted in the kernel. This is likely to be a difficult proposition.

The latency in this approach comprises entirely of the execution of the inference function. Depending on the size of the model, computational overhead is shown in Fig. 5.10. For the 50-tree model that provides sufficient accuracy on our traces, the average latency is $4.3 \mu s$.

5.6.2 Hardware implementation

We investigate if offloading inference to specialized hardware could improve latency. While such logic could be built into NIC hardware, the FPGAs already in use in some data centers [15] provide an avenue for experimenting with this approach.

An FPGA is a reconfigurable chip with tens of thousands of distributed generic logic cells and hundreds of small memory blocks (few kilo bytes) distributed between logic cells. These distributed logic and memory resources allow us to create tens of custom circuits, each of which has its own low-latency local memory and can operate on either same or different data in parallel. For our decision-tree based inference model, we extend the FPGA implementation from Owaida et al. [129].

This FPGA implementation creates 32 custom circuits each, each of which is capable of processing, in parallel, up to 8 trees of depth up to 10. This effectively provides random access with a fixed latency and bandwidth. Another useful feature of the custom circuits is their programmability with new decision tree models at runtime without the need to change the FPGA implementation.

The FPGA thus provides predictable and deterministic latency. The custom circuits take the same amount of time to perform inference on a tree regardless of the traversed path from the root node to the leaf node. In addition, on FPGAs available to us, this implementation can run up to 256 trees 10-level deep in parallel. As a result, tree ensembles with 256 of fewer trees experience the same latency. The FPGA latency grows linearly with the increase in tree depth or with the number of trees over 256.

Fig. 5.10 and 5.11 compare the latency of inference across different model sizes on a CPU and using an FPGA. The mean latency incurred for the model with 50 trees is $4.3\mu s$ for the CPU implementation, and $1.23\mu s$ using the FPGA. In each case, this includes the end-to-end time elapsed from when a new flow's packet arrives in the kernel to when it has the result for packet tagging. The worst-case latency is bounded by $53\mu s$ on the CPU, and $1.25\mu s$ on the FPGA. (Note the log scale in Fig. 5.10.)

Thus, with flow completion times being at least of the order of a few tens of microseconds, the inference latency can be made negligible in comparison.

5.6.3 More complex environments

So far we discussed a simple, controlled environment, where we have full access to every part of the system, but in practice that is not always the case, with virtualization and network stack offload.

Virtualization: The application may run in a virtualized environment in a guest OS or inside a container. In these settings, *Flux* cannot modify the guest or the container. The container setting is easier, and similar to a non-virtualized environment. For virtualization using a guest OS, *Flux* needs to be interposed in the guest-hypervisor interface. This does not change much in terms of what we can expect in terms of accuracy and performance, because ultimately, the guest is like an "application" running on the host hypervisor, and the network interface is similar at this level, except in cases where some networking functionality is additionally offloaded to hardware.

Hardware offload: In some cases, part of the network stack may be implemented in hardware, or virtual machines may interact directly with hardware, such as with hypervisor bypass using SR-IOV [130]. Nevertheless, these environments still must expose a similar *send*, *rcv* API to underlying layers, which *Flux* can intercept. However, implementation in these settings may necessitate *Flux* being part of a smart NIC. This is fundamental to *any* method of using such packet tagging for network scheduling, because the hypervisor (with bypass) may not have the ability to tag packets at all.

RDMA stacks: RDMA has a significantly different API than TCP. However, even for RDMA networking stacks, the API exposes similar information about sent and received data, which *Flux* can exploit.

5.7 *Flux* limitations

It should be clear that the learning approach is not a panacea. There are several scenarios where it falls short. First and foremost, the prediction context should be clear, *i.e.,* the learning module has to identify the program that is responsible for sending a flow and monitor all features of interest for that flow, as described in §5.2.2. For Spark, the prediction context assumes knowing start time of a job as well as its ID. This is not unreasonable, as noted in §5.5.2.

However, for Web servers, we would have to tie disk and memory reads to particular requests. To demonstrate the effect of missing context, we run Apache Tomcat, serving concurrent clients, so that it is not obvious how to match HTTP requests with corresponding disk reads and responses. In this case, disk reads become almost useless as an indicator, and we can only rely on system calls. The prediction accuracy can be made arbitrarily bad by tweaking the experiment parameters, so we omit a concrete accuracy number.

One possibility for obtaining such context is to apply *vertical context injection* [131], which is deployed in Google's data centers; it tags system calls with application information for easier monitoring and debugging. Further, for the execution of one-shot jobs without repetitive internal structure, there is clearly no learning potential. Likewise, for jobs where large, non-deterministic data volumes are generated (*e.g.,* computationally) for transmission, and there is little repetition across executions, it is unlikely that this approach can succeed.

Thus, for many workloads of practical interest, despite our best efforts, this approach will also be limited.

5.8 Summary

For a wide set of cloud applications, machine learning techniques are a powerful tool for obtaining workload specifications in advance. Moreover, we show that those specifications are accurate and can be obtained with low latency and minimal performance overhead. Further, we use advance workload specifications to enable sophisticated network scheduling systems and demonstrate significant performance improvements in terms of reducing flow completion time.

To motivate cloud providers to deploy specification-dependent control systems on top of their infrastructure, we next focus on providing a set of guarantees and constraints that must be met before these sistems become available in practical deployments.

6

WORKLOAD SPECIFICATION IN PRACTICE

In the example of building Data Center Interconnect (Chapter 3), we demonstrated how to collect and use coarse-grained workload specifications about past behavior of cloud workloads. However, telling something about the future is significantly more difficult (§5). Although advance knowledge is very valuable and can bring an order of magnitude performance improvement (§5.4), obtaining that knowledge is challenging, and in some cases even fundamentally impossible (§5.7). On the other hand, we also demonstrated that having partial knowledge can provide significant improvement in efficiency, and in certain situation, that knowledge can be easily obtained using simple methods and heuristics(§4.3, §5.4).

In reality, cloud providers must operate with *partially-known* workload specifications. Some of those specifications are known, others are not, while there is also a subset of them that are inaccurate due to (ML) methods used for obtaining them. Thus, cloud scheduling and control algorithms must be robust and designed to support each of these classes of workload specifications.

It is important to note that cloud providers and cloud tenants are able to *change* the amount of knowledge about workload specifications. By deploying proposed techniques and methods in various cloud-managed systems, cloud providers are able to increase the amount of workload specification available. Also, cloud tenants can help by specifying and communicating their explicit workload specifications directly to the providers, *e.g.,* declare their flow size before the flow starts, which would be simple for applications like file transfer.

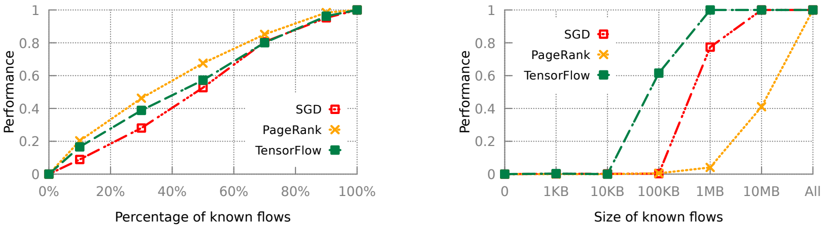
Then, the question is *how to motivate cloud tenants and cloud providers to invest substantial effort and increase the amount of workload specifications?*. Obviously, the effort must be justified by achieving higher efficiency or at least providing a strict guarantee that the performance will not deteriorate with more knowledge added to the system.

Although it sounds intuitive, more knowledge about a particular workload does not always lead to better performance. In fact, adding knowledge about a particular workload in some cases can deteriorate the performance of that very workload as well as of the system as a whole. Thus, we demonstrate that if one is not careful when designing network scheduling and control algorithms, adding knowledge about system's behavior can degrade the performance. In this chapter we discuss algorithms that formally guarantee that more knowledge leads to *nondecreasing* performance, but also indentify systems that do not have this important property. Furthermore, we discuss other properties that current and future systems need to provide in order to efficiently leverage workload specifications and motivate users to cooperate in obtaining more knowledge about cloud workloads.

Outline Sections 6.1 and 6.2 discuss the problem of adding more knowledge about workload specifications in the context of flow and coflow scheduling respectively. Section 6.3 explores how to provide a formally provable guarantee that performance increases when more knowledge is added to the system. Then, Section 6.4 discusses challenges in assuring strict performance guarantees in practise. Finally, Section 6.5 exposes limitations of our approach and discusses future steps that need to be taken in order to make the usage of workload specifications practical in the cloud environment.

6.1 Known vs.unknown in flow scheduling

In this section we explore what happens to the system when we *change* the amount of knowledge about workloads in that system. We demonstrate the effects of such change on the example of flow and coflow scheduling.



(a) *SRF-age, % of flows known* (b) *SRF-age, flows known by size*

FIG. 6.1: As more flow sizes are known, performance improves.

In the case of flow scheduling, we combine two techniques that are known to have great performance for known and unknown traffic. We schedule known flows using the shortest remaining first policy, while for unknown traffic we use flow Aging (§4.5). This is not a new approach. SOAP [132] and Karuna [133] have explored similar scheduling techniques in settings with a *fixed* proportion of flows of known and unknown sizes. For instance, Karuna combines pFabric's shortest remaining first (SRF) approach for known flows with Aging[1] for unknown flows. We refer to this policy as SRF-age, but consider a version with infinitely many priorities, and explore what what happens when the ratio of know and unknown traffic *changes*.

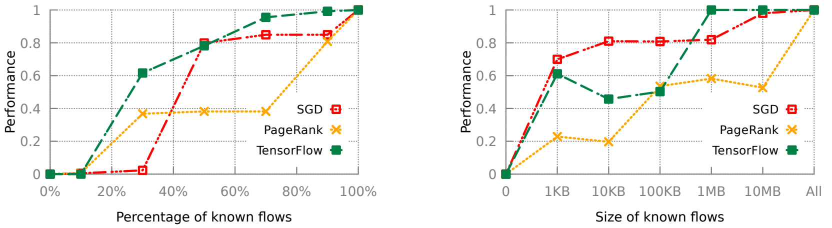
Knowing x% of all flows: If $x = 0$, SRF-age devolves to Aging, and if $x = 100$, it becomes Perfect (pFabric with full knowledge). We define performance with an arbitrary x% of flows known as the following normalization, where FCT_P is mean FCT with policy P:

$$Perf(x) = \frac{FCT_{SRF-age(x)} - FCT_{Aging}}{FCT_{Perfect} - FCT_{Aging}}$$

Fig. 6.1 shows the results on this normalized metric for a sample of workloads from §5.4. As more flow sizes become known, performance improves as we expected.

---

[1] We are simplifying here; Karuna actually uses a multi-level feedback queue, with queue thresholds set based on the flow size distribution.



(a) *Sincronia, % of flows known* (b) *Sincronia, flows known by size*

FIG. 6.2: As more flow sizes are known, performance generally improves but for Sincronia, as larger and larger flows become known, performance sometimes degrades.

Knowing all flows of size up to x bytes: Given that some approaches, like using the initial system call, are more effective at estimating smaller flows, it is worth asking how much benefit knowledge of small flows gives. To evaluate this, we modify SRF-age as follows: We assign priorities to known flows following standard SRF, but for flows larger than x, we use $max(age, x)$. This reflects our confidence that any unknown flow is larger than x bytes.

Fig. 6.1(b) shows that just knowing small flows will not improve performance drastically in terms of mean flow completion time, because they finish near the highest priority even in case of zero knowledge. However, their performance can improve if other, *larger* flows are known, and do not compete with small flows at the same high priority.

6.2 Known vs.unknown in coflow scheduling

Using Sincronia, a state of the art coflow scheduler (§5.3), we also explore the effects of having partial knowledge on coflow scheduling. We generate coflows in the same manner as in §5.4. We run Sincronia offline with 1000 coflows. For known traffic we rely on Sincronia's specific scheduling approach that assumes perfect knowledge. On the other hand, for unknown flows we

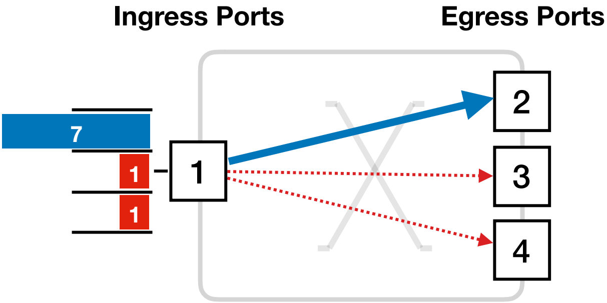


Fig. 6.3: In this scenario, Sinc-mean scheduling policy leads to priority inversion and performance degradation when knowledge about certain flows is added to the system.

had to assume that they are all of the mean flow size for the whole trace, due to the lack of a known or intuitive translation of Aging. We refer to this policy as Sinc-mean.

We normalize performance with partial knowledge in the same manner as in the previous section in the case of flow scheduling, except using coflow completion times (CCT). The results with $x\%$ of flow sizes known and all flows smaller than x bytes known are shown in Fig. 6.2(a) and Fig. 6.2(b) respectively.

In some cases, knowing large fractions of flows does not improve CCT substantially. For instance, for the PageRank workload, knowing 70% of flows still gives more than 60% worse results than with perfect knowledge (Fig. 6.2(a)). This is due to unknown flows within a coflow acting like stragglers.

Fig. 6.2(b), oddly, indicates that sometimes adding knowledge decreases performance. We explain this with an example scenario, following a brief (simplified) overview of Sincronia. Sincronia finds a bottleneck port, *i.e.,* one with the largest number of bytes accumulative across flows; and then assigns the lowest priority to the largest coflow on that link. Flows within a coflow share the same priority.

Now consider a scenario with two coflows, with all their flows going from the same ingress port to different egress ports as shown in Fig. 6.3. Coflow c_1 contains only one flow with 7 packets, and coflow c_2 contains two flows of 1 packet each. The mean *flow* size is thus 3 packets. Regardless of which flows are un/known, with Sinc-mean, the ingress port would correctly be identified as the bottleneck. If all flows are unknown, Sinc-mean would consider all of them to be of the size 3. Sinc-mean would give c_1 higher priority, because its total estimated coflow size is 3 (compared to 6 for c_2), and, thus, finish c_1 first, within 7 time units. Now instead, say we had disclosed the size of c_1's single constituent flow. This leads Sinc-mean to detect c_1 as the larger coflow (with size 7 for c_1 vs. an estimated 6 for c_2) and give higher priority to c_2. In this case, c_1 finishes after c_2 with a coflow completion time of 9 time units. Thus, for c_1, making its size known results in worse performance under Sinc-mean.

6.3 Guaranteed performance improvement

Ideally, we would like the assurance that investing in learning about more flows only improves performance. Otherwise, there are no incentives for data center operators and/or users to change their applications and expose flow size information or to deploy machine learning methods to estimate it.

This property clearly does not hold for Sinc-mean. Also, it is yet unclear how Aging could be incorporated into Sincronia, and whether a partial-knowledge variant can be developed that does not have the quirk of (sometimes) deteriorating when given additional knowledge. However, for the much simpler pFabric/SRF in the context of flow scheduling, we can prove a positive result in this direction, showing that for SRF-age, making a certain flow's size known can never deteriorate its performance, at least when interpreted in a worst-case manner.

To demonstrate this important propery of SRF-age, we used a simplified network model that assumes that all flows go through one link with unlimited output queuing. This output buffer queues packets in flow priority order. This implies that across different flows, packets leave the queue in priority order, but *within* flow packets leave in the same order as they arrive. At every timestep, either a packet leaves the queue, or a new flow arrives. When flows arrive, all their packets are immediately added to this priority queue in their respective positions, with priority ties being broken randomly. To tackle this randomness, we define *worst-case scheduling* for a particular flow f_x as the schedule where any and all ties for f_x's packets break against f_x.

For some flows, their flow sizes are known, and for others, they are not. For flows with unknown sizes, each packet uses the flow's age so far as its priority. The first packet of such a flow has the priority set to zero (highest), with successive packets seeing increments in priority value (*i.e.,* decreasing priority with more packets sent). (For brevity, we omit the distinction between packets and bytes and assume all packets are the same size.) In line with SRF, for known flows, the priority value for their last packet is zero (highest). If the size of a flow f is known, we denote it with f^k; otherwise with f^u. We define priorities such that if $P(p)$ and $P(q)$ are the priorities of packets p and q, then $P(p) > P(q)$ implies p has higher priority, and is scheduled before q.

Theorem 6.3.1. *All else fixed, with SRF-age, learning the flow size of a particular flow f_x cannot deteriorate its worst-case completion time, i.e.,* $FCT(f_x^k) \leq FCT(f_x^u)$.

Proof. To prove the result, we shall show that every packet of any other flow that is scheduled before the end of f_x^k would have also been scheduled before the end of f_x^u, assuming worst-case scheduling for either. It is easy to see that this would imply that the FCT for f_x^k in a worst-case schedule cannot be worse than the FCT of f_x^u.

Suppose a packet r of some other flow is scheduled before a packet p_x^k in f_x^k, given worst-case scheduling for f_x^k. This scheduling implies r has priority higher than or equal to p_x^k, *i.e.,* $P(r) \geq P(p_x^k)$.

Now, say the last packet of f_x^u is l_x^u. Notice that this last packet of f_x^u must have priority lower than or equal to *all* packets of f_x^k, including p_x^k, *i.e.*, $P(l_x^u) \leq P(p_x^k)$. This follows from the definition of SRF-age. If the size of a flow f is $|f|$ packets, then the last packet of f^u (per aging) has priority value $|f|-1$. The n^{th} packet of f^k has priority value $|f|-n$.

Putting the above two inequalities together yields $P(r) \geq P(l_x^u)$. Thus, r would also be scheduled before l_x^u (which is the end of f_x^u), at least in the worst-case schedule for f_x^u. □

A few remarks about this result are in order:

- The theorem and the proof specify worst-case tie-breaking for the flow under consideration. It is easy to produce counterexamples to the theorem statement without the worst-case addendum.
- The definition of SRF-age is central to the result, and it is easy to produce counterexamples for an analogous statement for SRF-mean.
- For systems with a limited number of priority queues, like Karuna or PIAS, the theorem still holds if both known and unknown flows share the same priority thresholds.
- The result can appear counter-intuitive; after all, large unknown flows benefit from high priority in the beginning, which they wouldn't if they were known. While this is true, unknown flows keep slowing down with aging, while known flows keep speeding up with SRF. The proof formalizes this idea.
- Our model, like past work, assumes that scheduling does not change packet inputs to the queue. This is *not true* for TCP flows entering a finite queue.

Mean completion time across all co/flows: Although we have shown that SRF-age cannot deteriorate the performance of a particular flow when its size is made known, it is easy to produce examples where it hurts mean flow

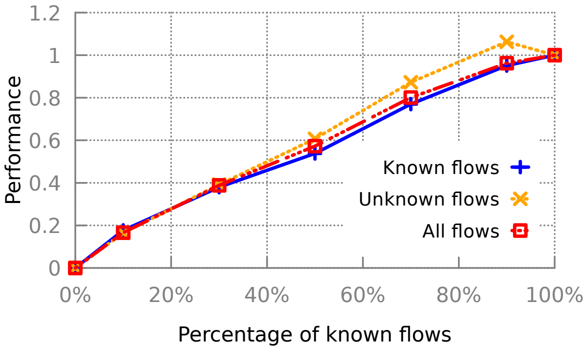


FIG. 6.4: For the TensorFlow workload, with TCP, unknown flows finish faster under SRF-age.

completion time across the set of all flows[2]. While our empirical results show improved mean FCT with SRF-age (and an overall trend for improvement even for mean CCT with Sinc-mean), a fuller analysis of this issue is left to future work.

6.4 Toward practical performance guarantees

Although the theorem from the previous section provides a strict performance guarantee, it does not hold (completely) in practice. Namely, the theorem works only on a simplified network model. For some practical scenarios that include the network software stack, congestion control algorithms, or a multihop topology, the theorem might not hold.

To illustrate the impact of realistic network environment in practice, we take a closer look at the TensorFlow trace in Fig. 6.1(a), separating out the FCTs for known and unknown flows. As Fig. 6.4 shows, unknown flows finish somewhat faster. This apparent deviation from our theorem's result stems from our simple model which ignores TCP dynamics, assuming instead

---

[2] This is also true of Sinc-mean for mean coflow completion time.

that all packets of a flow are available for scheduling at its arrival time. The TensorFlow workload comprises nearly 90% flows of sizes smaller than 100 KB. For unknown flows of this type, Aging results in higher priority in the beginning, allowing TCP's exponential slow-start to grow such flows faster than flows with known sizes.

Incorporating TCP dynamics into our model to potentially bound the disadvantage that known flows can suffer will require substantial additional effort, which is left to future work. While this discrepancy and its impact on scheduling results should be examined in greater detail, this does not take away from our results on incremental benefits from having greater knowledge with SRF-age scheduling overall.

6.5 Limitations

Having a performance guarantee like the one we discussed in the previous section is necessary to motivate both users and cloud operators to start investing time and effort into obtaining workload specifications. However, to make some of the techniques for performance improvement practical in today's cloud, operators need to design and deploy additional systems and adapt existing ones to better support the usage of cloud workload specifications.

Workload Specification API The most accurate and straightforward approach cloud providers can take to obtain workload specifications is to let users communicate those specifications explicitly. For that, besides guranteeing better performance to users who share their workload specifications, cloud operators need to build an efficient API that will be expressive enough to capture key properties of various cloud workloads. Creating such an API would require designing a new set of concepts for describing workload specifications and defining an expressive language together with a communication protocol. Finally, using the API must assure the minimal performance overhead to applications that expose their specifications thorugh it.

Obtaining workload specifications automatically Instead of relying on users to invest the effort and use the API explicitly, cloud providers could deploy systems like *Flux* that obtain worklaod specifications automaticaly. In some cases, these systems do not have to be as sophistcated as *Flux*. Often, workload specifications are already known to the application framework, and can be easily obtained from there. For instance, popular data processing frameworks like Hadoop [134] or Spark [135] already know the size of data they are going to exchange in the shuffle phase, so if cloud providers implement only modest modifications to these two frameworks, the flow size information could be exposed automatically [106].

Protection against cheating Optimizations based on workload specifications, especially user-provided specifications, are vulnerable to intentionally providing misinformation to achieve higher performance. For instance, in the case of flow size scheduling and the shortest remaining time first policy, users can declare all their flows to be very short and achieve the highest flow priority throughout the entire flow execution. To prevent that, cloud operators need to build control mechanisms to monitor the correctness of workload specifications and create a penalty system for users who try to cheat.

Security and privacy concerns Disclosing detailed information about the behavior of a cloud workload poses a security threat for that workload. Knowing how much data a particular program will exchange, when, and with who, can allow attackers to exploit that information and disrupt the regular operation of that program, for instance, by launching a DDoS attack. On top of that, providing too much visibility into the application poses a privacy challenge. Detailed workload specifications can be used to identify and track applications, together with users who use them, or even infer what kind of data a particular application is processing. Thus, cloud providers must handle workload specifications with great care and provide the same security guarantees as for storing other mission-critical data.

Robustness to imprecise workload specifications As we discussed, it is impossible to provide perfect workload specifications in certain situations.

Due to various noise sources like randomness in the workload or using an imprecise method for obtaining the specification, cloud providers must provide mechanisms that can handle errors in workload specifications without negatively impacting the system performance. In the example of flow scheduling (§5.4), we demonstrated that systems like pFabric and FastPass successfully handle relatively small errors in flow size estimates. However, systems like *Iris*, where a decision about network reconfiguration may be made based on those predictions, can have more significant consequences for the overall system performance due to noise in traffic pattern predictions. Thus, cloud providers must protect users from imprecise workload specifications or at least bound their negative influence.

Cost model adjustments In some situations, cloud users cannot see any performance improvement by providing their workload specifications. For instance, in the case of *Iris*, the specification allows providers only to make cheaper infrastructure with the same performance as specification agnostic systems. In those cases, cloud providers must change their billing model and transfer some of the savings they make directly to the clients who enabled the savings in the first place by providing detailed workload specifications.

6.6 Summary

In this section, we demonstrated that the ability to obtain workload specifications is just one step towards improving performance. The equally important aspect of leveraging those specifications is how to motivate both cloud providers and cloud users to start working together on their common goal – making the cloud infrastructure as efficient as possible.

We showed how cloud providers could guarantee performance improvements to those users who put effort into sharing their workload specifications (§6.3). However, to unlock the full potential of using workload specifications in the cloud, there is more to be done. Cloud providers need to build additional infrastructure to tolerate imprecise and incorrect specifications, respect privacy, have provable performance guarantees and bounds, and incentivize users to collaborate in the process by providing higher performance and lower cost of cloud resources.

7

CONCLUSION

7.1 Summary

Having workload specifications that describe the communication patterns and behavior of modern workloads is essential for deploying sophisticated control and scheduling techniques that maximize the efficiency of today's cloud infrastructure.

Workload specifications can describe various aspects of cloud programs. Certain specifications capture coarse-grained insights about the long history of execution of the entire cloud infrastructure. In contrast, others can be much more detailed and describe the fine-grained behavior of a particular application in the future.

Historical insights about cloud workloads are essential for designing physical network infrastructure that are relatively static and must support a wide variety of cloud applications. We demonstrate the power of historical workload specifications by proposing a new Data Center Interconnect architecture, *Iris*, that achieves an order of magnitude improvement in terms of infrastructure cost and flexibility thanks to the insights about the traffic stability across data centers.

On the other hand, for sophisticated network control and scheduling, more valuable are fine-grained specifications that describe the future behavior of individual applications. We explore this part of the workload specification space by proposing a system named *Flux* that automatically obtains knowledge about future events in the cloud system. *Flux* estimates the size of individual network flows and utilizes it for enabling clairvoyant schedulers that provide an order of magnitude improvement in terms of flow completion time and network queue occupancy compared to the systems that are deployed in the cloud today. Besides that, *Flux* demonstrates how to obtain those advance specifications in a timely manner with low latency, which is critical for providing the desired performance benefits.

However, the ability to obtain workload specifications does not make them immediately useful. As we have shown, it is also important to motivate cloud users to collaborate on creating and exposing workload specifications. For that, it is critical that cloud operators provide strict performance guarantees for those users that expose information about their workloads. To further facilitate the specification exchange, cloud providers should focus on building new interfaces and frameworks that allow users to quickly and securely describe their workloads. Also, the providers should change the billing model to stimulate the users who invest additional effort into obtaining workload specifications.

Leveraging workload specifications is necessary for maximizing the performance of modern hardware. Thus, cloud users and providers have to work together toward better defining and understanding workload specifications and utilizing them to create a new generation of workload-specification-based cloud infrastructure.

Chapter 1 introduced the problem of obtaining and using workload specifications, defined the workload specification space, discussed the motivation for deploying specification-based algorithms and systems in the modern cloud, and summarized the major contributions of this thesis.

In *Chapter 2*, we started exploring the workload specification space from the side of coarse-grained historical specifications. We did that in the context of designing and deploying regional cloud networks, which are one of the most expensive components of the modern cloud network infrastructure. First, we ran an analysis of real cloud regions, their physical equipment, and fiber maps, which resulted in a set of constraints that must be met when building a regional network. We also demonstrated the flexibility and latency benefits of distributed regional networks over today's centralized design. As it is implemented today, we show that distributed design comes at a cost so high that it is prohibitive for modern cloud providers to deploy it. Thus, we leveraged the insight based on historical workload specifications about traffic demand and stability to propose a new regional network architecture that minimizes the infrastructure cost and reduces the network complexity.

Chapter 3 introduced *Iris*, a new all-optical regional network design that reduces the cost and complexity of data center interconnect by order of magnitude, compared to today's standard electrical implementations. To minimize the need for expensive transceivers in the context of DCI, *Iris* relies on two major insights. First, there is plenty of relatively cheap fiber in metropolitan areas. Since that fiber has a substantially lower cost than the transceivers necessary to saturate that fiber, *Iris* leverages optical fiber switching to favor having small fiber overhead for completely removing the need for in-network transceivers. Second, since optical fiber-switching infrastructure requires reconfiguration under changes in traffic demand, *Iris* minimizes the negative performance impact of network reconfiguration by leveraging the insights about traffic stability between data center pairs in a cloud region.

Chapter 4 moves the focus from one extreme in the workload specification space to the other. Instead of looking at coarse-grained historical information about cloud workloads, here we explored the potential of automatically obtaining fine-grained details about the future behavior of particular applications. More precisely, we looked at the problem of obtaining flow size information in advance. Flow size information is a critical input to many scheduling techniques that provide an order of magnitude performance improvements compared to flow-size-agnostic approaches. This chapter explored simple methods and heuristics for exposing flow size information in a realistic cloud environment.

Chapter 5 introduced *Flux*, a system that leverages machine learning for obtaining flow size information in advance. *Flux* tries to identify dependencies between resource utilization, various system-level parameters, and past traffic, to estimate the size of the next network flow. This chapter also explores the utility of these, sometimes imprecise, estimates for improving network scheduling in the cloud. Although imperfect, those estimates can still achieve 1.1× to 11.1× performance improvement in terms of flow completion time compared to methods used today that do not rely on advance knowledge.

Finally, *Chapter 6* discussed current and future challenges of using various workload specifications in practical cloud deployments. Obtaining and leveraging workload specifications can require substantial effort from both cloud users and cloud operators. Thus, it is critical to create systems and algorithms that justify the effort and provide strict performance guarantees and bounds, secure specification exchange, and develop expressive languages and abstractions to help users better describe their programs' behavior.

7.2 Research outlook

We made initial steps towards obtaining and leveraging workload specifications and characteristics to improve the efficiency of data centers and cloud networks. However, as we discussed in §6.5, there are more research challenges to be solved before cloud providers can use those specifications to operate and manage their networks optimally.

On the other hand, workload specifications help improve infrastructure efficiency beyond only cloud networks. More general knowledge about application behavior help cloud operators to improve machine utilization, optimize job scheduling decisions, reduce resource fragmentation, save power, and enhance many other aspects of their entire infrastructure.

7.2.1 Future of general workload specifications

Understanding communication patterns and network behavior is only one piece in the general workload specification puzzle. It is also important to obtain knowledge about other components of cloud applications to improve the overall infrastructure efficiency. Previous work has shown that understanding features like changes in CPU utilization, job execution time, memory requirements, or physical hardware preferences of individual applications, all have significant impact on both workload and data center performance [136]–[138]. These detailed workload specifications about general application behavior and resource requirements allow better scheduling and workload placement decisions, minimize the interference between collocated programs that share the same physical hardware, and improve resource utilization [113], [139], [140].

Leveraging general workload specifications in practice has many challenges in common with using network-related specifications discussed in this work. Obtaining general specifications is usually difficult and requires serious effort both from cloud tenants and application developers. For instance, estimating job execution time or CPU utilization in advance has been shown to be problematic at best [141], [142]. Thus, to justify the effort, systems that leverage general job specifications must provide the same set of guarantees described in §6.5 – strict performance improvement bounds, deal with inaccurate specifications, and help users provide specifications and better define their applications' behavior. Many of the ideas presented in this thesis can help in this direction and they are directly applicable to improving efficiency of other resources in the cloud, *e.g.,* our ML model with its low-latency FPGA implementation can be used to predict other characteristics of cloud workloads like future CPU and Memory requirements, and the scheduling technique presented in §6.1 that mixes known and unknown workloads can be adjusted to improve CPU scheduling and provide strict performance guarantees.

In the long term, modern cloud environments need a logically centralized workload specification repository that would collect both automatic and user-defined specifications and make them available for various cloud control and scheduling mechanisms. Initial steps have already been made in this direction [125]. However, the major challenge in the future will be how to combine multiple of these specifications, both automatically obtained and user-defined, to develop comprehensive insights into application needs and improve efficiency across the entire cloud stack.

7.2.2 Integration with user-level metrics

From the perspective of improving infrastructure efficiency, the most important workload specifications are related to the physical resource requirements of cloud programs – how many CPU cycles a particular program needs, how much network traffic will be generated, or what will be the peak memory utilization of one program. Although cloud users may be motivated by performance improvements or cost reductions to invest additional effort and provide explicit resource specifications of their programs, this process is tiring and not natural to them. Users care about metrics that are at a higher level of abstraction and closer to their business problems – 99th percentile request latency, maximum throughput, or achieving 99.99% availability of their service.

Recent work showed progress in understanding and utilizing user-level metrics that describe application goals [143]–[145]. So far, the focus of this body of work was on finding a resource configuration that best fits the needs of a particular application, *e.g.,* correctly allocate CPU and memory resources for a virtual machine. Although useful in some cases, high-level application goals do not contain a sufficient amount of information to be used in situations where fine-grained workload specifications are necessary, *e.g.,* dynamic network reconfiguration or bandwidth reservation. The challenge for future work will be to efficiently capture all user-level metrics and combine them with automatically obtained fine-grained workload specifications to improve the efficiency of the entire cloud infrastructure.

BIBLIOGRAPHY

[1] H. Meyer, J. C. Sancho, J. V. Quiroga, F. Zyulkyarov, D. Roca, and M. Nemirovsky, “Disaggregated computing. an evaluation of current trends for datacentres”, *Procedia Computer Science*, vol. 108, 685, 2017.

[2] A. D. Papaioannou, R. Nejabati, and D. Simeonidou, “The benefits of a disaggregated data centre: A resource allocation approach”, in *2016 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2016, 1.

[3] J. Wang, J. Liu, and N. Kato, “Networking and communications in autonomous driving: A survey”, *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, 1243, 2018.

[4] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, *et al.*, “Towards fully autonomous driving: Systems and algorithms”, in *2011 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2011, 163.

[5] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D. O. Wu, “Improving cloud gaming experience through mobile edge computing”, *IEEE Wireless Communications*, vol. 26, no. 4, 178, 2019.

[6] R. D. Yates, M. Tavan, Y. Hu, and D. Raychaudhuri, “Timely cloud gaming”, in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, IEEE, 2017, 1.

[7] S. S. Sabet, S. Schmidt, S. Zadtootaghaj, B. Naderi, C. Griwodz, and S. Möller, “A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience”, in *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020, 15.

[8] W. Zhang, J. Chen, Y. Zhang, and D. Raychaudhuri, “Towards efficient edge cloud augmentation for virtual reality mmogs”, in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, 1.

[9] T. Kämäräinen, M. Siekkinen, J. Eerikäinen, and A. Ylä-Jääski, “Cloudvr: Cloud accelerated interactive mobile virtual reality”, in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, 1181.

[10] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, *et al.*, “Jupiter rising: A decade of clos topologies and centralized control in google’s datacenter network”, *ACM SIGCOMM computer communication review*, vol. 45, no. 4, 183, 2015.

[11] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, “Inside the social network’s (datacenter) network”, in *ACM SIGCOMM*, 2015.

[12] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, “Achieving high utilization with software-driven wan”, in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013, 15.

[13] X. Zhou, H. Liu, and R. Urata, “Datacenter optics: Requirements, technologies, and trends”, *Chinese Optics Letters*, vol. 15, no. 5, 120008, 2017.

[14] OIF, *400ZR*, `https://www.oiforum.com/technicalwork/hot-topics/400zr-2/`.

[15] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, “A cloud-scale acceleration architecture”, in *IEEE/ACM MICRO*, 2016.

[16] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, “Jellyfish: Networking data centers randomly”, in *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 2012, 225.

[17] M. Besta and T. Hoefler, “Slim fly: A cost effective low-diameter network topology”, in *SC’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 2014, 348.

[18] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture”, in *ACM SIGCOMM*, 2008.

[19] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp)”, in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM ’10, New Delhi, India: ACM, 2010, 63.

[20] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, “Re-architecting datacenter networks and stacks for low latency and high performance”, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, 29.

[21] G. Kumar, N. Dukkipati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, *et al.*, “Swift: Delay is simple and effective for congestion control in the datacenter”, in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, 514.

[22] A. Kuzmanovic, “The power of explicit congestion notification”, in *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, 2005, 61.

[23] K. Ramakrishnan, *The Addition of Explicit Congestion Notification (ECN) to IP*, RFC 3168, 2001.

[24] M. Kühlewind, S. Neuner, and B. Trammell, “On the state of ecn and tcp options on the internet”, in *International Conference on Passive and Active Network Measurement*, Springer, 2013, 135.

[25] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, *The cost of a cloud: Research problems in data center networks*, 2008.

[26] D. Popescu, N. Zilberman, and A. Moore, “Characterizing the impact of network latency on cloud-based applications’ performance”, 2017.

[27] *Akamai online retail performance report: Milliseconds are critical*, `https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp`, (Accessed May 3, 2021).

[28] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications", in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, 350.

[29] R. Potharaju and N. Jain, "When the network crumbles: An empirical study of cloud network failures and their impact on services", in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, 1.

[30] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect", in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, 216.

[31] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen, *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching", in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, 782.

[32] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers", in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, 339.

[33] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng, "Mirror mirror on the ceiling: Flexible wireless links for data centers", *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, 443, 2012.

[34] J. Tatum, G. Landry, D. Gazula, J. Wade, and P. Westbergh, "Vcsel-based optical transceivers for future data center applications", in *Optical Fiber Communication Conference*, Optical Society of America, 2018, M3F.

[35] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu, “Dc. p4: Programming the forwarding plane of a data-center switch”, in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, 1.

[36] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, “P4: Programming protocol-independent packet processors”, *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, 87, 2014.

[37] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, “pFabric: Minimal near-optimal datacenter transport”, in *ACM SIGCOMM*, 2013.

[38] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, “Fastpass: A centralized zero-queue datacenter network”, in *ACM SIGCOMM*, 2014.

[39] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, “Achieving high utilization with software-driven wan”, in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, 2013, 15.

[40] B. Vamanan, J. Hasan, and T. Vijaykumar, “Deadline-aware datacenter tcp (d2tcp)”, in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM ’12, Helsinki, Finland: ACM, 2012, 115.

[41] J. Perry, H. Balakrishnan, and D. Shah, “Flowtune: Flowlet control for datacenter networks”, in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, 421.

[42] M. Chowdhury, Y. Zhong, and I. Stoica, “Efficient coflow scheduling with Varys”, in *ACM SIGCOMM*, 2014.

[43] M. Chowdhury and I. Stoica, “Efficient coflow scheduling without prior knowledge”, in *ACM SIGCOMM*, 2015.

[44] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a globally-deployed software defined wan", *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, 3, 2013.

[45] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)", *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, 115, 2012.

[46] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. Ryan, "C-through: Part-time optics in data centers", in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, 327.

[47] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, "Firefly: A reconfigurable wireless data center fabric using free-space optics", in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, 319.

[48] V. Dukic, G. Khanna, C. Gkantsidis, T. Karagiannis, F. Parmigiani, A. Singla, M. Filer, J. L. Cox, A. Ptasznik, N. Harland, W. Saunders, and C. Belady, "Beyond the mega-data center: Networking multi-data center regions", in *ACM SIGCOMM*, 2020.

[49] V. Dukic, S. A. Jyothi, B. Karlas, M. Owaida, C. Zhang, and A. Singla, "Is advance knowledge of flow sizes a plausible assumption?", in *USENIX NSDI*, 2019.

[50] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally-distributed database", in *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, USENIX Association, 2012, 261.

[51] S. Alam, P. Agnihotri, and G. Dumont, *Aws re:invent. enterprise fundamentals: Design your account and vpc architecture for enterprise operating models.* `https://www.slideshare.net/AmazonWebServices/aws-reinvent-2016-enterprise-fundamentals-design-your`

`-account-and-vpc-architecture-for-enterprise-operating-models-ent203`.

[52] Yevgeniy Sverdlik, *Facebook Rethinks In-Region Data Center Interconnection*, `https://www.datacenterknowledge.com/networks/facebook-rethinks-region-data-center-interconnection`, 2018.

[53] X. Zhou and H. Liu, *Pluggable dwdm: Considerations for campus and metro dci applications*, `https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45713.pdf`, 2016.

[54] M. Filer, J. Gaudette, Y. Yin, D. Billor, Z. Bakhtiari, and J. L. Cox, “Low-margin optical networking at cloud scale”, *J. Opt. Commun. Netw.*, vol. 11, no. 10, C94, 2019.

[55] J. Dean and L. A. Barroso, “The tail at scale”, *Communications of the ACM*, vol. 56, no. 2, 74, 2013.

[56] I. Bozkurt, A. Aguirre, B. Chandrasekaran, P. Godfrey, G. Laughlin, B. Maggs, and A. Singla, “Why is the internet so slow?!”, 2017, 173.

[57] R. Durairajan, P. Barford, J. Sommers, and W. Willinger, “Intertubes: A study of the us long-haul fiber-optic infrastructure”, *Proceedings of the ACM SIGCOMM*, vol. 45, 565, 2015.

[58] D. C. Frontier, *Vertical data centers: ’watts per acre’ guides construction economics*, `https://datacenterfrontier.com/vertical-data-centers-watts-per-acre-guides-construction-economics/`.

[59] S. K. Barker and P. Shenoy, “Empirical evaluation of latency-sensitive application performance in the cloud”, in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, 2010, 35.

[60] I. Pelle, J. Czentye, J. Dóka, and B. Sonkoly, “Towards latency sensitive cloud native applications: A performance study on aws”, in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, IEEE, 2019, 272.

[61] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, “A flexible model for resource management in virtual private networks”, in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM ’99, Cambridge, Massachusetts, USA: Association for Computing Machinery, 1999, 95.

[62] R. Essiambre, G. Kramer, P. J. Winzer, G. J. Foschini, and B. Goebel, “Capacity limits of optical fiber networks”, *Journal of Lightwave Technology*, vol. 28, no. 4, 662, 2010.

[63] T. L. Koch, *Optical Fiber Telecommunications IIIA (Optics and Photonics).* Academic Press, 1997.

[64] A. S. Ahsan, C. Browning, M. S. Wang, K. Bergman, D. C. Kilper, and L. P. Barry, “Excursion-free dynamic wavelength switching in amplified optical networks”, *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 9, 898, 2015.

[65] Polatis, *Series 7000 - 384x384 port software-defined optical circuit switch.*

[66] Calient, *Edge 640 Optical Circuit Switch*, `https://www.calient.net/products/edge640-optical-circuit-switch/`.

[67] A. Mokhtar and M. Azizoglu, “Adaptive wavelength routing in all-optical networks”, *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, 197, 1998.

[68] Q. Cheng, M. Bahadori, M. Glick, and K. Bergman, “Scalable Space-and-Wavelength Selective Switch Architecture Using Microring Resonators”, in *Conference on Lasers and Electro-Optics*, Optical Society of America, 2019, STh1N.4.

[69] A. Bechtolsheim, *400G and 800G Ethernet and Optics*, `https://pc.nanog.org/static/published/meetings/NANOG75/1954/20190220_Martin_Building_The_400G_v1.pdf`.

[70] ACG Research, *Migration to 100G campus connectivity*, `https://www.inphi.com/pdfs/ACG-100G-Campus-Connectivity-Analysis.pdf`.

[71] Q. Cheng, S. Rumley, M. Bahadori, and K. Bergman, “Photonic switching in high performance datacenters”, *Opt. Express*, vol. 26, no. 12, 16022, 2018.

[72] N. Farrington, A. Forencich, G. Porter, P.-C. Sun, J. Ford, Y. Fainman, G. Papen, and A. Vahdat, “A multiport microsecond optical circuit switch for data center networking”, *Photonics Technology Letters, IEEE*, vol. 25, 1589, 2013.

[73] M. Ghobadi, R. Mahajan, A. Phanishayee, N. R. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. C. Kilper, “Projector: Agile reconfigurable data center interconnect”, in *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, 2016, 216.

[74] A. Funnel, K. Shi, P. Costa, H. Ballani, and B. Thomsen, “Hybrid Wavelength Switched-TDMA High Port Count All-Optical Data Centre Switch”, *OSA Journal of Lightwave Technology*, vol. 35, no. 20, 2017.

[75] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, “Helios: A hybrid electrical/optical switch architecture for modular data centers”, in *ACM SIGCOMM*, 2010.

[76] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. L. Rexford, “Optimizing bulk transfers with software-defined optical wan”, English (US), in *SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication, Association for Computing Machinery, Inc, 2016, 87.

[77] A. Juttner, I. Szabo, and A. Szentesi, “On bandwidth efficiency of the hose resource management model in virtual private networks”, in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 1, 2003, 386.

[78] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “Vl2: A scalable and flexible data center network”, *Commun. ACM*, vol. 54, no. 3, 95, 2011.

[79] Y. Li, H. Liu, W. Yang, D. Hu, and W. Xu, “Inter-data-center network traffic prediction with elephant flows”, in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2016, 206.

[80] J. Simsarian, M. Larson, H. Garrett, H. Xu, and T. Strand, “Less than 5-ns wavelength switching with an sg-dbr laser”, *Photonics Technology Letters, IEEE*, vol. 18, 565, 2006.

[81] Ciena, *Optical amplifier modules.*

[82] Ko, K.Y. and Demokan, M.S. and Tam, H.Y., “Transient analysis of erbium-doped fiber amplifiers”, *Photonics Technology Letters, IEEE*, vol. 6, 1436, 1995.

[83] LIGHTWAVE, *Polatis goes large with 192x192 all-optical switch*, `https://www.lightwaveonline.com/optical-tech/transport/article/16664779/polatis-goes-large-with-192x192-alloptical-switch`, 2012.

[84] S. Han, T. J. Seok, N. Quack, B.-W. Yoo, and M. Wu, “Monolithic 50x50 mems silicon photonic switches with microsecond response time”, 2014.

[85] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, and et al., “B4: Experience with a globally-deployed software defined wan”, in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM ’13, Hong Kong, China: Association for Computing Machinery, 2013, 3.

[86] D. M. Marom, P. D. Colbourne, A. D’errico, N. K. Fontaine, Y. Ikuma, R. Proietti, L. Zong, J. M. Rivas-Moscoso, and I. Tomkos, “Survey of photonic switching architectures and technologies in support of spatially and spectrally flexible optical networking [invited]”, *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, no. 1, 1, 2017.

[87] G. Wang, D. G. Andersen, M. Kaminsky, M. Kozuch, T. S. E. Ng, K. Papagiannaki, M. Glick, and L. B. Mummert, “Your data center is a router: The case for reconfigurable optical circuit switched paths.”, in *HotNets*, L. Subramanian, W. E. Leland, and R. Mahajan, Eds., ACM SIGCOMM, 2009.

[88] H. H. Bazzaz, M. Tewari, G. Wang, G. Porter, T. S. E. Ng, D. G. Andersen, M. Kaminsky, M. A. Kozuch, and A. Vahdat, “Switching the optical divide: Fundamental challenges for hybrid electrical/optical datacenter networks”, in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC ’11, Cascais, Portugal: Association for Computing Machinery, 2011.

[89] H. Ballani, P. Costa, I. Haller, K. Jozwik, K. Shi, B. Thomsen, and H. Williams, “Bridging the last mile for optical switching in data centers”, in *Optical Fiber Communication Conference (OFC’18)*, OSA, 2018.

[90] OFC Conference, *Optical Data Center Interconnect: Hot and Highly Competitive*, `https://www.ofcconference.org/en-us/home/about/ofc-blog/2018/february-2018/optical-data-center-interconnect-hot-and-highly/`, 2018.

[91] M. Filer, S. Searcy, Y. Fu, R. Nagarajan, and S. Tibuleac, “Demonstration and performance analysis of 4 tb/s dwdm metro-dci system with 100g pam4 qsfp28 modules”, in *2017 Optical Fiber Communications Conference and Exhibition (OFC)*, 2017.

[92] E. Maniloff, S. Gareau, and M. Moyer, “400g and beyond: Coherent evolution to high-capacity inter data center links”, in *Optical Fiber Communication Conference (OFC) 2019*, Optical Society of America, 2019.

[93] N. Dukkipati and N. McKeown, “Why flow-completion time is the right metric for congestion control”, *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, 59, 2006.

[94] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, “pHost: Distributed Near-optimal Datacenter Transport over Commodity Network Fabric”, in *ACM CoNEXT*, 2015.

[95] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, “Sincronia: Near-optimal network design for coflows”, in *ACM SIGCOMM*, 2018.

[96] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, “Homa: A receiver-driven low-latency transport protocol using network priorities”, in *ACM SIGCOMM*, 2018.

[97] C. Li, M. K. Mukerjee, D. G. Andersen, S. Seshan, M. Kaminsky, G. Porter, and A. C. Snoeren, “Using indirect routing to recover from network traffic scheduling estimation error”, in *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, IEEE, 2017, 13.

[98] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, “CODA: Toward automatically identifying and scheduling coflows in the dark”, in *ACM SIGCOMM*, 2016.

[99] C.-Y. Hong, M. Caesar, and P. B. Godfrey, “Finishing flows quickly with preemptive scheduling”, in *ACM SIGCOMM*, 2012.

[100] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, “Better never than late: Meeting deadlines in datacenter networks”, in *ACM SIGCOMM*, 2011.

[101] Y. Lu, G. Chen, L. Luo, K. Tan, Y. Xiong, X. Wang, and E. Chen, “One more queue is enough: Minimizing flow completion time with explicit priority notification”, in *IEEE INFOCOM*, 2017.

[102] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, “Managing data transfers in computer clusters with Orchestra”, in *ACM SIGCOMM*, 2011.

[103] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, “Decentralized task-aware scheduling for data center networks”, in *ACM SIGCOMM*, 2014.

[104] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. Ryan, “C-through: Part-time optics in data centers”, in *ACM SIGCOMM*, 2010.

[105] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers.", in *USENIX NSDI*, 2015.

[106] H. Wang, L. Chen, K. Chen, Z. Li, Y. Zhang, H. Guan, Z. Qi, D. Li, and Y. Geng, "FlowProphet: Generic and accurate traffic prediction for data-parallel cluster computing", in *IEEE ICDCS*, 2015.

[107] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu, "Hadoopwatch: A first step towards comprehensive traffic forecasting in cloud computing", in *IEEE INFOCOM*, 2014.

[108] I. A. Rai, G. Urvoy-Keller, M. K. Vernon, and E. W. Biersack, "Performance analysis of LAS-based scheduling disciplines in a packet switched network", in *ACM SIGMETRICS*, 2004.

[109] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization", in *ACM SIGCOMM*, 2018.

[110] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection", in *IEEE INFOCOM*, 2011.

[111] A. Mushtaq, R. Mittal, J. McCauley, M. Alizadeh, S. Ratnasamy, and S. Shenker, *Datacenter congestion control: Identifying what is essential and making it practical*, `https://people.eecs.berkeley.edu/~radhika/adsrpt.pdf`, 2017.

[112] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts", in *ACM IMC*, 2017.

[113] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, Í. Goiri, S. Krishnan, J. Kulkarni, *et al.*, "Morpheus: Towards Automated SLOs for Enterprise Clusters.", in *USENIX OSDI*, 2016.

[114] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format + schema", Google Inc., Mountain View, CA, USA, Technical Report, 2011, Revised 2014-11-17 for version 2.1. Posted at `https://github.com/google/cluster-data`.

[115] O. A. Abdul-Rahman and K. Aida, “Towards understanding the usage behavior of Google cloud users: The mice and elephants phenomenon”, in *IEEE CloudCom*, 2014.

[116] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and dynamicity of clouds at scale: Google trace analysis”, in *ACM SoCC*, 2012.

[117] K. LaCurts, J. C. Mogul, H. Balakrishnan, and Y. Turner, “Cicada: Introducing predictive guarantees for cloud networks”, in *USENIX HotCloud*, 2014.

[118] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, “The only constant is change: Incorporating time-varying network reservations in data centers”, in *ACM SIGCOMM*, 2012.

[119] C. Reiss, J. Wilkes, and J. L. Hellerstein, “Google cluster-usage traces: Format + schema”, *Google Inc., White Paper*, 1, 2011.

[120] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, in *Advances in neural information processing systems*, 2012, 1097.

[121] L. C. Jain and L. R. Medsker, *Recurrent Neural Networks: Design and Applications*, 1st. Boca Raton, FL, USA: CRC Press, Inc., 1999.

[122] F. Chollet *et al.*, *Keras*, `https://github.com/keras-team/keras`, 2015.

[123] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A system for large-scale machine learning”, in *USENIX OSDI*, 2016.

[124] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators”, *Neural Netw.*, vol. 2, no. 5, 1989.

[125] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, “Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms”, in *ACM SOSP*, 2017.

[126] G. Kumar, A. Narayan, and P. Gao, *YAPS network simulator*, `https://github.com/NetSys/simulator`, 2015.

[127] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system”, in *ACM SIGKDD*, 2016.

[128] *Treelite : Toolbox for decision tree deployment*, `http://treelite.readthedocs.io/en/latest/`, 2017.

[129] M. Owaida, H. Zhang, C. Zhang, and G. Alonso, “Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms”, in *FPL*, 2017.

[130] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan, “High performance network virtualization with SR-IOV”, in *IEEE HPCA*, 2010.

[131] D. Ardelean, A. Diwan, and C. Erdman, “Performance analysis of cloud applications”, in *USENIX NSDI*, 2018.

[132] Z. Scully, M. Harchol-Balter, and A. Scheller-Wolf, “SOAP: One clean analysis of all age-based scheduling policies”, in *ACM SIGMETRICS*, 2018.

[133] L. Chen, K. Chen, W. Bai, and M. Alizadeh, “Scheduling mix-flows in commodity datacenters with Karuna”, in *ACM SIGCOMM*, 2016.

[134] *Apache hadoop*, `https://hadoop.apache.org/`, (Accessed May 3, 2021).

[135] *Apache spark*, `https://spark.apache.org/`, (Accessed May 3, 2021).

[136] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, *et al.*, “Protean:{vm} allocation service at scale”, in *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, 2020, 845.

[137] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, “Apollo: Scalable and coordinated scheduling for cloud-scale computing.”, in *OSDI*, vol. 14, 2014, 285.

[138] Z. Gong, X. Gu, and J. Wilkes, “PRESS: PRedictive Elastic ReSource Scaling for cloud systems”, in *Network and Service Management (CNSM), 2010 International Conference on*, Ieee, 2010, 9.

[139] F. Caglar and A. Gokhale, "Ioverbook: Intelligent resource-overbooking to support soft real-time applications in the cloud", in *2014 IEEE 7th International Conference on Cloud Computing*, IEEE, 2014, 538.

[140] P. Ambati, Í. Goiri, F. Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, *et al.*, "Providing slos for resource-harvesting vms in cloud platforms", in *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, 2020, 735.

[141] S. Mustafa, I. Elghandour, and M. A. Ismail, "A machine learning approach for predicting execution time of spark jobs", *Alexandria engineering journal*, vol. 57, no. 4, 3767, 2018.

[142] F. Farahnakian, P. Liljeberg, and J. Plosila, "Lircup: Linear regression based cpu usage prediction algorithm for live migration of virtual machines in data centers", in *2013 39th Euromicro conference on software engineering and advanced applications*, IEEE, 2013, 357.

[143] A. Mahgoub, A. M. Medoff, R. Kumar, S. Mitra, A. Klimovic, S. Chaterji, and S. Bagchi, "{optimuscloud}: Heterogeneous configuration optimization for distributed databases in the cloud", in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 2020, 189.

[144] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics", in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, 469.

[145] V. Dukic and A. Singla, "Happiness index: Right-sizing the cloud's tenant-provider interface", *USENIX HotCloud*, 2019.

[146] K. Kikuchi, "Fundamentals of coherent optical fiber communications", *Journal of Lightwave Technology*, vol. 34, no. 1, 157, 2016.

A

APPENDIX

A.1 Physical layer experimental details

More details of the experiment reported in §3.7.2 are discussed here for completeness. Four dual polarization (DP) 200 Gbit/s 16 quadrature amplitude modulation (QAM) optical signals are generated by commercially available real-time coherent transceivers, Acacia AC200 and AC400, to produce 2^{31} pseudo random bit sequences. They are spectrally shaped with a root-raised cosine with a 0.2 roll-off factor and with 15% overhead. An amplified spontaneous emission (ASE) source emulates dense wavelength division multiplexed (DWDM) channels (“Channel emulation”), which are then split and multiplexed with the signals in two separate single mode fibers (SMFs) via two to emulate full C-band lines. It is worth pointing out that at the wavelengths of the live signals no ASE was present by the channel emulator since it was properly filtered by the . In the experiment the signals wavelengths were tuned within the C-band with similar achieved results. At the receiver side the optical signals under test are demultiplexed and sent to coherent receivers [146] to be converted in the electrical domain. The optical-to-electrical converted signals are fed to the application-specific integrated circuit (ASIC)’s analogue to digital conversion for further processing by the ASIC’s digital signal processing, which includes signal recovery, polarization mode dispersion and chromatic dispersion compensation, before SD-FEC decoding. Pre-FEC BER measurements are taken every 10 msec and the received powers are kept within the range of the receiver’s optimal performance.

Insets of Fig. 3.7 show examples of typical constellation diagrams of the tested signals in our experiments. The constellation diagrams of the tested signals are shown once converted in the electrical domain at different points in the system. They display the signals as a two-dimensional plane diagram in the complex plane at symbol sampling instants. The angle of a point, measured counter-clockwise from the horizontal axis, represents the phase shift of the carrier wave from a reference phase, given by the local oscillator in the coherent receiver. The distance of a point from the plane origin represents a measure of the amplitude or power of the signal. As expected for 16 QAM signals, 16 distinct symbols are visible for both polarizations, x and y. The cloud associated to each symbol is caused by noise. Due to transmission impairments, the constellation diagrams at the end of the system are characterized by a higher degree of noise as compared to the ones before transmission, but are within the range of acceptable received performance, as confirmed by the BER measurements reported in Fig.3.8. Insets of Fig.3.7 further show spectral traces that cover the whole C-band before and after the fiber spans. The traces show how *Iris* emulates missing channels to fill the unused spectrum while at the same time keeping per-channel power roughly constant so that no per-channel power management is required. These traces are measured in the frequency/wavelength domain using an optical spectrum analyzer.

CURRICULUM VITAE

Personal data

Name	Vojislav Dukic
Place of birth	Novi Sad, Serbia

Education

2016 – 2021	PhD candidate, distributed systems and networking *ETH Zurich*, Switzerland
2014 – 2015	Master of science, applied computer science *FTN Novi Sad*, Serbia
2010 – 2014	Bachelor of science, Electrical and computer engineering *FTN Novi Sad*, Serbia

Employment

2019	Research intern *MSR*, Cambridge, UK
2015	Research intern *EPFL*, Lausanne, Switzerland