Adaptive Protocols and Reconfigurable Optical Interconnects for Datacenter Networks

vorlegt von

Vamsi Addanki, M.Sc.

ORCID: 0000-0002-0577-0413

an der Fakultät IV – Elektrotechnik und Informatik, Technische Universität Berlin

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Axel Küpper
Gutachter: Prof. Dr. Stefan Schmid
Gutachter: Prof. Dr. Rachit Agarwal
Gutachter: Prof. Dr. Michael Schapira
Gutachterin: Prof. Dr. Anja Feldmann
Gutachterin: Prof. Dr. Maria Apostolaki

Tag der wissenschaftlichen Aussprache: 13. Dezember 2024

Berlin 2025

To Mom and Dad

Acknowledgments

I am deeply grateful to my PhD advisor, Stefan Schmid, for the opportunity to pursue this thesis. His guidance, unwavering support, and encouragement have been invaluable throughout my research. I feel fortunate to have had an advisor who strikes the perfect balance — and has shown me how to find it myself — between addressing fundamental questions and designing practical, real-world systems. Stefan not only pointed me to the right direction but also taught me fundamental concepts, techniques, and gave me the freedom to explore and develop my ideas. I am grateful for his mentorship, which has been instrumental in shaping my research and my career.

I would like to extend my heartfelt thanks to Chen Avin for his thought-provoking discussions, insightful feedback, and invaluable guidance. I am deeply grateful to Maciej Pacut for introducing me to the field of online algorithms. I would also like to thank Maria Apostolaki, whose co-supervision of my MSc thesis and continuous encouragement inspired me to pursue a PhD. I am also grateful to Manya Ghobadi for her guidance, for hosting me as a visiting student at MIT, and for her valuable insights and feedback. My sincere gratitude goes to Laurent Vanbever for his supervision during my MSc thesis and for supporting me through the PhD application process. I am thankful to Wei Bai, Prateesh Goyal and Ilias Marinos, for introducing me to timely and important problems in hyperscale datacenters. Additionally, I wish to thank my former supervisors, Dario Rossi, Leonardo Linguaglossa, and Luigi Iannone, whose guidance and support laid the foundation for this journey.

I would like to extend my sincere gratitude to the members of my thesis committee for their invaluable time in reviewing this work. I am also deeply grateful to the anonymous reviewers in the double-blind peer-review process of Networking conferences for their thoughtful comments and constructive suggestions, which greatly enriched this work. Additionally, I have had the privilege of collaborating with many talented researchers from around the world: Maria Apostolaki, Chen Avin, Wei Bai, Manya Ghobadi, Prateesh Goyal, Theo Jepsen, Ilias Marinos, Oliver Michel, Maciej Pacut, Gábor Rétvári, Laurent Vanbever, and many others, whose insights and perspectives have profoundly shaped my research.

Heartfelt thanks to the members of the INET group for their engaging discussions, unwavering support, random coffee breaks, and memorable weekends. Thanks not only for random whiteboard sessions but also for joining random concert plans.

I am profoundly grateful to my family for their boundless love, encouragement, and support. Special thanks to my sister for her invaluable support on various occasions. I also want to honor my late grandfathers, whose guidance and wishes inspired me to pursue my PhD. A heartfelt thanks to my dad, who has always been my blueprint for work ethic.

Lastly, I want to express special thanks to all the great musicians, including my mom, whose music and life journeys have been a constant source of inspiration and motivation in all my work.

Abstract

Datacenter networks form the backbone of modern computing and storage, driving the expansion of online services and applications. As these networks evolve, the demand for higher bandwidth and lower latency has increased significantly. Recently, GPU clusters have emerged within datacenters for large-scale distributed training, presenting unique networking challenges.

This thesis addresses several critical challenges in datacenter networks—congestion control, load balancing, buffer sharing, and reconfigurable optical interconnects—and is structured into three main parts.

First, we address the transport protocol problem in modern datacenters. We propose PowerTCP, a novel congestion control algorithm that dynamically adjusts the congestion window based on the bandwidth-window product (or "power"), a new congestion indicator. Through both analytical and empirical validation, we demonstrate PowerTCP's practicality in datacenter networks, showing that it meets key requirements for high throughput and low latency. We further introduce Ethereal, a transport protocol specifically designed for distributed training workloads in leaf-spine topologies. Ethereal achieves optimal load balancing, akin to packet spraying, by minimally splitting a few application flows while maintaining single-path transport semantics from the network's perspective. Empirically, we show that Ethereal surpasses state-of-the-art algorithms in collective communication completion times.

Second, we tackle the buffer-sharing problem in datacenter switches. We propose ABM, an innovative buffer-sharing algorithm that ensures isolation across different traffic classes while improving burst absorption. As an extension, we introduce Reverie, a solution that enables lossy and lossless traffic to coexist within the same network. Reverie preserves isolation between these traffic types while enhancing burst absorption and flow completion times for both. Additionally, we propose Credence, the first buffer-sharing algorithm to integrate machine-learned predictions. Our analysis shows that Credence achieves near-optimal throughput under perfect predictions and performs effectively even with imperfect predictions, significantly improving flow completion times in empirical tests.

Finally, as Moore's law approaches its limits, we address the challenge of high-performance optical interconnects in datacenter networks. We present the first formal result on the throughput of periodic networks, establishing an equivalence to a corresponding static network. Based on this result, we propose Mars, a demand-oblivious reconfigurable optical interconnect that achieves near-optimal throughput and low latency across various traffic patterns, even with shallow buffers. Additionally, we introduce Vermilion, a demand-aware optical interconnect that dynamically reconfigures according to traffic patterns. Our analysis and empirical results show that Vermilion delivers high throughput across diverse traffic patterns, exceeding the throughput capabilities of demand-oblivious interconnects.

Zusammenfassung

Rechenzentrumsnetzwerke bilden das Rückgrat moderner Computer- und Speichersysteme und treiben die Expansion von Online-Diensten und -Anwendungen voran. Mit der Weiterentwicklung dieser Netzwerke steigt die Nachfrage nach höherer Bandbreite und geringerer Latenz deutlich. In jüngster Zeit sind GPU-Cluster für groß angelegte verteilte Trainingsaufgaben in Rechenzentren entstanden und stellen besondere Herausforderungen an die Netzwerktechnik.

Diese Arbeit behandelt mehrere zentrale Herausforderungen in Rechenzentrumsnetzwerken—Staukontrolle, Lastverteilung, Pufferspeicherung und rekonfigurierbare optische Interconnects—und ist in drei Hauptteile gegliedert.

Zunächst beschäftigen wir uns mit dem Transportprotokollproblem in modernen Rechenzentren. Wir schlagen PowerTCP vor, einen neuartigen Algorithmus zur Staukontrolle, der das Staukontrollfenster dynamisch auf Grundlage des Produktes aus Bandbreite und Fenster (oder „Power“), einem neuen Stauindikator, anpasst. Durch analytische und empirische Validierung zeigen wir, dass PowerTCP für Rechenzentrumsnetzwerke praktisch ist und wichtige Anforderungen an hohen Durchsatz und geringe Latenz erfüllt. Zusätzlich stellen wir Ethereal vor, ein Transportprotokoll, das speziell für verteilte Trainingsaufgaben in Leaf-Spine-Topologien entwickelt wurde. Ethereal erreicht eine optimale Lastverteilung, ähnlich dem „Packet Spraying“, indem es einige wenige Anwendungsflüsse minimal aufteilt, während es aus Sicht des Netzwerks eine Single-Path-Transportsemantik beibehält. Empirisch zeigen wir, dass Ethereal in Bezug auf die Kollektivkommunikationsabschlusszeiten den aktuellen Stand der Technik übertrifft.

Im zweiten Teil befassen wir uns mit dem Pufferspeicherungsproblem in Switches von Rechenzentren. Wir schlagen ABM vor, einen innovativen Pufferspeicherungsalgorithmus, der eine Isolation zwischen verschiedenen Verkehrsklassen gewährleistet und gleichzeitig die Aufnahme von Bursts verbessert. Als Erweiterung stellen wir Reverie vor, eine Lösung, die ermöglicht, dass verlustbehafteter und verlustfreier Verkehr im gleichen Netzwerk koexistieren. Reverie bewahrt die Isolation zwischen diesen Verkehrstypen und verbessert gleichzeitig die Burstaufnahme und Abschlusszeiten von Flüssen beider Typen. Darüber hinaus schlagen wir Credence vor, den ersten Pufferspeicherungsalgorithmus, der maschinell erlernte Vorhersagen integriert. Unsere Analyse zeigt, dass Credence bei perfekten Vorhersagen nahezu optimalen Durchsatz erreicht und auch bei unvollkommenen Vorhersagen effektiv arbeitet. Unsere empirischen Tests zeigen, dass Credence die Abschlusszeiten für Rechenzentrums-Workloads erheblich verbessert.

Schließlich, da das Ende von Moore's Law näher rückt, befassen wir uns mit der Herausforderung leistungsstarker optischer Interconnects in Rechenzentrumsnetzwerken. Wir präsentieren das erste formale Ergebnis über den Durchsatz periodischer Netzwerke und zeigen dabei eine Äquivalenz zu einem entsprechenden statischen Netzwerk auf. Auf Grundlage dieses Ergebnisses schlagen wir Mars vor, einen nachfrageunabhängigen rekonfigurierbaren optischen Interconnect, der nahezu optimalen Durchsatz und geringe Latenz über verschiedene Verkehrsmuster hinweg erreicht, selbst bei geringen Puffern. Zusätzlich stellen wir Vermilion vor, einen nachfragebewussten optischen Interconnect, der sich dynamisch je nach

Verkehrsmustern rekonfiguriert. Unsere Analyse und empirischen Ergebnisse zeigen, dass Vermilion bei verschiedenen Verkehrsmustern einen hohen Durchsatz erzielt und die Durchsatzgrenzen nachfrageunabhängiger Interconnects ßbertrifft.

Contents

1

Preface

Datacenter networks are the backbone of modern computing and storage, driving the rapid expansion of online services and applications. As datacenter networks evolve, the demand for higher bandwidth and lower latency has increased significantly. In fact, the aggregate traffic generated by servers in Google datacenters increased by 50x within a span of six years between 2008 and 2014, indicating an exponential growth [246]. The aggregate traffic and network demand is expected to increase even faster with the emergence of GPU clusters within datacenters, in the context of large-scale distributed training.

The performance of applications running in a datacenter critically depends on the underlying infrastructure and the protocol stack. More than a decade after the inception of hyperscale datacenters and the advent of software-defined networking [160], several challenges remain in the design and operation of datacenter networks. Novel network architectures such as RDMA [58, 105, 194, 212, 287], NIC offload [200], programmable switches [73], reconfigurable networks [104], and optical interconnects [60, 79, 109, 189, 221] have been shown to improve scalability and performance but have also introduced novel challenges.

1.1 Problems

In this thesis, we focus on three key challenges in datacenter networks: *(i)* transport protocol, *(ii)* buffer sharing and *(iii)* network topology.

Transport protocol: Server networking stack typically employs a congestion control algorithm to control the rate of transmission, effectively dictating the network utilization and the latency observed by the applications in a datacenter [164]. Several algorithms have been proposed in the past decade [35, 193, 259] that are tailored for datacenter networks. The gradual shift from Kernel networking to fully hardware-accelerated networking has also necessitated the redesign of these algorithms. Further, the increase in bandwidth of network links and the decrease in latency of network devices have also necessitated a rethinking of the design of these algorithms. The challenge is to design a congestion control algorithm that is responsive, fair, and efficient, while ensuring that it can be deployed in practice. Importantly, RDMA networks have also introduced a new class of constraints e.g., ordered packet delivery, zero packet loss, and flow control, that need to be considered in the design of these algorithms. Unfortunately, existing algorithms [164, 173, 193, 287] have limitations in terms of throughput, convergence, and responsiveness. While a congestion control algorithm controls the rate of transmission, a load balancing algorithm determines the path taken by packets in the network, aiming to uniformly distribute the load across the network topology.

Equal cost multipath (ECMP) routing based on 5-tuple hash has long been used by network switches to achieve load balancing. However, hash collisions in ECMP can lead to suboptimal load balancing, congestion, long queueing delays, and increased flow completion times. These limitations are more pronounced in the context of distributed training workloads [13, 71] since the tail completion time of collective communication is critical for the overall training time. In view of these problems, this thesis aims at answering the following questions:

Question 1 (Congestion control)**.** *If the switches in the network support fine-grained congestion feedback with in-band network telemetry, then can we design an end-host congestion control algorithm that converges to near-zero queue lengths without losing throughput?*

Question 2 (Load balancing)**.** *Given a large-scale fattree datacenter topology running a distributed training workload, can we design a load balancing algorithm that achieves near-optimal performance without requiring significant modifications to commodity network hardware?*

Buffer sharing: Network switches utilize buffers to manage transient traffic bursts and enhance network utilization. Typically, buffer space is shared across all the ports of a switch. In the early stages of networking, on-chip buffer space was abundant, leading to concerns about bufferbloat [107]. However, in modern high-speed datacenter networks, the slowdown of Moore's law has made buffers a limited and costly resource, resulting in the widespread use of shallow buffers [75]. This trend in shrinking buffer sizes poses a significant challenge to consistently absorb transient bursts in the network traffic. Buffer sharing has become a critical issue in datacenter switches, particularly in lossless RDMA networks. The primary challenge lies in developing buffer sharing algorithms that not only maximize network utilization but also improve burst absorption, which is crucial for datacenter workloads. We tackle the following questions in this thesis, aiming to answer both the practical and theoretical aspects of buffer sharing algorithms:

Question 3 (Practical buffer sharing)**.** *Can we design a buffer sharing algorithm for datacenter switches that provides predictable burst absorption, fairness, and isolation across traffic classes, while remaining implementable in hardware?*

Question 4 (Optimal buffer sharing)**.** *Under practical constraints, how close can we get to the performance of an ideal buffer sharing algorithm that has complete foresight of future traffic?*

Reconfigurable optical interconnects: The topology of a datacenter network is a key factor in determining the performance of the applications it supports. It influences the routing of packets, the number of hops traversed, and the latency experienced. The topology also directly impacts the network's throughput, fault tolerance, and overall cost. Traditionally, datacenter topologies have been constructed using electrical packet switches, represented as static graphs [31,65,157,179,256,281]. However, with the advent of optical switches and reconfigurable networks, new possibilities have emerged for datacenter topology design. Optical switches offer better bandwidth scalability and energy efficiency compared to electrical switches,

with the potential to provide higher throughput and lower latency for datacenter workloads. This has spurred substantial research into reconfigurable optical interconnects [60,188,189,246]. However, there is limited understanding in the literature regarding the throughput bounds of such networks, and how demand-oblivious and demand-aware network designs compare in terms of performance. We tackle several fundamental questions on the design and performance of reconfigurable networks in this thesis:

Question 5 (Demand-oblivious periodic reconfigurable network)**.** *What is the design space for demand-oblivious reconfigurable networks, and what are the tradeoffs between throughput, delay and buffer requirements?*

Question 6 (Demand-aware periodic reconfigurable network)**.** *To what extent can demand-aware periodic networks improve throughput compared to their demand-oblivious counterparts?*

Question 7 (Demand-aware network design under demand uncertainty)**.** *Can we design demand-aware networks that are robust to demand estimation errors, without compromising throughput when the estimations are correct?*

1.2 Thesis Outline & Contributions

This thesis is organized into three parts, focusing on the three problems described in the previous section. Each part consists of chapters that delve into the problem and propose solutions that answer the main questions posed in the previous section. The chapters are organized as follows:

Part I: Datacenter Transport

- Chapter 2 presents **PowerTCP**, a congestion control algorithm that is tailored for the needs of modern datacenter networks. Increasingly stringent throughput and latency requirements in datacenter networks demand fast and accurate congestion control. We observe that the reaction time and accuracy of existing datacenter congestion control schemes are inherently limited. They either rely only on explicit feedback about the network state (e.g., queue lengths in DCTCP [35]) or only on variations of state (e.g., RTT gradient in TIMELY [193]). To overcome these limitations, we propose a novel congestion control algorithm, PowerTCP, which achieves much more fine-grained congestion control by adapting to the bandwidth-window product (referred as ‘‘Power’’ in this work). PowerTCP leverages in-band network telemetry [156] to react to changes in the network instantaneously without loss of throughput and while keeping queues short. Due to its fast reaction time, our algorithm is particularly well-suited for dynamic network environments and bursty traffic patterns. We show analytically and empirically that PowerTCP can significantly outperform the state-of-the-art in both traditional datacenter topologies and emerging reconfigurable datacenters where frequent bandwidth changes make congestion control challenging. In traditional datacenter networks, PowerTCP reduces tail flow completion times of short flows by 80% compared to DCQCN [287] and TIMELY [193], and by 33% compared to HPCC [173] even at 60% network load. In reconfigurable datacenters, PowerTCP achieves 85% circuit

utilization without incurring additional latency and cuts tail latency by at least 2x compared to existing approaches.

- Chapter 3 presents **Ethereal**, a load balancing algorithm tailored for modern GPU clusters. Large-scale distributed training in production datacenters constitutes a challenging workload bottlenecked by network communication. In response, both major industry players (e.g., Ultra Ethernet Consortium [13]) and parts of academia [71] have surprisingly, and almost unanimously, agreed that packet spraying is *necessary* to improve the performance of large-scale distributed training workloads. In this work, we challenge this prevailing belief and pose the question: *How close can a singlepath transport approach an optimal multipath transport?* We demonstrate that singlepath transport (from a NIC's perspective) is sufficient and can perform nearly as well as an ideal multipath transport with packet spraying, particularly in the context of distributed training in leaf-spine topologies. Our assertion is based on four key observations about workloads driven by collective communication patterns: *(i)* flows within a collective start almost simultaneously, *(ii)* flow sizes are nearly equal, *(iii)* the completion time of a collective is more crucial than individual flow completion times, and *(iv)* flows can be split upon arrival. We analytically prove that singlepath transport, using minimal flow splitting (at the application layer), is equivalent to an ideal multipath transport with packet spraying in terms of maximum congestion. Our preliminary evaluations support our claims. This work suggests an alternative agenda for developing next-generation transport protocols tailored for large-scale distributed training.

Part II: Datacenter Switch Buffer Sharing

- Chapter 4 presents **ABM**, a buffer sharing algorithm for datacenter switches. Today's network devices share buffer across queues to avoid drops during transient congestion and absorb bursts. As the buffer-per-bandwidth-unit in datacenter decreases, the need for optimal buffer utilization becomes more pressing. Typical devices use a hierarchical packet admission control scheme: First, a Buffer Management (BM) scheme decides the maximum length per queue at the device level and then an Active Queue Management (AQM) scheme decides which packets will be admitted at the queue level. Unfortunately, the lack of cooperation between the two control schemes leads to *(i)* harmful interference across queues, due to the lack of isolation; *(ii)* increased queueing delay, due to the obliviousness to the per-queue drain time; and *(iii)* thus unpredictable burst tolerance. To overcome these limitations, we propose ABM, Active Buffer Management which incorporates insights from both BM and AQM. Concretely, ABM accounts for both total buffer occupancy (typically used by BM) and queue drain time (typically used by AQM). We analytically prove that ABM provides isolation, bounded buffer drain time and achieves predictable burst tolerance without sacrificing throughput. We empirically find that ABM improves the 99th percentile flow completion times for short flows by up to 94% compared to the state-of-the-art buffer management. We further show that ABM improves the performance of advanced datacenter transport protocols in terms of flow

completion times by up to 76% compared to DCTCP [35], TIMELY [193] and PowerTCP [21] under bursty workloads even at moderate load conditions.

- Chapter 5 presents **Reverie**, a buffer sharing algorithm that builds upon ABM in order to serve both lossless and lossy traffic classes. The switch buffers in datacenters today are dynamically shared by traffic classes with different loss tolerance and reaction to congestion signals. In particular, while legacy applications use loss-tolerant transport, e.g., DCTCP [35], newer applications require lossless datacenter transport, e.g., RDMA over Converged Ethernet [116]. Unfortunately, as we analytically show in this work, the buffer-sharing practices of today's datacenters pose a fundamental limitation to effectively *isolate* RDMA and TCP while also maximizing *burst absorption*. We identify two root causes: *(i)* the buffer-sharing for RDMA and TCP relies on two independent and often conflicting views of the buffer, namely ingress and egress; and *(ii)* the buffer-sharing scheme micromanages the buffer and overreacts to the changes in its occupancy during transient congestion. In this work, we present Reverie, a buffer-sharing scheme, which, unlike prior works, is suitable for both lossless and loss-tolerant traffic, providing isolation and better burst absorption than state-of-the-art buffer-sharing schemes. At the core of Reverie lies a unified (consolidated ingress and egress) admission control that jointly optimizes the buffers for both RDMA and TCP. Reverie allocates buffer based on a low-pass filter that naturally absorbs bursty queue lengths during transient congestion within the buffer limits. Our evaluation shows that Reverie can improve the performance of RDMA as well as TCP in terms of flow completion times by up to 33%.

- Chapter 6 presents **Credence**, a first buffer sharing algorithm augmented with machine-learned predictions. Packet buffers in datacenter switches are shared across all the switch ports in order to improve the overall throughput. The trend of shrinking buffer sizes in datacenter switches makes buffer sharing extremely challenging and a critical performance issue. Literature suggests that push-out buffer sharing algorithms have significantly better performance guarantees compared to drop-tail algorithms. Unfortunately, switches are unable to benefit from these algorithms due to lack of support for push-out operations in hardware. Our key observation is that drop-tail buffers can emulate push-out buffers if the future packet arrivals are known ahead of time. This suggests that augmenting drop-tail algorithms with predictions about the future arrivals has the potential to significantly improve performance. This work is the first research attempt in this direction. We propose Credence, a drop-tail buffer sharing algorithm augmented with machine-learned predictions. Credence can unlock the performance only attainable by push-out algorithms so far. Its performance hinges on the accuracy of predictions. Specifically, Credence achieves near-optimal performance of the best known push-out algorithm LQD (Longest Queue Drop) with perfect predictions, but *gracefully* degrades to the performance of the simplest drop-tail algorithm Complete Sharing when the prediction error gets arbitrarily worse. Our evaluations show that Credence improves throughput by 1.5x compared to traditional approaches. In terms of flow completion times, we show that Credence improves upon the state-of-the-art approaches by up to 95% using off-the-shelf machine learning techniques that are also

practical in today's hardware. We believe this work opens several interesting future work opportunities both in systems and theory that we discuss at the end of Chapter 6.

Part III: Reconfigurable Datacenter Networks

- Chapter 7 presents **Mars**, an oblivious optical interconnect that maximizes throughput within the delay and buffer constraints of the network. This chapter also presents fundamental throughput and latency bounds for oblivious optical interconnects, including their buffer requirements. The performance of large-scale computing systems often critically depends on high-performance communication networks. Dynamically reconfigurable topologies, e.g., based on optical circuit switches, are emerging as an innovative new technology to deal with the explosive growth of datacenter traffic. Specifically, *periodic* reconfigurable datacenter networks (RDCNs) such as RotorNet [189], Opera [187] and Sirius [60] have been shown to provide high throughput, by emulating a *complete graph* through fast periodic circuit switch scheduling.

  However, to achieve such a high throughput, existing reconfigurable network designs pay a high price: in terms of potentially high delays, but also, as we show as a first contribution in this work, in terms of the high buffer requirements. In particular, we show that under buffer constraints, emulating the high-throughput complete graph is infeasible at scale, and we uncover a spectrum of unvisited and attractive alternative RDCNs, which emulate regular graphs, but with lower node degree than the complete graph.

  We present Mars, a periodic reconfigurable topology which emulates a d-regular graph with near-optimal throughput. In particular, we systematically analyze how the degree d can be optimized for throughput given the available buffer and delay tolerance of the datacenter. We further show empirically that Mars achieves higher throughput compared to existing systems when buffer sizes are bounded.

- Chapter 8 presents **Vermilion**, a simple demand-aware optical interconnect that breaks the throughput bounds of existing periodic demand-oblivious networks. The increasing gap between datacenter traffic volume and the capacity of electrical switches has driven the development of reconfigurable network designs utilizing optical circuit switching. Recent advancements, particularly those featuring periodic fixed-duration reconfigurations, have achieved practical end-to-end delays of just a few microseconds. However, current designs rely on multi-hop routing to enhance utilization, which can lead to a significant reduction in worst-case throughput and added overhead from congestion control and routing complexity. These factors pose significant operational challenges for the large-scale deployment of these technologies. We present Vermilion, a reconfigurable optical interconnect that breaks the throughput barrier of existing periodic reconfigurable networks, without the need for multi-hop routing — thus eliminating congestion control and simplifying routing to direct communication. Vermilion adopts a demand-aware approach while retaining the simplicity of periodic fixed-duration reconfigurations, similar to RotorNet [189]. We establish throughput bounds for Vermilion, demonstrating that it achieves at least 33% more throughput in

the worst-case compared to existing designs. The key innovation of Vermilion is its *short* demand-aware periodic schedule, derived using a matrix rounding technique. This schedule is then combined with a demand-oblivious periodic schedule to efficiently manage any residual demand. Our evaluation results support our theoretical findings, revealing significant performance gains for datacenter workloads.

- Chapter 9 augments **Vermilion** with machine-learned predictions about the network demand matrix and analyzes its performance under demand uncertainty. As Moore's Law approaches its limits, reconfigurable optical interconnects are becoming increasingly vital for accommodating the rapid growth of datacenter traffic. In this context, periodic and fixed-duration circuit-switching technologies have emerged as viable solutions. However, the scheduling of these circuits is typically derived offline in a demand-oblivious manner, failing to adapt to evolving traffic patterns. Recent studies have established a throughput bound of $\frac{1}{2}$ for demand-oblivious periodic circuit-switching networks [18, 42]. Notably, the existing algorithm Vermilion [17] breaks this bound, achieving a throughput of $\frac{2}{3}$ through a demand-aware approach while utilizing fixed-duration periodic circuit switching. This advancement relies on the assumption that communication patterns — represented by the underlying demand matrix — are known in advance. A critical and unresolved question remains: *To what extent can demand-aware networks enhance throughput in the presence of demand uncertainty?*

  In this work, we build upon the Vermilion algorithm [17] to enhance the design of reconfigurable optical networks using machine-learned predictions. Our main contribution is the formalization of demand-aware network design in an online setting aimed at maximizing throughput, relying solely on historical communication patterns without foresight into future demands. We augment Vermilion with machine-learned predictions regarding the underlying demand matrix and analyze its performance through competitive analysis. We demonstrate that the augmented Vermilion achieves $\frac{\gamma-1}{\gamma}$-consistency under perfect predictions and exhibits $\frac{1}{2\cdot\gamma}$-robustness under arbitrarily poor predictions, where γ serves as a trust factor. Moreover, we introduce a randomized algorithm based on Vermilion, revealing an entire spectrum of consistency-robustness tradeoff curves. We evaluate the performance of Vermilion with predictions through extensive simulations on both real-world and synthetic traffic matrices under varying prediction accuracies. Our results underscore the potential of machine-learned predictions to enhance the throughput of reconfigurable optical interconnects while highlighting the challenges posed by prediction errors.

  This work is a first research effort and represents an initial exploration towards integrating machine learning into the design of reconfigurable optical networks. We believe this work opens several opportunities for future research in both systems and theory, which we discuss at the end of Chapter 9.

During the course of my work on this thesis, several papers have been published or are currently under submission. Some chapters in this thesis are based on these papers, while others have been excluded to ensure a cohesive and comprehensive

narrative. The papers included in this thesis primarily reflect my contributions, supported by valuable intellectual input from my co-authors. Notably, Maria Apostolaki is a co-first author on ABM [16], where she identified the main drawbacks of the state-of-the-art. My contributions involved the analysis of the existing literature, as well as the design and analysis of ABM. In the other works, the first author served as the primary contributor.

[17] **Vermilion: A Traffic-Aware Reconfigurable Optical Interconnect with Formal Throughput Guarantees**
Vamsi Addanki, Chen Avin, Goran Dario Knabe, Giannis Patronas, Dimitris Syrivelis, Nikos Terzenidis, Paraskevas Bakopoulos, Ilias Marinos, Stefan Schmid.
(Under Review) Paper

[22] **Vermilion, Pt. 2: Tradeoffs between Throughput and Prediction Accuracy in Reconfigurable Optical Interconnects**
Vamsi Addanki, Maciej Pacut, Leon Kellerhals, Goran Dario Knabe and Stefan Schmid.
(Under Review)

[20] **Ethereal: Divide and Conquer Network Load Balancing in Large-Scale Distributed Training**
Vamsi Addanki, Prateesh Goyal, Ilias Marinos, and Stefan Schmid.
(Under Review) Paper

[177] **Pyrrha: Congestion-Root-Based Flow Control to Eliminate Head-of-Line Blocking in Datacenter**
Kexin Liu, Zhaochen Zhang, Chang Liu, Yizhi Wang, Qingyue Wang, Vamsi Addanki, Stefan Schmid, Wei Chen, Xiaoliang Wang, Jiaqi Zheng, Wenhao Sun, Tao Wu, Ke Meng, Fei Chen, Weiguang Wang, Wanchun Dou, Guihai Chen, and Chen Tian.
USENIX NSDI 2025 Paper

[28] **Dequeue Rate-Agnostic Switch Buffer Sharing through Packet Queueing Delay**
Krishna Agrawal, Vamsi Addanki, and Habib Mostafaei.
CoNEXT-SW 2024 Paper

[122] **Starlink Performance through the Edge Router Lens**
Sarah-Michelle Hammer, Vamsi Addanki, Max Franke, and Stefan Schmid.
LEO-NET 2024 Paper

[24] **Credence: Augmenting Datacenter Switch Buffer Sharing with ML Predictions**
Vamsi Addanki, Maciej Pacut, and Stefan Schmid.
USENIX NSDI 2024 Paper Slides Code

[19] **Reverie: Low Pass Filter-Based Switch Buffer Sharing for Datacenters with RDMA and TCP Traffic**
Vamsi Addanki, Wei Bai, Stefan Schmid, and Maria Apostolaki.
USENIX NSDI 2024 Paper Slides Code

[127] **TCP's Third-Eye: Leveraging eBPF for Telemetry-Powered Congestion Control**
Jörn-Thorben Hinz, Vamsi Addanki, Csaba Györgyi, Theo Jepsen, and Stefan Schmid.
eBPF 2023 (SIGCOMM Workshop) Paper Slides Code

[18] **Mars: Near-Optimal Throughput with Shallow Buffers in Reconfigurable Datacenter Networks**
Vamsi Addanki, Chen Avin, and Stefan Schmid.
ACM SIGMETRICS 2023 Paper Slides

[23] **Self-Adjusting Partially Ordered Lists**
Vamsi Addanki, Maciej Pacut, Arash Pourdamghani, Gábor Rétvári, Stefan Schmid, and Juan Vanerio.
IEEE INFOCOM 2023 Paper

[16] **ABM: Active Buffer Management in Datacenters**
Vamsi Addanki, Maria Apostolaki, Manya Ghobadi, Stefan Schmid, and Laurent Vanbever.
ACM SIGCOMM 2022 Paper Slides Code

[21] **PowerTCP: Pushing the Performance Limits of Datacenter Networks**
Vamsi Addanki, Oliver Michel, and Stefan Schmid.
USENIX NSDI 2022 Paper Slides Code

Part I

Datacenter Transport

2

Congestion Control

The performance of more and more cloud-based applications critically depends on the underlying network, requiring datacenter networks (DCNs) to provide extremely low latency and high bandwidth. For example, in distributed machine learning applications that periodically require large data transfers, the network is increasingly becoming a bottleneck [173]. Similarly, stringent performance requirements are introduced by today's trend of resource disaggregation in datacenters where fast access to remote resources (e.g., GPUs or memory) is pivotal for the overall system performance [173]. Building systems with strict performance requirements is especially challenging under bursty traffic patterns as they are commonly observed in datacenter networks [53, 83, 219, 271, 282].

These requirements introduce the need for fast and accurate network resource management algorithms that optimally utilize the available bandwidth while minimizing packet latencies and flow completion times. Congestion control (CC) plays an important role in this context being "a key enabler (or limiter) of system performance in the datacenter" [164]. In fact, fast reacting congestion control is not only essential to efficiently adapt to bursty traffic [149, 230], but is also becoming increasingly important in the context of emerging reconfigurable datacenter networks (RDCNs) [55, 60, 109, 163, 187, 189, 234]. In these networks, a congestion control algorithm must be able to quickly ramp up its sending rate when high-bandwidth circuits become available [201].

Traditional congestion control in datacenters revolves around a bottleneck link model: the control action is related to the state i.e., queue length at the bottleneck link. A common goal is to efficiently control queue buildup while achieving high throughput. Existing algorithms can be broadly classified into two types based on the feedback that they react to. In the following, we will use an analogy to electrical circuits[1] to describe these two types. The first category of algorithms react to the absolute network state, such as the queue length or the RTT: a function of network "effort" or **voltage** defined as the sum of the bandwidth-delay product and in-network queuing. The second category of algorithms rather react to variations, such as the change of RTT. Since these changes are related to the network "flow", we say that these approaches depend on the **current** defined as the total transmission rate. We tabulate our analogy and corresponding network quantities in Table 2.1. According to this classification, we call congestion control protocols such as CUBIC [118], DCTCP [35], or Vegas [74] **voltage-based CC** algorithms as they react to absolute properties such as the bottleneck queue length, delay, Explicit Congestion Notification (ECN), or loss. Recent proposals such as

[1]This analogy is inspired from S. Keshav's lecture series based on mathematical foundations of computer networking [154]. We emphasize that our power analogy is meant for the networking context considered in this work and it should not be applied to other domains of science.

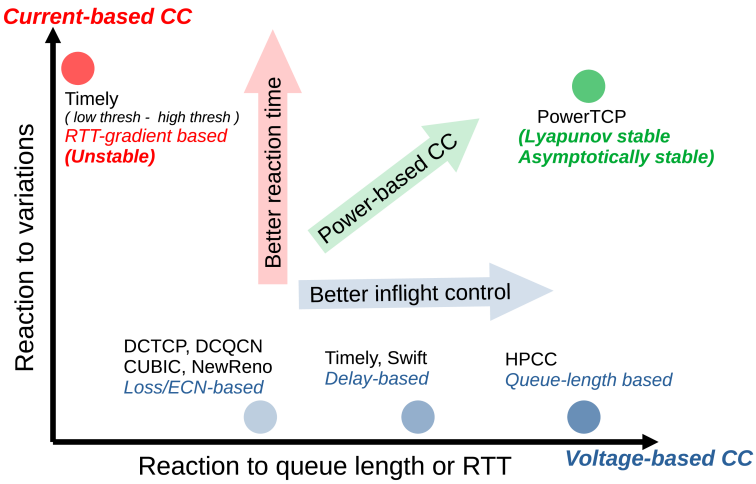


Figure 2.1: Existing congestion control algorithms are fundamentally limited to a single dimension in their window (or rate) update decisions and are unable to distinguish between two scenarios across multiple dimensions.

Quantity	Analogy
Total transmission rate (network flow)	Current (λ)
BDP + buffered bytes (network effort)	Voltage (ν)
Current $\times$ Voltage	Power (Γ)

Table 2.1: Analogy between metrics in networks and in electrical circuits. Note that the network here is the ‘‘pipe’’ seen by a flow and not the whole network.

TIMELY [193] are **current-based CC** algorithms as they react to the variations, such as the RTT-gradient. In conclusion, we find that existing congestion control algorithms are fundamentally limited to one of the two dimensions (voltage or current) in the way they update the congestion window.

We argue that the input to a congestion control algorithm should rather be a function of the two-dimensional state of the network (i.e., both voltage and current) to allow for more informed and accurate reaction, improving performance and stability. In our work, we show that there exists an accurate relationship between the optimal adjustment of the congestion window, the network voltage and the network current. We analytically show that the optimal window adjustment depends on the product of network voltage and network current. We call this product **network power**: current $\times$ voltage, a function of both queue lengths and queue dynamics.

Figure 2.1 illustrates our classification. Existing protocols depend on a single dimension, voltage or current. This can result in imprecise congestion control as the protocol is unable to distinguish between fundamentally different scenarios, and, as a result, either reacts too slowly or overreacts, both impeding performance. Accounting for both voltage and current, i.e., power, balances accurate inflight control and fast reaction, effectively providing the best of both worlds.

In this chapter we present PowerTCP, a novel *power*-based congestion control algorithm that accurately captures both *voltage* and *current* dimensions for every control action using measurements taken within the network and propagated through in-band network telemetry (INT). PowerTCP is able to utilize available

bandwidth within one or two RTTs while being stable, maintaining low queue lengths, and resolving congestion rapidly. Furthermore, we show that PowerTCP is Lyapunov-stable, as well as asymptotically stable and has a convergence time as low as five update intervals (§2.2.4). This makes PowerTCP highly suitable for today's datacenter networks and dynamic network environments such as in reconfigurable datacenters.

PowerTCP leverages in-network measurements at programmable switches to accurately obtain the bottleneck link state. Our switch component is lightweight and the required INT header fields are standard in the literature [173]. We also discuss an approximation of PowerTCP for use with non-programmable, legacy switches.

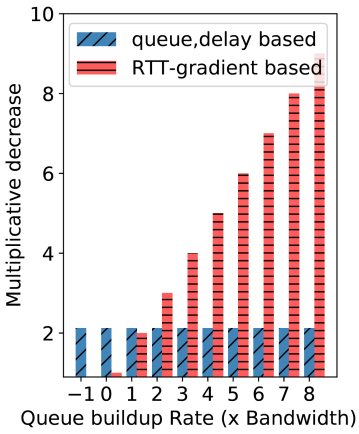
To evaluate PowerTCP, we focus on a deployment scenario in the context of RDMA networks where the CC algorithm is implemented on a NIC. Our results from large-scale simulations show that PowerTCP reduces the 99.9-percentile short flow completion times by 80% compared to DCQCN [287] and by 33% compared to the state-of-the-art low-latency protocol HPCC [173]. We show that PowerTCP maintains near-zero queue lengths without affecting throughput or incurring long flow completion times even at 80% load. As a case study, we explore the benefits of PowerTCP in reconfigurable datacenter networks where it achieves $80 - 85\%$ circuit utilization and reduces tail latency by at least $2\times$ compared to the state-of-the-art [201]. Finally, as a proof-of-concept [127], we implemented PowerTCP in the Linux kernel and the telemetry component on an Intel Tofino programmable line-rate switch using P4 [90].

In summary, our key contributions in this chapter are:

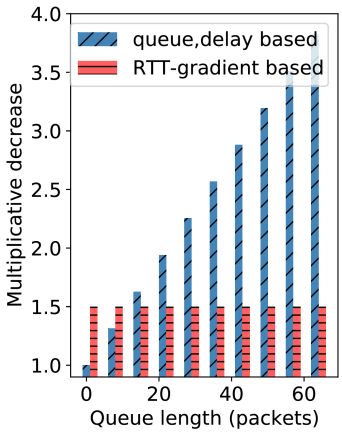
- We reveal the shortcomings of existing congestion control approaches which either only react to the current state or the dynamics of the network, and introduce the notion of *power* to account for both.
- PowerTCP, a power-based approach to congestion control at the end-host which reacts faster to changes in the network such as an arrival of burst, fluctuations in available bandwidth etc.,
- An evaluation of the benefits of PowerTCP in traditional DCNs and RDCNs.
- As a contribution to the research community and to facilitate future work, all our artefacts have been made publicly available at: https://github.com/inet-tub/ns3-datacenter.

2.1 Motivation

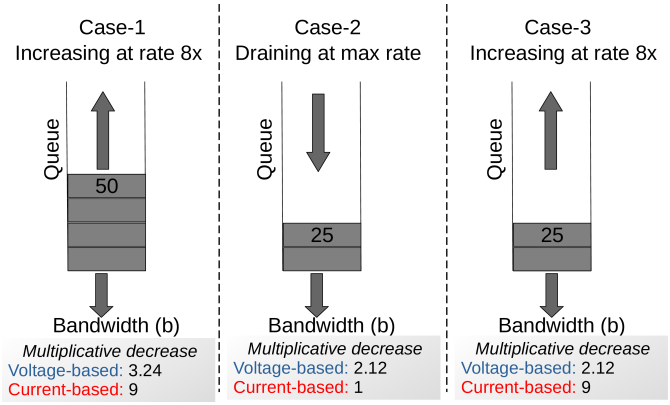
We first provide a more detailed motivation of our work by highlighting the benefits and drawbacks of existing congestion control approaches. In the following, **voltage-based CC** refers to the class of end-host congestion control algorithms that react to the state of the network in absolute values related to the bandwidth-delay product, such as bottleneck queue length, delay, loss, or ECN; **current-based CC** refers to the class of algorithms that react to changes in the state, such as the RTT-gradient. Voltage-based CC algorithms are likely to exhibit better stability but are fundamentally limited in their reaction time. Current-based CC algorithms



(a) Voltage-based CC is oblivious to queue buildup rate.

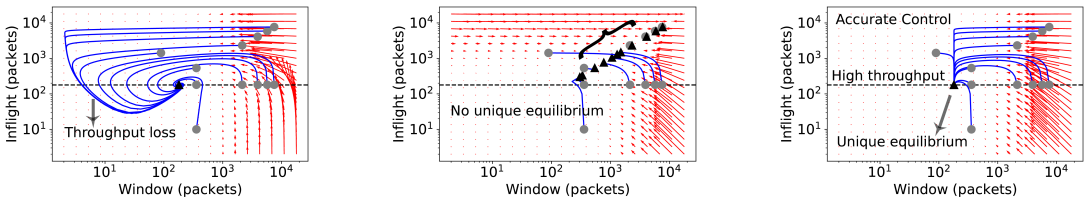


(b) Current-based CC is oblivious to queue lengths.



(c) Voltage-based CC cannot differentiate case-2 vs case-3; whereas current-based CC cannot differentiate case-1 vs case-3.

Figure 2.2: Existing CC schemes, classified as voltage and current-based, are orthogonal in their response to queue length and queue buildup rate.



(a) Voltage-based CC (RTT or queue length) exhibits equilibrium properties but has an imprecise reaction leading to throughput loss.

(b) Current-based CC (RTT-gradient) reacts faster but has no unique equilibrium point, and is thereby unable to stabilize queue lengths.

(c) PowerTCP, a power-based CC, exhibits equilibrium properties and has a precise reaction to perturbations.

Figure 2.3: Phase plots showing the trajectories of existing schemes and our approach PowerTCP from different initial states (circles) to equilibrium (triangles). At each point on the plane, arrows show the direction in which the system moves. An example is depicted with bottleneck link bandwidth 100Gbps and a base RTT of 20μs. BDP is shown by a horizontal dotted line and any trajectory going below this line indicates throughput loss.

detect congestion faster but ensuring stability may be more challenging. Indeed, TIMELY [193], a current-based CC, deployed at Google datacenters, turned out to be unstable [288] and evolved to SWIFT [164], a voltage-based CC.

Orthogonal to our approach, receiver-driven transport protocols [124, 131, 199] have been proposed which show significant performance improvements. A receiver-driven transport approach relies on the assumption that datacenter networks are well-provisioned and claims that congestion control is unnecessary; for example "NDP performs no congestion control whatsoever in a Clos topology" [124]. The key difference is that receiver-driven approaches take feedback from the ToR downlink at the receiver which can only identify congestion at the last hop, whereas sender-based approaches rely on a variety of feedback signals to identify congestion anywhere along the path. In this work, we focus on the sender-based congestion control approach which can in principle handle congestion anywhere along the round-trip path between a sender and a receiver, even in oversubscribed datacenters.

To take a leap forward and design fine-grained datacenter congestion control algorithms, we present an analytical approach and study the fundamental problems faced by existing algorithms. We first formally express the desirable properties of a datacenter congestion control law (§2.1.1) and then analytically identify the drawbacks of existing control laws (§2.1.3). Finally, we discuss the lessons learned and formulate our design goals (§2.1.4).

2.1.1 Desirable Control Law Properties

Among various desired properties of datacenter congestion control, high throughput and low tail latency are most important [35, 173, 193] with fairness and stability being essential as well [277, 288]. Achieving these properties simultaneously can be challenging. For example, to realize high throughput, we may aim to keep the queue length at the bottleneck link large; however, this may increase latency.

Thus, an ideal CC algorithm must be capable of maintaining near-zero queue lengths, achieving both high throughput and low latency. It must further minimize throughput loss and latency penalty caused by perturbations, such as bursty traffic.

In order to formalize our requirements, we consider a single-bottleneck link model widely used in the literature [129, 192, 277, 288]. Specifically, we assume that all senders use the same protocol, transmit long flows[2] sharing a common bottleneck link with bandwidth b, and have a base round trip time τ (excluding queuing delays). In this model, equilibrium is a state reached when the window size and bottleneck queue length stabilize. We now formally express the desired equilibrium state that captures our performance requirements in terms of the sum of window sizes of all flows (aggregate window size) $w(t)$, bandwidth delay product $b \cdot \tau$, and bottleneck queue length $q(t)$:

$$0 < q(t) < \epsilon \tag{2.1}$$

$$b \cdot \tau \leq w(t) < b \cdot \tau + \epsilon$$

$$\dot{q}(t) = 0;\ \dot{w}(t) = 0$$

where ϵ is a positive integer. First, this captures the requirement for high throughput i.e., when $w(t) > b \cdot \tau$ and $q(t) > 0$, the number of inflight bytes are greater than the bandwidth-delay product (BDP) and the queue length is greater than zero. Second, from $w(t) < b \cdot \tau + \epsilon$ and $q(t) < \epsilon$, the queue length is at most ϵ, thereby achieving low latency. Finally, for the system to stabilize, we need that $\dot{q}(t) = 0$ and $\dot{w}(t) = 0$.

As simple as these requirements are, it is challenging to control the aggregate window size $w(t)$ while CC operates per flow. In addition to the equilibrium state requirement, we need fast response to perturbations. The response must minimize the distance from the equilibrium i.e., minimize the latency or throughput penalty caused by a perturbation (e.g., incast or changes in available bandwidth).

In this work, we ask two fundamental questions:

***(Q1)* Equilibrium point:** Do existing algorithms satisfy the equilibrium state in Eq. 2.1 for the aggregate window size?

In addition to the equilibrium behavior, we are also interested in the reaction to a perturbation.

***(Q2)* Response to perturbation:** What is the trajectory followed after a perturbation, i.e., the dynamics of the bottleneck queue as well as the TCP window sizes, from an initial point to the equilibrium point?

2.1.2 A Simplified Analytical Model

We now aim to analytically answer our questions above and shed light on the inefficiencies of existing protocols, both voltage-based and current-based. Table 2.2 tabulates the main notations used in this chapter. We begin by simplifying the congestion avoidance model of existing CC approaches we are interested in, specifically delay, queue length, and RTT-gradient based CC approaches as follows:

$$w_i(t + \delta t) = \gamma \cdot \left(w_i(t) \cdot \frac{e}{f(t)} + \beta \right) + (1 - \gamma) \cdot w_i(t) \tag{2.2}$$

[2]Note that, although most DC flows are short flows, most DC traffic volume (bytes) is from long flows [35, 37].

Notation	Description
b	bottleneck bandwidth
q	bottleneck queue length
τ	base RTT
t^f	sender to bottleneck delay
θ	round trip time RTT
w_i	window size of a flow i
w	aggregate window size (of all flows)
γ	EWMA parameter
β	additive increase
e	desired equilibrium point
f	feedback
λ_i	sending rate of a flow i
λ	Current: aggregate sending rate
ν	Voltage
Γ	Power

Table 2.2: Key notations used in this chapter. Additionally for any variable say x, $\dot{x}$ denotes its derivative with respect to time i.e., $\frac{dx}{dt}$.

Here w_i is the window of a flow i, β is the additive increase term, e is the equilibrium point that the algorithm is expected to reach, $f(t)$ is the measured feedback and γ is the exponential moving average parameter. A queue length-based CC [173] sets the desired equilibrium point e as $b \cdot \tau$ (BDP) and the feedback $f(t)$ as the sum of bottleneck queue length and BDP i.e., voltage (ν). A delay-based CC [164] sets e to τ (base RTT) and the feedback $f(t)$ as RTT which is the sum of queuing delay and base RTT i.e., $\frac{\text{voltage}}{\text{bandwidth}}$ ($\frac{\nu}{b}$). Similarly, the RTT-gradient approach [193] sets e to 1 and the feedback $f(t)$ as one plus RTT-gradient i.e., $\frac{\text{current}}{\text{bandwidth}}$ ($\frac{\lambda}{b}$). In the following, we further justify how Eq. 2.2 captures existing control laws[3]. Note that our simplified model does not capture loss/ECN-based CC algorithms; however, there exists rich literature on the analysis of loss/ECN-based CC algorithms [129, 183] including DCTCP [35, 36]. We now use Euler's first order approximation to obtain the window dynamics as follows:

$$\dot{w}_i(t) = \frac{\gamma}{\delta t} \cdot \left(w_i(t) \cdot \frac{e}{f(t)} - w_i(t) + \beta \right) \tag{2.3}$$

Each flow i has a sending rate λ_i and hence the bottleneck queue experiences an aggregate arrival rate of λ. In our analogy, λ is the network current. We additionally use the traditional model of queue length dynamics which is independent of the control law [129, 192]:

$$\dot{q}(t) = \begin{cases} \lambda(t - t^f) - \mu(t) & q(t) > 0 \\ 0 & otherwise \end{cases} \tag{2.4}$$

where $\lambda(t) = \frac{w(t)}{\theta(t)}$. An equilibrium point is the window size w_e and queue length q_e that satisfies $\dot{w}(t) = 0$ and $\dot{q}(t) = 0$.

[3]TIMELY [193], for example, is rate-based while our simplification is window-based. However, window and rate are interchangeable for update calculations.

We now justify how the simplified model approximately captures the existing control laws. Our simplified model for congestion window update at time $t + \delta t$ is defined in Eq. 2.5 as a function of current congestion window size, a target e, the feedback $f(t)$, an additive increase β and an exponential moving average parameter γ.

$$w_i(t+\delta t) = \underbrace{\gamma \cdot \underbrace{\left(w_i(t) \cdot \frac{e}{f(t)} + \beta\right)}_{update} + (1-\gamma) \cdot w_i(t)}_{EWMA} \tag{2.5}$$

where e and $f(t)$ are given by,

$$e = \begin{cases} b \cdot \tau & queue\text{-}length\ based\ CC \\ \tau & delay\text{-}based\ CC \\ 1 & RTT\text{-}gradient\ based\ CC \end{cases} \tag{2.6}$$

$$f(t) = \begin{cases} q(t-\theta(t)+t^f) + b \cdot \tau & queue\text{-}length\ based\ CC \\ \frac{q(t-\theta(t)+t^f)}{b} + \tau & delay\text{-}based\ CC \\ \frac{\dot{q}(t-\theta(t)+t^f)}{b} + 1 & RTT\text{-}gradient\ based\ CC \end{cases} \tag{2.7}$$

We first use Euler's first order approximation and obtain the aggregate window ($\sum w$) dynamics for the simplified model,

$$\dot{w}(t) = \frac{\gamma}{\delta t} \cdot \left(w(t) \cdot \frac{e}{f(t)} - w(t) + \beta\right) \tag{2.8}$$

In order for the system to stabilize, we require $\dot{q}(t) = 0$ and $\dot{w}(t) = 0$. Using Eq. 2.17 and Eq. 2.8 and applying equilibrium conditions and assuming that $f(t)$ stabilizes,

$$q_e = w_e - b \cdot \tau \tag{2.9}$$

$$w_e = \frac{\hat{\beta}}{1 - \frac{e}{f}} \tag{2.10}$$

Recall that $\hat{\beta} = \sum_{\beta_i}$, the sum of additive increase terms of all flows sharing a bottleneck. To show whether there exists a unique equilibrium point, it remains to show whether Eq. 2.9 and Eq. 2.10 have a unique solution for w_e and q_e. We now show how the simplified model captures existing control laws and show the equilibrium properties.

Queue length or inflight-based control law: Substituting $e = b \cdot \tau$ and $f(t) = q(t - \theta(t) + t^f) + b \cdot \tau$, we express the simplified queue length based control law as,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t) \cdot b \cdot \tau}{q(t-\theta(t)+t^f) + b \cdot \tau} + \beta\right) + (1-\gamma) \cdot w_i(t) \tag{2.11}$$

notice that the update is an MIMD based on inflight bytes. Eq. 2.11 captures control laws based on inflight bytes; for example HPCC [173].

There exists a unique equilibrium point for a system defined by a queue length-based control law given Eq. 2.11 and the queue length dynamics given by Eq. 2.17. It can be observed that Eq. 2.10 for queue length based control law gives $w_e = b \cdot \tau + \hat{\beta}$ and $q_e = \hat{\beta}$.

Delay-based control law: Substituting $e = \tau$ and $f(t) = \frac{q(t-\theta(t)+t^f)}{b} + \tau$, we express the simplified delay-based control law as,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t) \cdot \tau}{\frac{q(t-\theta(t)+t^f)}{b} + \tau} + \beta \right) + (1-\gamma) \cdot w_i(t) \tag{2.12}$$

where the window update is an MIMD based on RTT. Eq. 2.12 captures control laws based on RTT; for example FAST [144].

Similar to queue-length based CC, a system defined by delay-based control law (Eq. 2.12 and the queue length dynamics (Eq. 2.17, there exists a unique equilibrium point. It can be observed that Eq. 2.10 for delay-based control law gives $w_e = b \cdot \tau + \hat{\beta}$ and $q_e = \hat{\beta}$.

RTT-gradient based control law: Substituting $e = 1$ and $f(t) = \frac{\dot{q}(t-\theta(t)+t^f)}{b} + 1$, we express the simplified RTT-gradient based control law as,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t) \cdot 1}{\frac{\dot{q}(t-\theta(t)+t^f)}{b} + 1} + \beta \right) + (1-\gamma) \cdot w_i(t) \tag{2.13}$$

where the window update is an MIMD based on RTT-gradient. Eq. 2.13 by rearranging the terms, captures control laws based on RTT-gradient such as TIMELY [193].

In contrast to queue-length and delay-based CC, RTT-gradient based CC has no unique equilibrium point since $f(t) = \frac{\dot{q}(t-\theta(t)+t^f)}{b} + 1$ stabilizes when $\dot{q} = 0$. However only $\dot{q} = 0$ leads to window dynamics Eq. 2.13 also to stabilize ($\dot{w} = 0$) at any queue lengths. As a result under RTT-gradient control law, Eq. 2.9 and Eq. 2.10 do not have a unique solution and consequently we can state that RTT-gradient based CC has no unique equilibrium point.

2.1.3 Drawbacks of Existing Control Laws

We now analyze the drawbacks of existing congestion control algorithms based on the simplified model. We first show that existing control laws do not satisfy the equilibrium state properties in Eq. 2.1 and then show the orthogonal behavior in their response to queue length and queue buildup rate.

Equilibrium point: It is well-known from literature that loss/ECN-based schemes operate by maintaining a standing queue [36,129,138]. For example, TCP NewReno flows fill the queue to maximum (say q_{max}) and then react by reducing windows by half. Consequently, the bottleneck queue-length oscillates between q_{max} and $q_{max} - b \cdot \tau$ or zero if $q_{max} < b \cdot \tau$. DCTCP flows oscillate around the marking threshold $K > \frac{b \cdot \tau}{7}$ which depends on BDP [35]. This does not satisfy our stringent requirement in Eq. 2.1. While ECN-based schemes reduce the amount of standing queue required, we still consider the standing queue which is proportional to bandwidth to be unacceptable given the increasing gap between bandwidth vs switch buffers.

It can be shown that there exists a unique equilibrium point for queue length and delay approaches (voltage-based CC) defined by Eq. 2.2. However, current-based CC and, in particular, RTT-gradient approaches do not have a unique equilibrium point suggesting a lack of control over queue lengths. Intuitively, RTT-gradient approaches quickly adapt the sending rate to stabilize the RTT-gradient ($\dot{\theta} = \frac{\dot{q}}{b}$) which in turn only stabilizes the queue length gradient $\dot{q}(t)$ but fails to control the absolute value of the queue length. It has indeed been shown that TIMELY, a current-based CC does not have a unique equilibrium [288].

Figure 2.3 visualizes the system behavior according to the window dynamics in Eq. 2.3 and the queue dynamics in Eq. 2.4. In Figure 2.3a we can see that voltage-based CC eventually reaches a unique equilibrium point. In contrast, in Figure 2.3b we see that current-based CC reaches different final points for different initial points, indicating that there exists no unique equilibrium point thereby violating the desired equilibrium state properties (Eq. 2.1). To give more context on this observation, in Figure 2.2 we show the reactions of different schemes for observed queue lengths and queue buildup rate. In Figure 2.2b, we can see that current-based CC has the same reaction for different queue lengths but exhibits a proportional reaction to queue buildup rate (Figure 2.2a); consequently, current-based CC cannot stabilize at a unique equilibrium point.

■ ***Takeaway.*** While voltage-based CC can in principle meet the desired equilibrium state requirements in Eq. 2.1, current-based CC cannot.

Response to perturbation: We observe an orthogonal behavior in the responses of voltage-based CC and current-based CC. In Figure 2.2b we show that voltage-based CC has a proportional reaction to increased queue lengths but a current-based CC approach has the same response for any queue length. Further in Figure 2.2a we observe that current-based CC has a proportional reaction to the rate at which queue is building up but a voltage-based CC has the same reaction for any rate of queue build up. This orthogonality in existing schemes often results in scenarios with either insufficient reaction or overreaction. To underline our observation, we use the system of differential equations (Eq. 2.3 and Eq. 2.4) to observe the trajectories taken by different control laws after a perturbation. We show the trajectories in Figure 2.3. Specifically, Figure 2.3a shows that voltage-based CC (queue length or delay based) eventually reaches a unique equilibrium point but overreacts in the response and losing throughput (window $<$ BDP and $q(t) = 0$) almost for every initial point. In Figure 2.3b we observe that current-based CC (RTT-gradient) reaches different end points for different initial states and consequently does not have a single equilibrium point. However, we see that the initial response is faster with current-based CC due to their use of RTT-gradient which is arguably a superior signal to detect congestion onset even at low queue lengths.

■ ***Takeaway.*** Current-based CC is superior in terms of fast reaction but lacks equilibrium state properties while voltage-based CC eventually reaches a unique equilibrium but overreacts in its response for almost any initial state resulting in long trajectories from initial state to equilibrium state.

2.1.4 Lessons Learned and Design Goals

From our analysis we derive two key observations. First, both voltage and current-based CC have individual benefits. Particularly, voltage-based CC is desirable for the stringent equilibrium properties we require and current-based CC is desirable for fast reaction. Second, both voltage and current-based CC have drawbacks. On one hand, voltage-based CC is oblivious to congestion onset at low queue lengths and on the other hand current-based CC is oblivious to the absolute value of queue lengths. Moreover, voltage-based CC overreacts when the queue drains essentially losing throughput immediately after.

Based on these observations, our goal is to design a control law that systematically combines both voltage and current for every window update action. Specifically our aim is to design a congestion control algorithm with *(i)* equilibrium properties from Eq. 2.1 exhibited by voltage-based CC and *(ii)* fast response to perturbation exhibited by current-based CC. The challenges are to avoid inheriting the drawbacks of both types of CC, stability and fairness. However in order to design such a control law we face the following challenges:

- Finding an accurate relationship between window, voltage and current.
  ▷ Property 1
- Ensuring stability, convergence and fairness.
  ▷ Theorem 1, 2, 3

2.2 Power-Based Congestion Control

Reflecting on our observations in §2.1, we seek to design a congestion control algorithm that systematically reacts to both the absolute value of the bottleneck queue length and its rate of change. Our aim is to address today's datacenter performance requirements in terms of high throughput, low latency, and fast reaction to bursts and bandwidth fluctuations.

2.2.1 The Notion of Power

To address the challenges faced by prior datacenter congestion control algorithms and to optimize along both dimensions, we introduce the notion of *power* associated with the network pipe. Following the bottleneck link model from literature [129,192], from Eq. 2.4 we observe that the window size is indeed related to the product of network voltage and network current which we call *power* (Table 2.1). This corresponds to the product of *(i)* total sending rate λ (current) and *(ii)* the sum of BDP plus the accumulated bytes q at the bottleneck link (voltage), formally expressed in Eq. 2.14.

$$\underbrace{\Gamma(t)}_{\text{power}} = \underbrace{(q(t) + b \cdot \tau)}_{\text{voltage}} \cdot \underbrace{\lambda(t - t^f)}_{\text{current}} \tag{2.14}$$

Notice that the unit of power is $\frac{bit^2}{second}$. We will show the useful properties of power specifically under congestion. Using Eq. 2.4, we can rewrite Eq. 2.14 in terms of queue length gradient $\dot{q}$ and the transmission rate μ as,

$$\Gamma(t) = (q(t) + b \cdot \tau) \cdot (\dot{q}(t) + \mu(t)) \tag{2.15}$$

We now derive a useful property of power using Eq. 2.15 and Eq. 2.4 showing an accurate relationship of power and window.

Property 1 (Relationship of Power and Congestion Window)**.** *Power is the bandwidth-window product*

$$\Gamma(t) = b \cdot w(t - t^f)$$

Note that the property is over the aggregate window size i.e., the sum of window sizes of all flows sharing the common bottleneck. We emphasize that our notion of power is intended for the networking context and cannot be applied to other domains of science. In the following, we outline the benefits of considering the notion of power and how Property 1 can be useful in the context of congestion control.

2.2.2 Benefits of Power-Based CC

A power-based control law can exploit Property 1 to precisely update per flow window sizes. Accurately controlling aggregate window size is a key challenge for an end-host congestion control algorithm. A power-based CC overcomes this challenge by gaining precise knowledge about the aggregate window size from measured power. First, using power enables the window update action to account for the bottleneck queue lengths as well as the queue build-up rate. As a result, a power-based CC can rapidly detect congestion onset even at very low queue lengths. At the same time, a power-based CC also reacts to the absolute value of queue lengths, effectively dampening perturbations. Second, calculating power at the end-host requires no extra measurement and feedback mechanisms compared to INT based schemes such as HPCC [173].

2.2.3 The PowerTCP Algorithm

Driven by our observations, we carefully designed our control law based on power, capturing a systematic reaction to voltage (related to bottleneck queue length), as well as to current (related to variations in the bottleneck queue length).

Control law: PowerTCP is a window-based congestion control algorithm and updates its window size upon receipt of an acknowledgment. For a flow i, every window update is based on *(i)* current window size $w_i(t)$, *(ii)* additive increase β, *(iii)* window size at the time of transmission of the acknowledged segment $w_i(t-\theta(t))$, and *(iv)* power measured from the feedback information. We refer the reader to Table 2.2 for the general notations being used. Formally, PowerTCP's control law can be expressed as

$$w_i(t) \leftarrow \gamma \cdot \left(w_i(t - \theta(t)) \cdot \frac{e}{f(t)} + \beta \right) + (1 - \gamma) \cdot w_i(t) \tag{2.16}$$

$$e = b^2 \cdot \tau; \qquad f(t) = \Gamma(t - \theta(t) + t^f)$$

where $\gamma \in (0, 1]$ and β are parameters to the control law. The base round trip time τ must be configured at compile time. If baseRTT is not precisely known, an alternative is to keep track of minimum observed RTT. We first describe

how power Γ is computed and then present the pseudocode of PowerTCP in Algorithm 1.

Feedback: PowerTCP's control law is based on power. Note that power (Eq. 2.14) is only related to variables at the bottleneck link. In order to measure power, we leverage in-band network telemetry. Specifically, the workings of INT and the header fields required are the same as in HPCC (Figure. 4 in [173]). When a TCP sender sends out a packet P into the network, it additionally inserts an INT header INT into the packet. Each switch along the path then pushes metadata containing the egress queue length ($qlen$), timestamp (ts), so far transmitted bytes ($txBytes$), and bandwidth (b). All values correspond to the time when the packet is scheduled for transmission. At the receiver, the received packet P INT 1,2,...r is read and the INT information is copied to the acknowledgment ACK packet A INT 1,2,...r. The sender then receives an ACK with an INT header and metadata inserted by all the switches along the path from sender to receiver and back to sender A INT 1,2,...r INT ..., n. Here, the INT header and meta-data pushed by switches along the path serve as feedback and as an input to the CC algorithm.

Accounting for the old window sizes: PowerTCP's control law (Eq. 2.16) uses the past window size in addition to the current window size to compute the new window size. PowerTCP accounts for old window size by remembering current window size once per RTT.

Algorithm: Putting it all together, we now present the workflow of PowerTCP in Algorithm 1. Upon the receipt of a new acknowledgment (line 2), PowerTCP: *(i)* retrieves the old *cwnd* (line 3), *(ii)* computes the normalized power (line 19) i.e., $\frac{f(t)}{e}$ in Eq. 2.16, *(iii)* updates *cwnd* (line 5), *(iv)* sets the pacing rate (line 6), and *(v)* remembers the INT header metadata and updates the old *cwnd* once per RTT based on the *ack* sequence number (line 7).

Specifically, power is calculated in the function call to **normPower**. First, the gradient of queue lengths is obtained from the difference in queue lengths and difference in timestamps corresponding to an egress port (line 12). Then the transmission rate of the egress port is calculated from the difference in $txBytes$ and timestamps (line 13). Current is calculated by adding the queue gradient and transmission rate (line 14). Then, the sum of BDP and the queue length gives voltage (line 16). Finally, power is calculated by multiplying current and voltage (line 17). We calculate the base power (line 18) and obtain the normalized power (line 19). The normalized power is calculated for each egress port along the path and the maximum value is smoothed and used as an input to the control law.

Finally, the congestion window is updated in the function call to **updateWindow** (line 26) where γ is the exponential moving average parameter and β is the additive increase parameter, both being parameters to the control law (Eq. 2.16)

Parameters: PowerTCP has only two parameters, that is the EWMA parameter γ and the additive increase parameter β. γ dictates the balance in reaction time and sensitivity to noise. We recommend $\gamma = 0.9$ based on our parameter sweep over wide range of scenarios including traffic patterns that induce rapid fluctuations in the bottleneck queue lengths. Reflecting the intuition for additive increase in prior work [173], we set $\beta = \frac{HostBw \times \tau}{N}$ where N is the expected number of flows sharing host NIC, $HostBw$ is the NIC bandwidth at the host and τ is the base-RTT. This is to avoid queuing at the local interface or, in other words, to avoid making the host NIC a bottleneck, assuming a maximum of N flows share

Algorithm 1: PowerTCP

/* ack contains an INT header with sequence of per-hop egress port meta-data accessed as ack.H[i] */
Input : *ack* and *prevInt*
Output: *cwnd*, *rate*
procedure `newAck`(*ack*):
  $cwnd_{old}$ = getCwnd(*ack.seq*)
  $normPower$ = normPower(*ack*)
  updateWindow($normPower$, $cwnd_{old}$)
  $rate = \frac{cwnd}{\tau}$
  $prevInt = ack.H$; updateOld($cwnd, ack.seq$)
function `normPower`(*ack*):
  $\Gamma_{norm} = 0$
  for *each egress port i on the path* **do**
    $dt = ack.H[i].ts - prevInt[i].ts$
    $\dot{q} = \frac{ack.H[i].qlen - prevInt[i].qlen}{dt}$ ▷ $\frac{dq}{dt}$
    $\mu = \frac{ack.H[i].txBytes - prevInt[i].txBytes}{dt}$ ▷ txRate
    $\lambda = \dot{q} + \mu$ ▷ λ : Current
    $BDP = ack.H[i].b \times \tau$
    $\nu = ack.H[i].qlen + BDP$ ▷ ν : Voltage
    $\Gamma' = \lambda \times \nu$ ▷ Γ' : Power
    $e = (ack.H[i].b)^2 \times \tau$
    $\Gamma'_{norm} = \frac{\Gamma'}{e}$ ▷ Γ'_{norm} :Normalized power
    if $\Gamma' > \Gamma_{norm}$ **then**
      $\Gamma_{norm} = \Gamma'$; $\Delta t = dt$
    end if
  end for
  $\Gamma_{smooth} = \frac{\Gamma_{smooth} \cdot (\tau - \Delta t) + \Gamma_{norm} \cdot \Delta t}{\tau}$ ▷ Smoothing
  return Γ_{smooth}
function `updateWindow`(*power*, *ack*):
  $cwnd = \gamma \times (\frac{cwnd_{old}}{normPower} + \beta) + (1 - \gamma) \times cwnd$
    ▷ γ : EWMA parameter
    ▷ β: Additive Increase
  return *cwnd*

the host NIC bandwidth. Finally, all flows transmit at line rate in the first RTT and use $cwnd_{init} = HostBw \times \tau$. By transmitting at line rate, a new flow is able to discover the bottleneck link state and reduce its *cwnd* accordingly without getting throttled due to the presence of existing flows.

2.2.4 Properties of PowerTCP

PowerTCP comes with strong theoretical guarantees. We show that PowerTCP's control law achieves asymptotic stability with a unique equilibrium point that satisfies our desired equilibrium state properties (Eq. 2.1). PowerTCP also guarantees rapid convergence to equilibrium and achieves proportional fairness.

Our analysis is based on a a single bottleneck link model widely used in the literature [129, 192, 277, 288]. Specifically, we assume that all senders use the same protocol, transmit long flows sharing a common bottleneck link with bandwidth b, and have a base round trip time τ (excluding queuing delays). We denote at time t queue length as $q(t)$, aggregate window size as $w(t)$, window size of a sender i as $w_i(t)$, forward propagation delay between sender and bottleneck queue as t^f, the round-trip time as $\theta(t)$ and a base round-trip time as τ. Here $w(t) = \sum_i w_i(t)$. We additionally use the traditional model of queue length dynamics which is independent of the control law [129, 192]

$$\dot{q}(t) = \frac{w(t - t^f)}{\theta(t)} - b \tag{2.17}$$

where $\theta(t)$ is given by,

$$\theta(t) = \frac{q(t)}{b} + \tau \tag{2.18}$$

Power at time t denoted by $\Gamma(t)$ as defined in §2.2.1 is expressed as,

$$\Gamma(t) = \underbrace{(q(t) + b \cdot \tau)}_{voltage} \cdot \underbrace{(\dot{q}(t) + \mu(t))}_{current} \tag{2.19}$$

PowerTCP's control law at a source i is given by,

$$w_i(t + \delta t) = \gamma \cdot \left(\frac{w_i(t - \theta(t)) \cdot e}{f(t)} + \beta \right) + (1 - \gamma) \cdot w_i(t) \tag{2.20}$$

where e and $f(t)$ are given by,

$$e = b^2 \cdot \tau$$

$$f(t) = \Gamma(t - \theta(t) + t^f)$$

and β is the additive increase term and $\gamma \in (0, 1]$ serves as the weight given for new updates using EWMA. Both β and γ are parameters to the control law.

Using the properties of power (Property 1), the aggregate window size at time $t - \theta(t)$ can be expressed in terms of power as,

$$w(t - \theta(t)) = \frac{\Gamma(t - \theta(t) + t^f)}{b} = \frac{f(t)}{b} \tag{2.21}$$

Suppose an *ack* arrives at time t acknowledging a segment, time $t - \theta(t)$ corresponds to the time when the acknowledged segment was transmitted.

Theorem 1 (Stability)**.** *PowerTCP's control law is Lyapunov-stable as well as asymptotically stable with a unique equilibrium point.*

Proof. First, we rewrite Eq. 2.20 as follows to obtain the aggregate window w,

$$\sum_i w_i(t + \delta t) = \sum_i \gamma \cdot \left(\frac{w_i(t - \theta(t)) \cdot e}{f(t)} + \beta \right) + \sum_i (1 - \gamma) \cdot w_i(t)$$

let $\hat{\beta} = \sum_i \beta$

$$w(t + \delta t) = \gamma \cdot \left(\frac{w(t - \theta(t)) \cdot e}{f(t)} + \hat{\beta} \right) + (1 - \gamma) \cdot w(t)$$

by rearranging the terms in the above equation we obtain,

$$w(t+\delta t) - w(t) = \gamma \cdot \left(-w(t) + \frac{w(t-\theta(t)) \cdot e}{f(t)} + \hat{\beta} \right)$$

dividing by δt on both sides in the above equation and using Euler's first-order approximation, we derive the window dynamics for PowerTCP as follows,

$$\dot{w}(t) = \gamma_r \cdot \left(-w(t) + \frac{w(t-\theta(t)) \cdot e}{f(t)} + \hat{\beta} \right) \tag{2.22}$$

where $\gamma_r = \frac{\gamma}{\delta t}$. Using Eq. 2.21 and substituting $e = b^2 \cdot \tau$, Eq. 2.22 reduces to,

$$\dot{w}(t) = \gamma_r \cdot \left(-w(t) + b \cdot \tau + \hat{\beta} \right) \tag{2.23}$$

In the system defined by Eq. 2.17 and Eq. 2.22, when the window and the queue length stabilize i.e., $\dot{w}(t) = 0$ and $\dot{q}(t) = 0$, it is easy to observe that there exists a unique equilibrium point $(w_e, q_e) = (b \cdot \tau + \hat{\beta}, \hat{\beta})$. We now apply a change of variable from t to $t - t^f$ in Eq. 2.23 and linearize Eq. 2.23 and Eq. 2.17 around (w_e, q_e),

$$\delta\dot{w}(t - t^f) = -\gamma_r \cdot \delta w(t - t^f) \tag{2.24}$$

$$\delta\dot{q}(t) = -\frac{\delta q(t)}{\tau} + \frac{\delta w(t - t^f)}{\tau} \tag{2.25}$$

We now convert the above differential equations to matrix form,

$$\begin{bmatrix} \delta\dot{q}(t) \\ \delta\dot{w}(t) \end{bmatrix} = \begin{bmatrix} -\frac{1}{\tau} & \frac{1}{\tau} \\ 0 & -\gamma_r \end{bmatrix} \times \begin{bmatrix} \delta q(t) \\ \delta w(t) \end{bmatrix}$$

It is then easy to observe that the eigenvalues of the system are $-\frac{1}{\tau}$ and $-\gamma_r$. Since τ (base RTT) and $\gamma_r = \frac{\gamma}{\delta t}$ are both positive, we see that both the eigenvalues are negative. This proves that the system is both lyapunav stable and asymptotically stable. □

Theorem 2 (Convergence)**.** *After a perturbation, PowerTCP's control law exponentially converges to equilibrium with a time constant* $\frac{\delta t}{\gamma}$ *where* δt *is the window update interval.*

Proof. A perturbation at time $t = 0$ causes the window to shift from $w_e = c \cdot \tau + \hat{\beta}$ to say w_{init}. We solve the differential equation in Eq. 2.23 and obtain the following equation,

$$w(t) = w_e + \underbrace{(w_{init} - w_e) \cdot e^{-\gamma_r \cdot t}}_{exponential\ \ decay} \tag{2.26}$$

From Eq. 2.26 we can see that, for any error $e = w_e - w_{init}$ caused by a perturbation, e exponentially decays with a time constant $\frac{1}{\gamma_r} = \frac{\delta t}{\gamma}$. Hence for e to decay 99.3%, it takes $\frac{5 \cdot \delta}{\gamma}$ time. □

Theorem 3 (Fairness)**.** *PowerTCP is β_i weighted proportionally fair, where β_i is the additive increase used by a flow i.*

Proof. Recall that PowerTCP's control law for each flow i is defined as,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t-\theta(t)) \cdot e}{f(t)} + \beta_i \right) + (1-\gamma) \cdot w_i(t)$$

From the proof of Theorem 1, we know that the equilibrium point for aggregate window size and queue length is $(w_e, q_e) = (b \cdot \tau + \hat{\beta}, \hat{\beta})$. Using this equilibrium we can also obtain the equilibrium value for $f(t)$ as,

$$f_e = (\hat{\beta} + b \cdot \tau) \cdot b$$

We can then show that w_i has an equilibrium point.

$$(w_i)_e = \frac{\hat{\beta} + b \cdot \tau}{\hat{\beta}} \cdot \beta_i$$

We use the argument that window sizes and rates are synonymous especially that PowerTCP uses pacing with rate $r_i = \frac{w_i}{\tau}$. We can then easily observe that the rate allocation is approximately max-min fair if β_i are small enough but β_i proportionally fair in general. □

Theorem 1 and Theorem 2 state the key properties of PowerTCP. First, the convergence with time constant of $\frac{\delta t}{\gamma}$ shows the fast reaction to perturbations. Second, the system being asymptotically stable at low queue lengths satisfies our stringent equilibrium property discussed in §2.1. Indeed, **power** and Property 1 play a key role in the proof of Theorem 1 and Theorem 2 revealing its importance in congestion control. In Figure 2.3c, we see the trajectories of PowerTCP from different initial states to a unique equilibrium without violating throughput and latency requirements, showing the accurate control enabled by power-based congestion control.

2.2.5 θ-PowerTCP: Standalone Version

PowerTCP's control law requires in-network queue length information which can be obtained by using techniques such as INT. In order to widen its applicability, PowerTCP can still be deployed in datacenters with legacy, non-programmable switches through accurate RTT measurement capabilities at the end-host. In this case, we rearrange term $\frac{e}{f}$ in Eq. 2.16 as follows,

$$\frac{e}{f} = \frac{b^2 \cdot \tau}{\Gamma} = \frac{b^2 \cdot \tau}{(\dot{q}+b) \cdot (q + b \cdot \tau)} = \frac{\tau}{(\frac{\dot{q}}{b}+1) \cdot (\frac{q}{b} + \tau)}$$

finally, using the fact that $\frac{q}{b} + \tau = \theta$ (RTT) and $\frac{\dot{q}}{b} = \dot{\theta}$ (RTT-gradient), we reduce $\frac{e}{f}$ to,

$$\frac{e}{f} = \frac{\tau}{(\dot{\theta}+1) \cdot (\theta)} \tag{2.27}$$

where $\dot{\theta}$ is the RTT-gradient and θ is RTT. Using Eq. 2.27 in Eq. 2.16 allows for deployment even when INT is not supported by switches in the datacenter. Algorithm 2 presents the pseudocode of θ-PowerTCP. This algorithm demonstrates

Algorithm 2: θ-PowerTCP (w/o switch support)

```
/* t_c is the timestamp upon ack arrival                                  */
Input  : ack
Output : cwnd, rate
procedure newAck(ack):
    cwnd_old = getCwnd(ack.seq)
    normPower = normPower(ack)
    updateWindow(normPower, cwnd_old)
    rate = cwnd / τ
    prevRTT = RTT
    t_c^prev = t_c
    updateOld(cwnd, ack.seq)
function normPower(ack):
    dt = t_c − t_c^prev
    θ̇ = (RTT − prevRTT) / dt                                  ▷ dRTT/dt
    Γ_norm = ((θ̇ + 1) × RTT) / τ                  ▷ Γ_norm :Normalized power
    Γ_smooth = (Γ_smooth · (τ − Δt) + Γ_norm · Δt) / τ
    return Γ_smooth
function updateWindow(power, ack):
    if ack.seq < lastUpdated then                                  ▷ per RTT
        return cwnd
    end if
    cwnd = γ × (cwnd_old / normPower + β) + (1 − γ) × cwnd
                                                  ▷ γ : EWMA parameter
                                                  ▷ β: Additive Increase
    lastUpdated = snd_nxt
    return cwnd
```

how PowerTCP's control law can be mimicked by using a delay signal without the need for switch support. However, as we will show later in our evaluation, there are drawbacks in using RTT instead of queue lengths. First, notice how queue lengths are changed to RTT, where we assume bottleneck $txRate$ (μ) as bandwidth (b). The implication is that, when using $txRate$ which is essentially obtained from INT, the control law knows the exact transmission rate and rapidly fills the available bandwidth. But, when using RTT, the control law assumes the bottleneck is at maximum transmission rate and does not react by multiplicative increase and rather relies on slow additive increase to fill the available bandwidth. Secondly, in multi-bottleneck scenarios, the control law precisely reacts to the most bottlenecked link when using INT but reacts to the sum of queuing delays when using RTT. Nevertheless, under congestion, both PowerTCP and θ-PowerTCP have the same properties in a single-bottleneck scenario.

2.2.6 Deploying PowerTCP

Modern programmable switches are able to export user-defined header fields and device metrics [90, 156]. These metrics can be embedded into data packets, a mechanism commonly referred to as in-band network telemetry (INT). PowerTCP

leverages INT to obtain fine-grained, per-packet feedback about queue occupancies, traffic counters, and link configurations within the network. For deployment with legacy networking equipment, we have proposed θ-PowerTCP which only requires accurate timestamps to measure the RTT.

We imagine PowerTCP and θ-PowerTCP to be deployed on low-latency kernel-bypass stacks such as SNAP [47] or using NIC offload. Yet, in this work, instead of implementing our algorithms for these platforms, we show how PowerTCP and θ-PowerTCP can readily be deployed by merely changing the control logic of existing congestion control algorithms. In particular, we compare our work to HPCC [173] which is based on INT feedback and SWIFT [164] which is based on delay feedback.

PowerTCP requires the same switch support and header format as HPCC, as well as packet pacing support from the NIC. Additionally, it does not maintain additional state compared to HPCC but requires one extra parameter γ, the moving average parameter for window updates. Similar to SWIFT and TIMELY, θ-PowerTCP requires accurate packet timestamps from the NIC but it does not require any switch support. The simpler logic of θ-PowerTCP (compared to PowerTCP) only reacts once per RTT and reduces the number of congestion control function calls.

The core contribution of this work is the design of a novel control law and we do not explore implementation challenges further at this point since PowerTCP does not add additional complexity compared to existing algorithms. Still, to confirm the practical feasibility of our approach, we implemented PowerTCP as a Linux kernel congestion control module. We also implemented the INT component as a proof of concept for the Intel Tofino switch ASIC [90]. The switch implementation is written in P4 and uses a direct counter associated with the egress port to maintain the so far transmitted bytes and appends this metric together with the current queue occupancy upon dequeue from the traffic manager to each segment. We leverage a custom TCP option type to encode this data and append 64 bit per-hop headers to a 32 bit base header. The implementation uses less than one out of 12 stages of the Tofino's ingress pipeline (where the headers are prepared and appended) and less than one out of 12 stages in the egress pipeline (where the measurements are taken and inserted). The processing logic runs at line rate of 3.2 Tbit per second.

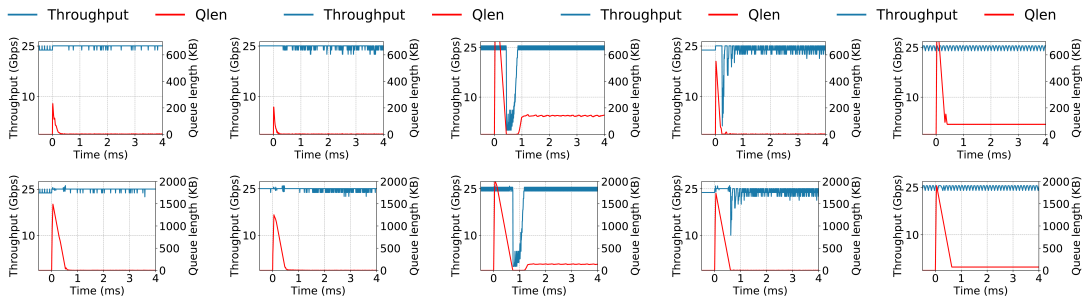
Later, we have also tested our algorithm [127] on a real network setup using Tofino switches (with TCP-INT [141]) and Intel NICs.

2.3 Evaluation

We evaluate the performance of PowerTCP and θ-PowerTCP and compare against existing CC algorithms. Our evaluation aims at answering four main questions.

(Q1) How well does PowerTCP react to congestion?
We find that PowerTCP outperforms the state-of-the-art congestion control algorithms, reducing tail buffer occupancy and consequently tail latency under congestion by 30% when compared to HPCC and at least by 60% compared to TIMELY and DCQCN.

(Q2) Does PowerTCP introduce a tradeoff between throughput and latency?



(a) PowerTCP (b) θ-PowerTCP (c) TIMELY (d) HPCC (e) HOMA

Figure 2.4: State-of-the-art congestion control algorithms vs PowerTCP in response to an incast. For each algorithm, we show the corresponding reaction to $10:1$ incast in the top row and to $255:1$ incast in the bottom row.

Our evaluation shows that PowerTCP does not trade throughput for latency and that PowerTCP rapidly converges to near-zero queue lengths without losing throughput.

(Q3) How much can we benefit under realistic workloads?
We show that PowerTCP improves 99th-percentile flow completion times for short flows ($< 10KB$) by 33% compared to HPCC, by 99% compared to HOMA and by 74% compared to TIMELY and DCQCN even at moderate network loads. At the same time, we find that PowerTCP does not penalize long flows ($> 1MB$). In fact, we find that θ-PowerTCP performs equally well for short flows compared to PowerTCP but performs similarly to TIMELY for medium and long flows.

(Q4) How does PowerTCP perform under high load and bursty traffic patterns?
Our evaluation shows that the benefits of PowerTCP are further enhanced under high loads and that PowerTCP remains stable even under bursty traffic.

2.3.1 Setup

Our evaluation is based on network simulator NS3 [209].

Topology: We consider a datacenter network based on a FatTree topology [31] with 2 core switches and 256 servers organized into four pods. Each pod consists of two ToR switches and two aggregation switches. The capacity of all the switch-to-switch links are 100Gbps and server-to-switch links are all 25Gbps leading to $4:1$ oversubscription similar to prior work [232]. The links connecting to core switches have a propagation delay of $5\mu s$ and all the remaining links have a propagation delay of $1\mu s$. We set up a shared memory architecture on all the switches and enable the Dynamic Thresholds algorithm [86] for buffer management across all the ports, commonly enabled in datacenter switches [15, 75]. Finally we set the buffer sizes in our topology proportional to the bandwidth-buffer ratio of Intel Tofino switches [90].

Traffic mix: We generate traffic using the web search [35] flow size distribution to evaluate our algorithm using realistic workloads. We evaluate an average load (on the ToR uplinks) in the range of $20\% - 95\%$. We also use a synthetic workload similar to prior work [33] to generate incast traffic. Specifically, the synthetic workload represents a distributed file system where each server requests a file from a set of servers chosen uniformly at random from a different rack. All the

servers which receive the request respond at the same time by transmitting the requested part of the file. As a result, each file request creates an incast scenario. We evaluate across different request rates and request sizes.

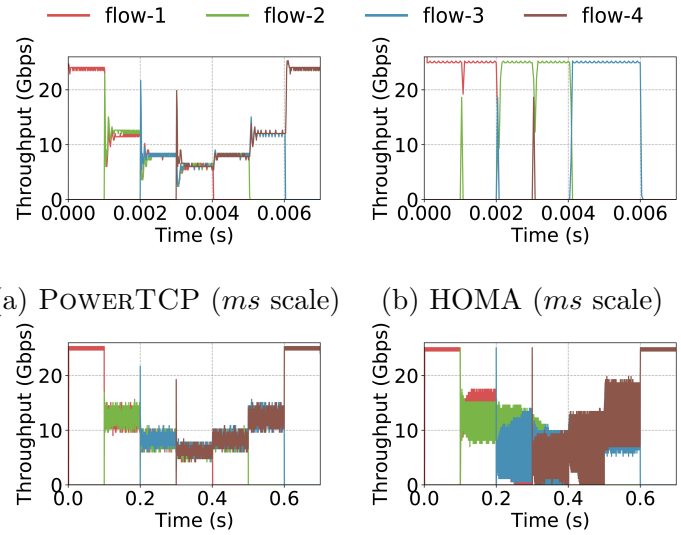
Comparisons and metrics: We evaluate PowerTCP with and without switch support and compare to HPCC [173], DCQCN [287], and TIMELY [193] representing sender-based control law approaches similar to PowerTCP and HOMA [199] representing receiver-driver transport. We report flow completion times and switch buffer occupancy metrics.

Configuration: We set $\gamma = 0.9$ for PowerTCP and θ-PowerTCP. Both HPCC and PowerTCP are configured with base-RTT (τ) set to the maximum RTT in our topology and $HostBw$ is set to the server NIC bandwidth. The product of base-RTT and $HostBw$ is configured as RTTBytes for HOMA and the over-commitment level is set to 1 where HOMA performed best across different overcommitment levels in our setup. We set the parameters for DCQCN following the suggestion in [173] which is based on experience and TIMELY parameters are set according to [193].

2.3.2 Results

PowerTCP reacts rapidly yet accurately to congestion: We evaluate PowerTCP's reaction to congestion in two scenarios: *(i)* $10:1$ small-scale incast and *(ii)* $255:1$ large-scale incast. Figure 2.4 shows the aggregate throughput and the buffer occupancy at the bottleneck link for PowerTCP, TIMELY, HPCC and HOMA. First, at time $t = 0$, we launch ten flows simultaneously towards the receiver of a long flow leading to a **10:1** incast. We show in Figure 2.4a and Figure 2.4b that PowerTCP quickly mitigates the incast and reaches near zero queue lengths without losing throughput. In Figure 2.4d we see that HPCC indeed reacts quickly to get back to near-zero queue lengths. On one hand, however, HPCC does not react enough during the congestion onset and reaches higher buffer occupancy $\approx$ 2x compared to PowerTCP and on the other hand loses throughput after mitigating the incast as opposed to PowerTCP's stable throughput. TIMELY as shown in Figure 2.4c does not control the queue-lengths either and loses throughput after reacting to the incast. While HOMA sustains throughput, we observe from Figure 2.4e that HOMA does not accurately control bottleneck queue-lengths. Second, at time $t = 0$, in addition to the $10:1$ incast, the 256^{th} server sends a query request (§2.3.1) to all the other 255 servers which then respond at the same time, creating a **255:1** incast. From Figure 2.4a and Figure 2.4b (bottom row), we observe similar benefits from both PowerTCP and θ-PowerTCP even at large-scale incast: both react quickly and converge to near-zero queue-lengths without losing throughput. In contrast, from Figure 2.4c and Figure 2.4d we see that TIMELY and HPCC lose throughput immediately after reacting to the increased queue length. From Figure 2.4e we observe that HOMA reaches approximately 500KB higher queue-length compared to PowerTCP and cannot converge to near-zero queue-lengths quickly.

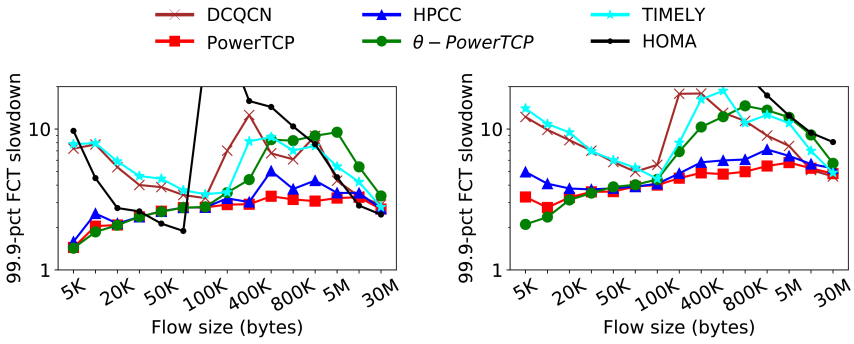
PowerTCP is stable and achieves fairness: PowerTCP not only reacts rapidly to reduce queue lengths but also features excellent stability. Figure 2.5 shows how bandwidth is shared by multiple flows as they arrive and leave. We see that PowerTCP stabilizes to a fair share of bandwidth quickly, both when



(a) PowerTCP (ms scale) (b) HOMA (ms scale)

(c) θ-PowerTCP (d) TIMELY

Figure 2.5: Fairness and stability



(a) 20% load (b) 60% load

Figure 2.6: 99.9 percentile flow completion times with websearch workload *(a)* even at low network load, PowerTCP outperforms existing algorithms and *(b)* as the load increases the benefits of PowerTCP are enhanced. However, only short flows benefit from θ-PowerTCP.

flows arrive and leave, confirming PowerTCP's fast reaction to congestion as well as the available bandwidth.

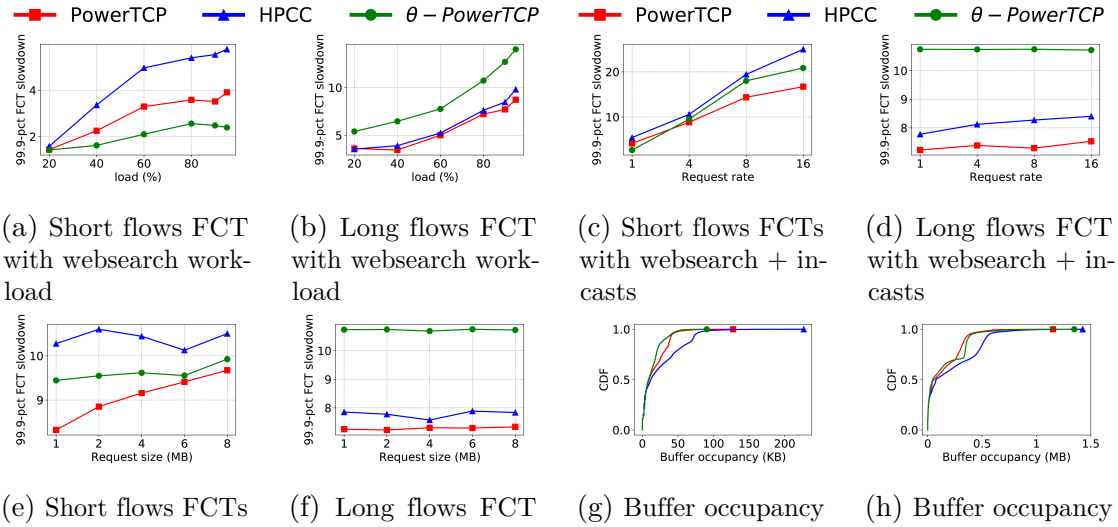
Figure 2.4a showing convergence and Figure 2.5a showing fairness and stability confirm the theoretical guarantees of PowerTCP. Hereafter, all our results are based on the setup described above, §2.3.1, using realistic workloads.

PowerTCP significantly improves short flows FCTs: In Figure 2.6 we show the 99.9-percentile flow completion times using PowerTCP and state-of-the-art datacenter congestion control algorithms. At 20% network load (Figure 2.6a), PowerTCP and θ-PowerTCP improve 99.9-percentile flow completion times for short flows ($< 10KB$) by 9% compared to HPCC and by 80% compared to TIMELY, DCQCN and HOMA. Even at moderate load of 60% (Figure 2.6b), short flows significantly benefit from PowerTCP as well as θ-PowerTCP. Specifically, PowerTCP improves 99.9 percentile flow completion times for short flows by 33% compared to HPCC, by 99% compared to HOMA and by 74% compared to TIMELY and DCQCN. θ-PowerTCP provides even greater benefits to short flows showing an improvement of 36% compared to HPCC and 82% compared to TIMELY and DCQCN. Indeed, web search workload being buffer-intensive, our results confirm the observations made in §2.1. TIMELY being a current-based CC, does not explicitly control queuing latency, while HPCC, a voltage-based CC, does not react as fast as PowerTCP to mitigate congestion resulting in higher flow completion times. Surprisingly, HOMA performs the worst, showing an order-of-magnitude higher FCTs for short flows at high loads as shown in Figure 2.6b.

We also evaluate across various loads in the range $20\% - 95\%$ and show the 99.9-percentile flow completion times for short flows in Figure 2.7a. In particular, we see that the benefits of PowerTCP and θ-PowerTCP are further enhanced as the network load increases. PowerTCP (and θ-PowerTCP) improve the flow completion times of short flows by 36% (and 55%) compared to HPCC. Short flows particularly benefit from PowerTCP due its accurate control of buffer occupancies close to zero. In Figure 2.7g we show the CDF of buffer occupancies at 80% load. PowerTCP consistently maintains lower buffer occupancy and cuts the tail buffer occupancy by 50% compared to HPCC.

Medium sized flows also benefit from PowerTCP: We find that PowerTCP not only improves short flow performance but also improves the 99.9-percentile flow completion times for medium sized flows ($100KB - 1M$). In Figure 2.6 we see that PowerTCP consistently achieves better flow completion times for medium sized flows. Specifically, at 20% network load (Figure 2.6a), PowerTCP improves 99.9-percentile flow completion times for medium flows by 33% compared to HPCC, by 76% compared to HOMA and by 62% (and 50%) compared to TIMELY (and DCQCN). In Figure 2.6b, we observe similar benefits even at 60% load.

We notice from Figure 2.6a and Figure 2.6b that the performance of θ-PowerTCP deteriorates sharply for medium sized flows. θ-PowerTCP uses RTT for window update calculations. While RTT can be a good congestion signal, it does not signal under-utilization as opposed to INT that explicitly notifies the exact utilization. As a result, medium flows with θ-PowerTCP experience 60% worse performance on average compared to PowerTCP and HPCC. We also observe similar performance for TIMELY that uses RTT as a congestion signal. Although delay is simple and effective for short flows performance even at the



(a) Short flows FCT with websearch workload

(b) Long flows FCT with websearch workload

(c) Short flows FCTs with websearch + incasts

(d) Long flows FCT with websearch + incasts

(e) Short flows FCTs with websearch + incasts

(f) Long flows FCT with websearch + incasts

(g) Buffer occupancy with websearch workload at 80% load

(h) Buffer occupancy with websearch + incasts

Figure 2.7: A detailed comparison of PowerTCP, θ-PowerTCP and the state-of-the-art showing the benefits of PowerTCP and the trade-offs of θ-PowerTCP. Particularly PowerTCP outperforms the state-of-the-art across a range of network loads even under bursty traffic. However, θ-PowerTCP performs well for short flows but long flows cannot benefit from θ-PowerTCP.

tail, our results show that delay as a congestion signal is not ideal if not worse for medium sized flows.

PowerTCP does not penalize long flows: Fast reaction to available bandwidth makes PowerTCP ideal for best performance across all flow sizes. We observe from Figure 2.6 that PowerTCP achieves flow completion times comparable to existing algorithms, indicating that PowerTCP does not trade throughput for low latency. Further, in Figure 2.7b we show the 99.9-percentile flow completion times for long flows across various loads. At low load, PowerTCP performs similar to HPCC and performs 9% better compared to HPCC at 90% network loads. However, we see that θ-PowerTCP is consistently 35% worse on average across various loads compared to PowerTCP and HPCC.

PowerTCP outperforms under bursty traffic: We generate incast-like traffic described in §2.3.1 in addition to the web search workload at 80% load. In Figure 2.7c and Figure 2.7d we show the 99.9-percentile flow completion times for short and long flows across different request rates for a request size of $2MB$. Note that by varying request rates, we are essentially varying the frequency of incasts. We observe that even under bursty traffic, PowerTCP improves 99.9-percentile flow completion times on average for short flows by 24% and for long flows by 10% compared to HPCC. Further PowerTCP outperforms at high request rates showing 33% improvement over HPCC for short flows. On the other hand, θ-PowerTCP improves flows completion times for short flows but performs worse across all request rates compared to HPCC.

We further vary the request size at a request rate of four per second. Note that by varying the request size, we also vary the duration of congestion. In Figure 2.7e and Figure 2.7f, we show the 99.9-percentile flow completion times for short and long flows. Specifically, in Figure 2.7e we observe that flow completion times with

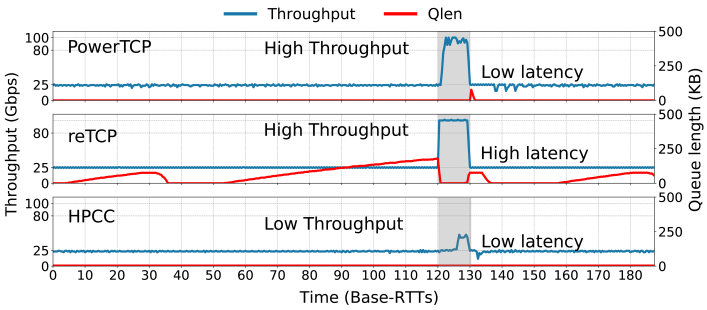
PowerTCP gradually increase with request size. PowerTCP, compared to HPCC, improves flow completion times of short flows by 20% at $1MB$ request size and improves by 7% at $8MB$ request size. At the same time, PowerTCP does not sacrifice long flows performance under bursty traffic. PowerTCP improves flow completion times for long flows by 5% on average compared to HPCC. θ-PowerTCP's performance similar to previous experiments is on average 30% worse for long flows but 9% better for short flows compared to HPCC. We show the CDF of buffer occupancies under bursty traffic with $2MB$ request size and 16 per second request rate. Both PowerTCP and θ-PowerTCP reduce the 99 percentile buffer by 31% compared to HPCC.

We note that HOMA's performance in our evaluation is not in line with the results presented in [199]. Recent work [131] reports similar performance issues with HOMA. We suspect two possible reasons: *(i)* HOMA's accuracy in controlling congestion is specifically limited in our network setup with an oversubscribed Fat-Tree topology where congestion at the ToR uplinks is a possibility which cannot be controlled by a receiver-driven approach such as HOMA. *(ii)* As pointed out by [131], HOMA's original evaluation considered practically infinite buffers at the switches whereas switches in our setup are limited in buffer and use Dynamic Thresholds to share buffer. Further, even at 20% load, asymmetric RTTs in a Fat-Tree topology (consequently RTTBytes) across ToR pairs contributes to HOMA's inaccuracy in controlling congestion. Further, our evaluation of Homa is based on its NS3 simulator that the authors later confirmed to be incomplete and may not capture the intended performance of HOMA. It remains to be seen how the full version of HOMA performs.

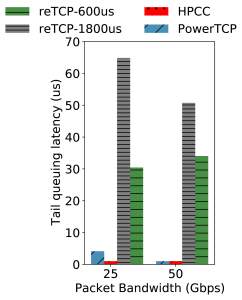
2.4 Case Study: Reconfigurable DCNs

Given PowerTCP's rapid reaction to congestion and available bandwidth, we believe that PowerTCP is well suited for emerging reconfigurable datacenter networks (RDCN) [205]. We now examine PowerTCP's applicability in this context through a case study. Congestion control in RDCNs is especially challenging as the available bandwidth rapidly fluctuates due to changing circuits. In this section, we evaluate the performance of PowerTCP and compare against the state-of-the-art reTCP [201] and HPCC using packet-level simulations in NS3. We implement both PowerTCP and HPCC in the transport layer and limit their window updates to once per RTT for a fair comparison with reTCP. PowerTCP and HPCC flows initialize the TCP header with the unused option number 36. Switches are configured to append INT metadata to TCP options. It should be noted that TCP options are limited to 40 bytes. As a result, our implementation can only support at most four hops round-trip path length.

We evaluate in a topology with 25 ToR switches with 10 servers each and a single optical circuit switch connected to all the ToR switches. ToR switches are also connected to a separate packet switched network with 25Gbps links. The optical switch internally connects each input port to an output port and cycles across 24 matchings in a permutation schedule where the switch stays in a specific matching for $225\mu s$ (one day) and takes $20\mu s$ to reconfigure to the next matching (one night). In this setting, each pair of ToR switches has direct connectivity through the circuit switch once over a length of 24 matchings (one week). We



(a) PowerTCP reacts rapidly to the available bandwidth achieving good circuit utilization.



(b) PowerTCP significantly reduces the tail latency

Figure 2.8: The benefits of PowerTCP in reconfigurable datacenter networks showing its ability to achieve good circuit utilization while significantly reducing the tail latency compared to reTCP.

use single-hop routing in the circuit network and a maximum base RTT is 24μs. Note that circuit-on time (i.e., one day) is approximately 10 RTTs. The links between servers and ToR switches are 25Gbps and circuit links are 100Gbps. We configure the ToR switches to forward packets exclusively on the circuit network when available. Switches are further equipped with per-destination virtual output queues (VOQs). Our setup is in line with prior work [201]. We set reTCP's prebuffering to 1800μs based on the suggestions in [201] and set to 600μs based on our parameter sweep for the minimum required prebuffering in our topology. We compare against both versions.

In Figure 2.8a, we show the time series of throughput and VOQ length for a pair of ToR switches. Specifically, the gray-shaded area in Figure 2.8a highlights the availability of high bandwidth through the circuit-switched network. On one hand, reTCP instantly fills the available bandwidth but incurs high latency due to prebuffering before the circuit is available. On the other hand, HPCC maintains low queue lengths but does not fill the available bandwidth. In contrast, PowerTCP fills the available bandwidth within one RTT and maintains near-zero queue lengths and thereby achieves both high throughput and low latency. We show the tail queuing latency incurred by reTCP, HPCC and PowerTCP in Figure 2.8b. We observe that PowerTCP improves the tail queuing latency at least by 5× compared to reTCP. Our case study reveals that fine-grained congestion control algorithms such as PowerTCP can alleviate the circuit utilization problem in RDCNs without trading latency for throughput.

2.5 Related Work

Dealing with congestion has been an active research topic for decades with a wide spectrum of approaches, including buffer management [46, 86, 87], scheduling [37, 130, 216, 217] and resource allocation [26]. In the following, we will focus on the most closely related works on end-host congestion control.

Approaches such as [35, 259, 287] (e.g., DCTCP, D^{2}TCP) rely on ECN as the congestion signal and react proportionally. Such algorithms require the bottleneck queue to grow up to a certain threshold, which results in queuing delays. ECN-based schemes remain oblivious to congestion onset and intensity. Protocols such as TIMELY [193], SWIFT [164], CDG [125], DX [166] rely on RTT measurements for window update calculations. TIMELY and CDG partly react to congestion based on delay gradients, remaining oblivious to absolute queue lengths. TIMELY, for instance, uses a threshold to fall back to proportional reaction to delay instead of delay gradient. SWIFT, a successor of TIMELY, only reacts proportionally to delay. As a result, SWIFT cannot detect congestion onset and intensity unless the distance from target delay significantly increases. In contrast, θ-PowerTCP also being a delay-based congestion control algorithm updates the window sizes using the notion of power. As a result, θ-PowerTCP accurately detects congestion onset even at near-zero queue lengths.

XCP [151], D^3 [269], *RCP* [95] rely on explicit network feedback based on rate calculations within the network. However, the rate calculations are based on heuristics and require parameter tuning to adjust for different goals such as fairness and utilization. HPCC [173] introduces a novel use of in-band network telemetry and significantly improves the fidelity of feedback. Our work builds on the same INT capabilities to accurately measure the bottleneck link state. However, as we show analytically and empirically, HPCC's control law then adjusts rate and window size solely based on observed queue lengths and lacks control accuracy compared to PowerTCP. Our proposal PowerTCP uses the same feedback signal but uses the notion of power to update window sizes leading to significantly more fine-grained and accurate reactions. NCF [99] introduces explicit NACK feedback from the network switches, leveraging p4 switches. Bolt [50] and SFC [165] use explicit network feedback to control congestion. However, these approaches require changes to the network infrastructure. In contrast, PowerTCP uses INT capabilities that are already available in modern datacenter networks.

Receiver-driven transport protocols such as NDP [124], HOMA [199], and Aeolus [131] have received much attention lately. Such approaches are conceptually different from classic transmission control at the sender. Importantly, receiver-driven transport approaches make assumptions on the uniformity in datacenter topologies and oversubscription [124]. PowerTCP is a sender-based classic CC approach that uses our novel notion of power and achieves fine-grained control over queuing delays without sacrificing throughput.

Orthogonal to network congestion control, recent works have also identified congestion problems within the host interconnect [25, 27, 260]. Similar to In-band Network Telemetry, host interconnects expose several performance counters that can be used to derive congestion signals. In future, we plan to explore control-theoretic approaches for host congestion control, leveraging the insights from PowerTCP's design philosophy.

2.6 Summary

We presented PowerTCP, a novel fine-grained congestion control algorithm. By reacting to both the current state of the network as well as its trend (i.e., power), PowerTCP improves throughput, reduces latency, and keeps queues within the network short. We proved that PowerTCP has a set of desirable properties, such as fast convergence and stability allowing it to significantly improve flow completion times compared to the state-of-the-art. Its fast reaction makes PowerTCP attractive for many dynamic network environments including emerging reconfigurable datacenters which served us as a case study in this work. In our future work, we plan to explore more such use cases.

3

Load Balancing

The rapid increase in the computational needs of emerging deep learning models (e.g., GPT-4 [6]) necessitates that the training process be distributed across a large-scale GPU cluster (e.g., thousands of GPUs) in a datacenter. To the surprise of datacenter operators and researchers, communication has turned out to be a major bottleneck in distributed training [180]. To speed up the training process, significant research efforts have recently been made, including parallelism strategies [143, 206, 207, 244], topology engineering [266, 285], and collective optimizers [10, 180, 237]. Unsurprisingly, congestion control and load-balancing are the main root causes of communication bottlenecks, two of the most widely studied problems in the literature, especially in the context of datacenters.

In the wake of this emerging problem, both industry and academia have almost unanimously agreed that multipath transport is *necessary* to improve the performance of large-scale distributed training workloads [13, 71]. Multipath transport protocols are motivated by their ability to exploit path diversity and effectively balance the load across all network paths [63, 124, 199, 210, 213, 270]. While, in theory, multipath transport protocols can achieve similar objectives to an optimal multi-commodity flow problem, several practical challenges remain. For instance, packet ordering (and the required reorder buffers), loss recovery, and telemetry (for traffic analysis) complicate NIC design and operational costs. This begs the question:

Is multipath transport necessary for improving large-scale distributed training performance, and how close can singlepath transport get to the optimal performance?

To our surprise, our findings indicate a likely *no* i.e., a singlepath transport can perform nearly as well as multipath transport. While multipath transport protocols are ideally superior to singlepath variants, certain properties of training workloads allow us to demonstrate that multipath transport is not necessary to improve performance.

Our assertion is based on four main properties of distributed training workloads (collective communication). Figure 3.1 illustrates our perspective. First, from a NIC perspective, flows originating from a GPU corresponding to a collective operation (e.g., allReduce) arrive nearly simultaneously at the NIC. Additionally, a single GPU does not participate in multiple collectives at the same time. Second, within each step of a single collective (e.g., ring-allReduce), all the flows are of the same size. Third, in contrast to traditional datacenter workloads, individual flow completion time is less critical; rather, the completion time of a collective is more crucial and directly impacts the training duration. Fourth, sending packets of the same flow over a set of k equal-cost paths is equivalent, in terms

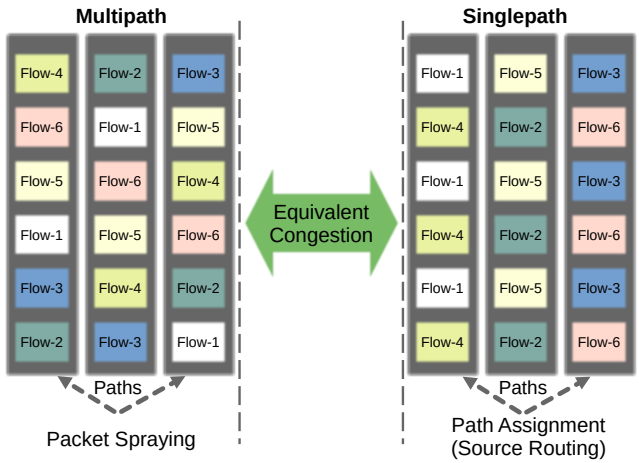


Figure 3.1: In contrast to traditional datacenter workloads, distributed training workloads exhibit certain properties in terms of flow sizes, the number of concurrent flows, and arrival times that allow singlepath transport to achieve nearly the same performance as an optimal multipath transport. The problem essentially boils down to assigning paths to each flow in order to minimize congestion.

of congestion, to sending packets of each flow over a single path when there are multiples of k flows and each flow is of the same size. Although the number of flows within a collective varies depending on the collective algorithm, the communication library provides several knobs to maintain a sufficient number of flows at all times without impacting the bandwidth cost of a collective algorithm. For instance, NCCL allows a user to configure `NCCL_IB_QPS_PER_CONNECTION` and `NCCL_IB_SPLIT_DATA_ON_QPS` which split the data transmitted between a GPU pair into multiple queue pairs uniformly [8], effectively increasing the number of flows if needed without any involvement from the NIC. Given these properties, uniformly spreading load across multiple equal-cost paths in the network (similar to multipath transport) essentially boils down to uniformly assigning paths to each flow in a collective when they arrive at the NIC.

To strengthen our observation, we analytically prove that a singlepath transport (from NIC's perspective) with uniform path assignment is equivalent to an optimal multipath transport with packet spraying in terms of congestion under collective communication workloads. Our proof relies on the feasibility to split a flow (at application layer) and we show that the required splitting is minimal.

We develop a novel singlepath transport, Ethereal, that greedily assigns paths to each flow arriving at the NIC such that the load is uniformly spread across each path in the network. Our preliminary evaluations on leaf-spine topology, under data-parallel distributed training workloads, show that Ethereal can achieve nearly the same completion times for a collective compared to a multipath transport with packet spraying. This challenges the prevailing belief that packet spraying is *necessary* for improving the performance of distributed training workloads and offers an alternative to complex NIC designs with reorder buffers and loss recovery mechanisms.

Finally, we outline future research directions to enhance the practicality of singlepath transport protocols like Ethereal for state-of-the-art GPU clusters, particularly under various parallelism strategies and in scenarios involving link failures.

In the remainder of this chapter, our distinction[1] *between singlepath and multipath is solely from the network interface card (NIC) flow state and congestion control point-of-view.*

3.1 Motivation

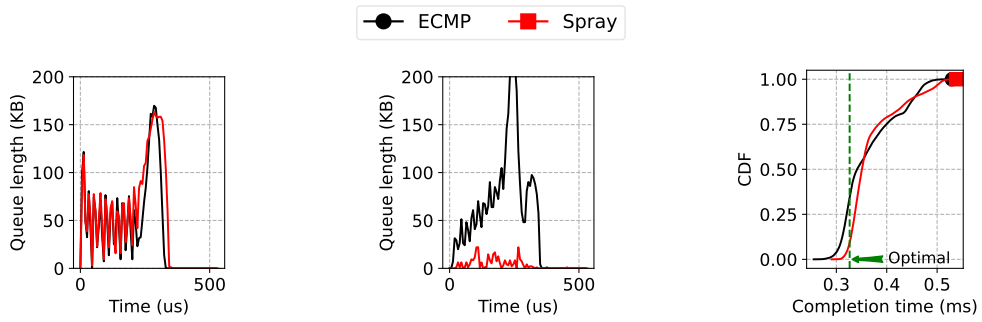
We dissect the transport protocol problem of distributed training workloads using a running example. We simulate a leaf-spine topology consisting of 256 nodes, arranged into 8 leaves and 8 spines, with 100 Gbps links. We run an allReduce collective, the most common communication pattern used for gradient synchronization across multiple GPUs during the backward pass phase of training. For simplicity, we use an all-to-all algorithm to implement allReduce, i.e., every node communicates with every other node. We set the transfer size between each pair of GPUs to 16 KB, much lower than the bandwidth-delay product, ensuring that the congestion control algorithm does not significantly impact our observations and claims.

3.1.1 Repetitive Incasts at the Edge

A leaf-spine topology theoretically supports (without congestion) any traffic pattern within the capacity limits of each sender and receiver [204]. Given the uniform communication pattern of an allReduce, it is typical to expect smooth transmission between each pair of GPUs without experiencing any congestion. However, the allReduce collective induces repetitive incast at the receivers. Figure 3.2a shows multiple incasts at a particular receiver in our simulation, under equal-cost multipath routing (ECMP) and packet spraying (Spray). ECMP chooses one path among the available equal-cost paths based on a 5-tuple hash, leading to congestion due to hash collisions. Packet spraying randomly chooses a path in the network core for every packet, uniformly spreading the load across all paths. Surprisingly, both ECMP and packet spraying fail to mitigate these repetitive incasts, as seen in Figure 3.2a. Further, since we set all flow sizes to less than the bandwidth-delay product, any congestion control algorithm would incur these repetitive incasts. We conclude that the incast problem in collective communication is fundamentally unrelated to load-balancing and congestion control. Interestingly, this is a synchronization problem across flows within a collective. Although flows arrive nearly at the same time from a NIC's perspective, there is a specific order in which they arrive due to the sequential nature of how the flows are launched by the underlying communication library. For instance, NCCL launches all-to-all transmissions starting from the same rank (say rank 0) at each GPU. As a result, an incast initially appears at rank 0, then at rank 1, and so on, repetitively. To mitigate these repetitive incasts from a NIC's perspective, simple randomization techniques are essential to desynchronize the transmissions.

■ ***Takeaway.*** Repetitive incasts during collective communication do not fundamentally stem from inefficiencies in load-balancing but rather from the synchronization of flow arrivals.

[1] We chose to view this as the complexity of the transport implementation and congestion control at the NIC.



(a) leaf→ server (b) leaf→ spine (c) CCT

Figure 3.2: Flow synchronization and non-uniform load-balancing critically impact the completion times of collective communication workloads.

3.1.2 Non-uniform Load in the Core

It is well-known that ECMP routing causes congestion in the core due to hash collisions. Our example in Figure 3.2b shows this effect, with ECMP accumulating more than 200KB of queue length even for the small transfer size used in our setup. Although there are a large number of flows in an all-to-all transmission (65280 flows in our setup), ECMP does not achieve uniform load balancing. Even doubling the number of flows (increasing the entropy) in our setup resulted in similar congestion levels at the core links. As such, we note that controlling the entropy is a non-trivial task and load-balancing techniques relying on entropy are prone to similar problems. In contrast, packet spraying effectively balances the load across all core links, keeping the queue lengths close to zero. The near-optimal load-balancing property of packet spraying, essentially a multipath transport, makes it a more attractive choice for distributed training workloads. However, this approach requires implementing reorder buffers and sophisticated loss recovery mechanisms in hardware, complicating NIC design. We seek to understand whether a particular choice of paths for each flow (unlike ECMP) can balance the load on each path uniformly, similar to packet spraying. Intuitively, if there are 6 flows of equal size and 3 paths to choose from, assigning 2 flows to each path can achieve uniform load across the 3 paths, similar to packet spraying, as illustrated in Figure 3.1. The question remains whether such a path assignment can be achieved without requiring a centralized controller, which would introduce additional delays and complexity to the design.

■ ***Takeaway.*** Packet spraying achieves near-optimal load-balancing in the core but complicates NIC design. The question remains: can any alternative to ECMP achieve similar load balancing without a centralized controller?

3.1.3 Poor Completion Time

The primary concern in distributed training workloads is the completion time of collective communications. Figure 3.2c shows the CDF of flow completion times achieved by ECMP and packet spraying in our example. Our focus is on the tail completion time that indicates the collective completion time (CCT). Neither of these approaches achieve satisfactory completion times. First, the poor completion time with ECMP is attributed to its inefficient load-balancing. Second, even with near-optimal load balancing, packet spraying is no better than ECMP, primarily

due to flow synchronization (§3.1.1).

■ ***Takeaway.*** Improving collective completion time requires better congestion management techniques *both* in the network core as well as at the edge i.e., across the entire network.

3.2 Singlepath vs Multipath

In this section, we seek to understand whether a singlepath (per-flow) load-balancing algorithm can achieve nearly the same objective as a multipath (packet spraying) load-balancing algorithm. Specifically, we are interested in simple, distributed algorithms that operate on each end-host's NIC without relying on a centralized controller. In the following, we formally establish an equivalence between singlepath and multipath load-balancing under certain traffic patterns.

Theorem 4 (Equivalence)**.** *Given a leaf-spine topology, with ℓ leaves, s spines and k server nodes, a set of demands $\mathcal{M} = \{f_i \times n_{i,j} \mid f_i, n_{i,j} \in \mathbb{N}, i \in [1,k], j \in [1,\ell]\}$, where f_i is the flow size and $n_{i,j}$ is the number of flows that a server node i has towards any set of destination nodes within leaf j, then a greedy distributed algorithm (at each node) that splits the least number of flows and assigns each flow to the least congested (local perspective) uplink (leaf↔spine), is equivalent to packet spraying in terms of the objective to minimize the maximum congestion.*

We make three key observations about the communication patterns of distributed training that put our result in Theorem 4 into context for transport protocol design. We present the formal proof at the end of this section.

A single flow can be split to multiple flows: While multipath transport protocols split the flow only logically, from a transport point of view, in the context of distributed training, the communication library (traffic source) allows certain user configurations such that each flow can be split into multiple flows with corresponding queue pairs (separate connections) at the NIC. For instance, environment variables in NCCL, such as `NCCL_IB_QPS_PER_CONNECTION`, enable splitting the data uniformly into multiple queue pairs. As a result, the number of flows can be controlled in a much more fine-grained manner. We use this property later in the proof of Theorem 4 to show that splitting (if needed) only a few number of flows from each source is sufficient to achieve the same properties of an optimal multipath load-balancing algorithm in terms of the maximum congestion.

GPU flows are equal in size within each collective step: Each GPU in a cluster can perform computation and communication (collectives) simultaneously but it does not participate in multiple collective operations simultaneously in order to maintain consistent updates to the model. Further, flows originating from a GPU corresponding to each collective step are of equal size eg., due to chunking, and padding used by NCCL. As result, each i^{th} GPU may any number of flows each of size f_i that fits within the definition of the demands considered in Theorem 4.

Fairness across flows is not a requirement: Given that our primary concern is the completion time of a collective, fairness across flows is not necessary. Scheduling the flows in any particular order while maintaining uniform load across the network core and high link utilization is critical to achieve fast collective completion times.

Given the workload properties, our result in Theorem 4 indicates that a singlepath transport is sufficient for large-scale distributed training. We now present a formal proof.

Proof of Theorem 4. We begin with node $i = 1$ and account for the total amount of demand assigned to each uplink and downlink[2]. For simplicity, we assume that all demands exit the leaves (use uplinks), focusing on the congestion on both the uplinks and downlinks. There are $n_{1,j}$ flows, each of size f_1, towards each destination leaf j. Our greedy algorithm (depicted in Algorithm 3), say ALG, operates in steps for each destination leaf j. By fixing the destination leaf j, choosing an uplink from the source's perspective also implies selecting the corresponding downlink (via the chosen leaf→spine uplink). Consequently, the demand assigned to each uplink from source i equals the demand assigned to the corresponding downlink. ALG first assigns $\lfloor \frac{n_{1,j}}{s} \rfloor$ flows to each uplink. At this point, there are $r = n_{1,j} \bmod s$ flows left to be assigned. Let $g = \gcd(r, s)$[3]. ALG splits each of the r flows into $\frac{s}{g}$ flows, each of size $\frac{f_1 \cdot g}{s}$, and assigns $\frac{r}{g}$ (an integer) flows of size $\frac{f_1 \cdot g}{s}$ to each uplink. In total, ALG assigns $f_1 \cdot \lfloor \frac{n_{1,j}}{s} \rfloor + \frac{f_1 \cdot g}{s} \cdot \frac{r}{g} = f_1 \cdot (\lfloor \frac{n_{1,j}}{s} \rfloor + \frac{r}{s}) = f_1 \cdot \frac{n_{i,j}}{s}$ of demand to each uplink. An optimal multipath load-balancing algorithm (OPT) splits the total demand evenly across all uplinks, assigning $f_1 \cdot \frac{n_{i,j}}{s}$ of demand to each uplink. This demonstrates that ALG and OPT assign equal demand to each uplink (and consequently to each downlink) from each source i, and the proof follows. Additionally, at each source, ALG only splits $r = n_{1,j} \bmod s$ flows, and these are split into $\frac{s}{g}$ flows. The extra flows created by ALG at each source are limited to $\frac{r \cdot (s-g)}{g}$. To prove that the splitting is minimal, assume a better splitting is feasible, with each of the r remaining flows split into $\gamma < \frac{s}{g}$ flows. Thus, the size of each remainder flow is $\frac{f_1}{\gamma}$. We need to assign $r \cdot \gamma$ flows to s uplinks such that the assignment results in $\frac{f_1 \cdot r}{s}$ demand on each uplink to be optimal in terms of congestion. This requires $\gamma \geq 1$ to be an integer (for splitting) and $\frac{r \cdot \gamma}{s}$ to be an integer (for assignment). The smallest value of γ that satisfies these conditions is $\frac{s}{g}$, contradicting our initial assumption. □

We emphasize that the same result cannot be derived for hash-based ECMP since it does not explicitly converge based on the number of flows, but rather on the entropy of the input to the hash function, which is a non-trivial task to control in practice. Furthermore, such properties cannot be derived for singlepath transport in general. The problem of finding the minimum congestion unsplittable flow is known to be NP-hard [85]. The best-known result for CLOS topologies establishes a tight bound of 4-approximation for a greedy algorithm [85]. However, the specific properties of collective communication workloads allow us to establish the above results. We note that although the splitting is minimal, splitting still implies additional state space at the NIC. However, the required number of splittings depends on the total number of flows $n_{i,j}$ towards a destination leaf j and the number of uplinks s. For instance, an allReduce collective implemented using all-to-all algorithm in a non-oversubscribed topology would *not* require any splitting in order to achieve optimal load-balancing using our algorithm.

[2]We refer to leaf→spine links as uplinks and spine→leaf links as downlinks.

[3]$\gcd(r, s)$ is the greatest common divisor of r and s.

Algorithm 3: ETHEREAL

Input : Batch of flows $\mathcal{F}$ ($n_{i,j} \geq 0$ towards each leaf)
Number of uplinks s

```
function selectPath(F):                                   ▷ Potentially by GPU
    ▷ Uniform load-balancing
    for each leaf do
        assign ⌊n_{i,j}/s⌋ flows to each uplink
        r = remaining flows
        g = gcd(r, s)
        split each remaining flow to s/g flows
        for each uplink do
            assign r/g remaining flows
        end for
    end for
function flowArrival(F):
    ▷ Mitigates repetitive incasts
    for each flow ∈ F do
        Schedule after small random interval:
            f → nextAvail = now
            Insert f to a random position in the list of queue pairs
    end for
function Ack():
    DcTcp()                                               ▷ Any CCA could be used
/* ------------------------------------------------------ */
/* Rare scenarios                                         */
function NAck():
    Select new path                                       ▷ Reroute
    GoBackN()                                             ▷ Loss recovery
function flowTimeout(f):
    ▷ Handles link failures
    Select new path
    GoBackN()
function receive():                                       ▷ At the receiver
    ▷ Handles slow links (NICs) in the network
    if f(ECN) > threshold then
        sendNack()                                        ▷ Triggers rerouting
        Ignore out-of-order packets for the next RTT
    else
        sendAck()
    end if
```

3.3 Ethereal Transport for AI

Reflecting on our observations in §3.1 and §3.2, we design a singlepath[4] transport, ETHEREAL, suitable for distributed training workloads in leaf-spine topologies. ETHEREAL has three main components *(i)* path assignment (load-balancing) *(ii)* randomization *(iii)* loss recovery and handling failures. Importantly, ETHEREAL

[4]We note that the notion of singlepath and multipath in our context is from a NIC's state and congestion control point-of-view.

uses a singlepath transport control logic. Implementing ETHEREAL only requires a few changes to the NIC hardware. We describe each of the components of ETHEREAL conceptually and leave the concrete implementation details to future work. Algorithm 3 outlines ETHEREAL's logic.

Path assignment: ETHEREAL achieves uniform load-balancing using a novel technique to assign flows to paths in the network, summarized by the pathSelect function in Algorithm 3. ETHEREAL's load-balancing is greedy in nature i.e., it assigns a batch of flows towards each leaf to the set of uplinks such that the overall congestion on each uplink as well as each downlink is equivalent to packet spraying. Upon path assignment, every packet transmitted by the NIC carries a path ID and switches in the network perform routing based on the pathID of each packet, requiring minimal changes to both NIC as well as the switches. Although currently we assume that the NIC performs path selection, we envision that the corresponding source GPU and collective communication library can potentially assist in this calculation, further reducing the need for significant changes to NIC hardware design.

Randomization: ETHEREAL randomizes the start time of each flow and the position of a flow in the active list of queue pairs. This simple technique addresses the major problem of flow synchronization in GPU clusters, discussed in §3.1.1.

Congestion control: We currently use DCTCP as the congestion control algorithm for ETHEREAL. In principle, any ECN-based algorithm could be used provided the following properties are satisfied: *(i)* quickly ramps up throughput without causing congestion *(ii)* reacts to congestion scenarios and rapidly converges to equilibrium (stable queue lengths) without losing throughput. Developing a fine-grained congestion control algorithm for ETHEREAL is part of our future work.

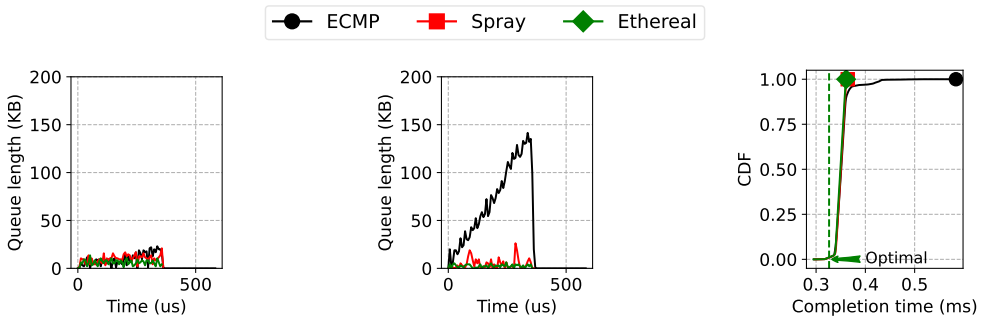
Loss recovery: In view of compatibility with existing RoCE implementations, ETHEREAL assumes that the network supports PFC and relies on NACK with GoBackN for packet drops in the network. Further, ETHEREAL uses a timeout per-flow that upon expiry triggers GoBackN and recovery.

Handling Failures: ETHEREAL relies on two mechanisms to quickly detect failures (or slow links) and reroute flows: *(i)* The receiver immediately sends NACK and ignores any out-of-order packets for the next RTT, triggering a GoBackN and recovery at the sender upon receiving a NACK. The sender then selects a new ''good'' path. *(ii)* If a timeout expires, a new ''good'' path is chosen. We leave further details of loss recovery and failures for future work.

3.4 Preliminary Results

We evaluate ETHEREAL using packet-level simulations using ns-3 [209]. Our preliminary evaluations focus on data parallelism workloads i.e., all the GPUs in the network participate in collective communication. Our goal is to empirically understand how close to packet spraying can ETHEREAL perform and to validate our equivalence claim in §3.2.

We consider a leaf-spine topology consisting of 256 servers, 16 leaves and 16 spines, connected using 100Gbps links with a propagation delay of 500ns. We run All-to-All and Ring communication patterns representative of allReduce collective in GPU clusters. Specifically, in All-to-All, every server communicates with every other server (255 flows each) and in Ring each server communicates



(a) leaf→ server (b) leaf→ spine (c) CCT

Figure 3.3: ETHEREAL's randomization technique effectively mitigates the repeated incasts problem (see Figure 3.2) and improves the completion times.

with only one other server (cross-rack) using 4 channels. We vary the transfer (flow) size and report the CDF of completion times as well as the shared buffer occupancy of switches. Our main metric is on the collective completion time i.e., tail completion time. We compare ETHEREAL with Equal Cost Multipath (ECMP), packet spraying and REPS [71]. We use DCTCP [35] with PFC enabled[5] as the congestion control algorithm for all our baselines including ETHEREAL. Figure 3.3 and Figure 3.4 show our results.

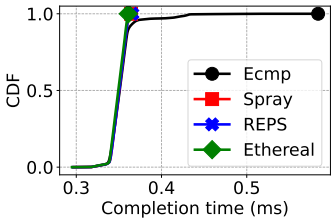
Desynchronization improves the completion times: Figure 3.3 shows the results for the same setup used for our discussed in §3.1 (Figure 3.2). From Figure 3.3a, we see that randomization effectively mitigates repetitive incasts and synchronization problems. We apply the randomization technique of ETHEREAL to all our baselines in the following.

Ethereal significantly improves the completion times: Figure 3.4a, 3.4b and Figure 3.4e, 3.4f show the completion times for All-to-All and Ring (one iteration) respectively. We observe that ETHEREAL not only outperforms ECMP (singlepath), but also outperforms packet spraying and REPS for large transfers (1MB). When the transfers are short (16KB), while ECMP significantly underperforms, ETHEREAL performs similar to packet spraying and REPS. In the case of All-to-All (A2A), ETHEREAL does not incur any flow splitting since $\frac{s}{g} = 1$. However, in the case of 4-channel Ring, ETHEREAL splits each flow into $\frac{s}{g} = 4$ flows (16 in total at each NIC) and uniformly balances the load across the network. As shown by Figure 3.4e, 3.4f, this minimal splitting allows ETHEREAL to significantly outperform, even with a singlepath transport protocol.

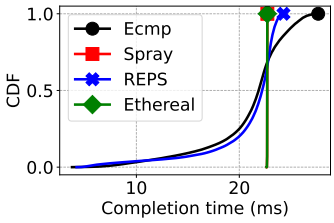
Ethereal reduces the buffer requirements: Given that datacenter switch buffer sizes are shallow, it is desirable that the transport protocol and load-balancing have low buffer requirements. Figure 3.4c, 3.4d and Figure 3.4g, 3.4h show the buffer utilization for All-to-All and Ring. ETHEREAL requires less buffers compared to alternative approaches, highlighting the effectiveness of ETHEREAL's greedy algorithm for path assignment.

Ethereal outperforms even with low entropy: Unlike ECMP and REPS, ETHEREAL does not rely on entropy for load-balancing. Our 4-Ring communication pattern launches only 4 flows at each sender. Note that our topology consists of 16 uplinks and downlinks. As a result, achieving uniform load balancing is a

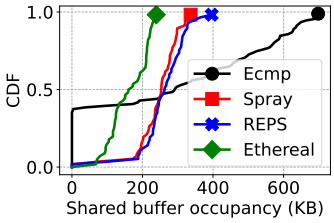
[5]We use reorder buffers for multipath protocols and do not trigger NACK for out-of-order packets.



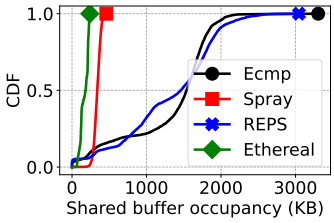
(a) A2A Transfer size = 16KB



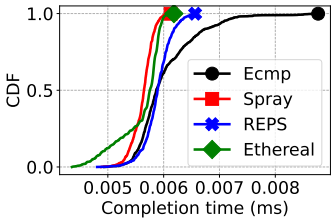
(b) A2A Transfer size = 1MB



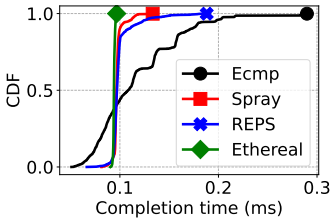
(c) A2A Transfer size = 16KB



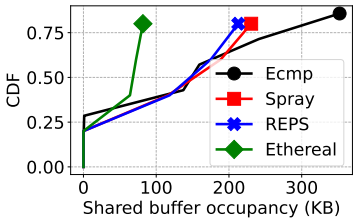
(d) A2A Transfer size = 1MB



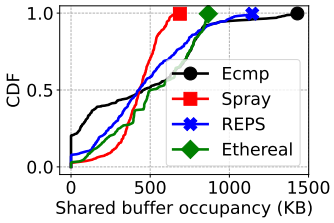
(e) 4-Ring Transfer size = 16KB



(f) 4-Ring Transfer size = 1MB



(g) 4-Ring Transfer size = 16KB



(h) 4-Ring Transfer size = 1MB

Figure 3.4: Completion times and buffer requirements under All-to-All (A2A) and Ring communication patterns.

much more difficult task due to the high likelihood of collisions. Although REPS performs packets spraying in the first RTT, it uses a cached entropy from then on, while switching (exploring) paths based on ECN. Figure 3.4e and Figure 3.4f show how REPS performs much worse compared to packet spraying and Ethereal for Ring communication patterns, highlighting the drawbacks of entropy-based approaches. In contrast, Ethereal uses minimal splitting and assigns paths based on the least congested path.

3.5 Epilogue

This work challenges the prevailing belief that packet spraying is necessary to improve the performance of large-scale distributed training workloads. We present Ethereal, an alternative approach that relies solely on the existing singlepath transport protocol from a NIC's perspective. We analytically demonstrate that optimal load balancing can be achieved by splitting a tiny fraction of the flows upon arrival. Many interesting open questions remain: *(i)* reducing the state space needed by the NIC to serve a large number of flows, e.g., maintaining a small window of active flows, *(ii)* quickly adapting to failures, and *(iii)* extending Ethereal to multi-tiered folded CLOS topologies. Our results warrant further research on transport protocols like Ethereal. We urge the community to rethink the design space, viewing this work as a prologue.

Part II

Datacenter Switch Buffer Sharing

4

Buffer Sharing with Lossy Traffic

Network devices are equipped with a buffer to avoid drops during transient congestion and to absorb bursts. To reduce cost and maximize utilization, the on-chip buffer is shared across its queues. This sharing naturally leads to various problems. Concretely, the excessive growth of a queue might harm the performance of another queue, which might be starved, deprived of throughput, etc. Worse yet, such harmful interference might occur across queues that are seemingly independent e.g., queues that are mapped to different ports or queues that are formed by independent applications.

Network devices typically employ a hierarchical packet admission control to orchestrate the use of the shared space. First, a Buffer Management (BM) algorithm [46,86,161] dynamically splits the buffer space across queues. Second, an Active Queue Management (AQM) algorithm [102,139,215] manages the buffer slice that BM allocates to each individual queue by selectively admitting the incoming packets. Historically, BM and AQM evolved independently with orthogonal goals. We visualize this in Figure 4.1. BM aims at achieving *isolation* across queues by managing the *spatial* allocation of the buffer at the device level. Intuitively, BM's goal is to avoid long-lived queue starvation, effectively enforcing fairness in the steady state. For instance, Dynamic Thresholds [86] aims at weighted fairness across multiple queues in a device. In contrast, AQM's goal is to maintain stable queuing delays by managing the *temporal* allocation of the buffer at the queue level. Intuitively, AQM prevents bufferbloat by avoiding packets stay in the buffer for "too long" [102, 139, 215]. For instance, ECN-based AQM such as RED [102] control the queue lengths via ECN marking; delay-based AQM such as Codel [139, 215] control the queueing delays at a fixed reference value.

While this decoupling has been reasonable and successful in the past as it allowed BM and AQM schemes to evolve further, two recent datacenter trends make the need for coordination between them pressing. First, buffer size is not keeping up with the increase in switch capacity [59, 112]. In effect, BM no longer has enough buffer available to provide isolation to each queue. Second, as traffic becomes more bursty and incast scenarios more prevalent [53,230,282], the transient state of the buffer needs to be controlled at the device-level [59]. To keep up with these trends, a buffer-sharing scheme needs to provide isolation, bounded drain time and high burst tolerance.

In this work, we show that today's BM and AQM schemes are fundamentally unable to independently satisfy these requirements. Driven by this insight, we propose ABM, an Active Buffer Management algorithm that incorporates the insights from both BM and AQM to achieve high burst absorption without sacrificing throughput. Concretely, ABM leverages both total buffer occupancy at the device level and the individual queue drain time. Essentially, ABM is a function

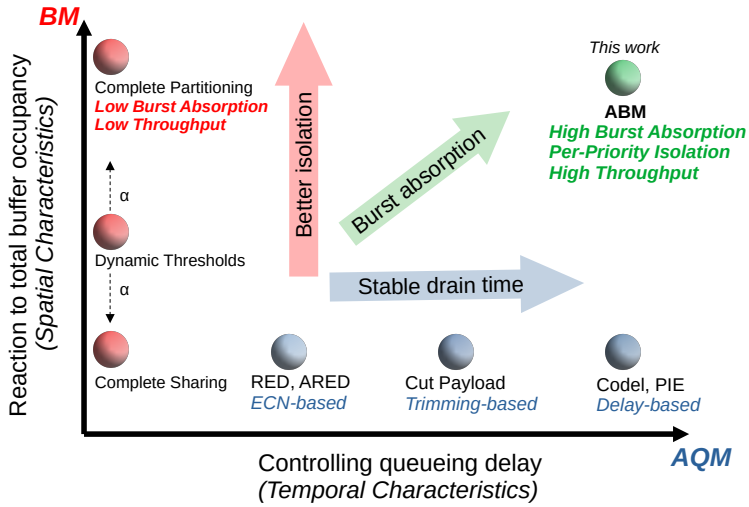


Figure 4.1: BM and AQM are orthogonal in their goals, and the hierarchical scheme fundamentally limits the burst absorption capabilities of the buffer.

of both spatial (used by BM) and temporal (used by AQM) characteristics of the buffer, effectively providing the best of both worlds as shown in Figure 4.1. We analytically show that unlike state-of-the-art, ABM achieves strong isolation properties and maintains stable buffer drain time. This allows ABM to provide high and predictable burst absorption while achieving high throughput. We consider ABM practical, as it operates using statistics that are already used by BM or AQM algorithms, thus ABM is well within the capabilities of today's devices.

Our results from large-scale simulations show that ABM improves the flow completion times for short flows by up to 94% compared to existing BM and AQM schemes. Moreover, ABM is not only compatible with advanced congestion control algorithms (e.g., TIMELY, DCTCP and PowerTCP) but improves their performance in terms of tail FCTs by up to 76% under bursty workloads. Finally, we show that unlike traditional buffer management schemes, ABM works well on various buffer sizes, including shallow buffers (e.g., Tomahawk [1,3]).

We view our work as the beginning towards a new class of ABM algorithms which react to both total buffer occupancy and the queuing delay.

In summary, our contributions in this chapter are:

- We reveal the fundamental limitations of BM and AQM schemes that prevent optimally absorbing bursts (§ 4.1.2).
- We analytically show the critical limitations of the state-of-the-art buffer management scheme (§ 4.1.3).
- We design Active Buffer Management (ABM), an algorithm that achieves high burst absorption and maintains high throughput by leveraging both total buffer occupancy and queue drain time (§ 4.2).
- An extensive evaluation that demonstrates the benefits of ABM in the datacenter context (§ 4.4).
- As a contribution to the research community, to ensure reproducibility and facilitate future work, we made all of our artifacts publicly available at https://github.com/inet-tub/ns3-datacenter.

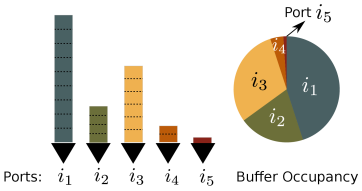


Figure 4.2: The queue at port i_5 is starved (deprived of buffer) due to other queues that are using the buffer.

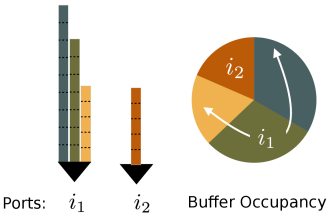


Figure 4.3: While one of the queues at port i_1 has the same length as the queue at port i_2, it has 3x queuing delay.

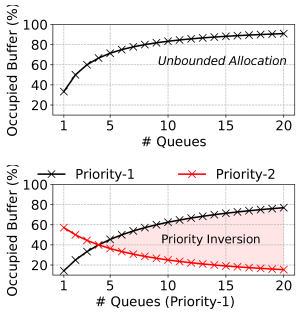


Figure 4.4: As the number of congested queues increases (Equation 4.6), DT allocates a unboundedly large amount of buffer (top) and cannot address isolation (bottom).

4.1 Motivation

In this section, we make a case for cooperation between Buffer Management (BM) and Active Queue Management (AQM) to reap the best out of the precious but limited on-chip buffer space. Hereafter, we say **buffer-sharing** scheme to refer to any scheme within the two dimensions of BM and AQM as shown in Figure 4.1. After describing our model, we explain the desirable properties of a buffer-sharing scheme (§ 4.1.1). We then discuss the limitations of existing approaches (§ 4.1.2). Finally, we analytically reveal the pitfalls of the state-of-the-art BM scheme, namely Dynamic Thresholds (§ 4.1.3).

Model: We consider an output-queued shared-memory packet switching chip. Two schemes affect the allocation of the shared buffer. First, a BM scheme dynamically decides the maximum length of each queue. Second, an AQM scheme decides whether an incoming packet will be enqueued, marked, trimmed, or dropped. We consider that packets are grouped into a small set of priorities $\mathcal{P}$. Each priority exclusively uses a separate queue at each port.

4.1.1 Desirable Properties

To maximize the benefits of the shared buffer, a buffer-sharing scheme needs to satisfy three key properties: *(i)* isolation; *(ii)* bounded drain time; and *(iii)* predictable burst tolerance. Next, we describe these properties and motivate them through intuitive examples.

4.1.1.1 Isolation

As the buffer is shared across multiple queues, the excessive use of buffer by a small set of queues in aggregate might interfere with the ability of other queues in the switch to use the shared buffer. Such interference can be particularly harmful if the competing queues belong to different traffic priorities. As an intuition, consider the case illustrated in Figure 4.2. A queue serving a traffic priority that is particularly critical to the operator (e.g., loss-sensitive traffic) starves because the buffer is occupied by queues formed on other ports and even by best-effort traffic. To avoid such harmful interference across queues, different traffic priorities must be isolated. Concretely, we require that each priority must be allowed to occupy a

configurable minimum amount of buffer at any given time, independently of the buffer state.

Formally, let $T_p(t)$ denote the total allocated buffer to a priority $p \in \mathcal{P}$ in the set of all priorities $\mathcal{P}$ at any time t. Then $T_p(t)$ must always be greater than B_p^{min}, a configurable static value. However, since the total buffer space, B is limited, it is necessary that the total allocation is within B. Thus, each priority must also be upper bound in its allocation (B_p^{max}) to prevent monopolizing the buffer.

$$\textit{Isolation (Minimum Guarantee):} \quad T_p(t) \geq B_p^{min} \tag{4.1}$$

$$\textit{Isolation (Preventing Monopoly):} \quad T_p(t) \leq B_p^{max} \tag{4.2}$$

4.1.1.2 Bounded drain time

Queueing delays are the root cause of high flow completion times for short flows in a datacenter [35, 178, 199]. Hence, various schemes including ECN-based AQM aim at reducing the queue lengths. Indeed, queuing delay also affects the buffer's drain time, namely, how fast the occupied buffer can be made available for incoming traffic (e.g., a burst or an incast). Importantly, though, queueing delay is not equivalent to a queue's length. In fact, two queues with the same queue length might experience different queueing delays. This is possible as a variable number of queues can share the bandwidth of a single port. Consider the example shown in Figure 4.3: a queue in port i_1 experiences 3x queueing delay compared to the queue of equal length in port i_2. This is the case as the former queue can use $\frac{1}{3}$ of the port's bandwidth while the latter can use the full bandwidth of the port. Even within the same port, queues of equal length might experience different queueing delay depending on the underlying scheduling algorithm (e.g., weighted round-robin).

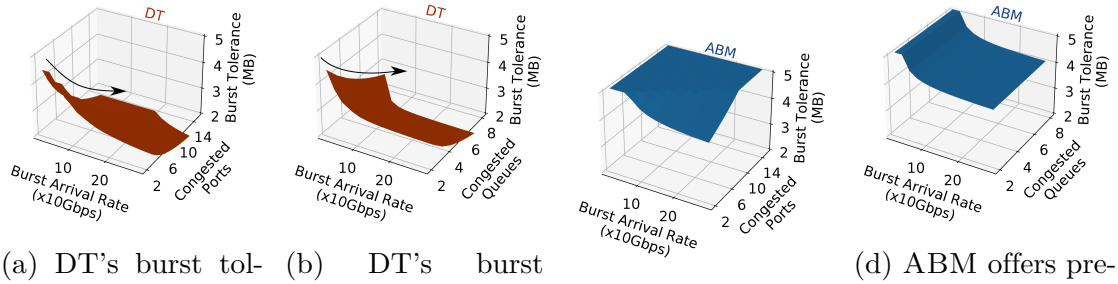
To avoid the harmful consequences of high queuing delays, a buffer-sharing scheme needs to bound the per-queue drain time. Concretely, a buffer-sharing scheme needs to bound the occupied buffer $q(t)$ of a queue with service rate $\mu(t)$ by a configurable static value Γ. In fact, the following condition summarizes the bufferbloat problem [107, 139, 215]: the desired property is to avoid packets staying ''too long'' in the buffer.

$$\textit{Bounded drain time:} \quad \frac{q(t)}{\mu(t)} \leq \Gamma \tag{4.3}$$

4.1.1.3 Predictable burst tolerance

Incast and bursty traffic are key challenges in a datacenter. Dropping packets of a burst results in costly timeouts and is thus undesirable. We define burst tolerance as the maximum burst of packets that the buffer-sharing scheme can store in the buffer until they can be transmitted via the corresponding port.

Intuitively, a buffer-sharing scheme can absorb an incoming burst either if there is burst-size amount of empty buffer available upon arrival (relates to isolation) or if the occupied buffer can drain fast enough to accommodate the burst (relates to drain time). While these conditions are sufficient to absorb a burst, they could have adverse effects. On the one hand, maintaining burst-size amount of empty buffer at all times deprives queues of the precious buffer, resulting in potential loss of throughput. On the other hand, allocating all newly released buffer (from



(a) DT's burst tolerance reduces with the number of congested ports due to its unbounded buffer allocation.

(b) DT's burst tolerance reduces with the number of congested per-port queues due to its obliviousness to drain rate.

(c) ABM offers predictable burst tolerance by bounded allocation regardless of the number of congested ports.

(d) ABM offers predictable burst tolerance by dynamically adapting allocations to the buffer drain time (number of queues per port).

Figure 4.5: DT's burst tolerance is unpredictable as it depends on unpredictable factors. On the contrary, ABM's burst tolerance remains high even at times of high load.

draining queues) to an incoming burst will starve the draining queues (drop each of their incoming packets). In conclusion, to provide predictable burst tolerance while avoiding buffer waste, a buffer-sharing scheme needs a combination of both isolation and bounded drain time.

4.1.2 Limitations of Existing Approaches

Next, we show that the existing approaches in BM and AQM cannot satisfy the desirable properties of a buffer-sharing scheme discussed in § 4.1.1. We focus on BM and AQM as these "actively" control the worst-case (maximum) queue lengths.

We first generalize the steady-state workings of both BM and AQM using a fluid-flow model. We are in steady state when the load conditions and buffer occupancy remain unchanged. We define the threshold of a queue as the value of queue length after which incoming packets are not admitted in the buffer. Thus, the steady-state of both BM and AQM can be generalized as shown in Equation 4.4, where T_p^i is the steady-state length of a queue at port i and of priority p. In the following, Ψ is a function that any BM scheme uses to calculate the per-queue thresholds based on buffer-wide metrics i.e., the instantaneous total buffer occupancy Q, while Φ is a function that any AQM scheme uses to calculate the per-queue threshold based on per-queue metrics e.g., length q and queue drain rate μ_p^i. The effective threshold per queue T_p^i is the minimum between the two thresholds,

$$T_p^i = \min\left(\underbrace{\Psi_p^i(Q)}_{BM}\ ,\ \underbrace{\Phi(q, \mu_p^i)}_{AQM}\right) \tag{4.4}$$

Limitations of Active Queue Management (AQM): AQM schemes control the queue lengths/delay at a fixed reference value. For instance, RED [102] queues used in DCTCP [35] would set $\Phi = K$, where K is the marking threshold (a constant). Delay-based AQM such as PIE [215] queues would set $\Phi = K \cdot \frac{\mu_p^i}{b}$ where μ_p^i is the drain rate of the queue and b is the port bandwidth; $\frac{K}{b}$ is the delay

reference value. In principle, delay-based AQM schemes which set $\Phi = K \cdot \frac{\mu_p^i}{b}$ can maintain constant queue drain time. As discussed earlier in § 4.1.1, bounded drain time helps in absorbing bursts, since the occupied buffer can rapidly react to accommodate the incoming burst. However, the total shared buffer occupancy with AQM controlled queues is $n \cdot \Phi$ where n is the total number of queues using the buffer. AQM has no visibility over other queues using the buffer and thus cannot bound the overall buffer occupancy. As a result, AQM cannot satisfy the isolation property we describe in § 4.1.1.

■ ***Takeaway.*** While AQM can, in principle, guarantee bounded drain time to help with burst tolerance, AQM cannot fundamentally satisfy the isolation property due to its inability to control the shared buffer.

Limitations of Buffer Management (BM): Buffer Management schemes assign thresholds to every queue on a device. Thus, a BM scheme can, in principle, achieve isolation across queues. For instance, complete partitioning sets $\Psi_p^i(Q) = \frac{B}{N}$ where N is the total number of queues in the buffer. This naturally isolates each queue, but at the cost of extremely low buffer utilization. More dynamic schemes, such as Dynamic Thresholds (DT) [86] improve buffer utilization but sacrifice isolation. We analytically show this and other limitations of DT in § 4.1.3. Further, BM schemes are oblivious to the drain time, since they only react to total buffer occupancy i.e., BM schemes cannot ensure that packets leave the buffer ''fast enough''. As a result, BM schemes cannot satisfy the drain time and burst-tolerance property.

■ ***Takeaway.*** BM schemes can, in principle, achieve isolation but are fundamentally limited in burst tolerance as a result of being oblivious to buffer drain time.

4.1.3 Drawbacks of the State-of-the-Art Buffer Management Scheme

In this section, we shed light on the important drawbacks of the state-of-the-art buffer management algorithm used in today's datacenter switches, namely Dynamic Thresholds (DT) [86]. DT is the most common buffer management today [46, 126, 191, 211, 252]. We first explain how DT calculates its thresholds. Next, we describe why it is fundamentally unable to achieve the desirable properties described in § 4.1.1.

DT's workings: DT dynamically adapts the instantaneous maximum length of each queue of priority p, namely its threshold according to the remaining buffer and a configurable parameter α_p often configured per priority.[1] Formally, let B be the total shared buffer and $Q(t)$ be the total buffer occupancy at time t. DT calculates the threshold $T_p^i(t)$ for a queue of priority p at port i as follows:

$$T_p^i(t) = \alpha_p \cdot \underbrace{(B - Q(t))}_{\text{Total Remaining}} \tag{4.5}$$

The α parameter of a queue affects its maximum length relative to the other queues. An operator is likely to set higher (resp. lower) α values for high-priority (resp. low-priority) traffic. Different vendors and operators reportedly use different α

[1] While α can be configured per queue, it is often configured per priority.

values. For instance, Yahoo uses $\alpha = 8$ [126] while Cisco uses $\alpha = 14$ [252] and Arista $\alpha = 1$.

In the following, we analytically show that Dynamic Thresholds (DT) suffers from low and unpredictable burst tolerance. The key reasons for this limitation are: *(i)* the unbounded buffer allocation; and *(ii)* the obliviousness to the buffer drain time.

Unbounded buffer allocation: To explain why DT is fundamentally unable to bound its allocation, we walk through the workings of DT in steady-state. We refer to steady-state as the time during which the load conditions, the total buffer occupancy, and the thresholds remain stable (unchanged) i.e., $\dot{T}^i_p(t) = 0; \quad \dot{Q}(t) = 0$. The threshold calculated by DT for a queue of priority p at port i in steady-state is given by,

$$T^i_p(t) = \frac{\alpha_p \cdot B}{1 + \sum_{p \in \mathcal{P}} n_p \cdot \alpha_p} \tag{4.6}$$

where $\mathcal{P}$ is the set of priorities using the buffer and n_p is the number of congested queues of priority p. As we see in Equation 4.6, DT's threshold for a queue of priority p is dependent on the configurable α_p value and the number of congested queues of each priority using the buffer. As the number of congested queues n_p increases, the threshold decreases arbitrarily close to zero. In effect, DT cannot offer any minimum available buffer to any priority, i.e., cannot offer any isolation.

Figure 4.4, illustrates this effect. First, the remaining buffer tends to zero as the number of congested queues increases. Second, the threshold for a queue of certain priority (e.g., loss-sensitive AF) drops as the number of congested queues of another priority (e.g., best-effort BE) increases. Observe that this occurs even though the α_p value is higher for the loss-sensitive priority.

Unpredictable burst tolerance: Intuitively, DT's threshold calculation results in unpredictable burst tolerance for two reasons. First, DT's unbounded buffer allocation (which we described before) cannot predict the amount of buffer available for an incoming burst. Second, DT is oblivious to drain time, thus unaware of the rate at which the buffer can be made available for an incoming burst.

To formally verify this intuition, we analyze the state of the buffer throughout the arrival of a packet burst that arrives in a queue at port i, priority p and with a drain rate of μ^i_p. We describe the burst as incoming traffic with an arrival rate r. Before the burst arrives, given the number of congested ports, we denote the aggregate drain rate as μ. Upon the arrival of a burst at time $t = 0$, the buffer enters a transient-state: the total buffer occupancy, remaining buffer and consequently the thresholds change until they stabilize to a steady-state.

For time $t = 0^+$, the queue (hosting the burst) starts to grow, and its threshold changes. The change in the queue's threshold depends on the change in the total remaining buffer (see Equation 4.5). To study the rate of change of the threshold, we take its derivative with respect to time and obtain the following:

$$\dot{T}^i_p(t) = -\alpha_p \cdot \dot{Q}(t) \tag{4.7}$$

We integrate on both sides of Equation 4.7 in the interval $t = 0$ to a time τ when the queue hosting the burst reaches its threshold i.e., $T^i_p(t) = Q^i_p(t)$. We substitute the initial conditions for $t = 0$ from the steady-state occupancy (Equation 4.6) and solve for the time τ. In essence, the queue experiences zero loss until τ and the

corresponding queue length at τ i.e., $(r - \mu_p^i) \cdot \tau$ indicates the burst tolerance of the buffer given the initial steady-state occupancy. Depending on the arrival rate r of the burst, we split into two cases.

First, if the arrival rate r is such that the threshold of each queue at $t = 0$ drops at a rate less than the corresponding queue's drain rate, then all the queues are able to drain according to the changes in the thresholds. In this case, the burst occupies the steady-state allocation corresponding to its queue (Equation 4.6). Notice that DT's burst tolerance even when the arrival rate r is low, critically depends on the number of congested queues of each priority due to its unbounded buffer allocation. Figure 4.5a illustrates the poor burst tolerance of DT across different burst arrival rates and initial steady-state conditions.

Second, if the arrival rate r is such that the threshold of each queue at $t = 0$ drops at a rate greater than the corresponding queue's drain rate, then the queues cannot keep up with the changes in the threshold and the burst experiences drops at time τ before reaching its steady-state allocation. In this case, we obtain the burst tolerance of DT $(r - \mu_p^i) \cdot \tau$ as follows:

$$\overbrace{(r - \mu_p^i) \cdot \tau}^{\text{Burst Tolerance}} = \frac{\alpha_p \cdot B}{\left(1 + \sum_{p \in \mathcal{P}} n_p \cdot \alpha_p\right) \cdot \left(1 + \alpha_p \cdot \frac{(r - \mu_p^i) - \mu}{r - \mu_p^i}\right)} \tag{4.8}$$

Observe that DT's burst tolerance critically depends not only on the number of congested queues n_p of each priority but also on the difference between the burst arrival rate r and the aggregate buffer drain rate μ. Even worse, DT's obliviousness to drain rate allows queues to increase in lengths irrespective of their drain time --- effectively increasing n_p even with low aggregate drain rate μ. As a result, DT significantly suffers from low burst tolerance. Figure 4.5b illustrates the consequences of the obliviousness of drain time in buffer allocation to burst tolerance.

We have so far proved our intuition, namely that the burst tolerance depends on two factors *(i)* the total buffer occupancy at $t = 0$ in the steady-state (Equation 4.6) and *(ii)* the drain rate of the buffer in the transient-state (Equation 4.7).

4.2 Active Buffer Management

Driven by our observations in § 4.1.3, we design a buffer-sharing scheme that systematically combines the insights of both BM and AQM while avoiding the pitfalls of existing schemes. Our goal is to satisfy the properties we identified in § 4.1.1:

- Provide isolation ▷ Theorem 5, 6
- Maintain bounded drain time ▷ Theorem 7
- Achieve predictable burst tolerance ▷ Equation 4.11

4.2.1 The ABM Algorithm

ABM assigns thresholds to each queue considering both spatial, buffer-wide and temporal, per-queue statistics. Formally, ABM assigns a threshold $T_p^i(t)$ to a queue of priority p at port i according to Equation 4.9 i.e., using a configurable value α_p, the port's bandwidth b and three dynamically changing factors: *(i)* the number of congested queues of priority p (n_p) that contributes to isolation property; *(ii)* the drain rate of the queue (μ_p^i) that contributes to maintain bounded drain time property; and the remaining buffer space ($B - Q(t)$).

$$T_p^i(t) = \overbrace{\alpha_p \cdot \frac{1}{n_p} \cdot (B - Q(t))}^{Buffer\ Management} \cdot \overbrace{\frac{\mu_p^i}{b}}^{AQM} \tag{4.9}$$

α_p is the only parameter the operator needs to configure in ABM. Similarly to DT, a higher α_p value in ABM results in a higher allocation *on average*. Unlike DT though, the α_p defines the minimum and maximum buffer available to each priority, as we show in § 4.2.2

n_p denotes the number of congested queues of priority p. ABM considers a queue congested if the queue length is close to the corresponding threshold. In our evaluation, we consider a queue as congested if its length is greater than or equal to 0.9 of its threshold.

$\frac{\mu_p^i}{b}$ denotes the normalized drain rate, where μ_p^i is the drain rate of the queue and b is the bandwidth per port.[2] In effect, $\frac{\mu_p^i}{b}$ is the portion of the port's bandwidth that is available to the particular queue. In our analysis, we assume that the drain rate μ_p^i is a continuous value that changes according to the offered load and the scheduling policy. For instance, each of four queues that are mapped to the same port will have $\frac{\mu_p^i}{b} = 0.25$ if the scheduling is Round Robin. To ensure practicality, in our evaluations, we measure μ_p^i periodically and use the measurement for the threshold calculation. We also perform a sensitivity analysis on the periodic update intervals.

$(B - Q(t))$ denotes the unused and unreserved buffer space. Importantly, this factor is identical to what DT uses.

4.2.2 ABM's Properties

We now show how ABM satisfies the desirable properties, namely isolation, bounded drain time, and predictable burst absorption. We also provide formal arguments for these properties in the next section (§4.3).

ABM offers isolation across priorities. Concretely, ABM *(i)* bounds the total buffer occupied by each priority; and *(ii)* offers minimum buffer guarantees to each priority. In effect, no priority can monopolize the buffer and starve others. ABM achieves this by considering the number of congested queues per priority when calculating the per-queue thresholds. As an intuition, the per-queue threshold of a given priority decreases as more queues of that priority are congested.

[2] If the switch ports are not symmetric in bandwidth, b is the bandwidth of the port with the highest bandwidth.

Concretely, Theorem 5 addresses the isolation property of minimum guarantee, while Theorem 6 addresses the isolation property of preventing monopoly. We formally define and discuss the need for those properties in § 4.1.1.

Theorem 5 (Isolation - Minimum guarantee)**.** *The total amount of buffer available for any priority p is lower bounded by B_p^{min} given by,*

$$B_p^{min} \geq \left(\frac{B \cdot \alpha_p}{1 + \sum_{p \in \mathcal{P}} \alpha_p} \right) \cdot \frac{\hat{\mu}_p}{b}$$

where $\hat{\mu}_p$ is the average dequeue rate of the congested queues of priority p.

Theorem 6 (Isolation - Preventing monopoly)**.** *The total amount of buffer available for any priority p is upper bounded by B_p^{max} given by,*

$$B_p^{max} \leq \frac{B \cdot \alpha_p}{1 + \alpha_p}$$

Notably, both bounds depend only on the α_p parameter which the operator configures, and not on the instantaneous state of the buffer as in DT.

ABM bounds drain time by allocating buffer space proportionately to the drain rate of each queue. In effect, ABM bounds queuing delay and total buffer drain time irrespective of the number of congested queues in a port, or the scheduling policy. We formally express the drain time properties of ABM in Theorem 7.

Theorem 7 (Bounded drain time)**.** *The thresholds assigned by ABM upper bounds the drain time Γ for any queue of priority p given by,*

$$\Gamma \leq \frac{B \cdot \alpha_p}{(1 + \alpha_p) \cdot b}$$

Notably, this bound is only dependent on the α_p which the operator configures, and b which is the switch port bandwidth which is static at run-time.

Intuition for proof of Theorem 5, 6, 7: We first define Ω (formally defined in Definition 1) as $\Omega_p^i = \alpha_p \cdot \frac{1}{n_p} \cdot \frac{\mu_p^i}{b}$. Notice that ABM's allocation scheme (Equation 4.9) is essentially $T_p^i(t) = \Omega_p^i \cdot (B - Q(t))$. We can then view Ω as an adaptive α parameter according to DT's allocation (Equation 4.5)[3]. We derive an important property that the sum of Ω_p^i values of all congested queues of a certain priority p is upper bounded by α_p (Lemma 1). The buffer allocation for a priority p in steady state turns out to be $\frac{\sum_i \Omega_p^i \cdot B}{1 + \sum_i \sum_{p \in \mathcal{P}} \Omega_p^i}$ where $\mathcal{P}$ is the set of priorities using buffer. Based on the property of Ω, we then derive lower and upper bounds (Theorem 5, 6) for the buffer allocated to a certain priority. Similarly, using the upper bound for buffer allocation and dividing by the drain rate μ_i^p, we bound the drain time in Theorem 7.

ABM's allocation scheme in Equation 4.9 has its roots in the properties of Ω that offer isolation and bounded drain time properties. For full proofs, we refer the interested reader to our complete analysis in §4.3.

ABM offers predictable burst tolerance thanks to the previous properties. Intuitively, ABM's bounded allocation makes it ready to absorb small bursts and

[3]Setting $\Omega_p^i = \alpha_p$ reduces to DT's allocation scheme.

the stable drain time property further enhances ABM's burst tolerance as the occupied buffer can readily react to incoming bursts. To formalize ABM's burst tolerance, we analyze the state of the buffer throughout the arrival of a packet burst that arrives on a queue at port i, priority p and with a drain rate of μ_p^i. Similar to our analysis of DT in § 4.1.3, we describe the burst as incoming traffic with an arrival rate r. Upon the arrival of a burst at time $t = 0$, the buffer enters a transient state.

First, if the arrival rate r is such that the threshold of each queue at $t = 0$ reduces at a rate less than the corresponding queue's drain rate, then the aggregate buffer drains according to the changes in the thresholds and the burst occupies its steady-state allocation as follows:

$$\overbrace{(r - \mu_p^i) \cdot \tau}^{\text{Burst Tolerance}} = \frac{\alpha_p \cdot \frac{1}{n_p} \cdot B \cdot \frac{\mu_p^i}{b}}{1 + \sum_{p \in \mathcal{P}} \alpha_p} \tag{4.10}$$

Notice that ABM's burst tolerance is independent of the number of the congested queues of *other* priorities. Rather, the burst tolerance only reduces due the number of congested queues of the *same* priority.

Second, if the arrival rate r is such that the threshold of each queue at $t = 0$ reduces at a rate greater than the corresponding queue's drain rate, then the aggregate buffer cannot drain according to the changes in the thresholds. In this case, we obtain ABM's burst tolerance given by Equation 4.11, where μ is the aggregate drain rate of the buffer at time $t = 0$.

$$\overbrace{(r - \mu_p^i) \cdot \tau}^{\text{Burst Tolerance}} = \frac{\alpha_p \cdot \frac{1}{n_p} \cdot B \cdot \frac{\mu_p^i}{b}}{\left(1 + \sum\limits_{p \in \mathcal{P}} \alpha_p\right) \cdot \left(1 + \alpha_p \cdot \frac{1}{n_p} \cdot \frac{\mu_p^i}{b} \cdot \frac{(r - \mu_p^i) - \mu}{r - \mu_p^i}\right)} \tag{4.11}$$

Finally, notice from Equation 4.11 that ABM's burst tolerance on a queue of a certain priority remains independent of the number of congested queues of other priorities. Further, unlike DT, ABM accounts for the buffer drain time and achieves a high burst tolerance even with low aggregate drain rate. In essence, ABM's burst tolerance is a function of the burst arrival rate r. Observe from Equation 4.11 that the burst tolerance of a certain priority still depends on the number of congested queues of the *same* priority. The buffer occupancy of a priority self-inflicts its own burst tolerance. In § 4.2.3, we further optimize ABM's thresholds to prevent this effect.

Figure 4.5 illustrates ABM's burst tolerance properties under various buffer states. Observe that ABM's burst tolerance for a priority remains high and only depends on *rate* at which the burst arrives regardless of the number of ports that are congested (Figure 4.5c) and of the number of congested queues per port (Figure 4.5d). Recall that DT's burst tolerance depends on both these factors and is thus unpredictable as we observe in Figure 4.5a, 4.5b.

4.2.3 Optimizing for Datacenter Workloads

As mentioned above, although ABM alleviates the dependency on other priorities in its burst tolerance of a given priority, the buffer occupancy of each priority

self-inflicts its own burst tolerance. To prevent this effect, we further optimize ABM's thresholds to maximize its burst tolerance.

To this end, ABM prioritizes all unscheduled packets by using a higher α_p value in allocating buffer for two reasons. First, bursty traffic in a datacenter originates mainly from unscheduled (first RTT) packets of a flow, since congestion control cannot fundamentally act within the first RTT. Second, short flows that finish within the first RTT are of utmost importance in a datacenter. Even a single packet loss could lead to costly timeouts and long flow completion times.

Specifically, even the unscheduled packets are destined to a specific queue at each port based on their default traffic priority. However, ABM uses a higher α_p value in its thresholds while admitting such packets to the buffer. We assume that unscheduled packets arrive with a tag attached by the end-hosts. Observe that such a tag can also be dynamically obtained if the switch is programmable. In essence, by prioritizing unscheduled packets, ABM prevents the self-inflicting effect as seen in Equation 4.11 i.e., a higher α_p for unscheduled packets diminishes the effect of self-inflicting n_p factor. Note that unscheduled packets relate to the transient state of the buffer, and prioritizing such packets does not affect the steady-state properties of ABM. As a result, ABM maximizes the burst tolerance properties and remains independent of the number of congested ports, as well as the number of congested queues at each port.

4.2.4 ABM's Practical Considerations

ABM is attractive in practice for three reasons.

ABM uses statistics that are available to today's switches. ABM cannot be easily implemented today, as the MMU is not programmable even on programmable devices [242]. Still, it is important to note that ABM only uses statistics that are used either by BM or by AQM schemes, thus is not fundamentally impossible. α_p and $(B - Q(t))$ are used by DT [86] which is implemented on most datacenter switches [126, 191, 211, 252]. n_p, i.e., the number of congested queues of priority p only requires visibility over queue lengths, which is provided to both AQM and BM schemes to decide whether a packet can to be admitted. $\frac{\mu_p^i}{b}$ depends on the port's bandwidth, the scheduling algorithm, and the number of congested queues mapped to the same port. The two former do not change. Thus, if the number of congested queues mapped to the same port is static, then $\frac{\mu_p^i}{b}$ also becomes static. If the number of congested queues (i.e., exceeding a threshold) mapped to the same port changes over time, then we only need this number to calculate the rate. For example, if scheduling is round-robin and there are two congested queues in a port, then the normalized drain rate is $\frac{\mu_p^i}{b} = 0.5$. Further, several congestion control algorithms implemented in real datacenters already use in-band telemetry and insert μ_p^i in packet headers (e.g., HPCC [173]). Finally, ABM's threshold also requires a floating point operation similar to DT which calculates $\alpha_p \cdot (B - Q(t))$.

ABM teaches essential lessons on how to configure α values. Although ABM and DT have major differences in their properties, their thresholds are in fact similar (cf. Equations 4.6, vs 4.9). Thus, ABM's insights can help an operator configure DT. Concretely, the operator could divide α_p values to the number of queues mapped to the same port or to the number of congested ports they expect to have. Finally, an operator can use our mathematical analysis (also

illustrated in Fig. 4.5) to find an approximation of the burst tolerance of their DT configuration leveraging their insights about the usual state of the buffer (i.e., number of congested queues per port and the number of congested ports).

One can approximate ABM on top of DT. As ABM's threshold (Equation 4.9) is so similar to DT's formula (Equation 4.5) an operator could approximate ABM using DT. To that end, the operator would need to implement a control-plane function that periodically pulls queue statistics and reconfigures α_p per queue according to ABM 's thresholds. Observe that most vendors today expose queue statistics that are required to calculate $\frac{\mu_p^i}{b}$ and n_p i.e., queue lengths [2, 7, 9, 14]. How close this approximation would be to ABM depends on two factors *(i)* how frequently a device can be reconfigured; and *(ii)* how dynamic the traffic patterns are. In § 4.4.4, we evaluate the effect of such an approximation by varying the time intervals in which α_p values are updated. We find that the approximation performs similarly to ABM for small update intervals, but performs similarly to DT at high update intervals.

4.3 Analysis

In this section, we formally model and analyze a shared memory switch architecture with ABM's allocation scheme. The aim of our analysis is to show ABM's formal guarantees and its properties. We refer the reader to §4.1 for the terminology and definitions regarding ports, queues and priorities. Our analysis indeed generalizes both ABM and DT. Substituting static α_p values for ω_p^i (Definition 1) gives the analysis of DT.

4.3.1 Model

For generality, we model a switch with an arbitrary but fixed number of ports and queues per port. In particular, each port has only one queue per priority as defined in (§4.1). The switch in our model has a shared memory architecture with B buffer space. We denote the instantaneous occupied buffer at time t as $Q(t)$. Our analysis is based on a fluid model where packet (bits) arrivals and departures are assumed to be fluid and deterministic.

We denote by α_p, the parameter used by ABM for each priority in allocating the buffer. Each priority is associated with a separate queue at each port. We denote port indices by i and priority p. $\mathcal{P}$ is the set of priorities using the buffer. The number of congested queues of a priority p at time t is denoted by $n_p(t)$.

4.3.2 Formalizing ABM's Allocation

As described in §4.2, the threshold of a queue at port i and belonging to a priority p is calculated based on the alpha parameter α_p, the number of congested queues $n_p(t)$, the normalized drain rate of the queue indicated by $\gamma_p^i(t)$ and the remaining buffer $B - Q(t)$. Formally,

$$T_p^i(t) = \alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t) \cdot (B - Q(t)) \tag{4.12}$$

where, $\beta_p(t) = \frac{1}{n_p(t)}$ is the inverse of the total number of congested queues of priority p at time t.

Definition 1 (Omega - Adaptive α Parameter)**.** *For a queue belonging to a priority p, the product $\alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t)$ in ABM's buffer allocation scheme (Eq. 4.12) is defined as Omega denoted by $\omega_p^i(t)$ and is viewed as an adaptive alpha parameter.*

$$\omega_p^i(t) = \alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t) \tag{4.13}$$

Based on the above definition of ω, in the following we derive an upper bound on the sum of ω values for all the classes of a priority p. Later in our analysis, we will see how the sum of ω values plays a key role in ABM's buffer allocation scheme. We will later use this upper bound to derive several properties and formal guarantees provided by ABM.

Lemma 1 (Property of Omega)**.** *The instantaneous sum of $\omega_p^i(t)$ over all the queues belonging to a priority $p \in \mathcal{P}$ across all the ports is upper bounded by α_p.*

$$\sum_i \omega_p^i(t) \leq \alpha_p \tag{4.14}$$

Proof. Using Definition 1 and observing that $\beta_p(t)$ is the number of congested queues of a priority p is the same across all the queues of the same priority, we express the sum of $\omega_p^i(t)$ as follows:

$$\sum_i \omega_p^i(t) = \sum_i \alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t) = \alpha_p \cdot \beta_p(t) \cdot \sum_i \gamma_p^i(t)$$

Since $\gamma_p^i(t)$ is the normalized drain rate, $\gamma_p^i(t)$ is upper bounded by 1. Finally, we substitute $\beta_p(t) = \frac{1}{n_p(t)}$. We reduce the sum to an inequality as follows, where the last inequality holds since $\sum_i \gamma_p^i(t) \leq n_p(t)$ i.e., the sum of the normalized drain rates is upper bounded by the number of congested queues.

$$\alpha_c \cdot \beta_p(t) \cdot \sum_i \gamma_c^i(t) \leq \alpha_p \cdot \frac{1}{n_p(t)} \cdot n_p(t) \leq \alpha_p$$

□

4.3.3 Steady-State Analysis

We now analyze the steady-state behavior of ABM's buffer allocation scheme. Specifically, we say steady-state when the load-conditions remain stable and a steady buffer occupancy is achieved. Under steady-state, the queue lengths remain stable at less than or equal to their corresponding thresholds. To stress on the worst-case scenarios, we assume that any occupied queue is at the respective threshold. In our steady-state analysis, for simplicity of presentation, we drop the time variable in all the equations.

Under steady-state, we are interested in determining the overall buffer allocation and occupancy denoted by Q, the remaining buffer space $B - Q$ and ABM's threshold calculation per queue T_c^i.

Lemma 2 (Steady-state allocation)**.** *Under steady-state, given a set of congested queues, the overall buffer occupancy Q is given by Eq. 4.15, the remaining buffer $B - Q$ is given by Eq. 4.16 and the threshold per congested queue calculated by ABM is given by Eq. 4.17.*

$$Q = \frac{B \sum_i \sum_p \omega_p^i}{1 + \sum_i \sum_p \omega_p^i} \tag{4.15}$$

$$B - Q = \frac{B}{1 + \sum_i \sum_p \omega_p^i} \tag{4.16}$$

$$T_p^i = \frac{B \cdot \omega_p^i}{1 + \sum_i \sum_p \omega_p^i} \tag{4.17}$$

Proof. In the steady-state, from the assumption that the queue lengths are equal to their thresholds, we derive the overall buffer occupancy by summation of queue lengths of all the congested queues. Using Eq. 4.12 and Eq. 4.13 we express the total buffer occupancy Q as follows and solve for Q leading to the last equality.

$$Q = \sum_i \sum_p \omega_p^i \cdot (B - Q) = \frac{B \sum_i \sum_p \omega_p^i}{1 + \sum_i \sum_p \omega_p^i}$$

The remaining buffer space (Eq. 4.16) is then straight-forward by substituting Q. Finally, the threshold per queue T_p^i (Eq. 4.17) is obtained by definition from ABM's allocation scheme i.e., $T_p^i = \alpha_p \cdot \beta_p(t) \cdot \gamma_p^i(t) \cdot (B - Q) = \omega_p^i \cdot (B - Q)$. □

In the following, we derive ABM's formal guarantee on isolation i.e., ABM offers minimum buffer space per priority based on the α parameters.

Theorem 5 (Isolation - Minimum guarantee)**.** *The total amount of buffer available for any priority p is lower bounded by B_p^{min} given by,*

$$B_p^{min} \geq \left(\frac{B \cdot \alpha_p}{1 + \sum_{p \in \mathcal{P}} \alpha_p} \right) \cdot \frac{\hat{\mu}_p}{b}$$

where $\hat{\mu}_p$ is the average dequeue rate of the congested queues of priority p.

Proof. The threshold T_p^i for each queue of priority p is given by Equation 4.17. We sum across all ports and obtain the total allocated buffer as follows,

$$\sum_i T_p^i = \frac{B \cdot \sum_i \omega_p^i}{1 + \sum_i \sum_{p \in \mathcal{P}} \omega_p^i} \geq \frac{B \cdot \sum_i \omega_p^i}{1 + \sum_{p \in \mathcal{P}} \alpha_p} \geq \left(\frac{B \cdot \alpha_p}{1 + \sum_{p \in \mathcal{P}} \alpha_p} \right) \cdot \frac{\hat{\mu}_p}{b}$$

where the last inequality holds since $\sum_i \sum_p \omega_p^i \leq \sum_p \alpha_p$ from Lemma 1 and $\sum_i \omega_p^i = \alpha_p \cdot \sum_i \frac{\mu_p^i}{b} \geq \alpha_p \frac{\hat{\mu}_p}{b}$. □

Theorem 6 (Isolation - Preventing monopoly)**.** *The total amount of buffer available for any priority p is upper bounded by B_p^{max} given by,*

$$B_p^{max} \leq \frac{B \cdot \alpha_p}{1 + \alpha_p}$$

Proof. The proof is similar to Theorem 5. To obtain the upper bound, we use the property that $\sum_i \sum_{p \in \mathcal{P}} \omega_p^i \geq \alpha_p$, a case when only the priority p is using the buffer. □

Theorem 7 (Bounded drain time)**.** *The thresholds assigned by ABM upper bounds the drain time* Γ *for any queue of priority* p *given by,*

$$\Gamma \leq \frac{B \cdot \alpha_p}{(1 + \alpha_p) \cdot b}$$

Proof. Using Equation 4.17 from Lemma 2 and noting that the drain time is occupied buffer divided by its drain rate ($\gamma_p^i \cdot b$), we obtain the drain time Γ as follows for a queue at port i and of priority p,

$$\Gamma = \frac{B \cdot \alpha_p \cdot \frac{1}{n_p} \cdot \gamma_p^i}{\gamma_p^i \cdot b \cdot (1 + \sum_i \sum_p \omega_p^i)} \leq \frac{B \cdot \alpha_p}{b \cdot (1 + \alpha_p)}$$

The last inequality holds since $\sum_i \sum_p \omega_p^i \geq \alpha_p$ and $\frac{1}{n_p} \cdot \gamma_p^i \leq 1$. □

4.3.4 Transient-State Analysis

In this section we analyze ABM's transient-state properties. We define transient-state as a state when the buffer is initially in the steady-state and at time $t = 0$ load conditions change, creating a transient buffer state until the queue lengths stabilize. In particular, we consider that at time $t = 0$, a set of initially empty queues have incoming traffic. As a result, the thresholds and queue lengths undergo a transient state. Due to the appearance of new queues, ω_p^i of some of the existing queues get affected due to the changes in β_p (number of congested queues of a priority p) and γ_p^i (normalized drain rate). In the following, we introduce and describe certain notations specific to our transient-state analysis.

- The arrival rate of traffic at each *new* queue is denoted by r and the arrival process is fluid and deterministic. Note that we consider each port has a bandwidth of unit 1 and r is in the same unit.
- G_e denotes the set of queues whose ω_p^i gets affected.[4]
- G_{ne} denotes the set of queues whose ω_p^i does not get affected.
- For simplicity we denote the queue at port i and of priority p with ordered pairs as (i, p).
- The set of ordered pairs of existing queues is denoted as S_{old}. Observe that $S_{old} = G_{ne} \cup G_e$.
- The ordered pairs of new queues that trigger transient state are denoted as S_{new}.
- Dot over a variable denotes its rate of change i.e., derivative with respect to time. For example $\dot{x}$ denotes $\frac{dx}{dt}$.

[4]Note that the ω_p^i values of G_e only reduce. (It is not possible that ω_p^i increases due the appearance of a new queue)

4.3.4.1 Preliminaries

While the transient-state begins at $t = 0$, the initial buffer occupancy is based on the prior steady-state (Lemma 2) as expressed in Eq. 4.18 and Eq. 4.19.

$$T_p^i(0) = \frac{\omega_p^i \cdot B}{1 + \sum_{S_{old}} \omega_p^i} \tag{4.18}$$

$$Q_p^i(0) = \begin{cases} \frac{\omega_p^i \cdot B}{1 + \sum_{S_{old}} \omega_p^i} & , \text{for } \forall (i,p) \in S_{old} \\ 0 & , \text{for } \forall (i,p) \in S_{new} \end{cases} \tag{4.19}$$

At $t = 0^+$, ω_p^i of G_e change and remain same for the entire duration of transient state. At the same time, the ω_p^i of G_{ne} remains unchanged. Hence, such changes are assumed to occur at time $t = 0$ and the time variable is dropped for ω_p^i in the equations.

From Eq. 4.12, we express the rate of change of thresholds and queue lengths as follows,

$$\dot{T}_p^i(t) = -\omega_p^i \cdot \sum_{S_{old} \cup S_{new}} \dot{Q}_p^i(t) \tag{4.20}$$

$$\dot{Q}_p^i(t) = \begin{cases} max[-\gamma_p^i, min[\dot{T}_p(t), r - \gamma_p^i]] & \forall (i,p) \in S_{old} \\ r - \gamma_p^i & \forall (i,p) \in S_{new} \end{cases} \tag{4.21}$$

It can be proved by contradiction that $\frac{dT_p^i(t)}{dt} \leq 0 < r - \gamma_p^i$. Solving Eq. 4.20 and Eq. 4.21 for $t = 0+$,

$$\dot{T}_p^i(t) = -\omega_p^i \cdot \left(\sum_{S_{old}} max[-\gamma_p^i, \frac{dT_p(t)}{dt}_{(t=0+)}] \right) - \omega_c^i \cdot \sum_{S_{new}} (r - \gamma_p^i) \tag{4.22}$$

Recall that $S_{old} = G_e \cup G_{ne}$. All the queues belonging to G_e, will experience a change in their ω_p^i values at $t = 0^+$ resulting in their queue-lengths greater than threshold. As a result, the rate of change of their queue lengths is their corresponding drain rates. Eq. 4.22 can then be expanded as,

$$\dot{T}_p^i(t) = -\omega_p^i \cdot \left(\sum_{G_e} -\gamma_p^i + \sum_{G_{ne}} max[-\gamma_p^i, \dot{T}_p^i(t)] + \sum_{S_{new}} (r - \gamma_p^i) \right) \tag{4.23}$$

From Eq. 4.23, arrival rate of traffic in new queues i.e r can be expressed as,

$$r = \frac{\sum_{S_{new} \cup G_e} \gamma_p^i}{\sum_{S_{new}} 1} - \frac{\dot{T}_p^i(t) + \omega_p^i \cdot \left(\sum_{G_{ne}} max[-\gamma_p^i, \dot{T}_p(t)] \right)}{\omega_p^i \cdot \sum_{S_{new}} 1} \tag{4.24}$$

By applying summation over G_{ne} in Eq. 4.23 (will be seen later how this will be useful), r can be expressed as,

$$r = \frac{\sum\limits_{S_{new} \cup G_e} \gamma_p^i}{\sum\limits_{S_{new}} 1} - \frac{\sum\limits_{G_{ne}} \dot{T}_p^i(t) + \left(\sum\limits_{G_{ne}} max[-\gamma_p^i, \dot{T}_p(t)]\right) \cdot \sum\limits_{G_{ne}} \omega_p^i}{(\sum\limits_{G_{ne}} \omega_p^i) \cdot (\sum\limits_{S_{new}} 1)} \tag{4.25}$$

Now it can be observed that the value of r influences all $\forall (i,p) \in G_{ne}$, $\dot{T}_p^i(t)$. In other words, the value of r influences the total i.e $\sum_{G_{ne}} \dot{T}_p^i(t)$ which is the aggregate rate at which thresholds drop for the non-affected set of queues i.e G_{ne}.

4.3.4.2 Case-1: Aggregate drain rate tracks the threshold changes

In this case, the arrival rate r is such that, the queues belonging to G_{ne} are able to reduce in length exactly tracking the changes in their thresholds. As a result, their queue-lengths remain equal to the threshold throughout the transient state i.e,

$$\left(\frac{dT_p^i(t)}{dt}\right)_{(t=0^+)} \geq -\gamma_p^i \tag{4.26}$$

leading to,

$$\sum_{\forall (i,p) \in G_{ne}} \left(\frac{dT_p^i(t)}{dt}\right)_{(t=0+)} \geq \sum_{\forall (i,p) \in G_{ne}} -\gamma_p^i \tag{4.27}$$

Using Eq. 4.26 and Eq. 4.27 in Eq. 4.25, the condition on r can be expressed as,

$$r \leq \frac{\sum\limits_{S_{new} \cup G_e} \gamma_p^i}{\sum\limits_{S_{new}} 1} + \left(\sum_{G_{ne}} \gamma_p^i\right) \cdot \frac{1 + \sum\limits_{G_{ne}} \omega_p^i}{(\sum\limits_{G_{ne}} \omega_p^i) \cdot (\sum\limits_{S_{new}} 1)} \tag{4.28}$$

For such an arrival rate of traffic at new queues, in the following theorem we state the time up to which the new queues experience zero transient drops.

Theorem 8. *For an arrival rate r within Case-1 (Eq. 4.28) at a set of new queues S_{new}, given an initial state of the buffer at time $t = 0$, a new queue $(i,p) \in S_{new}$ experiences zero transient drops up to a time $t1_p^i$ given by Eq. 4.29*

$$t1_p^i = \frac{\omega_p^i \cdot B \cdot (1 + \sum\limits_{G_{ne}} \omega_p^i)}{X_1 \cdot Y_1}$$

$$X_1 = (1 + \sum_{S_{old}} \omega_p^i)$$

$$Y_1 = ((r - \gamma_p^i) \cdot (1 + \sum_{G_{ne}} \omega_p^i) + \omega_p^i \cdot (\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i))) \tag{4.29}$$

Proof. Substituting Eq.4.26 and Eq.4.27 in Eq. 4.23 and using the result in Eq. 4.21 gives,

$$\dot{T}_p(t) = \frac{-\omega_p^i \cdot \left(\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i) \right)}{1 + \sum_{G_{ne}} \omega_p^i} \tag{4.30}$$

$$\dot{Q}_p^i(t) = \begin{cases} \dfrac{-\omega_p^i \cdot \left(\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i) \right)}{1 + \sum_{G_{ne}} \omega_p^i} & ,\forall (i,p) \in G_{ne} \\ -\gamma_p^i & ,\forall (i,p) \in G_e \\ r - \gamma_p^i & ,\forall (i,p) \in S_{new} \end{cases} \tag{4.31}$$

These differential equations will be valid as long as $Q_p^i(t) = T_p^i(t)$ for $\forall (i,p) \in G_{ne}$, $Q_p^i(t) \geq T_p^i(t)$ for $\forall (i,p) \in G_e$ and $Q_p^i(t) < T_p^i(t)$ for newly created queues i.e $\forall (i,p) \in S_{new}$. Solving these equations, using the initial conditions, Eq. 4.18 and Eq. 4.19 leads to,

$$T_p^i(t) = \frac{\omega_p^i \cdot B}{1 + \sum_{S_{old}} \omega_p^i} - \frac{\omega_p^i \cdot \left(\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i) \right) \cdot t}{1 + \sum_{G_{ne}} \omega_p^i} \tag{4.32}$$

$$Q_p^i(t) = \begin{cases} \frac{\omega_p^i \cdot B}{1 + \sum_{S_{old}} \omega_p^i} - \frac{\omega_p^i \cdot t \cdot (\sum_{G_e} -\gamma_p^i + \sum_{S_{new}} (r - \gamma_p^i))}{1 + \sum_{G_{ne}} \omega_p^i} & \forall (i,p) \in G_{ne} \\ \dfrac{\omega_p^i \cdot B}{1 + \sum\limits_{\forall (i,p) \in S_{old}} \omega_p^i} - \gamma_p^i \cdot t & \forall (i,p) \in G_e \\ (r - \gamma_p^i) \cdot t & \forall (i,p) \in S_{new} \end{cases} \tag{4.33}$$

As we can observe from Eq. 4.32 and Eq. 4.33, the new queues will grow in length without dropping packets up to a time $t1_p^i$ when the threshold equals the queue length. The transient state continues after $t1_p^i$ until all the queues achieve a steady state occupancy. By equating Eq. 4.32 and Eq. 4.33 for the case of $\forall (i,p) \in S_{new}$, we obtain $t1_p^i$ as in Eq. 4.29. □

In order to offer guarantees, it is absolutely required that either γ_p^i is constant. The reason being that there is a dependency between γ_p^i and the number of queues of the same port using buffer, a dependency that is fundamentally impossible to evade unless γ_p^i is constant. As a result of this assumption, $G_e = \phi$ and $S_{old} = G_{ne}$ and Eq. 4.29 reduces to,

$$t1_p^i = \frac{\alpha_p \cdot \frac{1}{n_p} \cdot \gamma_p^i \cdot B}{(r - \gamma_p^i) \cdot (1 + \sum_{S_{old}} \omega_p^i + \omega_p^i \cdot \sum_{S_{new}} 1)} \tag{4.34}$$

As an example, we can further simplify for a case with one high priority and one low priority using the buffer where load variations occur for *High Priority* whose α value is α_H and the existing *Low Priority* in the queues have α value of α_L. We can then guarantee that for an arrival rate r that satisfies *Case-1* will experience zero drops i.e., no transient drops if its duration t satisfies the following condition:

$$t1_p^i = \frac{\alpha_H \cdot \frac{1}{n_p} \cdot \gamma_p^i \cdot B}{(r - \gamma_p^i) \cdot \left(1 + \alpha_L + \alpha_H \cdot \frac{1}{n_p} \cdot \gamma_p^i \cdot \sum\limits_{\forall (i,p) \in S_{new}} 1\right)} \tag{4.35}$$

Observe that Eq. 4.35 is independent of the number of queues of *Low Priority* and hence it can be said that *High Priority* isolation can be guaranteed.

4.3.4.3 Case-2: Aggregate drain rate is slower than the changes in thresholds

In this case, the arrival rate r is such that, the queues belonging to G_{ne} are unable to reduce in length in accordance with the changes in their thresholds. As a result, their queue-lengths remain greater than the threshold throughout the transient state i.e,

$$\left(\frac{dT_p^i(t)}{dt}\right)_{(t=0^+)} < -\gamma_p^i \tag{4.36}$$

leading to,

$$\sum_{\forall (i,p) \in G_{ne}} \left(\frac{dT_p^i(t)}{dt}\right)_{(t=0+)} < \sum_{\forall (i,p) \in G_{ne}} -\gamma_p^i \tag{4.37}$$

Using Eq. 4.36 and Eq. 4.37 in Eq. 4.25, the condition on r can be expressed as,

$$r > \frac{\sum\limits_{S_{new} \cup G_e} \gamma_p^i}{\sum\limits_{S_{new}} 1} + \left(\sum_{G_{ne}} \gamma_p^i\right) \cdot \frac{1 + \sum\limits_{G_{ne}} \omega_p^i}{(\sum\limits_{G_{ne}} \omega_p^i) \cdot (\sum\limits_{S_{new}} 1)} \tag{4.38}$$

Theorem 9. *For an arrival rate r within Case-2 (Eq. 4.38) at a set of new queues S_{new}, given an initial state of the buffer at time $t = 0$, a new queue $(i,p) \in S_{new}$ experiences zero transient drops up to a time $t1_p^i$ given by Eq. 4.39.*

$$\begin{aligned} t1_p^i &= \frac{\omega_p^i \cdot B}{X_2 \cdot Y_2} \\ X_2 &= 1 + \sum_{\forall (i,p) \in S_{old}} \omega_p^i \\ Y_2 &= (r - \gamma_p^i) + \omega_p^i \cdot \left(\sum_{\forall (i,p) \in S_{old}} -\gamma_p^i + \sum_{\forall (i,p) \in S_{new}} (r - \gamma_p^i)\right) \end{aligned} \tag{4.39}$$

We omit the proof of Theorem 9. The proof is similar to Theorem 8.
As an example, with one high priority (α_H) queue experiencing burst and low priority (α_L) queues occupying buffer, from Lemma 1 on the property of ω and observing that $\sum_{\forall(i,p)\in S_{old}} -\gamma_p^i$ is the *Total drain rate of the congested ports of* S_{old}, we derive the following relation where n is the number of congested ports belonging to S_{old} and BW is the bandwidth of each port assuming ports are of same bandwidth (the assumption is not critical to our analysis, it can easily generalized).

$$t1_p^i = \frac{\frac{\alpha_H}{(1+\alpha_L)} \cdot \frac{1}{n_p} \cdot \gamma_p^i \cdot B}{\left((r-\gamma_p^i) + \frac{\alpha_H}{n_p} \cdot \gamma_p^i \cdot \left(-n \cdot BW + \sum_{S_{new}} (r-\gamma_p^i)\right)\right)} \tag{4.40}$$

where n is the number of congested ports belonging to S_{old} i.e.,

$$n \cdot BW = \sum_{\forall(i,p)\in S_{old}} -\gamma_p^i \tag{4.41}$$

Notice that the presence of n in Eq. 4.40, is a dependency on the number of congested ports of *Low Priority*. However, n only creates a positive effect on $t1_p^i$ i.e., greater the n greater is $t1_p^i$. On the other hand, Eq. 4.40 is independent of negative dependencies as was in the traditional algorithm DT where a higher number of congested queues lead to lower $t1_p^i$ leading to faster transient drops.

4.3.4.4 Burst Tolerance

Building on the analysis in the previous sections, in this section, we discuss how the steady-state and transient-state behavior of ABM relates to burst tolerance. We denote the burst tolerance for a queue of priority p at port i as $Burst_p^i$ and is defined as follows:

$$Burst_p^i = r \cdot t1_p^i \tag{4.42}$$

where r is the arrival rate of traffic and $t1_p^i$ is the amount of time starting from $t = 0$ until the queue experiences zero drops.

Based on our analysis of transient state in §4.3.4, we generate analytical graphs as shown in Figure 4.5 (§4.1.3). For each arrival rate r, we first distinguish whether r falls under case-1 (§4.3.4.2) or case-2 (§4.3.4.3). We then calculate the corresponding time $t1_p^i$ until the burst experiences zero drops. We then multiply r and $t1_p^i$ to obtain the burst tolerance.

4.4 Evaluation

We evaluate the performance of ABM and compare it with state-of-the-art approaches in the datacenter setting. Our evaluation aims at answering the following key questions:

(Q1) *How does ABM perform compared to other BMs in burst absorption and isolation?*
We find that ABM improves the 99th percentile FCT slowdown of the flows contributing to bursts by up to 94% (12.6%) under high (low) load compared to existing schemes. Moreover, we show that (unlike DT) ABM offers isolation

across priorities as performance of a given priority is unaffected by the load of other priorities.

(Q2) *Does ABM sacrifice short flows or throughput?*
ABM does not sacrifice throughput or short flows' FCTs in exchange for burst absorption. In fact, ABM reduces the 99th percentile FCT slowdown for short flows by 28.3% on average even at 20% load with Cubic compared to existing BM schemes, while achieving on-par throughput.

(Q3) *Can ABM further improve FCTs of advanced congestion control schemes?*
We show that ABM improves the tail FCT slowdown under bursty workloads in DCTCP by 88%; in TIMELY by 33.3% and in PowerTCP by 2.13% even at moderate burst sizes.

(Q4) *How well does ABM perform in extremely shallow buffers?*
We show that ABM maintains its performance and improves the 99th percentile FCTs of bursty workloads by up to 92% compared to other BMs in extremely shallow buffers.

(Q5) *Can we reap any of ABM's benefits by approximating its allocation with DT?*
We perform a sensitivity analysis on the importance of the update interval in ABM's benefits. We find that one could benefit from ABM by re-configuring DT's α_p values at every ≈ 8 ms (100x RTT).

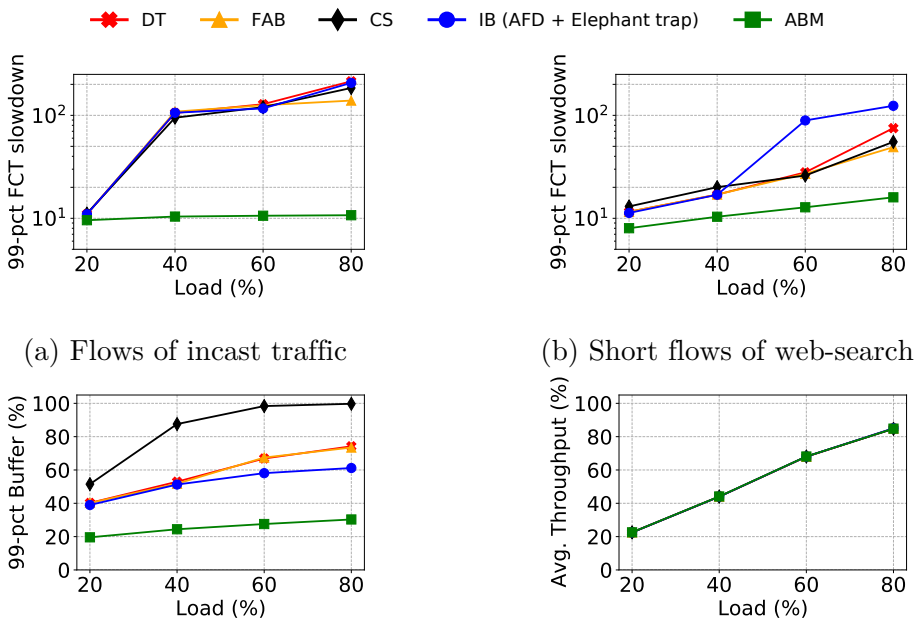
4.4.1 Setup

Our evaluation is based on network simulator NS3 [209].

Topology: We use a Leaf-Spine topology [33] with 8 spine switches and 256 servers organized into eight leaves. Each link has a capacity of 10Gbps (4:1 oversubscription) similar to prior work [21, 232]. Moreover, each link has $10\mu s$ propagation delay. Both leaf and spine switches have 9.6KB buffer-per-port-per-Gbps following the features of the Broadcom TridentII switch [12, 59].

Workload: We generate traffic using two workloads *(i)* web-search; and *(ii)* incast traffic. First, we generate traffic following the web-search flow size distribution [35] which is based on real-world datacenter measurements across various loads in the range 20%-80%. Second, we generate incast traffic using a synthetic workload similar to prior work [21, 33]. Specifically, our incast traffic simulates a distributed file system query-response behavior in a datacenter. Each server in our topology requests a file from a set of servers chosen uniformly at random from a different rack. All servers that receive the request respond by transmitting the requested part of the file. Each file request creates an incast. We evaluate across different request sizes from 10% -75% of the buffer.

Comparisons and metrics: We compare ABM with four alternative BM schemes: Dynamic Thresholds (DT) [86]; Flow Aware Buffer (FAB) [46], Complete Sharing (CS) and Cisco Intelligent Buffer (IB) [252]. DT allocates buffer proportionately to the remaining as we describe in § 4.1.3. FAB uses DT but prioritizes short flows. CS allows every queue in the switch to grow as long as there is remaining buffer in the shared space. IB uses Approximate Fair Dropping in combination with DT representing the typical hierarchical scheme.



(a) Flows of incast traffic

(b) Short flows of web-search

(c) Buffer occupancy

(d) Throughput

Figure 4.6: Buffer Management under various loads. ABM achieves lower tail FCT (a) for flows contributing to bursts (incast traffic) and (b) for short flows (web-search) compared to other BM schemes across various loads. In doing so ABM (c) uses less buffer; and (d) does not sacrifice throughput.

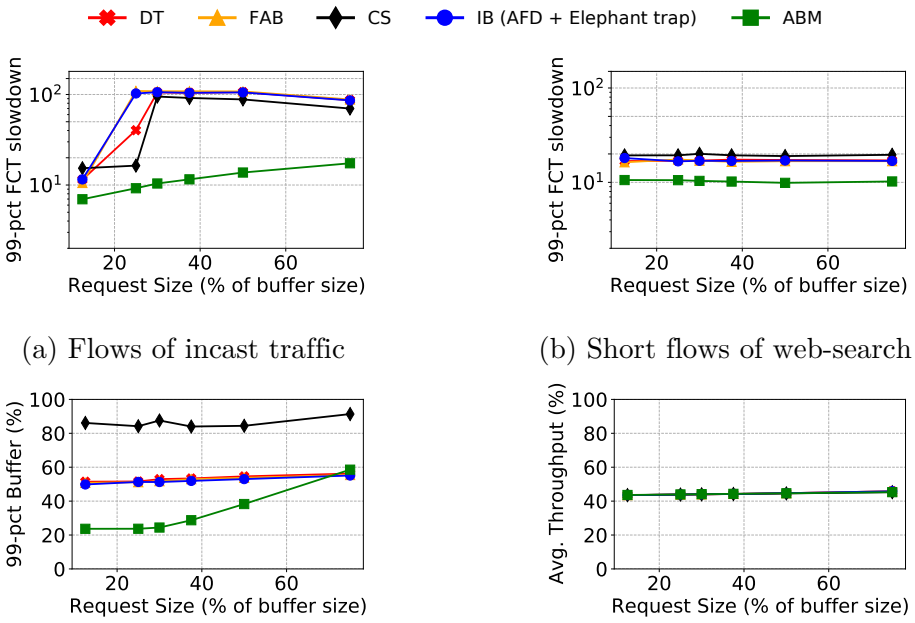
We compare the performance of various congestion control algorithms with and without ABM. In particular, we evaluate ABM with Cubic [118] (loss-based), DCTCP [35] (ECN-based), TIMELY [193] (RTT-gradient-based), PowerTCP and θ-PowerTCP [21] (power-based). We report the switch total buffer occupancy, throughput and Flow Completion Time (FCT) slowdown i.e., the actual FCT divided by the ideal FCT with no other traffic in the network.

Parameter setting: We configure ABM, DT and FAB with $\alpha = 0.5$ for all the queues unless otherwise specified. We update n_p and μ_p^i for ABM once per RTT. We configure IB according to [87]. ABM uses $\alpha = 64$ for unscheduled packets (§ 4.2.3) and uses headroom similar to IB. We set $K = 65$ packets for DCTCP according to [35] and TIMELY parameters according to [193]. For PowerTCP and θ-PowerTCP, we set $baseRTT$ to the minimum RTT of the longest path; $NicBW$ is set to 10Gbps and the γ parameter is set according to [21]. We set $minRTO = 10$ms.

4.4.2 ABM's Performance

ABM significantly improves incast-traffic FCTs: In Figure 4.6a, we show the 99th percentile FCT slowdown for the flows of incast workload with a fixed request size of 30% of the buffer size, as a function of the load created by the web-search workload. All flows use Cubic. We observe that at low load (20%) ABM reduces the 99th percentile FCT slowdown by 12.6% on average compared to DT, FAB, CS and IB. As the load increases, ABM outperforms alternatives in FCT slowdown by 90.12% at 40% load; and by 94% at 80% load.

In Figure 4.7a, we show the 99th percentile FCT slowdown for the flows of incast workload with fixed web-search load in 40% as a function of the request



(a) Flows of incast traffic

(b) Short flows of web-search

(c) Buffer occupancy

(d) Throughput

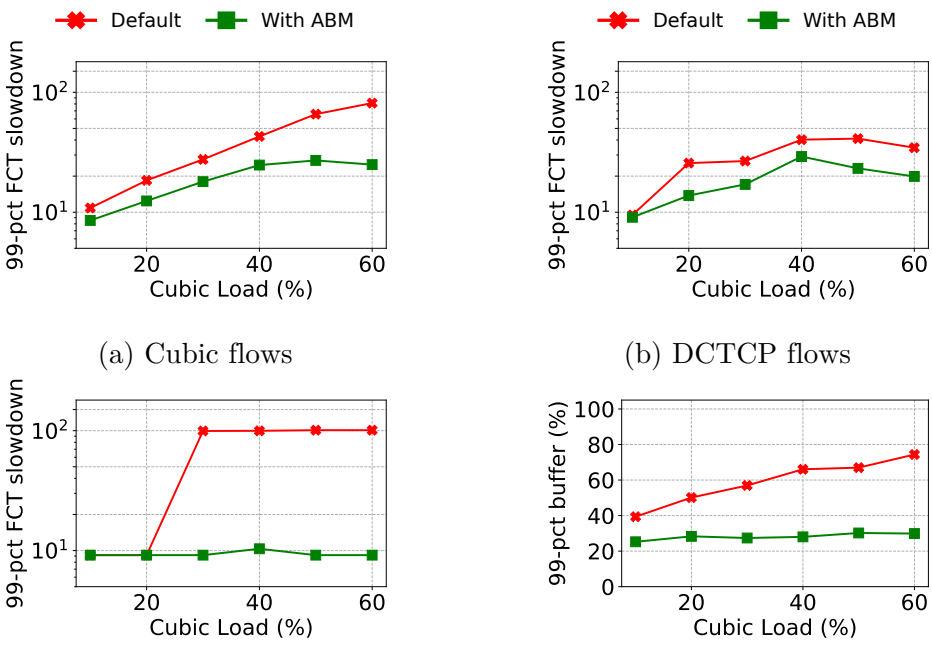
Figure 4.7: Buffer Management under various request sizes. ABM achieves lower tail FCT (a) for flows contributing to bursts (incast traffic) and (b) for short flows (web-search) compared to other BM schemes across various request sizes. (c) ABM's tail buffer occupancy increases with the request size, while (d) throughput stays untouched.

size of incast traffic. Even for a request size as small as 12.5% of the buffer size, ABM reduces the FCT slowdown by 39% on average compared to DT and IB; by 33% compared to FAB; and by 54% compared to CS. As the request size increases, ABM's benefits are more pronounced. At a request size of 50% of the buffer size, ABM reduces the FCT slowdown of incast workload by at least 75.1% on average compared to DT, FAB, CS and IB.

In essence, ABM's bounded buffer allocation (Theorem 6) effectively limits the used buffer by the web-search workload and provides enough headroom for the incast workload. Further, ABM's prioritization of unscheduled packets (incast traffic) fully exploits the available headroom.

ABM effectively isolates traffic priorities: To evaluate ABM's performance isolation across different priorities compared to the default, namely DT, we consider the following scenario. The network is shared across three priorities each using a different transport protocol among Cubic, DCTCP, and θ-PowerTCP. Each priority uses a distinct queue at each port. The most recent protocol, namely θ-PowerTCP is used for the incast workload, while Cubic and DCTCP serve web-search traffic. Figure 4.8 (a,b,c) shows the 99th percentile FCT slowdown of short flows belonging to each priority as a function of the load carried by Cubic.

At a high level, Figure 4.8 shows the inability of separated queues and DT to offer isolation. As the Cubic load increases, the default (i.e., DT) FCT slowdown performance of DCTCP, and θ-PowerTCP significantly increases. In contrast, ABM effectively protects θ-PowerTCP and DCTCP from the Cubic load and reduces the tail FCT slowdown by up to 90.92%. Concretely, we observe that when ABM manages the buffer the FCT slowdown of both Cubic and DCTCP stabilizes around 20 (i.e., stops degrading with increasing Cubic load) and θ-PowerTCP



(a) Cubic flows

(b) DCTCP flows

(c) θ-PowerTCP flows

(d) Overall buffer occupancy

Figure 4.8: Cubic, DCTCP and θ-PowerTCP harm each other even though they use different queues with default BM (i.e., DT). ABM isolates each congestion control algorithm, effectively reducing their tail FCT.

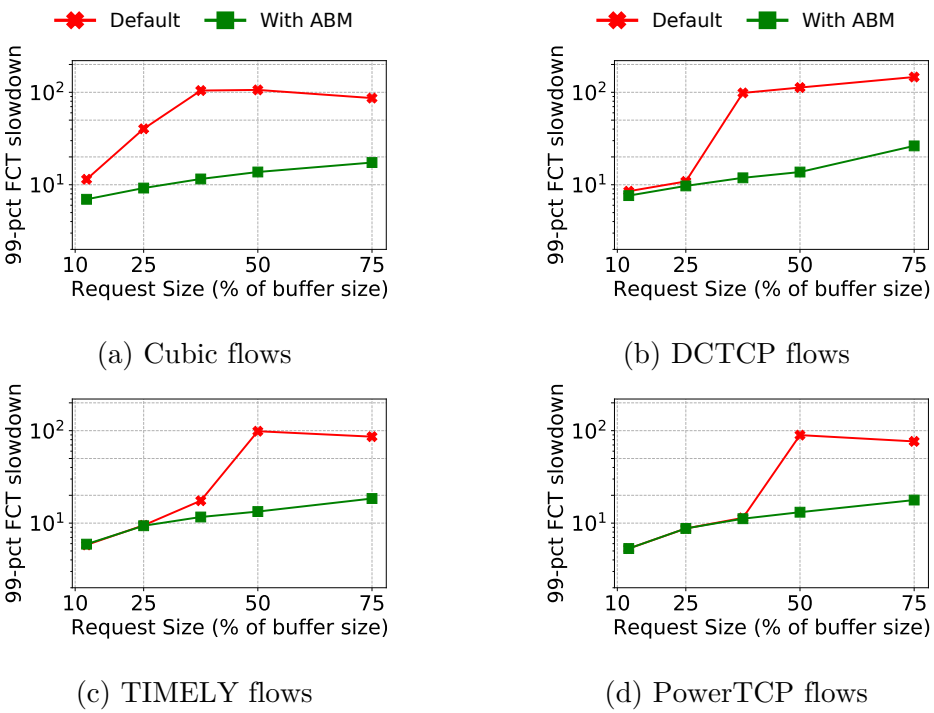
stabilizes at 10 despite the increasing Cubic load.

In essence, as ABM bounds the buffer drain time (Theorem 7) and the buffer occupancy of each traffic priority (Theorem 6), it reduces the impact of Cubic traffic in other priorities of the shared buffer.

ABM improves short flows FCTs (even if they are not part of a burst): ABM not only improves incast workload FCTs, but also improves FCTs of all short flows. In Figure 4.6b, we show the 99th percentile FCT slowdown of short flows belonging to the web-search traffic as a function of the load. We observe that even at 20% load, ABM reduces the FCT slowdown by 30.7% compared to DT and FAB; by 38.5% compared to CS; and by 28.9% compared to IB. At 80% load, ABM reduces the FCT slowdown for short flows by 76% on average. Similarly, Figure 4.7b we show the 99th percentile FCT slowdown of short flows belonging to the web-search traffic as a function of request size of the incast workload and at a fixed load of 40% from the web-search workload. We observe that ABM reduces the FCT slowdown for short flows of web-search workload by 41.8% on average.

ABM does not sacrifice throughput to serve short flows or bursts: We further evaluate the performance of ABM in terms of throughput. In Figure 4.6d, we observe that ABM achieves on-par throughput compared to alternative approaches. Even under large request sizes of the incast workload in Figure 4.7d, ABM does not sacrifice throughput for low FCTs of short flows.

ABM strategically increases buffer utilization to accommodate incast: ABM effectively limits the buffer occupied by medium and long flows. Even with a buffer-hungry transport algorithm such as Cubic, shown in Figure 4.6c, ABM reduces the buffer usage across various loads by 54.2% on average compared to DT, FAB and IB; and by 68.9% compared to CS while still achieving on-par throughput. At the same time, ABM strategically uses more buffer to accommodate bursts.



(a) Cubic flows

(b) DCTCP flows

(c) TIMELY flows

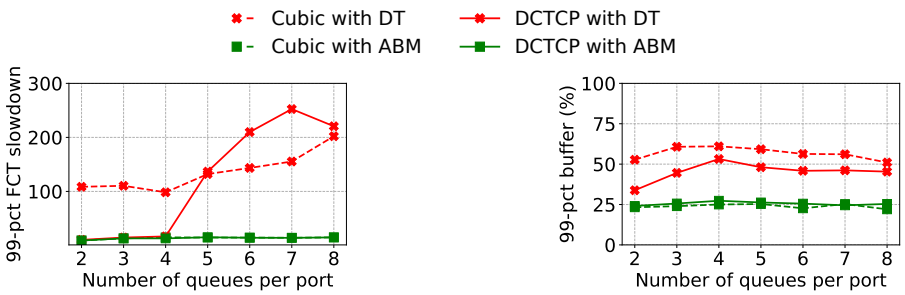
(d) PowerTCP flows

Figure 4.9: While advanced congestion control already offers low tail FCT, ABM allows for further improvements through better buffer allocations compared to the default buffer management (i.e., DT).

Figure 4.7c shows that ABM uses more buffer as the request size (incast workload) increases. Indeed, ABM uses 5.3% more buffer compared to DT, FAB and IB for large request sizes.

ABM's benefits increase as the number of queues-per-port increases under stable load: To evaluate the impact of μ_p^i factor on ABM's thresholds, we generate *(i)* web-search workload at 40% load; and *(ii)* incast with request size at 25% of the buffer size and vary the number of queues per port across which the load is distributed. Figure 4.10a shows the tail FCT slowdown of Cubic and DCTCP when combined with ABM and DT. We observe that as the number of queues per port increases, ABM shows higher benefits even with stable load conditions. Concretely, ABM and DT perform similarly to DCTCP until 4 queues per port, where ABM only improves over DT by 10%. At 5 queues per port, ABM reduces the 99th percentile FCT slowdown of Cubic and DCTCP by 88.9% on average. Furthermore, at 8 queues per port, ABM reduces the FCT slowdown of Cubic and DCTCP by 92.6%. Figure 4.10b shows how even DCTCP uses more buffer as the number of queues per port increases, even when the load is fixed. Figure 4.10 validates the importance of bounded drain time in § 4.1.1.

ABM can improve FCTs of advanced congestion control algorithms under bursty workloads: Aiming at understanding the benefit of ABM in conjunction with advanced congestion control, we evaluated it under the following four scenarios. In each scenario one congestion control algorithm from Cubic, DCTCP, TIMELY or PowerTCP runs at the hosts generating web-search at 40% load and incast traffic at varying request sizes. We then compare the tail FCT slowdown achieved when ABM manages the buffer compared to when the default buffer management algorithm does. In Figure 4.9, we observe that for small-sized request at 12.5% of the buffer size, ABM reduces FCT slowdown by



(a) (b)

Figure 4.10: As the number of queues-per-port increases under stable load ABM's buffer occupancy remains low (b), similarly to its tail FCT (a).

39.1% compared to Cubic (with DT) and by 10.3% compared to DCTCP For this request size ABM achieves no improvements compared to TIMELY and PowerTCP. For medium-sized requests at 37.5% of the buffer size, ABM reduces the FCT slowdown by 88% compared to Cubic and DCTCP; reduces TIMELY's FCT slowdown by 33.3%. For this request size ABM achieves no improvements compared to PowerTCP. As the request size increases further, at 50% of the buffer size, ABM also reduces the 99th percentile FCT slowdown of PowerTCP by 76%.

Although advanced congestion control can improve the tail buffer occupancy and consequently the tail FCTs for short flows, we observe from Figure 4.9 that ABM can improve FCTs of advanced congestion control algorithms under bursty workloads.

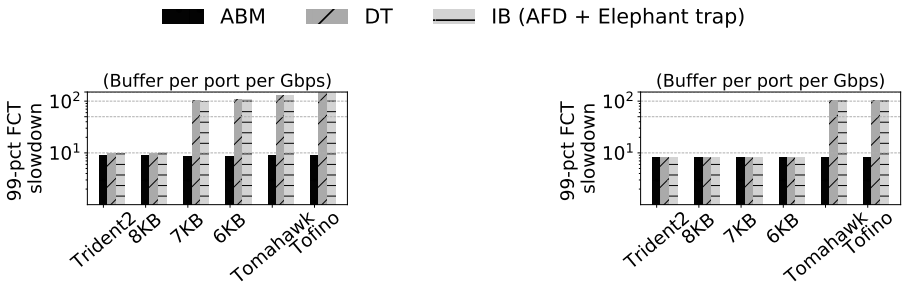
4.4.3 ABM's Performance in Shallow Buffers

Our evaluation setup so far considers a switch with 9.6KB buffer-per-port-per-Gbps that corresponds to a Trident2 switch. We further evaluate the benefits of ABM with smaller, yet realistic buffer sizes e.g., Tomahawk.

To that end, we vary the buffer-per-port-per-Gbps according to the specifications reported in [59]. We generate web-search workload at 40% load and incast workload at a request size at 25% of the buffer size corresponding to Trident2. We separately consider DCTCP and PowerTCP.

Figure 4.11, summarizes our results by showing the 99th percentile FCT slowdown of the incast flows for DCTCP (a) and PowerTCP (b). We observe that ABM's performance is robust to changes in buffer size even for extremely shallow buffers. Concretely, ABM achieves similar performance across various buffer sizes both with DCTCP and PowerTCP.

On the contrary, both DT and IB cannot effectively manage the buffer when its size is equal or smaller than 7KB per-port-per-Gbps. Indeed, at this buffer size the FCT slowdown of DT and IB with DCTCP increases by $\approx 10x$ compared to ABM with DCTCP as we observe in Figure 4.11a. PowerTCP hides DT's and IB's inability until 6KB per-port-per-Gbps but not further. Concretely, in Figure 4.11b, we observe that DT and IB cannot sustain low tail FCTs with 5.12KB per-port-per-Gbps (corresponding to Tomahawk). In summary, ABM effectively manages the buffer under bursty workloads even with a shallow buffer of 3.44KB (corresponding to Tofino). For instance, ABM reduces the 99th percentile FCT slowdown by 96% with DCTCP and by 92% with PowerTCP compared to alternatives i.e., DT and IB.



(a) DCTCP (b) PowerTCP

Figure 4.11: ABM maintains low tail FCT under various realistic buffer sizes that are smaller than Trident2 (which we use throughout § 4.4.2). On the contrary, DT and IB cause 10x tail FCT slowdown for multiple realistic buffer sizes.

4.4.4 ABM's Performance with Periodic & Infrequent α Updates

In § 4.2.4, we introduced the possibility of approximating ABM on top of DT via periodically reconfiguring α at the control plane. We now aim at evaluating how sensitive ABM is to the update interval and what benefits such a practical approximation of ABM can have.

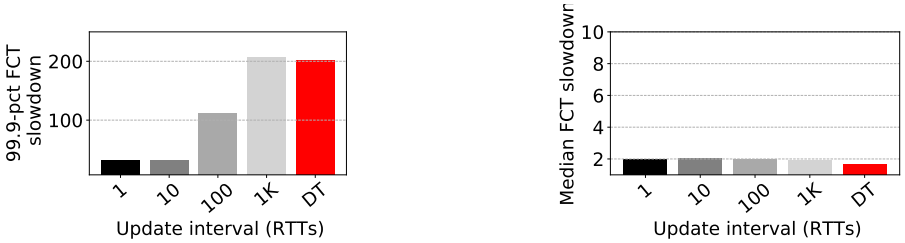
We generate traffic using web-search workload at 40% load and incast traffic with 75% request size. Flows are destined to one of the eight queues at each port chosen uniformly at random. We vary the update interval after which DT's α values are reconfigured to approximate ABM's allocation. We chose to have more queues per port compared to the previous experiment and a relatively large request size to make the scenario more challenging for ABM. Finally, in Figure 4.12a, we plot the 99.9th[5] percentile FCT slowdown for short flows and the median FCT slowdown for long flows for *(i)* ABM with varying update intervals; and *(ii)* DT (red bar).

At a high level, the 99.9th percentile FCT slowdown for short flows increases with larger update intervals, as the reconfiguration rate cannot keep up with the updates in n_p and μ_p^i used in ABM's thresholds. Thus, at around $1K$x RTT update interval, ABM's approximation is equivalent to DT for short flows FCT. Still, ABM's approximation significantly improves DT's performance for update intervals smaller than $1K$x RTT (i.e., 80ms) showing ABM's practical benefits. For instance, even with 10x RTT (100x RTT) update interval, the tail FCT slowdown of ABM decreases by 84.05% (44.78%) compared to DT. Importantly, the tail FCT of long flows is not affected by the intervals. Thus, the achieved throughput of ABM's approximation is on par with that of DT.

4.5 Related Work

Optimally managing switch buffers has been an active area of research for more than two decades with a wide range of approaches, including BMs [46, 49, 56, 64, 86, 97, 161, 231] and AQMs [102, 139, 215, 267], scheduling [37, 130, 242] and end-host congestion control [21, 35, 173, 193, 199, 202, 259].

[5]We report 99.9th rather than 99th to stress ABM further.



(a) Short flows FCT

(b) Long flows FCT

Figure 4.12: DT can approximate ABM's allocation if we reconfigure its α parameter every 100x RTT (i.e., every 8ms) according to ABM's formula. Such a coarse update interval will already improve DT's tail FCT by 44.78% compared to DT (red).

BM schemes such as FAB [46], Cisco's IB [5], TDT [133] and EDT [239] rely on DT [86] and attempt to give more of the remaining buffer to short flows. By relying on DT, these schemes inherit its pitfalls (§ 4.1.3).

AQM schemes such as RED [102], ARED [103], Codel [139] and PIE [215] control queue lengths or delays but under unrealistic assumptions. Indeed, AQM schemes assume per-queue buffer isolation i.e., that the maximum length per queue is static. Yet, in practice, this length dynamically changes, often depriving AQM from the required buffer to operate normally.

End-host congestion control algorithms have the potential to reduce the buffer requirements, but are orthogonal to our work. Control algorithms such as DCTCP [35], DCQCN [287] use marking schemes as feedback to adjust the sending rates. TIMELY [193] uses RTT-gradient approach to rapidly react to congestion onset. Even more advances algorithms use a variety of feedback signals e.g., HPCC [173] uses inflight bytes, Swift [164] uses delay and PowerTCP [21] uses the power. Yet, end-host congestion control algorithms cannot act on the first RTT packets, and are fundamentally unable to orchestrate the buffer-sharing at all times.

4.6 Summary

In this chapter, we demonstrate the fundamental inability of Buffer Management and Active Queue Management schemes to address the challenges that occur from sharing the on-chip buffer across queues. Furthermore, we analytically show the limitations of the state-of-the-art buffer sharing scheme. We present ABM, a novel buffer-sharing scheme that offers isolation, bounded drain time, and high burst tolerance. ABM is practical in that it only uses statistics that are available to the MMU. We show that ABM outperforms all other buffer management schemes in tail FCT while achieving on-par throughput. Importantly, ABM improves the FCTs of advanced congestion control algorithms (i.e., TIMELY, DCTCP, and PowerTCP) under bursty workloads.

5

Buffer Sharing with Lossy & Lossless Traffic

Network devices contain a buffer that can temporarily store excessive packets during congestion events. As the link speeds increase, maintaining a constant buffer-bandwidth ratio would require buffer memory to evolve faster than Moore's law and is hence impractical [185]. As a result, we observe buffer-per-Gbps to constantly shrink [59, 146] making performance problems rooted in buffer sharing more evident. Indeed, our expert survey, which included experts from six companies, revealed that buffer sharing is causing performance problems in most large-scale datacenters.

At a high level, the goal of a buffer-sharing scheme is to provide isolation between traffic classes, while maximizing the benefit of the buffer e.g., by absorbing bursts and achieving high throughput. Existing buffer management schemes (even recent ones) [16, 46, 86, 133] were designed considering exclusively loss-tolerant traffic (e.g., TCP variants). However, modern datacenters host traffic classes with different loss tolerance. Concretely, along with traditional loss-tolerant transport protocols, many clouds, e.g., Azure [58], Alibaba [105] and OCI [212], deploy RDMA over Converged Ethernet which requires lossless transport. In order to guarantee zero packet loss for RDMA, production datacenters enable Priority Flow Control (PFC) at the switches [58].

The co-existence of TCP and RDMA traffic in the switch buffer makes sharing the buffer particularly challenging. While, in principle, TCP and RDMA traffic have the same performance objectives (e.g., high throughput, low latency), their reaction to network events such as congestion is vastly different in terms of speed and granularity. A PFC pause proactively throttles RDMA traffic at *per-hop* granularity *before* the buffer fills up in order to prevent packet loss due to congestion. On the contrary, a packet drop throttles TCP at *per-flow* granularity once the buffer is filled up due to congestion. Moreover, the effect of PFC pause (in RDMA) on the buffer is not immediately evident as all incoming packets after the PFC has been triggered must be admitted in the buffer further increasing its occupancy. On the contrary, the effect of a packet drop (in TCP) on the buffer is immediately evident in the buffer as current packets do not further increase the buffer occupancy which can decrease proportionately to the aggregate port bandwidth. Since RDMA and TCP share the same switch buffer, congestion caused by TCP can result in excessive PFC pauses for RDMA and similarly the buffer occupied by RDMA (especially when it is paused) can result in excessive packet drops for TCP; leading to throughput degradation and poor burst absorption.

A naive approach for isolating RDMA and TCP in the shared buffer is to statically partition it e.g., dedicate 50% of the buffer to each class. However, such

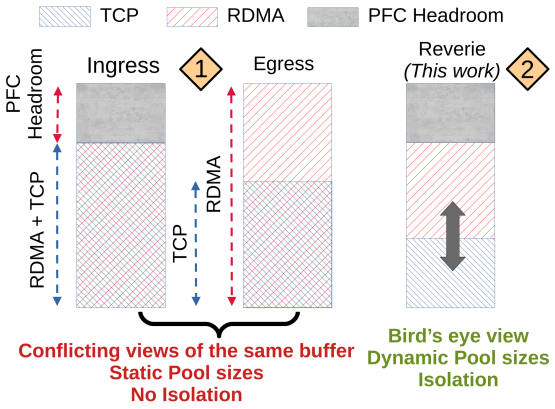


Figure 5.1: ① Current buffer-sharing practices maintain two independent and at times conflicting views of the buffer i.e., ingress and egress, which are subdivided into various pools; posing a challenge to achieve isolation across RDMA and TCP traffic. ② REVERIE maintains a bird's eye view of the buffer; effectively unifying ingress and egress admission control *without statically partitioning the buffer* to achieve isolation.

an approach will result in suboptimal burst absorption; and (in the worst case) poor throughput when one of the two classes is not using its dedicated portion of the buffer. On the one hand, production-grade buffer-sharing schemes do not significantly depart from static partitioning among TCP and RDMA. The root cause of this pitfall is the unnecessary complex buffer model together with the use of pre-configured buffer pools i.e., pieces of buffer dedicated to certain queues only. On the other hand, research-grade buffer-sharing schemes such as ABM [16] can — at best — achieve steady-state isolation across traffic priorities only for loss-tolerant traffic, but would fail to isolate lossless and lossy traffic, even if we extend them to work in such settings, as we show in §5.1.3. Our goal in this works is to formally navigate the trade-off between isolation and burst absorption in a setting where lossless and lossy traffic co-exist. Two key insights allow us to do so.

Our first key insight –after thoroughly studying the current buffer-sharing practices– is that although lossless and lossy traffic are, in practice, independently managed, the available buffer for both depends on each other's occupancy. Concretely, today's switches maintain two views of the buffer (i.e., ingress and egress); each of these views is virtually further split into pools i.e., buffer pieces configured to serve a subset of the queues. Figure 5.1 summarizes these views. The complexity of the buffer design stems from its evolution over the years from serving lossy traffic to serving both lossy and lossless. Unfortunately, as we show in this chapter, this complex buffer design of today's datacenter switches leads to unexpected buffer issues e.g., lossy traffic gets more buffer allocation than lossless traffic when they compete, against the high-level objective of the configuration. Our analysis shows that buffer pools and the *independent* views of the buffer at the ingress and egress are the root causes of such issues. To tackle this problem, we propose a simple buffer-sharing scheme in which both RDMA and TCP are managed *jointly* with a bird's eye view of the buffer. Such an allocation scheme facilitates novel admission control schemes that can efficiently isolate RDMA and TCP without statically partitioning the buffer.

Our second key insight is that absorbing RDMA bursts is extremely challenging because the decisions of a buffer-sharing scheme are on a per-packet basis but PFC

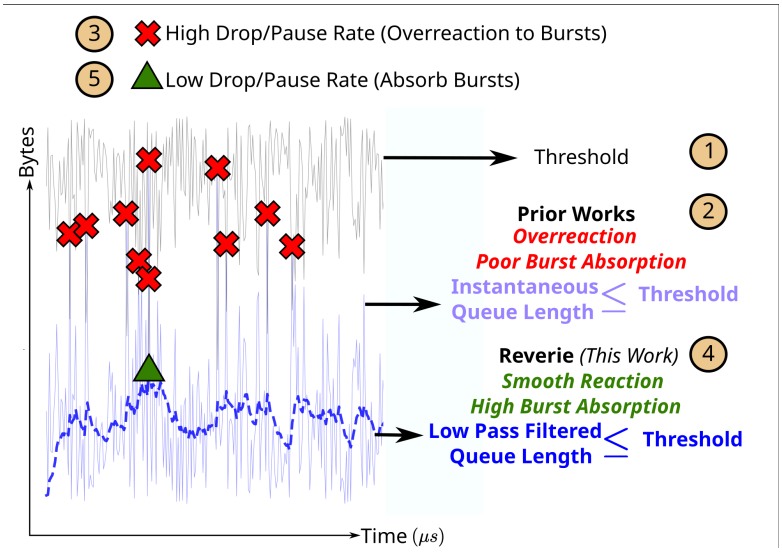


Figure 5.2: Prior works calculate (1) thresholds and compare against (2) instantaneous queue lengths, which leads to (3) overreaction to bursts and a high loss rate. REVERIE takes a different approach and compares the thresholds against (4) low pass filtered queue lengths. This allows REVERIE to (5) smoothly react to congestion while absorbing transient bursts.

pause (required for RDMA) is a per-hop signal affecting not just the incoming packet but all the future arrivals from the previous hop. Moreover, sudden and concurrent fluctuations in multiple queue lengths can rapidly change the buffer occupancy. Worse yet, the incoming rate at today's link speeds can very rapidly fill in a buffer. Existing schemes [16, 46, 133] that were designed specifically for loss-tolerant traffic apply large thresholds to those packets that are classified as short flows or incast flows. Unfortunately, as we later show in this chapter, these techniques cannot fundamentally achieve better burst absorption for lossless traffic since PFC works at *per-hop* or per-queue granularity and not per-packet i.e., burst absorption for RDMA requires prioritizing a queue experiencing burst (not just specific packets). To address this problem, we show that instead of increasing thresholds at per-packet granularity under bursty scenarios, it is sufficient to dampen the queue statistic (e.g., by a low-pass filter) against which thresholds are compared, when taking buffer decisions. Figure 5.2 illustrates our main idea. This essentially prioritizes queues experiencing bursts and improves the burst absorption capabilities of the buffer.

We present REVERIE, a buffer-sharing scheme suitable for modern datacenters hosting traffic with different loss tolerance. REVERIE jointly optimizes the buffer allocation for lossless and lossy traffic with a bird's eye view over the buffer; essentially unifying ingress and egress admission control as shown in Figure 5.1. Further, REVERIE significantly improves the burst absorption capabilities of the buffer by comparing low-pass-filtered queue lengths against thresholds as illustrated in Figure 5.2.

Our extensive evaluation of REVERIE based on large-scale simulations in NS3 [209] shows that REVERIE effectively isolates RDMA and TCP, reduces the overall number of PFC pauses by 60% on average and improves the flow completion times for bursty workloads by up to 33% compared to the state-of-the-art approaches.

In summary, our key contributions in this work are:

- The first analysis of a production-grade buffer model (exemplified by the open source SONiC [251]) that includes both ingress and egress admission control. This analysis generates multiple insights including the conflicting buffer views of ingress and egress that prevent effective isolation between lossless and lossy traffic.
- REVERIE, the first buffer-sharing scheme that can isolate lossless and lossy traffic while improving burst absorption for both.
- As a contribution to the research community and to facilitate future work, all our artifacts have been made publicly available at `https://github.com/inet-tub/ns3-datacenter`.

5.1 Motivation

We first present unintuitive outcomes (issues) that arise under typical configurations (§5.1.1). To explain the root cause of those issues (§5.1.3), we first describe in detail a representative buffer-sharing architecture of an open-source and widely-used switch OS, i.e., SONiC [251] (§5.1.2). Our buffer model has been endorsed by major Ethernet switch ASIC vendors, including Broadcom, Cisco and NVIDIA.

5.1.1 Buffer Issues in Datacenters

In this section, we walk through three issues that operators of large-scale RDMA deployments [58] can face while debugging buffer problems. We have verified that these issues are *(i)* possible by showing them analytically (§5.1.3); and *(ii)* realistic by direct communications with operators of large-scale RDMA deployments. Consider an operator who wants lossless traffic to get as much buffer as it needs, i.e., lossless is prioritized over lossy traffic under buffer contention. This is a typical use case in datacenters with large RDMA deployments. Although the operator closely follows the "best practices" (i.e., a set of heuristics) to configure the buffer, which we explain in §5.1.2.2, they observe the following issues.

Issue 1. *Lossy traffic gets more buffer allocation than lossless traffic when they both compete for buffer space.*

Issue 2. *Lossless traffic yields to the increase in buffer occupancy of lossy traffic, while the opposite is not true, i.e., the allocation for lossy traffic is not affected by the buffer occupancy of lossless traffic.*

Issue 3. *The buffer is more efficient in absorbing bursts of lossy traffic than bursts of lossless traffic.*

5.1.2 Buffer Sharing Practices

To understand the root cause of the issues (§5.1.3), we need to understand the buffer model used in today's datacenters and the "best practices" for configuring it. To the best of our knowledge, we are the first to present a detailed and

up-to-date description of a buffer model of a datacenter switch that serves both lossless (e.g., RDMA) and lossy (e.g., TCP) traffic. We use SONiC [251], an open-source network operating system that is the closest we can get to the modus operandi for buffer management. SONiC runs on switch ASICs from multiple vendors, e.g., Broadcom, NVIDIA, Cisco and Intel, and has been widely deployed in Microsoft [58], Alibaba [262], LinkedIn [276] and Tencent. Importantly, our buffer model aligns with that of NVIDIA Onyx [255]. Hence, we believe our buffer model is representative of a broad range of scenarios and settings. We next describe the terminology and the configurable parts according to the buffer model of SONiC. We tabulate the important notations we use in Table 5.1.

Hereafter, we denote lossless by $\bullet$ and lossy by $\circ$. The switch uses a memory management unit (MMU) to manage the packet buffer.

Ingress and Egress Counters (Queues): The MMU maintains two types of *counters*[1], ingress denoted by $\leftarrow$ and egress denoted by $\rightarrow$ that serve admission control purposes. We henceforth refer to these counters as *queues*. Let $\mathcal{Q}$ be the set of all queues maintained by the MMU. Once a packet arrives at the switch, the packet is mapped to an ingress queue $(s, p) \in \overleftarrow{\mathcal{Q}}$ based on the source port s and the packet's priority p[2]; and an egress queue $(d, p) \in \overrightarrow{\mathcal{Q}}$ based on the destination port d. Ingress (egress) admission control acts over ingress (egress) queues. Each packet is admitted to the buffer if and only if the corresponding ingress and egress queues pass the ingress and egress admission controls. An arriving and admitted packet increases both the corresponding ingress and egress queues, while a departure packet decreases the queues. A packet is only buffered once regardless of the number of counters it is accounted by. Note that once a packet is admitted, it cannot be *pushed out* by new packet arrivals. A queue carrying lossless (lossy) traffic is known as a lossless (lossy) queue. Overall, the MMU maintains four sets of queues i.e., ingress lossless $\overleftarrow{\mathcal{Q}}^{\bullet}$, ingress lossy $\overleftarrow{\mathcal{Q}}^{\circ}$, egress lossless $^{\bullet}\overrightarrow{\mathcal{Q}}$ and egress lossy $^{\circ}\overrightarrow{\mathcal{Q}}$.

Buffer Size and Pools: The packet buffer has a total size of b. Current datacenter practices define pools that can intuitively be viewed as the buffer available for certain types of queues. In other words, the pool is a group of queues. The user can configure the buffer allocation policy, including the allocation algorithm, per-queue limit, and total size, for this group of queues. SONiC defines the following four pools:

- Ingress pool of size $\overleftarrow{b}$ shared by *both* ingress lossless and lossy queues, with an occupancy of $\overleftarrow{q}(t)$ at time t.
- PFC headroom pool of size b_h used *only* by ingress lossless queues upon PFC pause (described next).
- Egress lossless pool of size $^{\bullet}\overrightarrow{b}$ used by egress lossless queues, with an occupancy of $^{\bullet}\overrightarrow{q}(t)$ at time t.
- Egress lossy pool of size $^{\circ}\overrightarrow{b}$ used by egress lossy queues, with an occupancy of $^{\circ}\overrightarrow{q}(t)$ at time t.

[1]Not to be confused with ingress and egress pipelines. Throughout this chapter, ingress and egress are merely counters (referred to as queues).

[2]Most of the switch ASICs support 8 priorities. Operators typically map a packet to a priority based on its DSCP value.

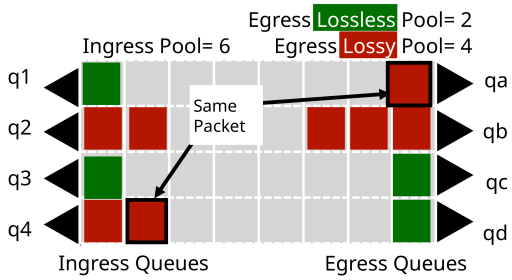


Figure 5.3: Buffer bookkeeping: packets are stored once but accounted twice; once in the ingress and once at the egress for admission control purposes.

Note that pool sizes and occupancy are also *counters*. A packet can be counted in multiple pools while being buffered once (pools may overlap), thus the sum of all pools may exceed the actual buffer occupancy.

Figure 5.3 illustrates an example of the buffer bookkeeping. Packets are physically stored only once, but are accounted twice. For example, packet 5 is accounted in the ingress queue (counter) $q4$ and in the egress queue (counter) qa. In essence, all packets are accounted in the ingress pool, but packets of lossless queues are also accounted in the egress lossless pool, while packets of lossy queues are also accounted in the egress lossy pool.

Admission Control: Each queue i.e., counter $(i, p) \in \mathcal{Q}$ at an input or output port denoted by i, corresponding to priority p is associated with a *threshold* $\Gamma_i^p(t)$ at time t. The admission control scheme compares the instantaneous length $q_p^i(t)$ of a queue against its threshold to make buffering decisions. Thresholds can be intuitively viewed as the maximum size of a queue. Once the queue hits the threshold, the switch will drop the incoming packet or send PFC pause frames to throttle the queue build-up. We emphasize that the switch cannot push out existing packets in the buffer to make room for the incoming packet.

5.1.2.1 Journey of a Packet in the Switch MMU

We walk through the various counters that are incremented and decremented during a packet's journey in the MMU. Recall that a packet can travel through the switch if and only if it satisfies both ingress and egress admission control.

Ingress Admission Control: The admission control in the ingress is different for lossy and lossless queues since as TCP (lossy) tolerates packet loss whereas RDMA (lossless) requires PFC and does not tolerate packet loss. The admission for ingress lossy queues is straightforward. If the ingress lossy queue hits the threshold (meaning if the length of the queue equals or exceeds the corresponding threshold devised by the buffer-sharing logic), the packet is dropped. Otherwise, both the ingress lossy queue and ingress pool counters are incremented upon admission and decremented as the packet departs to its destination.

In contrast, the admission control for ingress lossless queues is more complex and designed to achieve zero packet loss. If the ingress queue hits the threshold, the switch moves the queue to ‘‘paused’’ state and keeps sending PFC pause frames to the peer device. Then the arriving packet is admitted, but it increments the PFC headroom pool occupancy rather than the ingress pool occupancy. In

Notation	Description
$\leftarrow$	Ingress
$\rightarrow$	Egress
$\bullet$	Lossless
$\circ$	Lossy
$*$	Shared
b	Total buffer size
b_h	Headroom pool size
$\mathcal{Q}$	Set of all queues
$\overleftarrow{\mathcal{Q}}^{\bullet}$	Set of ingress lossless queues
(i,p)	Queue at input or output port i, priority p
$q_i^p(t)$	Length of queue (i,p) at time t
$\Gamma_i^p(t)$	Threshold of queue (i,p) at time t
$^{\circ}\overrightarrow{q}(t)$	Occupancy of egress lossy pool
α_i^p	Parameter for queue (i,p)
$\overleftarrow{\alpha}^{\bullet}$	α for ingress lossless queues (for simplicity)
$^{\circ}\overrightarrow{\alpha}$	α for egress lossy queues (for simplicity)
$\overleftarrow{n}^{\bullet}$	# ingress lossless queues using buffer
$^{\circ}\overrightarrow{n}$	# egress lossy queues using buffer

Table 5.1: Important notations used in this chapter.

other words, once an ingress lossless queue uses up its limit in the ingress pool, it starts to consume (or be accounted in) the PFC headroom pool. As the buffer drains, an ingress lossless queue under "paused" state first decrements its headroom pool occupancy, and then its ingress pool occupancy. When the headroom buffer occupancy is zero and the ingress pool occupancy is below the threshold, the switch moves the "paused" ingress lossless queue back to "resumed" state and sends PFC resume frames.

Egress Admission Control: Egress counters are straightforward. Egress queue length and pool occupancy based on the class of packet (lossy or lossless) are incremented upon admission and decremented as the buffer drains. The switch drops packets if egress queues hit thresholds.

5.1.2.2 Buffer Management

The MMU of the switch uses a buffer management algorithm that assigns thresholds to all ingress and egress queues. Dynamic Thresholds [86] (DT) is the state-of-the-art buffer management algorithm widely adopted by ASIC vendors [211, 252].

In a nutshell, DT calculates *dynamic* thresholds for each queue $(i,p) \in \mathcal{Q}$ as the product of a configurable parameter α_p^i and the *remaining* buffer space in the corresponding pool. We refer the reader to Table 5.1 for the list of notations we use. In the following, we summarize DT's buffer management for lossless and lossy traffic, at ingress and egress queues.

$$\Gamma_p^i(t) = \alpha_p^i \times \begin{cases} \overleftarrow{b} - \overleftarrow{q}(t) & \textit{Ingress Lossless: } (i,p) \in \overleftarrow{\mathcal{Q}}^{\bullet} \\ \overleftarrow{b} - \overleftarrow{q}(t) & \textit{Ingress Lossy: } (i,p) \in \overleftarrow{\mathcal{Q}}^{\circ} \\ \overrightarrow{b}^{\bullet} - \overrightarrow{q}^{\bullet}(t) & \textit{Egress Lossless: } (i,p) \in \overrightarrow{\mathcal{Q}}^{\bullet} \\ \overrightarrow{b}^{\circ} - \overrightarrow{q}^{\circ}(t) & \textit{Egress Lossy: } (i,p) \in \overrightarrow{\mathcal{Q}}^{\circ} \end{cases} \tag{5.1}$$

In addition to the above threshold checks, the switch also uses the physical packet buffer limit as the last defense.

Having a better understanding of how the buffer works, we can go back to the operator's goal to prioritize lossless over lossy and explain how they would in practice configure the buffer. To avoid lossless packet drops, they would want to control lossless traffic *only* at the ingress with PFC thresholds. To make the problem easier to debug and more intuitive, they would want to control lossy traffic *only* at the egress with drop thresholds. To this end, they would use the following buffer configuration heuristics. For simplicity, in this section, we set α_p^i to $\overleftarrow{\alpha}^{\bullet}$, $\overleftarrow{\alpha}^{\circ}$, $^{\bullet}\overrightarrow{\alpha}$ and $^{\circ}\overrightarrow{\alpha}$ for all ingress lossless, ingress lossy, egress lossless and egress lossy queues respectively.

Heuristic 1. *To avoid packet drops for lossless traffic at ingress, the sum of the ingress pool size* $\overleftarrow{b}$ *and headroom pool size* b_h *should be equal to (or smaller than) the total size of the switch buffer* b *i.e.,* $\overleftarrow{b} + b_h \leq b$.

Heuristic 2. *To bypass egress admission control for lossless traffic and to allow fully utilizing the buffer space i.e., to avoid lossless packet drops at egress, we should set egress lossless pool size* $^{\bullet}\overrightarrow{b}$ *to total switch buffer size* b *and use an infinitely large egress lossless threshold i.e.,* $^{\bullet}\overrightarrow{\alpha} \gg 1$.

Heuristic 3. *To avoid packet drops for lossy traffic at ingress, we should set ingress lossy threshold* $\overleftarrow{\alpha}^{\circ}$ *to an infinitely large value, and ensure that egress lossy pool size* $^{\circ}\overrightarrow{b}$ *is not larger than ingress pool size* $\overleftarrow{b}$ *i.e.,* $^{\circ}\overrightarrow{b} \leq \overleftarrow{b}$.

5.1.3 Root Causes of the Buffer Issues

To systematically analyze the problems, we consider a fluid flow model with deterministic packet arrivals and analyze the steady state[3] of the buffer, similar to prior works [16, 86]. In this section, we formally model the buffer sharing scheme of SONiC [251] and analyze its buffer allocation properties. Our analysis aims to understand the steady-state behavior of the buffer. Specifically, we analyze a shared buffer switch architecture whose buffer sharing scheme supports both lossless traffic and lossy traffic. Our analysis is based on a fluid flow model with deterministic packet arrival rates, extending [16, 86].

We begin by analyzing the drop thresholds for lossy traffic. As we study the steady-state of the buffer, we drop the time variables in our notation for ease of presentation. Since every packet is accounted both in the ingress and egress, the following relations hold at all times: *(i)* lossy traffic buffer occupancy $^{\circ}\overrightarrow{q}$ at egress equals its occupancy $\overleftarrow{q}^{\circ}$ at ingress; *(ii)* lossless traffic buffer occupancy at egress $^{\bullet}\overrightarrow{q}$ equals its occupancy at the ingress pool $\overleftarrow{q}^{\bullet}$ plus headroom occupancy q_h; *(iii)* ingress pool occupancy $\overleftarrow{q}$ equals the sum of occupancy of lossless $\overleftarrow{q}^{\bullet}$ (without PFC headroom) and lossy $\overleftarrow{q}^{\circ}$ traffic occupancy at ingress.

$$^{\circ}\overrightarrow{q} = \overleftarrow{q}^{\circ} \tag{5.2}$$

$$^{\bullet}\overrightarrow{q} = \overleftarrow{q}^{\bullet} + q_h \tag{5.3}$$

$$\overleftarrow{q} = \overleftarrow{q}^{\circ} + \overleftarrow{q}^{\bullet} \tag{5.4}$$

[3] A steady state is achieved when the queue lengths stabilize i.e., packet arrival rate equals departure rate.

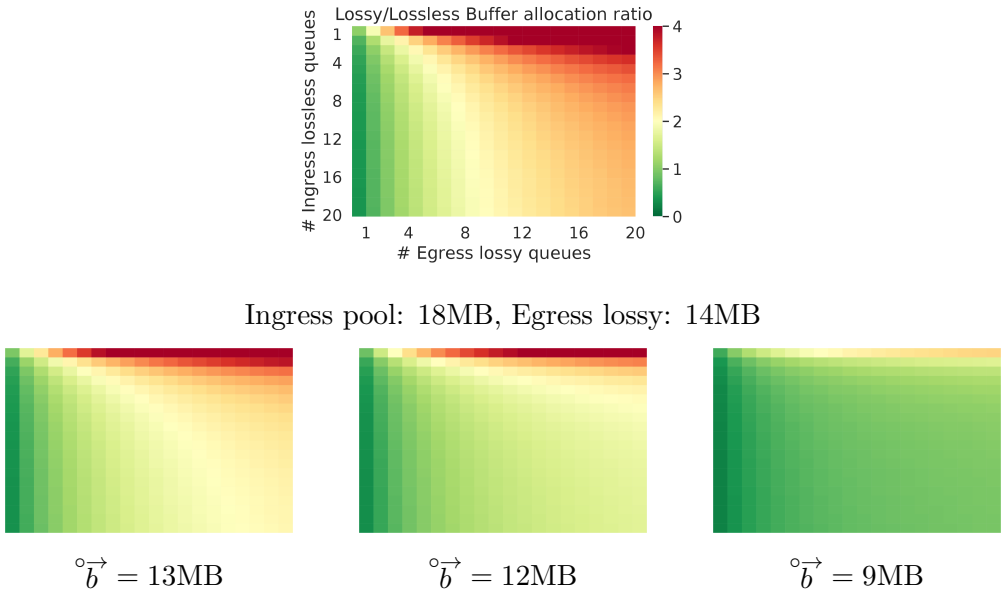


Figure 5.4: Isolation cannot be achieved with SONiC unless the ingress pool size $\overleftarrow{b}$ = 18MB is at least twice as large as egress lossy pool size $^{\circ}\overrightarrow{b}$ i.e., statically partitioning the buffer.

Based on the egress admission control for lossy traffic i.e., Dynamic Thresholds (see Equation 5.1 in §5.1), if $^{\circ}\overrightarrow{n}$ egress lossy queues are in the steady state, then the total buffer occupancy is $^{\circ}\overrightarrow{q} = {}^{\circ}\overrightarrow{n} \cdot (^{\circ}\overrightarrow{b} - {}^{\circ}\overrightarrow{q})$. By rearranging the terms, we obtain the egress lossy pool occupancy:

$$^{\circ}\overrightarrow{q} = \frac{^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha} \cdot {}^{\circ}\overrightarrow{b}}{1 + {}^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha}} \tag{5.5}$$

Similarly, based on the ingress admission control for lossless traffic, if $\overleftarrow{n}^{\bullet}$ lossless queues are in the steady state, then the total buffer occupancy is $\overleftarrow{q}^{\bullet} = \overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet} \cdot (\overleftarrow{b}^{\bullet} - \overleftarrow{q}^{\bullet})$. Substituting in Equation 5.4 and using Equation 5.5 for $^{\circ}\overrightarrow{q}$, we obtain the following:

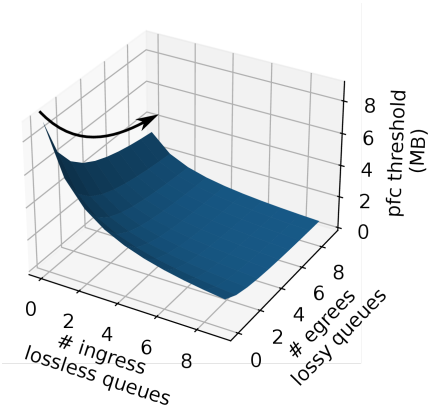
$$\overleftarrow{q} = \overleftarrow{b} \cdot \left(\frac{\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}}{1 + \overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}} \right) + {}^{\circ}\overrightarrow{b} \cdot \left(\frac{1}{1 + \overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}} \cdot \frac{^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha}}{1 + {}^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha}} \right)$$

Finally, substituting $\overleftarrow{q}$ from above in $\overleftarrow{q}^{\bullet} = \overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet} \cdot (\overleftarrow{b} - \overleftarrow{q})$, we obtain $\overleftarrow{q}^{\bullet}$, the buffer occupied by lossless traffic at the ingress pool.

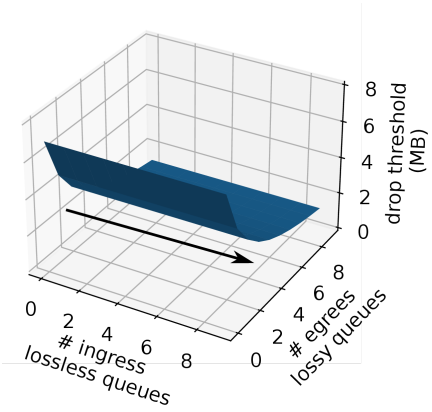
$$\overleftarrow{q}^{\bullet} = \overleftarrow{b} \cdot \left(\frac{\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}}{1 + \overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}} \right) - {}^{\circ}\overrightarrow{b} \cdot \left(\frac{\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}}{1 + \overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}} \cdot \frac{^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha}}{1 + {}^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha}} \right) \tag{5.6}$$

Overall, our steady-state analysis gives the amount of buffer occupied by lossless traffic (Equation 5.6) and lossy traffic (Equation 5.5) based on the buffer configuration and the state of the buffer i.e., the number of active queues of each class. Following the admission control of Dynamic Thresholds from Equation 5.1, we could compute the drop thresholds for egress lossy and PFC thresholds for ingress lossless by using each pool occupancy from above.

First, based on Heuristic 1, lossless traffic is allowed to use the entire buffer at the ingress i.e., $\overleftarrow{b} + b_h \leq b$. Further, lossless traffic is allowed to use the entire buffer at the egress i.e., $^{\bullet}\overrightarrow{b} = b$ based on Heuristic 2. Still, we end up with Issue 1.



(a) PFC thresholds (for ingress lossless queues) are affected by both the number of ingress lossless and egress lossy queues.



(b) Drop thresholds (for egress lossy queues) are affected only by lossy queues.

Figure 5.5: Contrary to the expectation that lossless and lossy traffic are admitted independently in the buffer, lossless traffic is throttled due to the buffer occupancy of lossy, whereas lossy is admitted independent of the presence of lossless i.e., seemingly prioritizing lossy over lossless.

Root cause of Issue 1: To shed light on this issue, we analytically derive the aggregate buffer allocation $\overleftarrow{q}^{\bullet}$ to lossless queues and the aggregate buffer allocation $^{\circ}\overrightarrow{q}$ to lossy queues. Let $\overleftarrow{n}^{\bullet}$ and $^{\circ}\overrightarrow{n}$ denote the number of ingress lossless queues and egress lossy queues using the buffer respectively. We have:

$$\overleftarrow{q}^{\bullet} = \overleftarrow{b} \cdot \left(\frac{\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}}{1 + \overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}} \right) - {}^{\circ}\overrightarrow{b} \cdot \left(\frac{\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}}{1 + \overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}} \cdot \frac{{}^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha}}{1 + {}^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha}} \right) \tag{5.7}$$

$${}^{\circ}\overrightarrow{q} = \frac{{}^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha} \cdot {}^{\circ}\overrightarrow{b}}{1 + {}^{\circ}\overrightarrow{n} \cdot {}^{\circ}\overrightarrow{\alpha}} \tag{5.8}$$

Figure 5.4 illustrates the ratio of buffer allocated to lossy and lossless. Notice that for a sufficiently large number of lossless queues $\overleftarrow{n}^{\bullet}$ and lossy queues $^{\circ}\overrightarrow{n}$, the buffer allocation to lossless queues tends to $\overleftarrow{b} - {}^{\circ}\overrightarrow{b}$ (based on Equation 5.7) and the allocation for lossy queues tends to $^{\circ}\overrightarrow{b}$ (based on Equation 5.8). Unless $\overleftarrow{b} \geq 2 \times {}^{\circ}\overrightarrow{b}$, we end up with Issue 1, caused by the buffer pools. Specifically, although lossless queues are allowed to fully utilize the buffer, the egress lossy pool *effectively* overlaps with both ingress pool and egress lossless pools as shown in Figure 5.1, leading to Issue 1.

■ ***Takeaway.*** The buffer is pre-fragmented in a way that makes enforcing high-level objectives through low-level configuration impossible. Current buffer sharing practices cannot prevent Issue 1 unless the ingress pool is at least twice as large as the egress lossy pool i.e., the buffer is statically partitioned.

Second, according to Heuristic 2 and Heuristic 3, since lossless (lossy) bypasses egress (ingress) admission control, we would expect that lossless and lossy traffic are isolated in the buffer. While Issue 1 already suggests that lossy traffic may effectively get more buffer allocation than lossless traffic, we find yet another issue that lossy and lossless traffic interact in a surprisingly unfair manner: lossy traffic is effectively prioritized over lossless traffic (Issue 2) although our expert heuristics are intended otherwise.

Root cause of Issue 2: Notice that the buffer occupancy of lossy traffic at the egress equals its occupancy at the ingress i.e., $^{\circ}\overrightarrow{q} = \overleftarrow{q}^{\circ}$, since every packet

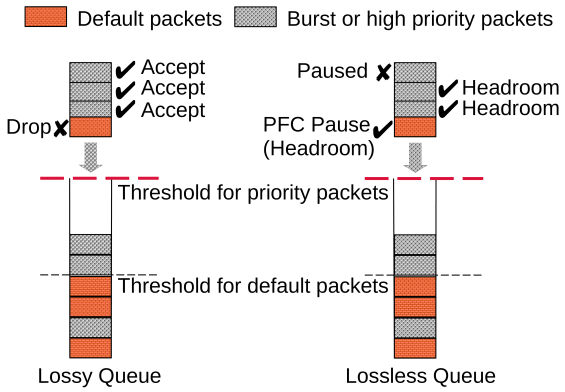


Figure 5.6: Per-packet prioritization cannot increase burst absorption for lossless traffic since PFC works at per-queue granularity.

is accounted both in the ingress and egress as shown in Figure 5.3. As a result, the overall ingress pool occupancy is the sum of egress lossy occupancy and the ingress lossless occupancy i.e., $\overleftarrow{q} = {}^{\circ}\overrightarrow{q} + \overleftarrow{q}^{\bullet}$. Using Equations 5.7 and 5.8, as well as the above relation, we derive the steady-state thresholds for ingress lossless (PFC thresholds) and egress lossy (drop thresholds) based on Equation 5.1.

Figure 5.5a shows how PFC thresholds for lossless queues vary depending on the number of ingress lossless queues using the buffer and the number of egress lossy queues. Interestingly, we find that the drop thresholds vary only according to the number of egress lossy queues (affected by the own buffer occupancy), but remain unchanged as the number of ingress lossless queues increases, see Figure 5.5b. We observe that current buffer sharing practices allow buffering lossy packets mostly independently of lossless traffic, but lossless traffic is suppressed due to lossy traffic i.e., effectively prioritizing lossy over lossless. Our analysis reveals two root causes of Issue 2: *(i)* the egress lossy pool occupancy ${}^{\circ}\overrightarrow{q}$ which is used to calculate egress lossy drop thresholds (see Equation 5.1) does not account for lossless traffic whereas *(ii)* the ingress pool occupancy $\overleftarrow{q}$ which is used to calculate ingress lossless PFC thresholds (see Equation 5.1) accounts for both lossless and lossy traffic.

■ ***Takeaway.*** Although lossless and lossy traffic are admitted seemingly independently by ingress and egress, the admission control for lossless traffic depends on both lossy and lossless occupancy, whereas the admission control for lossy traffic depends only on its own occupancy.

It is natural to ask here whether these issues are due to the underlying buffer management scheme DT [86], and hence whether recent proposals such as ABM [16] should be able to avoid them. Although ABM can address isolation across various traffic priorities, it only works for a buffer-sharing architecture that supports lossy traffic but not lossless traffic. While, in theory, one could extend ABM to calculate the thresholds accordingly (by replacing Equations 5.1 within the same buffer architecture with the corresponding pools described in §5.1.2), both Issue 1 and Issue 2 would still hold. Indeed, if ABM operates in the default buffer architecture, it will (similar to DT) control lossless and lossy independently since lossless (lossy) bypasses egress (ingress) admission control. As a result, ABM cannot jointly impose fairness and isolate lossless and lossy traffic. Importantly, the technique used for burst absorption in ABM and other recent proposals [46, 133], i.e., prioritizing burst packets by using a high α parameter for thresholds, also results in Issue 3.

Root cause of Issue 3: Indeed, several recent works [16,46,133] rely on selectively taking action on burst and non-burst packets i.e., if the queue length exceeds its threshold, the buffer management scheme still accepts burst packets selectively by increasing the thresholds using a higher α parameter value only for those packets identified as burst. Figure 5.6 intuitively summarizes our key points. For example, if a non-burst (e.g., long flow) packet interleaves burst packets, the non-burst packet would be dropped since the queue length exceeds its threshold. However, a key property of PFC is that once PFC is triggered due to a non-burst packet, it affects all arriving traffic (including bursts) to the queue due to PAUSE frames. Selectively accepting packets does not apply for PFC.

■ ***Takeaway.*** Optimally prioritizing bursts involves preferentially treating packets belonging to the burst only; this is possible for lossy but not for lossless traffic where congestion is signaled at a per-queue granularity.

5.2 Reverie

Based on the lessons learned from our analysis in §5.1, we design a buffer-sharing scheme Reverie which prevents harmful interactions between lossless and lossy traffic (isolation) while absorbing bursts of both. We first describe the two pillars on which Reverie relies: *(i)* consolidated admission control; and *(ii)* a low pass filter. Next, we explain how they fit together to form Reverie. Finally, we discuss Reverie's properties and its practicality.

5.2.1 Single Buffer Pool for Isolation

We argue that the first step towards achieving isolation is to have full visibility and control over the state of the buffer. Yet, current buffer-sharing practices maintain independent views and admissions at the ingress and egress, prohibiting global visibility and control. To address this, Reverie uses a single shared buffer pool as shown in Figure 5.1, in addition to a headroom pool dedicated to lossless queues. Further, Reverie uses a single admission control that jointly optimizes the buffer allocation for lossless and lossy queues. The single shared buffer pool and a single admission control offer Reverie a bird's eye view over the buffer.

Specifically, upon a packet arrival, Reverie first determines the packet's class (lossless or lossy). If the packet belongs to lossless • class, Reverie maps the packet to an ingress lossless queue $(s, p) \in \overleftarrow{\mathcal{Q}^{\bullet}}$, where s is the source port that received the packet and p is the packet priority. Similarly, if the packet belongs to the lossy class, Reverie maps the packet to an egress lossy queue $(d, p) \in \overrightarrow{\mathcal{Q}^{\circ}}$, where d is the port to which the packet is destined and p is the packet priority.

As illustrated in Figure 5.7, Reverie maintains only two types of queues (counters) i.e., ingress lossless and egress lossy, as opposed to the four types of queues in SONiC (see Figure 5.3). Further, Reverie accounts for each packet only once as opposed to twice (once at ingress and once at egress) in the current buffer-sharing practices. In essence, all the lossless and lossy queues are mapped to the same (single) shared buffer pool as shown in Figure 5.1.

Using a single shared buffer pool and a single admission control leaves Reverie solely responsible for fairly allocating the buffer across all queues to ensure isolation.

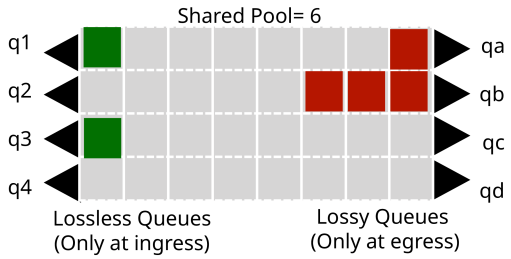


Figure 5.7: REVERIE's buffer bookkeeping: packets are stored once and accounted for once; lossless and lossy packets are accounted in the ingress and egress queues, respectively. All the queues are managed by a single admission control scheme.

In §5.2.3, we show how REVERIE's allocation achieves isolation across lossless and lossy.

5.2.2 Low-Pass Filter for Burst Absorption

A vast majority of prior works, including DT [86], FAB [46], ABM [16], TDT [133], calculate a threshold and compare it against *instantaneous* queue lengths in order to take buffer decisions. To improve burst absorption, prior works [16, 46, 133] selectively prioritize certain packets (e.g., short flows) by assigning them a larger threshold compared to the default threshold for other packets (e.g., long flows). However, as we explained in §5.1.3, selective packet prioritization cannot improve burst absorption for lossless traffic leading to Issue 3. It is essential to identify a queue experiencing a burst and prioritize all incoming traffic to the queue under bursty scenarios. A natural indicator for a queue that is experiencing a burst is its queue gradient, i.e., the rate of change of queue length. Thus, one could increase thresholds proportionally to the queue gradient. While intuitive, queue gradient is hard to monitor/calculate in practice in hardware, especially at microsecond granularity. To address this, we show an equivalence between an admission control based on queue gradient and an admission control based on first-order low-pass filtered queue lengths (Property 2). Leveraging this equivalence, REVERIE uses an exponential weighted moving average which is an easy-to-implement first-order low-pass filter. In essence, REVERIE compares *average* queue lengths against a threshold, unlike prior works that use instantaneous queue lengths.

Property 2 (Relationship of low pass filter and gradient)**.** *Let Ψ be an admission control scheme that compares first order low pass filtered queue length $\hat{q}(t-\delta t)$ against a threshold $\Gamma(t)$ i.e., $\hat{q}(t-\delta t) \leq \Gamma(t)$, where $t-\delta t$ denotes the previous time instance. Let Φ be an admission control that compares instantaneous queue length $q(t)$ against Ψ's threshold $\Gamma(t)$ incremented proportionally based on the average queue gradient $\frac{d\hat{q}(t)}{dt}$ i.e., $q(t) \leq \Gamma(t) + K \cdot \frac{d\hat{q}(t)}{dt}$; where K is a constant and $\frac{d\hat{q}}{dt}$ is the gradient. Then, there exists a constant K such that Ψ and Φ are equivalent.*

Proof. We consider exponentially weighted moving average for the first order low pass filter in this context. Let $q(t)$ denote the instantaneous queue length and let $\hat{q}(t)$ denote the average queue length. We denote the moving average parameter

by ν. The moving average of the instantaneous queue lengths is then as follows:

$$\hat{q}(t) = \nu \cdot q(t) + (1 - \nu) \cdot \hat{q}(t - \delta t)$$

where δt denotes the previous time when the average was updated. By rearranging the terms and dividing by δt,

$$\frac{\hat{q}(t) - \hat{q}(t - \delta t)}{\delta t} = \frac{\nu}{\delta t} \cdot (q(t) - \hat{q}(t - \delta t))$$

Let $K = \frac{\delta}{\nu}$. Using Euler's approximation method, we obtain the following:

$$K \cdot \frac{d\hat{q}(t)}{dt} = q(t) - \hat{q}(t - \delta t) \tag{5.9}$$

Using this relation, we now prove that Ψ and Φ admission control schemes are equivalent. We begin with Ψ which compares average queue lengths against a threshold $\Gamma(t)$ at time t.

$$\hat{q}(t - \delta t) \leq \Gamma(t)$$

Using Equation 5.9, we convert the above inequality as follows:

$$q(t) - K \cdot \frac{d\hat{q}(t)}{dt} \leq \Gamma(t)$$

By rearranging the terms, we obtain the the admission control scheme Φ. Hence Ψ and Φ are equivalent.

$$q(t) \leq \Gamma(t) + K \cdot \frac{d\hat{q}(t)}{dt}$$

□

5.2.3 The Workings of Reverie

In this subsection, we put all the pieces together, to describe Reverie's buffer-sharing architecture, admission control, and the underlying buffer management scheme.

Buffer sharing architecture: Let b be the total buffer space. Reverie dedicates a headroom pool of size b_h for lossless traffic similar to the existing architecture. The rest of the shared buffer space denoted by $\overset{*}{b} = b - b_h$ is shared by both lossless and lossy traffic dynamically. A lossless packet is mapped to a lossless queue $(s, p) \in \overleftarrow{\mathcal{Q}}^{\bullet}$ and a lossy packet is mapped to a lossy queue $(d, p) \in \overset{\circ}{\overrightarrow{\mathcal{Q}}}$ based on the source port s, destination port d and the packet priority p. In total, Reverie maintains a set $\overset{*}{\mathcal{Q}}$ of queues, where $\overset{*}{\mathcal{Q}} = \overleftarrow{\mathcal{Q}}^{\bullet} \cup \overset{\circ}{\overrightarrow{\mathcal{Q}}}$ consists of only two types of queues i.e., lossless and lossy instead of the four types maintained by SONiC.

Admission control: Reverie calculates a threshold $\Gamma^i_p(t)$ for each queue (i, p) and compares it against the moving averaged queue length $\hat{q}^i_p(t)$ of the corresponding queue at time t i.e.,

$$\hat{q}^i_p(t) \leq \Gamma^i_p(t) \tag{5.10}$$

where $\hat{q}^i_p(t)$ is given by,

$$\hat{q}^i_p(t) = \overbrace{\gamma \cdot \hat{q}^i_p(t - \delta t)}^{\textit{Capture steady congestion}} + \overbrace{(1 - \gamma) \cdot q^i_p(t)}^{\textit{Capture transient bursts}} \tag{5.11}$$

Here γ is a constant and a parameter for REVERIE, $q_p^i(t)$ is the instantaneous queue length, and $t - \delta t$ denotes the previous time instance. γ can be intuitively viewed as the degree of burst absorption. Since REVERIE's admission control (Equation 5.10) compares the threshold of a queue against its average queue length (Equation 5.11), a higher γ masks the impact of transient bursts on the average queue lengths and, as a result, allows the admission control to absorb more transient bursts as illustrated in Figure 5.2. However, setting γ arbitrarily close to 1 makes admission control more oblivious to persistent or steady-state congestion. We leave it for future work to study the optimal parameter setting.

Any arriving packet that satisfies Equation 5.10 is buffered in the shared pool. If a lossy packet does not satisfy the above condition, the packet is simply dropped. However, if a lossless packet does not satisfy the above condition, the queue enters pause state (sends a PFC pause frame to its peer) and all the subsequent packets to the queue are buffered in the dedicated headroom irrespective of Equation 5.10. The queue sends out a PFC resume frame once the queue's headroom drains out completely and when Equation 5.10 is satisfied again.

Buffer management: The threshold $\Gamma_p^i(t)$ for each queue $(i,p) \in \overset{*}{Q}$ calculated by REVERIE at time t, depends on *(i)* a configurable parameter α_p for each priority, *(ii)* the number of congested queues n_p of priority p and the remaining shared buffer space $\overset{*}{b} - \overset{*}{q}(t)$ where $\overset{*}{b}$ is the size of shared buffer pool and $\overset{*}{q}(t)$ is the pool occupancy at time t. The threshold $\Gamma_p^i(t)$ is given by,

$$\Gamma_p^i(t) = \alpha_p \cdot \frac{1}{n_p} \cdot (\overset{*}{b} - \overset{*}{q}(t)) \quad \forall (i,p) \in \overset{*}{Q} \tag{5.12}$$

The thresholds used by REVERIE are similar to ABM [16] but we drop the dequeue rate factor due to the complexity of measuring it, especially when queues pause in the case of lossless traffic. However, our design does not prevent using the dequeue rate factor as well if it can be systematically measured.

5.2.4 The Properties of Reverie

REVERIE inherits the steady-state isolation properties of ABM's thresholds for lossy as well as lossless traffic. Unlike ABM, though, which can only achieve isolation across priorities *within lossy* traffic, REVERIE can also achieve isolation *across lossless and lossy* priorities. In the following, for simplicity, we consider that all lossless and lossy queues are configured with the parameter value $\overset{\bullet}{\alpha}$ and $\overset{\circ}{\alpha}$ respectively.

In Theorem 10, we state the ratio in which buffer is allocated in the steady-state across lossless and lossy traffic in aggregate when both traffic classes compete for buffer. The ratio turns out to be the ratio of the configured α parameter. This makes it very intuitive and flexible to configure the buffer required for each traffic class, rather than the complicated pool sizes in the current practices.

Theorem 10 (Isolation)**.** *Under contention, REVERIE allocates buffer across lossless and lossy in the ratio of the corresponding α parameters i.e.,*

$$\frac{\overset{\bullet}{q}}{\overset{\circ}{q}} = \frac{\overset{\bullet}{\alpha}}{\overset{\circ}{\alpha}}$$

where $\overset{\bullet}{q}$ and $\overset{\circ}{q}$ denote the steady-state shared buffer occupancy of lossless and lossy traffic respectively; $\overset{\bullet}{\alpha}$ and $\overset{\circ}{\alpha}$ denote the parameter values for lossless and lossy queues respectively.

Proof. Since under steady-state, average and instantaneous converge, we simply use instantaneous values to prove our claim. Let $\overset{\bullet}{n}$ and $\overset{\circ}{n}$ be the number of congested queues of lossless and lossy. Reverie allocates a total of $\overset{\bullet}{q} = \overset{\bullet}{n} \cdot \overset{\bullet}{\alpha} \cdot \frac{1}{\overset{\bullet}{n}} \cdot (\overset{*}{b} - \overset{*}{q}) = \overset{\bullet}{\alpha} \cdot (\overset{*}{b} - \overset{*}{q})$ to $\overset{\bullet}{n}$ lossless and a total of $\overset{\circ}{q} = \overset{\circ}{n} \cdot \overset{\circ}{\alpha} \cdot \frac{1}{\overset{\circ}{n}} \cdot (\overset{*}{b} - \overset{*}{q}) = \overset{\circ}{\alpha} \cdot (\overset{*}{b} - \overset{*}{q})$ to $\overset{\circ}{n}$ lossy queues. Since both lossy and lossless are mapped to the shared pool, we have that $\overset{\bullet}{q} + \overset{\circ}{q} = \overset{*}{q}$. But substituting the previous relations, we obtain:

$$\overset{*}{q} = \frac{(\overset{\bullet}{\alpha} + \overset{\circ}{\alpha}) \cdot \overset{*}{b}}{1 + (\overset{\bullet}{\alpha} + \overset{\circ}{\alpha})} \tag{5.13}$$

$$\overset{\bullet}{q} = \frac{\overset{\bullet}{\alpha} \cdot \overset{*}{b}}{1 + (\overset{\bullet}{\alpha} + \overset{\circ}{\alpha})} \tag{5.14}$$

$$\overset{\circ}{q} = \frac{\overset{\circ}{\alpha} \cdot \overset{*}{b}}{1 + (\overset{\bullet}{\alpha} + \overset{\circ}{\alpha})} \tag{5.15}$$

From the above relations, it is easy to see that the ratio $\frac{\overset{\bullet}{q}}{\overset{\circ}{q}} = \frac{\overset{\bullet}{\alpha}}{\overset{\circ}{\alpha}}$. □

Based on Theorem 10, it is sufficient that the α parameter for lossless is greater than lossy in order to prevent issue 1. Further, since the thresholds are calculated with a bird's eye view of the buffer, the thresholds for both lossless and lossy depend on the overall buffer occupancy (see Equation 5.12). Hence, given that the α parameter for lossless is greater than lossy, Reverie assigns a larger threshold for lossless compared to lossy i.e., prioritizing lossless over lossy. Essentially, Reverie solves both issue 1 and issue 2 without statically partitioning the buffer.

When a single traffic class utilizes the buffer, Reverie allocates $\frac{\alpha}{1+\alpha}$ fraction of the shared buffer, where α corresponds to the parameter value of the traffic class using the buffer. Notice that Reverie allocates more buffer to a traffic class when it is not competing with the other class e.g., Reverie allocates $\frac{\overset{\bullet}{\alpha} \cdot \overset{*}{b}}{1+\overset{\bullet}{\alpha}}$ amount of buffer when only lossless traffic is using the buffer compared to $\frac{\overset{\bullet}{\alpha} \cdot \overset{*}{b}}{1+\overset{\bullet}{\alpha}+\overset{\circ}{\alpha}}$ amount of buffer allocation for lossless when both traffic classes are competing for buffer space. Intuitively, Reverie dynamically adapts the buffer allocation to lossless and lossy according to their load as opposed to the static pool sizes in the current practices. However, Reverie keeps some buffer idle.

Theorem 11 (Buffer waste)**.** *Reverie keeps idle a certain amount of buffer in the steady-state denoted by b_w given by,*

$$\frac{\overset{*}{b}}{1 + \overset{\bullet}{\alpha} + \overset{\circ}{\alpha}} \leq b_w \leq \frac{\overset{*}{b}}{1 + min(\overset{\bullet}{\alpha}, \overset{\circ}{\alpha})}$$

where $\overset{*}{b}$ is the shared buffer pool size; $\overset{\bullet}{\alpha}$ and $\overset{\circ}{\alpha}$ are the parameter values for lossless and lossy queues correspondingly.

Proof. Our proof follows from the proof of Theorem 11. Specifically, for $\overset{\bullet}{n}$ lossless queues and $\overset{\circ}{n}$ lossy queues, Reverie allocates $\overset{*}{q}$ in aggregate given by Equation 5.13. The remaining buffer $\overset{*}{b} - \overset{*}{q}$ which is wasted in the steady state is then given by,

$$b_w \geq \overset{*}{b} - \overset{*}{q} = \frac{\overset{*}{b}}{1 + (\overset{\bullet}{\alpha} + \overset{\circ}{\alpha})}$$

However, if a traffic class eg., lossless does not use the buffer, we can derive the remaining shared pool buffer similar to above. Depending on the smallest α value across all traffic classes, when such a class uses the buffer alone, then in this case the buffer waste is given by,

$$b_w \leq \overset{*}{b} - \overset{*}{q} = \frac{\overset{*}{b}}{1 + (min(\overset{\bullet}{\alpha}, \mathring{\alpha}))}$$

□

Although REVERIE keeps a tiny buffer portion idle in the steady state, this helps in absorbing transient bursts. REVERIE effectively absorbs transient bursts even for lossless traffic since it compares *average* queue lengths against the threshold. Indeed, upon burst arrival, the average queue length hits the threshold slower than the instantaneous queue length, essentially absorbing transient bursts even for lossless queues. As a result, REVERIE finally solves issue 3.

5.2.5 Implementation Feasibility

A prototype implementation of REVERIE is beyond the scope of this chapter and is part of our future work. Our discussions with NVIDIA already confirm that an approximation of the shared buffer pool model of REVERIE is feasible in hardware. In fact, we are currently discussing with a major Ethernet switch vendor on implementing REVERIE using their latest ASIC with programmable admission control features.

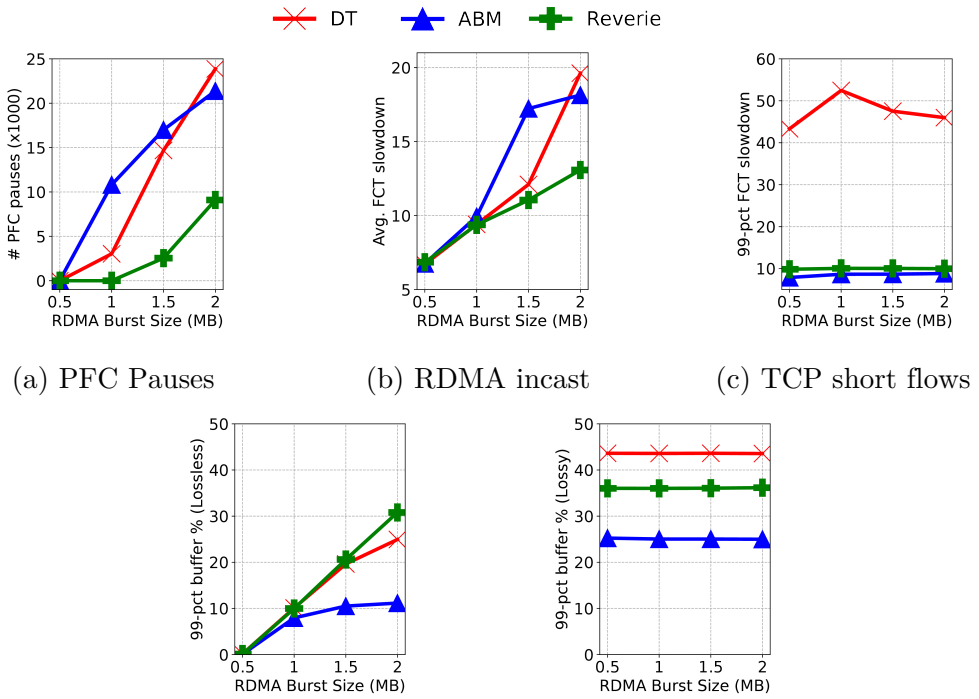
REVERIE is within reach of today's hardware because it does not significantly depart from commodity ASICs' buffer-sharing architecture and admission control mechanisms. REVERIE introduces two primary changes. First, REVERIE uses a simpler buffer-sharing architecture with a single shared pool (excluding PFC headroom) and two types of queues. We have confirmed with NVIDIA that this architecture is supported by existing NVIDIA Ethernet switch ASICs[4]. Second, REVERIE uses a first-order low pass filter to obtain the moving averaged queue lengths. We believe this is practical as moving averages are used by common AQM like RED [102]. As previous works (e.g., ABM [16], [239]) have noted, various queue-length statistics are available and already used by the native buffer management of the MMU (although switch vendors do not open up the API for operators to do so on their own).

5.3 Evaluation

We evaluate the performance of REVERIE and compare it against the state-of-the-art approaches in the datacenter. Our evaluation aims to answer four main questions:

(Q1) *Can REVERIE protect RDMA from TCP?*
We find that REVERIE shields the performance of RDMA from TCP under various loads. At increased loads, REVERIE reduces the number of PFC pauses by 60% on

[4]To implement this buffer-sharing architecture, we just need to map both egress lossy queues and egress lossless queues to a single egress buffer pool, and use an infinitely large egress lossless threshold.



(a) PFC Pauses (b) RDMA incast (c) TCP short flows

(d) RDMA buffer (e) TCP buffer

Figure 5.8: Buffer sharing under websearch TCP (Cubic) 80% load and across various burst sizes of incast RDMA (DCQCN) workload. As the burst size increases, the inability of DT and ABM to absorb RDMA bursts becomes more.

average compared to DT and by 71.2% compared to ABM, with DCQCN as the transport protocol. When using advanced congestion control for RDMA, Reverie reduces the number of PFC pauses by up to 100% compared to ABM.

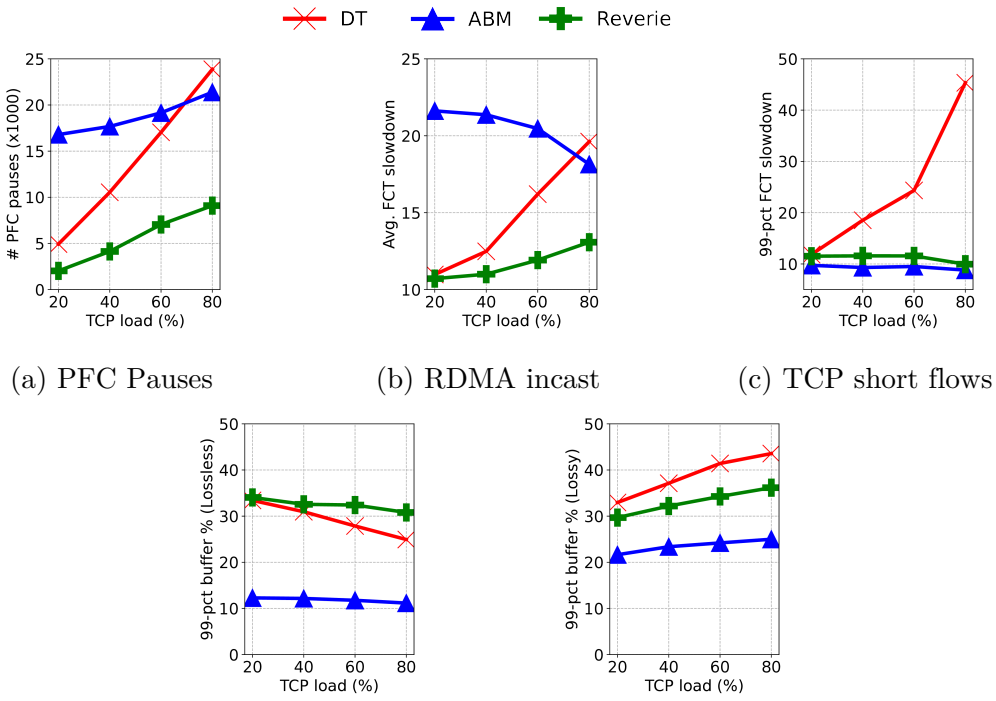
(Q2) *Can Reverie improve burst absorption of any class?*
We show that Reverie significantly improves the burst absorption for RDMA and for TCP. With background TCP traffic (websearch), Reverie improves the incast performance of RDMA by up to 33.3% compared to DT and by 50.4% compared to ABM. Under background RDMA traffic Reverie improves the incast performance of TCP by up to 46.8% compared to DT and by up to 2.1% compared to ABM.

(Q3) *Does Reverie penalize TCP?*
Reverie does not penalize TCP. We find that Reverie also improves the 99-percentile flow completion times (FCT) for short flows of TCP by 42.7% on average across various loads compared to DT. On this front, Reverie is on par with ABM.

(Q4) *How sensitive is Reverie to its parameters?*
We find an interesting characteristic of the parameter γ in Reverie: increasing γ arbitrarily close to 1 dramatically reduces the number of PFC pauses, and improves the FCT for incast flows. However, beyond a certain value of γ, the infrequent PFC pauses negatively affect FCT. Finding the optimal γ value for a given switch remains an open question.

5.3.1 Setup

Our evaluation is based on network simulator NS3 [209].



(a) PFC Pauses (b) RDMA incast (c) TCP short flows

(d) RDMA buffer (e) TCP buffer

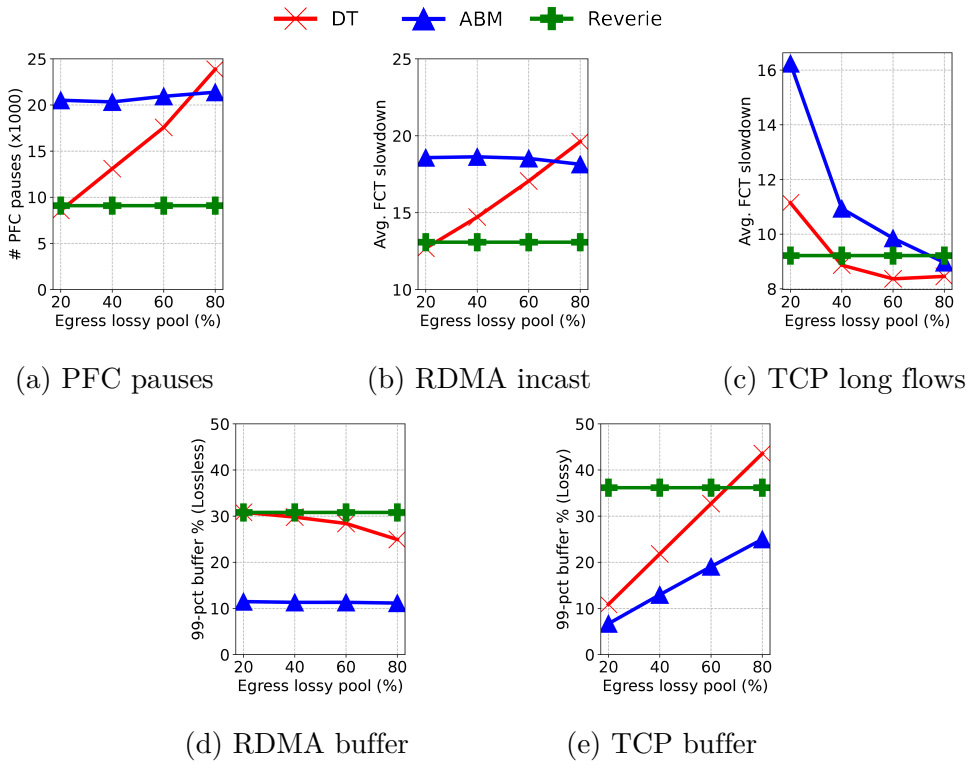
Figure 5.9: Buffer sharing under incast RDMA (DCQCN) workload and across various loads of websearch TCP (Cubic) workload. ABM achieves better performance for TCP but heavily penalizes RDMA, while REVERIE balances the two.

Topology: We consider a leaf-spine datacenter topology with 256 hosts organized into 4 spines and 16 leaves with 25Gbps links; link delay to 2μs (thus 17.28μs base RTT and 54KB bandwidth-delay product) and an oversubscription of 4, similarly to previous work [16, 21, 232]. All switches have 5.12KB buffer-per-port-per-Gbps similar to Broadcom Tomahawk [3][5]. All server NICs and switches are PFC enabled.

Traffic mix: We launch two types of workloads in our evaluation: *(i)* background and *(ii)* incast workloads. First, we generate background traffic across 20%-80% loads using websearch [35] flow size distribution, which is based on real-world datacenter measurements. Second, similar to prior works [16, 21, 33], we generate incast traffic using a synthetic workload that simulates the query-response behavior of a distributed file system. Specifically, each server in our topology sends out requests (queries) to all servers connected to a different leaf switch, chosen uniformly at random. These servers respond by sending a fraction of the file. We generate requests from each server based on a poisson process and we set the average request rate to 2 per second. We vary the file size (referred to as burst size). We use DCQCN [287] and PowerTCP [21] for RDMA congestion control; and Cubic [118] for TCP.

Baselines & metrics: We compare REVERIE with the SONiC [251] buffer model which is the state-of-the-art buffer sharing architecture widely deployed in today's datacenters. SONiC uses Dynamic Thresholds (DT) [86] as the buffer management scheme. The vast majority of the schemes in the literature are tailored for loss-

[5]While Tomahawk splits the buffer across 4 MMUs, for simplicity, we assume a single MMU manages the entire buffer.



(a) PFC pauses (b) RDMA incast (c) TCP long flows

(d) RDMA buffer (e) TCP buffer

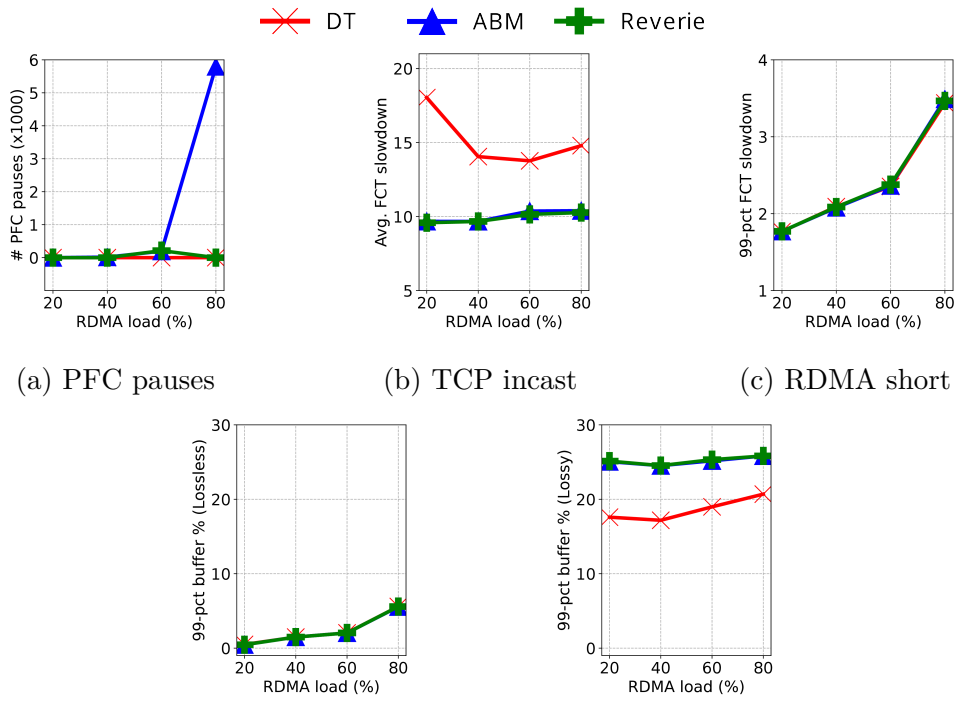
Figure 5.10: Buffer sharing under websearch TCP (Cubic) 80% load and incast RDMA (DCQCN) workload across various egress lossy pool sizes (% of ingress pool size) available for lossy TCP traffic. By changing the size of lossy pool, ABM and DT can only decide which traffic class will be prioritized against the other.

tolerant traffic; thus it is unclear how those schemes can be evaluated in a fair manner for lossless traffic. To address this, we extend ABM [16] to support lossless traffic within the SONiC buffer model (after discussing with the authors) by accounting for the drain rate as well as the number of saturated queues in the ingress, and use it as a baseline. We report the following metrics: *(i)* number of PFC pauses triggered, *(ii)* average FCT slowdown for incast traffic, *(iii)* 99-percentile FCT slowdown for short flows of background traffic *(iv)* 99-percentile buffer occupancy of RDMA and *(v)* 99-percentile buffer occupancy of TCP, as a percentage of the total shared buffer.

Switch buffer configuration: We set the headroom pool size based on the NIC bandwidth and link delay, according to [268]. The remaining buffer is configured as ingress pool size. We set the egress lossless pool to the total switch buffer size and the egress lossy pool to 80% of the ingress pool size. For Reverie, the headroom pool configuration is the same as stated earlier and the remaining buffer is configured as shared pool size. We set $\alpha = 1$ for all the schemes and set $\gamma = 0.999$ for Reverie. We configure DCQCN according to [173], which is based on industry experience, and PowerTCP according to [21]. We set TCP minRTO to $1ms$.

5.3.2 Results

Reverie significantly reduces PFC pause rate: We generate TCP traffic using websearch workload and RDMA traffic using the incast workload in Figure 5.9 and Figure 5.8. RDMA uses DCQCN for congestion control. Across various loads



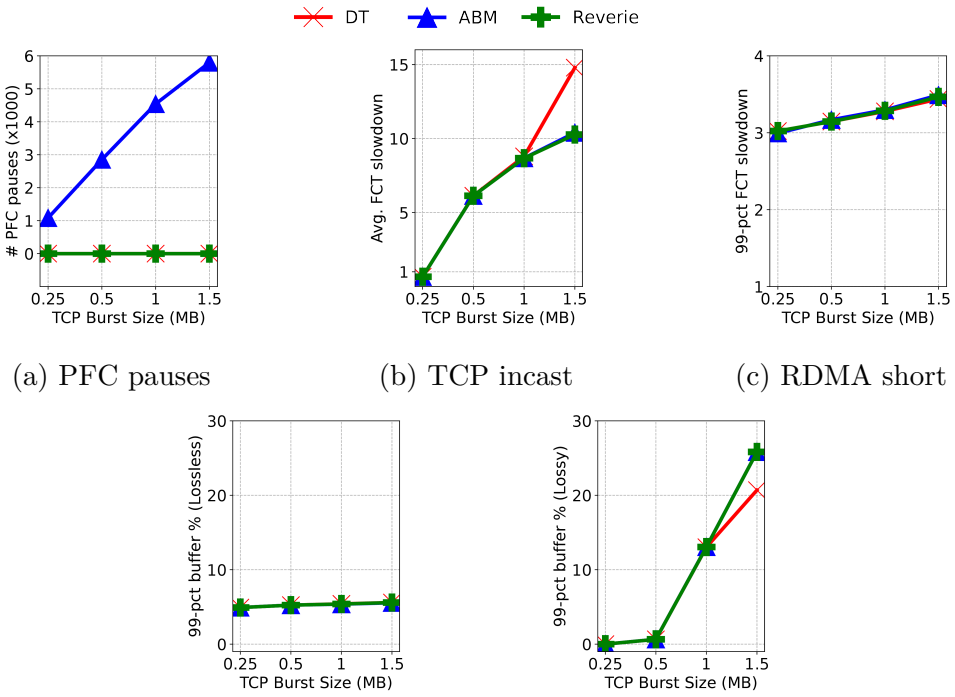
(a) PFC pauses (b) TCP incast (c) RDMA short

(d) RDMA buffer (e) TCP buffer

Figure 5.11: Buffer sharing under incast TCP (with Cubic) and across various loads of websearch RDMA (with PowerTCP). ABM can only deal with low loads of RDMA traffic as it cannot distinguish or priortize it.

of TCP and a burst size of 2MB for RDMA traffic, we observe from Figure 5.9a that Reverie reduces the number of PFC pauses by 60% on average compared to DT and by 71.2% on average compared to ABM. Specifically, even at 20% TCP load, Reverie reduces the number of PFC pauses by 58.9% compared to DT and by 87.9% compared to ABM. Further, across various burst sizes of RDMA with 80% TCP load, Reverie reduces the number of PFC pauses by 61.8% on average compared to DT and by 57.4% on average compared to ABM, as shown in Figure 5.8a. In Figure 5.11 and Figure 5.12, we generate RDMA traffic using websearch workload and TCP traffic using the incast workload. We use PowerTCP as the congestion control for RDMA. From Figure 5.11a showing various RDMA loads at 1.5MB TCP burst size, and Figure 5.12a showing various TCP bursts at 80% RDMA load, we observe that Reverie significantly reduces the PFC pauses by 100% compared to ABM, while Reverie and DT perform similarly in this case. This confirms our observations in §5.1 on SONiC that lossy severely interacts with lossless traffic even though they are controlled independently by ingress and egress. Reverie drastically reduces TCP's interference with RDMA.

Reverie improves burst absorption for RDMA & TCP: Figure 5.9b, shows that across various TCP loads, Reverie significantly reduces the average FCT for incast flows by 18.5% on average compared to DT and by 18.2% on average compared to ABM. At 80% TCP load, across various RDMA burst sizes, Figure 5.8b shows that Reverie improves the average FCT for incast flows by 10% on average compared to DT and by 17% compared to ABM. This shows that Reverie improves the overall burst absorption for RDMA. Although Reverie's thresholds are similar to ABM's, Reverie achieves better performance for lossless traffic due to its LPF-based admission control which favors transient bursts.



(a) PFC pauses (b) TCP incast (c) RDMA short

(d) RDMA buffer (e) TCP buffer

Figure 5.12: Buffer sharing under websearch RDMA (with PowerTCP) at 80% load varying burst sizes of incast TCP (with Cubic). As TCP traffic increases, ABM further penalizes RDMA to protect TCP traffic.

Across various loads of RDMA and 1.5MB TCP burst size, in Figure 5.11b, we see that Reverie significantly reduces the average FCT for incast flows by 33.7% on average compared to DT and by 1.08% compared to ABM. From Figure 5.12b, across various TCP burst sizes, we observe that Reverie reduces the average FCT for incast flows of TCP by up to 30% for large bursts compared to DT while Reverie performs similarly to ABM. Overall, Reverie's LPF-based admission control scheme improves burst absorption for RDMA as well as for TCP.

Reverie protects RDMA from TCP in the buffer: Given the better burst absorption of Reverie and significantly fewer PFC pauses even in the presence of TCP as seen in Figures 5.9-5.12, already shows that Reverie successfully prevents TCP's interactions with RDMA. Moreover, from Figure 5.9d, we see that DT *reduces* its buffer allocation significantly for lossless traffic as the load of TCP increases, while *increasing* buffer allocation for TCP (Figure 5.9e). However, Reverie gives lossless traffic its fair share in the buffer even at high TCP loads. Further, as the burst size of RDMA increases, from Figure 5.8d, we see that Reverie opportunistically allocates increasingly buffer to RDMA whereas DT and ABM fail to allocate more buffers to RDMA, thus significantly increasing PFC pauses (Figure 5.8a). Reverie's isolation properties allow RDMA to get its fair share of buffer even at high TCP loads. Reducing TCP's buffer share (egress lossy pool size) for DT and ABM improves the number of PFC pauses (Figure 5.10a) and flow completion times for RDMA incast flows, but TCP long flows suffer (Figure 5.10c) due to the reduced overall buffer available for TCP (Figure 5.10e). In contrast, Reverie dynamically utilizes the entire shared buffer space in a fair manner and protects RDMA from TCP in the buffer as seen in Figures 5.9, 5.8, 5.10.

Figure 5.13: As the parameter γ value approaches 1, Reverie becomes oblivious to the changes in the buffer and drastically reduces the number of PFC pauses. However, beyond a certain value, the lack of PFC pauses negatively impacts FCTs.

With advanced congestion control for RDMA (PowerTCP) under websearch workload, across various RDMA loads and TCP incasts (Figures 5.11d, 5.12d), Reverie and DT as well as ABM occupy a significantly small portion of buffer and achieve similar FCTs for short flows of RDMA (Figures 5.11c, 5.12c). However, Reverie triggers much lower PFC pauses than ABM as we observe in Figures 5.11a, 5.12a even with PowerTCP.

Reverie also protects TCP in the buffer: Under websearch workload for TCP and RDMA incasts, from Figures 5.9c and 5.8c, we see that Reverie and ABM achieve similar FCTs for short flows of TCP whereas DT severely penalizes TCP. DT penalizes TCP short flows even though it allocates more buffer to TCP compared to Reverie as seen in Figures 5.9c, 5.8e. This excessive buffering results in increased queueing delays for DT. Further, under incast workload for TCP and websearch workload for RDMA, while Reverie and ABM achieve similar FCTs for incast TCP flows, DT suffers from poor FCTs for TCP incasts (Figures 5.11b, 5.12b). Unlike DT, Reverie and ABM protect TCP in the buffer.

Impact of LPF filtering: As discussed in §5.2.3, the parameter γ balances the capturing of steady-state (long-term) congestion against transient-state (short-term) congestion is captured by the admission control scheme. To better understand the impact of γ, we generate RDMA traffic using websearch workload at 80% load along with incast workload at 2MB burst size. In Figure 5.13, we show the number of PFC pauses and the average FCT for incast flows as a function of γ value. We observe that PFC pauses dramatically reduce as γ increases. Average FCT for incast flows decreases as γ increases until $\gamma = 0.999$. Yet, for $\gamma = 0.999999$ (close to 1), the average FCT increases by 9%. Naturally, a small γ value makes the admission control scheme highly sensitive to instantaneous queue lengths, which triggers PFC more frequently upon transient bursts. Similarly, a high γ value makes the admission control scheme insensitive to queue length and PFC is not triggered even when the queues *steadily* grow. In such cases, the excessive buffer occupied by steady-state traffic leaves less buffer to absorb transient bursts. Finding an optimal γ value is not required for reaping the benefits of Reverie, as long as we avoid the extreme values that are easy to distinguish.

5.4 Related Work

Our work relates to *(i)* buffer management; and *(ii)* RDMA.

Multiple works focus on sharing the on-chip buffer across queues of the same switch [16, 45, 46, 49, 56, 64, 86, 97, 161, 231] and on sharing bandwidth across queues of the same port e.g., AQM and scheduling [37, 102, 103, 139, 215]. In fact, there are also proposals to combine the two [5]. Further, augementing buffer

sharing algorithms with machine-learned predictions has been shown to improve performance [24]. While useful, such works are designed exclusively for lossy traffic (i.e., TCP variants) and often with loss-based congestion control in mind. As a result, they are orthogonal to this work.

Many cloud providers have deployed RDMA over Ethernet to accelerate storage [58, 105], HPC, and ML [212]. To the best of our knowledge, all of these deployments [58, 105, 212] rely on PFC. Other research efforts related to RDMA include congestion control [21, 173, 287], efficient loss recovery [194], deadlock prevention [132], high performance RDMA applications [92, 146, 147, 170], testing [159], security [227, 254, 273] and performance isolation [283]. Among them, the most related topic is congestion control, but also in those works the buffer is only used by RDMA traffic [173, 287] (i.e., no TCP).

Coexistence of RDMA and TCP is an emerging new problem. Several recent parallel works proposed alternative solutions e.g., dynamically sharing the headroom buffer space under extremely shallow buffers [241]; using average occupancy time of packets to allocate buffers for each queue [181]. Yet, unlike Reverie, these works do not address the fundamental issues that arise due to the static buffer pool configurations in today's switches. We leave it for future work to evaluate how the emerging alternative approaches fare against Reverie's allocation.

5.5 Summary

This chapter addresses the tension in buffer sharing between lossy traffic (e.g., TCP variants) and lossless traffic (e.g., RDMA) on datacenter switches. To this end, we first uncover, and explain analytically three particular unexpected buffer behaviors (issues) that today's buffer-sharing scheme can cause. Next, we find the root cause of these inefficiencies, and design a new buffer sharing scheme, Reverie that can provide both isolation and high burst absorption to lossy and lossless traffic. In future work, we will try to closely work with a switch ASIC vendor to incorporate Reverie into an ASIC's programmable admission control features.

6

Augmenting Buffer Sharing with ML Predictions

Datacenter switches come equipped with an on-chip packet buffer that is shared across all the device ports in order to improve the overall throughput and to reduce packet drops. Unfortunately, buffers have become increasingly expensive and chip-manufacturers are unable to scale up buffer sizes proportional to capacity increase [59]. As a result, the buffer available per port per unit capacity of datacenter switches has been gradually reducing over time. Worse yet, datacenter traffic is bursty even at microsecond timescales [108,282]. This makes it challenging for a buffer sharing algorithm to maximize throughput. Recent measurement studies in large scale datacenters point-out the need for improved buffer sharing algorithms in order to reduce packet drops during congestion events [108]. To this end, buffer sharing under shallow buffers is an emerging critical problem in datacenters [16,58].

The buffer sharing problem has been widely studied in the literature from an online perspective [72] with the objective to maximize throughput [44, 111, 119, 155, 158]. Traditional online algorithms for buffer sharing can be classified into two types: **drop-tail** e.g., Dynamic Thresholds (DT) [86], Harmonic [155], ABM [16] and **push-out** e.g., Longest Queue Drop (LQD). The performance gap of these algorithms compared to an offline optimal (clairvoyant) algorithm can be expressed in terms of the competitive ratio [72]. For instance, we say that an online algorithm is 2-competitive if it performs at most 2x worse compared to an offline optimal algorithm. Figure 6.1 illustrates the performance spectrum of drop-tail and push out buffer sharing algorithms. In terms of throughput-competitiveness, it is well-known that push-out algorithms perform significantly better than drop-tail algorithms. In fact, no deterministic drop-tail algorithm can perform better than a certain throughput (a lower bound for competitive ratio), beyond which only push-out algorithms exist (Figure 6.1). Table 6.1 presents the competitive ratios of known algorithms. Interestingly, LQD pushes out packets when the buffer is full, and it is $\approx$ 2-competitive whereas Complete Sharing drops packets when the buffer is full, but it is $N+1$-competitive.

Intuitively, the poor throughput-competitiveness of drop-tail buffers owes it to the fundamental challenge that utilizing the buffer for some queues comes at the cost of deprivation of buffer for others [16]. To this end, drop-tail algorithms proactively drop packets i.e., packets are dropped even when the buffer has remaining space [46, 86, 119, 133, 155]. On one hand, maintaining remaining buffer space is necessary to serve future packet arrivals. On the other hand, maintaining remaining buffer space could lead to under-utilization, throughput loss and excessive packet drops. In contrast, the superior throughput-competitiveness of push-out algorithms owes it to their fundamental advantage to push out packets instead

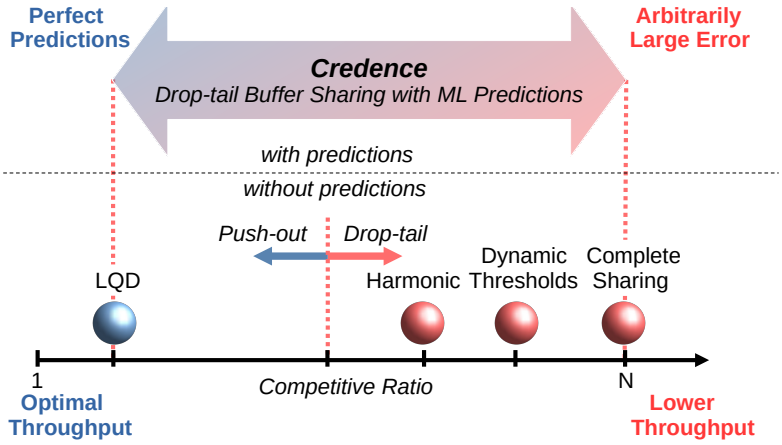


Figure 6.1: Augmenting drop-tail buffer sharing with ML predictions has the potential to significantly improve throughput compared to the best possible drop-tail algorithm (without predictions), and unlock the performance that was only attainable by push-out so far.

Algorithm	Competitive Ratio
Complete Sharing [119]	$N+1$
Dynamic Thresholds [86, 119]	$\mathcal{O}(N)$
Harmonic [155]	$\ln(N)+2$
LQD (push-out) [44, 119]	1.707
LateQD (clairvoyant) [67]	1
CREDENCE	$\min(1.707\ \eta,\ N)$

Table 6.1: CREDENCE's performance smoothly depends on the prediction error (η). CREDENCE outperforms traditional drop-tail buffer sharing algorithms and performs as good as push-out when the predictions are perfect ($\eta = 1$) but is also never worse than Complete Sharing even when the predictions are bad ($\eta \to \infty$). N denotes the number of ports.

of dropping them[1]. Hence, push-out algorithms can utilize the entire buffer as needed and only push out packets when multiple ports contend for buffer space. Although push-out algorithms offer far superior performance guarantees compared to drop-tail, hardly any datacenter switch supports push-out operations for the on-chip shared buffer. This begs the question: Are drop-tail buffer sharing algorithms ready for the trend of shrinking buffer sizes?

Our key observation is that every push-out algorithm can be converted to a drop-tail algorithm. However, such a conversion requires certain (limited) visibility into the future packet arrivals. Specifically, pushing out a packet is equivalent to dropping the packet when it arrives. Recent advancements in dataplane programmability and traffic predictions play a pivotal role in providing such visibility into the future packet arrivals [30, 76, 140, 167]: paving a way for better drop-tail buffer sharing algorithms.

In this work, we take the first step in this direction. Figure 6.1 illustrates our perspective. We propose CREDENCE, a drop-tail buffer sharing algorithm augmented with machine-learned predictions. CREDENCE's performance is tied to

[1]Push-out operation, similar to a drop operation, does not incur any transmission delays, unlike extract-out [245].

the accuracy of these predictions. As the prediction error decreases, CREDENCE unlocks the performance of push-out algorithms and reaches the performance of the best-known algorithm. Even when the prediction error grows arbitrarily large, CREDENCE offers at least the performance of the simplest drop-tail algorithm Complete Sharing. Table 6.1 gives the competitive ratio of CREDENCE as a function of the prediction error η. Importantly, CREDENCE's performance *smoothly* varies with the prediction error, generalizing the performance space between the known push-out and drop-tail algorithms. Hence, CREDENCE achieves the three goals of prediction-augmented algorithms, in the literature referred to as consistency, robustness and smoothness [198, 223].

In addition to the theoretical guarantees for CREDENCE's performance, our goal is also its practicality. Specifically, without predictions, CREDENCE's core logic only uses additions, subtractions, and does not add additional complexity compared to existing approaches. For predictions, we currently use random forests, which have been recently shown to be feasible on programmable switches at line rate [30, 76]. A full implementation of CREDENCE in hardware unfortunately requires switch vendor involvement since buffer sharing is merely a blackbox even in programmable switches. With this work, we wish to gain attention from switch vendors on the fundamental blocks required for such algorithms to be deployed in the dataplane. We currently implement CREDENCE in NS3 to evaluate its performance using realistic datacenter workloads. We present a detailed discussion on the practicality of CREDENCE later in this chapter.

Our evaluations show that CREDENCE performs 1.5x better in terms of throughput and up to 95% better in terms of flow completion times, compared to alternative approaches.

We believe CREDENCE is a stepping stone towards further improving buffer sharing algorithms. Especially, achieving better performance than CREDENCE under large prediction error remains an interesting open question. Our approach of augmenting buffer sharing with predictions is not limited to drop-tail algorithms, but push-out algorithms can also benefit from predictions. We discuss exciting future research directions both in systems and theory at the end of this chapter.

In summary, our key contributions in this chapter are:

- CREDENCE, the first buffer sharing algorithm augmented with predictions, achieving near-optimal performance with perfect predictions while also guaranteeing performance under arbitrarily large prediction error, and gradually degrading the performance as the prediction error increases.
- Extensive evaluations using realistic datacenter workloads, showing that CREDENCE outperforms existing approaches in terms of flow completion times.
- All our artifacts have been made publicly available at https://github.com/inet-tub/ns3-datacenter.

6.1 Motivation

In this section, we provide a brief background and motivate our approach by highlighting the drawbacks of traditional approaches. We show the potential

for reaching close-to-optimal performance when buffer sharing algorithms are augmented with machine-learned predictions. To this end, we first describe our model and throughput competitiveness (§6.1.1). We then discuss the drawbacks of existing approaches (§6.1.2). We show that a renewed hope for improved buffer sharing is enabled by the recent rise in algorithms with predictions (§6.1.3).

6.1.1 Buffer Sharing from Online Perspective

A network switch receives packets one after the other at each of its ports. The switch does not know the packet arrivals ahead of time. This makes buffer sharing inherently an *online* problem i.e., algorithms must take instantaneous decisions upon packet arrivals without the knowledge of the future. In order to systematically understand the performance of such algorithms, we take an online approach following the classical model in the literature [44, 111, 119, 155, 158]. Figure 6.2 illustrates the model.

Buffer model: We consider a network switch equipped with an on-chip buffer size of B units shared by N ports. Time is discrete, and we refer to each step as timeslot. Packets (each of size unit 1) arrive in an online manner as time progresses. Each timeslot is divided into two phases, arrival phase and departure phase. During each arrival phase, at most N number of packets (in aggregate) arrive destined to N ports. During each departure phase, every queue drains out one packet unless the queue is empty. A buffer sharing algorithm manages the shared buffer allocation across the N ports. We next define preemptive (push-out) and non-preemptive (drop-tail) buffer sharing.

Definition 2 (Preemptive buffer sharing)**.** *During every arrival phase, the buffer sharing algorithm is allowed to preempt i.e., drop any number of existing packets in the buffer.*

Definition 3 (Non-preemptive buffer sharing)**.** *During every arrival phase, the buffer sharing algorithm is only allowed to accept or drop the incoming packet. Every accepted packet must eventually be drained out from the corresponding queue.*

Online algorithm: When a packet arrives, a buffer sharing algorithm determines whether it should be accepted into the available buffer space. Drop-tail algorithms can only accept or discard incoming packets, while push-out algorithms can also remove packets from the buffer.

Objective: The network throughput is of utmost importance for datacenter operators since throughput often relates to the cost in typical business models (e.g., $ per bandwidth usage). We hence study the performance of a buffer sharing algorithm in terms of throughput i.e., our objective is to maximize the total number of packets transmitted over the entire arrival sequence. The throughput maximization objective is closely related to packet drops minimization objective. In this sense, our objective captures two important performance metrics i.e., throughput and packet drops.

Competitive Ratio: We use competitive ratio as a measure to compare the performance of an online algorithm to the optimal offline algorithm. Specifically,

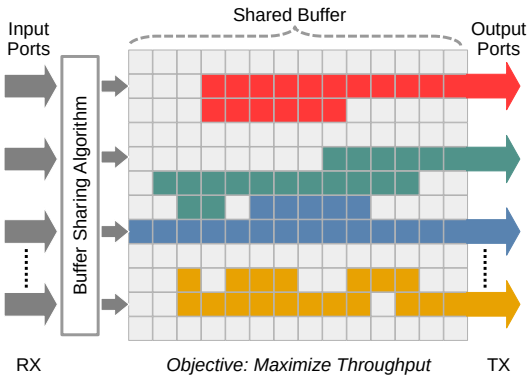


Figure 6.2: The switch has a buffer size of B shared across N output ports. Each color indicates the packets residing in the shared buffer corresponding to each port. A buffer sharing algorithm takes decisions (accept or drop) for each input packet.

let ALG and OPT be an online and optimal offline algorithm correspondingly. We denote by $\sigma(t) = (\sigma_i(t), \sigma_i(t), ..., \sigma_N(t))$, an N-tuple, where $\sigma_i(t)$ denotes the number of packets arriving at time t to queue i. Let $ALG(\sigma)$ be the throughput of ALG for the packet arrival sequence σ. We say an algorithm ALG is c-competitive if the following relation holds for any packet arrival sequence.

Definition 4 (Competitive ratio)**.** *Let ALG be an online algorithm and OPT be an offline optimal algorithm for the buffer sharing problem. Let $ALG(\sigma)$ and $OPT(\sigma)$ be the total number of packets transmitted by ALG and OPT for the arrival sequence σ. We say ALG is c-competitive if it satisfies the following condition for any arrival sequence σ.*

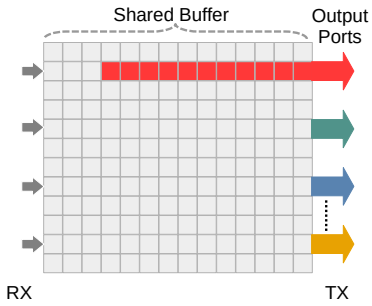
$$OPT(\sigma) \leq c \cdot ALG(\sigma) \tag{6.1}$$

Competitive ratio is a particularly interesting metric for buffer sharing since it offers performance guarantees without any assumptions on specific traffic patterns. For example, the buffer may face excessive packet drops or may temporarily experience throughput loss due to bursty traffic. One could argue that the buffer sharing algorithm is the culprit and should have allocated more buffer to the bursty traffic. While this may have solved the problem for a particular bursty arrival, the same solution could result in unexpected drops and throughput loss if there were excessive bursty arrivals i.e., large bursts could monopolize the buffer. Instead, from an online perspective, better competitive ratio indicates that the buffer sharing algorithm performs close to optimal under any traffic conditions.

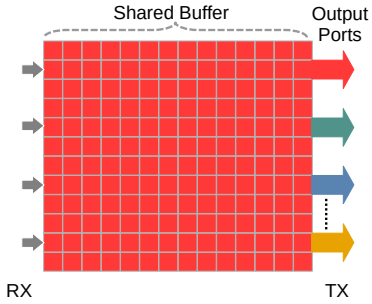
■ ***Takeaway.*** *A buffer sharing algorithm with lower competitive ratio improves the throughput of the switch and reduces packet drops under worst-case packet arrival patterns.*

6.1.2 Drawbacks of Traditional Approaches

We observe two main drawbacks of traditional buffer sharing algorithms, both affecting the competitive ratio. First, algorithms proactively and unnecessarily drop packets in view of accommodating future packet arrivals. Second, algorithms reactively drop packets when the buffer is full and incur throughput loss, which could have been avoided. We argue that these drawbacks are rather fundamental to drop-tail algorithms and cannot be addressed by traditional online approaches.

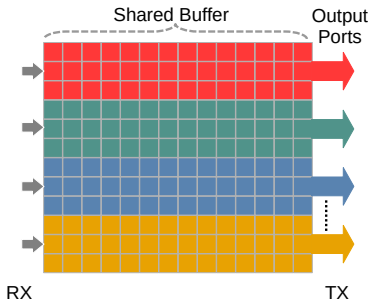


(a) ALG

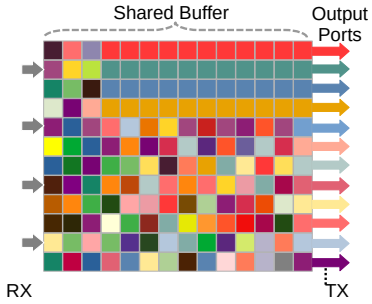


(b) OPT

Figure 6.3: Upon a large burst arrival, a typical drop-tail algorithm (ALG) *proactively* drops the incoming packets in anticipation of future bursts and significantly under-utilizes the buffer. In this case, an optimal offline algorithm accepts the entire burst without any packet drops.



(a) ALG



(b) OPT

Figure 6.4: In pursuit of high burst absorption, a drop-tail algorithm ALG may absorb bursts but this results in excessive *reactive* drops for the future packet arrivals. In this case, an optimal offline algorithm OPT drops few packets such that the overall throughput is maximized.

Proactive unnecessary packet drops → throughput loss: A drop-tail buffer sharing algorithm typically drops packets even if there is remaining buffer space available [16, 46, 86]. We refer to such drops as proactive drops. Being proactive is indeed necessary in order to accommodate transient bursts. However, proactive packet drops and the corresponding remaining buffer space ends up being wasteful if the future packet arrivals do not need additional buffer space (if the anticipated burst does not arrive). Figure 6.3a and Figure 6.3b illustrate an example. Consider a traffic pattern where there is little to no congestion on all the ports but once in a while, a large burst appears. Specifically, the buffer is empty initially and a large burst of size B appears. A deterministic drop-tail algorithm has two choices: *(i)* accept a portion of the burst and proactively drop the rest of the burst or *(ii)* accept the entire burst. Typical algorithms in the literature choose the former in view of accommodating future packet arrivals. An optimal offline algorithm that knows the arrivals ahead of time would accept the entire burst of size B in this case. This makes an online algorithm at least c-competitive for this particular arrival pattern, where $\frac{1}{c}$ is the fraction of the burst accepted: since the optimal solution accepts and transmits B packets over time, whereas an online algorithm only accepts and transmits only $\frac{B}{c}$ packets over time. We observe that recent works focus on minimizing proactive unnecessary packet drops by prioritizing bursty traffic to the extent that they allow burst on a single port to monopolize the buffer [16, 46, 133]. However, note that competitive ratio is not defined for a particular arrival sequence, but over all scenarios. To this end, accepting a larger burst size may be helpful in the above example but if there were indeed future packet arrivals on other ports that need buffer, the algorithm incurs excessive

reactive drops (described next) and throughput loss.

Reactive avoidable packet drops → throughput loss: Any drop-tail algorithm is forced to drop the incoming packets once the shared buffer is full. We call such drops reactive drops. Reactive drops result in throughput loss if the algorithm fills up significant portion of the buffer on a small set of ports but reactively drops incoming packets to other ports. Figure 6.4a and Figure 6.4b illustrate an example. Consider that the buffer is initially empty and four simultaneous bursts each of size B arrive to four ports. If an algorithm proactively drops a significant portion of the bursts, it would suffer under arrival sequences such as in the previous example (Figure 6.3a). Alternatively, the algorithm may choose to accept a larger portion of the bursts and ends up filling up the entire buffer in aggregate. At this point, several short bursts arrive to multiple other ports. An optimal offline algorithm accepts only a fraction of the large bursts such that it is able to accommodate upcoming short bursts. In doing so, the optimal algorithm benefits in throughput since the switch transmits packets from more number of ports. However, since the online algorithm fills up the entire buffer due to the initial large bursts, it is forced to reactively drop the upcoming short bursts, losing throughput. In fact, a similar arrival pattern for Dynamic Thresholds yields at least $\Omega\left(\sqrt{\frac{N}{\log(N)}}\right)$-competitiveness [119]. The known upper bound for Dynamic Thresholds is $\mathcal{O}(N)$ [119]. Further, it has been shown in the literature that no deterministic drop-tail algorithm can be better than $\Omega\left(\frac{\log(N)}{\log(\log(N))}\right)$-competitive [155].

Interestingly, push-out algorithms are not prone to the problems discussed above, since they can take revocable decisions i.e., to accept a packet and drop it later. Hence, push-out algorithms do not have to maintain free space in the buffer in order to accommodate transient bursts. Instead, such algorithms can defer the dropping decision until the moment the drop turns out to be necessary.

■ ***Takeaway.*** *Traditional drop-tail algorithms are fundamentally limited in throughput-competitiveness as they are unable to effectively navigate proactive and reactive drops due to the online nature of the problem i.e., future packet arrivals are unknown to the algorithm.*

6.1.3 Predictions: A Hope for Competitiveness

Given that the fundamental barrier in improving drop-tail buffer sharing algorithms is the lack of visibility into the future arrivals, we turn towards predictions. The recent rise of algorithms with predictions offers a renewed hope for competitive buffer sharing. Algorithms with predictions successfully enabled close to optimal performance for various classic problems [223]. The core idea is to guide the underlying online algorithm with certain knowledge about the future obtained via predictions. The machine-learned oracle that produces predictions is considered a blackbox with a certain error. The main challenge is to offer performance guarantees at the extremes i.e., close to optimal performance under perfect predictions and a minimum performance guarantee when the prediction error gets arbitrarily large. Further, it is desirable that the competitiveness of the algorithm *smoothly* degrades as the prediction error grows.

6.1.3.1 Prediction Model

In the context of the buffer sharing problem, there are several prediction models

that can be considered e.g., drops or packet arrivals. In this chapter, we assume that a blackbox machine-learned oracle predicts packet drops. Our choice is due to the fact that packet drops are the basic decisions made by an algorithm. Concretely, we consider an oracle that predicts whether an incoming packet would eventually be dropped (or pushed out) by the Longest Queue Drop (LQD) algorithm serving the same packet arrival sequence. We classify the predictions into four types: *(i)* true positive i.e., a correct prediction that a packet is eventually dropped by LQD, *(ii)* false negative i.e., an incorrect prediction that a packet is eventually transmitted by LQD, *(iii)* false positive i.e., an incorrect prediction that a packet is eventually dropped by LQD and *(iv)* true negative i.e., a correct prediction that a packet is eventually transmitted by LQD. Figure 6.5 summarizes this classification. Following the literature [198,223], our goals for prediction-augmented buffer sharing are consistency, robustness and smoothness.

α-Consistent buffer sharing algorithm has a competitive ratio α when the predictions are all true i.e., perfect predictions.

β-Robust buffer sharing algorithm has a competitive ratio β when the predictions are all false i.e., large prediction error.

Smoothness is a desirable property such that the competitive ratio degrades smoothly as the prediction error grows i.e., a small change in error does not drastically influence the competitive ratio.

Our goal is to design a prediction-augmented buffer sharing algorithm that is close to 1-consistent (with perfect predictions) i.e., near-optimal, at most N-robust (with arbitrarily large error) i.e., not worse than Complete Sharing algorithm, and has the desirable property of smoothness.

6.1.3.2 Common Pitfalls

It is intuitive that predictions can potentially improve the performance of a drop-tail algorithm. For instance, in the examples from Figure 6.3 and Figure 6.4, our prediction-augmented online algorithm could take nearly the same decisions as a push-out algorithm. However, the main challenge is to ensure robustness and smoothness. If an algorithm blindly trusts the predictions, we observe that false positive and false negative predictions have a significantly different impact on the performance.

Excessive false positives can lead to starvation: The worst case for a naive algorithm that blindly trusts predictions is when all the predictions are false positives. In this case, the algorithm ends up dropping every incoming packet. Blindly trusting false predictions could lead to a competitive ratio worse than the simplest drop-tail algorithm Complete Sharing i.e., the competitive ratio becomes unbounded (∞-robust) if the predictions are mostly false positives.

A single false negative can hurt throughput forever: A naive algorithm that blindly relies on false negative predictions is susceptible to adverse effects that propagate over time. Consider a packet arrival sequence that hits only one queue initially and consider that the predictions are all true negatives until the queue length reaches $B-1$, where B is the total buffer size. At this point, one more packet arrives and our prediction is a false negative. As a result, our naive algorithm has a queue of size B and the optimal algorithm has a queue of size $B-1$. Note that all non-empty queues drain one packet after each timeslot. From

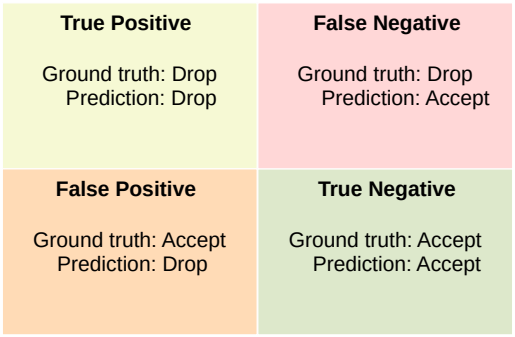


Figure 6.5: Confusion matrix for our prediction model.

here on, in every timeslot, one packet (first) arrives to the large queue and one packet (second) arrives to any other queue. Also consider that all the predictions are true from now on. The optimal algorithm accepts both first and second packet in every timeslot. However, in every timeslot our naive approach can only accept the first packet to the large queue and cannot accept the second packet since the buffer is full. Notice that relying on just one false negative resulted in cumulative drops in this case even though all other predictions were true. In fact, a tiny error such as just N number of false negatives even with all other predictions being true could result in a competitive ratio for a naive approach as worse as Complete Sharing.

■ ***Takeaway.*** *Augmenting drop-tail algorithms with predictions has the potential to unlock the optimal performance. Ensuring performance guarantees with inaccurate predictions remains a challenge.*

6.2 Prediction-Augmented Buffer Sharing

Reflecting on our observations in §6.1, our goal is to design a drop-tail buffer sharing algorithm that performs close to optimal with perfect predictions but also provides a minimum performance guarantee when the prediction error is arbitrarily large. In essence, our aim is to enable performance improvement in terms of throughput and packet drops in datacenter switches. To this end, we first present an overview of our algorithm (§6.2.1). We then present the workings of Credence (§6.2.2) and discuss its properties (§6.2.3). Finally, we discuss the practicality of Credence (§6.2.4).

6.2.1 Overview

In a nutshell, Credence relies on predictions and *follows* a push-out algorithm, reaching close to optimal performance under perfect predictions. Credence cleverly takes certain decisions independent of the predictions in order to guarantee a minimum performance. Further, Credence's competitiveness gradually degrades as prediction error grows (a property known as smoothness [198]), hence the algorithm still performs near-optimally when predictions are slightly inaccurate.

Credence *follows* Longest Queue Drop algorithm: Our design of Credence consists of two key ingredients. First, Credence uses thresholds as a drop condition irrespective of the predictions. Credence treats thresholds as queue

lengths of LQD and updates the thresholds based on the LQD algorithm (simply arithmetic) upon every packet arrival. Second, CREDENCE relies on predictions as long as the queue lengths satisfy the corresponding thresholds. The combination of thresholds and predictions allows CREDENCE to closely *follow* the Longest Queue Drop algorithm (LQD) without requiring push-out operations[2].

Credence guarantees performance under extremities: When all the predictions are perfectly accurate, CREDENCE achieves a competitive ratio of 1.707 (consistency) due to the straight-forward argument that the drops by CREDENCE and LQD are equivalent for true predictions. In order to guarantee a minimum performance even with arbitrarily large prediction error (robustness), CREDENCE bypasses the threshold and predictions as long as the longest queue is within $\frac{B}{N}$ size. Here, B is the buffer size and N is the number of ports. This allows CREDENCE to be most N-competitive even under large prediction error, similar to the Complete Sharing algorithm. We prove our claim formally in § 6.2.3.

Credence smoothly degrades with prediction error: We design our error function in terms of the performance of LQD and the predicted drops. We analyze the types of drops incurred by CREDENCE due to false positive and false negative predictions. This allows us to show that CREDENCE satisfies the smoothness property i.e., the competitive ratio smoothly degrades from 1.707 to N as the prediction error grows.

6.2.2 Credence

We now present CREDENCE and explain how it operates. Algorithm 4 presents the pseudocode of CREDENCE. Our pseudocode is simplified to discrete time for ease of presentation and for simplicity of analysis. It can be trivially extended to continuous time, and our implementation incorporates it[3].

Arrival: Upon a packet arrival, CREDENCE has three important steps that are highlighted in Algorithm 4. First, CREDENCE updates the threshold for the current queue (highlighted in blue). Second, CREDENCE takes a decision based on the thresholds and predictions whether or not to accept the incoming packet (highlighted in yellow). Finally, the packet is either accepted or dropped. We next describe each of these steps in detail. Third, depending on the state of the buffer, CREDENCE bypasses the thresholds and predictions with a safeguard condition in order to accept or drop the incoming packet (highlighted in green).

Thresholds: CREDENCE updates its thresholds based on the longest queue drop algorithm. Specifically, upon a packet arrival at time t to a queue i, CREDENCE increments the threshold $T_i(t)$ for queue i by the packet size. If upon arrival the sum of thresholds $\Gamma(t)$ is equal to the buffer size B, then CREDENCE first decrements the longest queue threshold by packet size and then increments the threshold for queue i by the packet size. Note that upon a packet arrival to a queue, the corresponding threshold is updated before accepting or dropping the packet.

Drop criterion: Similar to existing threshold-based algorithms, CREDENCE also uses thresholds as a drop criterion. CREDENCE compares the queue length $q_i(t)$ of a queue i against its threshold $T_i(t)$ and drops an incoming packet if the queue

[2]Recall that LQD is close to optimal with a competitive ratio of 1.707.

[3]Our source code is available is at https://github.com/inet-tub/ns3-datacenter.

length is larger than or equal to the corresponding threshold; and if the aggregate shared buffer occupancy $Q(t)$ equals the buffer size B. If an incoming packet satisfies the thresholds, then CREDENCE takes input from a machine-learned oracle that predicts whether to accept or drop according to our prediction model discussed in §6.1.3.1. Finally, based on the thresholds and predictions, CREDENCE either accepts or drop the incoming packet.

Safeguard: In order to bound CREDENCE's competitiveness under arbitrarily large prediction error, we bypass the above drop criterion under certain cases. Specifically, when the longest queue length is less than $\frac{B}{N}$, CREDENCE always accepts a packet irrespective of the thresholds and predictions. This ensures that CREDENCE is at least N-competitive even with large prediction error. Our safeguard is based on the observation that even the push-out longest queue drop algorithm cannot push out a packet from a queue less than $\frac{B}{N}$ size since the longest queue must be at least $\frac{B}{N}$ size when the buffer is full. In essence, CREDENCE circumvents the impact of large prediction error by accepting packets until a certain amount of buffer is filled up.

Predictions: CREDENCE can be used with any ML oracle that predicts whether to accept or drop a packet, according to our prediction model (see §6.1.3.1). We do not rely on the internal details of the oracle. However, certain choices of ML oracles are better suited to operate within the limited resources available in a switch hardware. We discuss further on our choice of oracle later in §6.2.4.

6.2.3 Properties of Credence

CREDENCE offers attractive theoretical guarantees in terms of competitive ratio. In this section, for simplicity, we refer an offline optimal algorithm as OPT.

In order to analyze the performane of CREDENCE, we first introduce a new online deterministic algorithm FollowLQD in the non-preemptive model, which is a non-predictive building block of CREDENCE. Intuitively, FollowLQD simply *follows* the Longest Queue Drop (LQD) queues in the preemptive model. In particular, FollowLQD maintains a threshold $T_i(t)$ for each queue at time t. The thresholds are updated for every packet arrival and departure according to LQD in the preemptive model. We present the pseudocode for FollowLQD in Algorithm 5. While FollowLQD tries to *follow* LQD queue lengths by accepting packets as long as the queue lengths are smaller than the thresholds, it may happen that FollowLQD queues are larger than their thresholds. This is since FollowLQD cannot preempt (remove) existing packets in the buffer whereas LQD can preempt and correspondingly the thresholds may drop below the queue lengths. FollowLQD simply drops an incoming packet if it finds that the corresponding queue exceeds its threshold.

Although LQD is known to be 1.707-competitive, we show that FollowLQD is still at least $\frac{N+1}{2}$-competitive. We present our lower bound based on a simple arrival sequence.

Observation 1. *FollowLQD is at least* $\frac{N+1}{2}$*-competitive.*

Proof. We construct an arrival sequence such that for every two packets transmitted by FollowLQD, the offline optimal algorithm OPT transmits $N+1$ packets. Consider that all the queues are empty at time $t = 0$. We then burst packets to a single queue say i until its queue length reaches B. Note that this is possible

Algorithm 4: CREDENCE

Input : Packet arrivals σ, Drop predictions $\phi'(\sigma)$

procedure `arrival`$(\sigma(t))$**:**
  for *each packet* $p \in \sigma(t)$ **do**
    Let i be the destination queue for the packet p
    updateThreshold$(i, arrival)$
    Let j be the longest queue ▷ *Guarantees N-competitiveness*
    if $q_j(t) < \frac{B}{N}$ **then**
      $q_i(t) \leftarrow q_i(t) + 1$ ▷ Accept
      Continue to next packet
    end if
    if $q_i(t) < T_i(t)$ **then** ▷ *Enables 1.707 η-competitiveness*
      if $Q(t) < B$ **then**
        $drop$ = GetPrediction()
        if *drop* **then**
          ▷ Drop
        else
          $q_i(t) \leftarrow q_i(t) + 1$ ▷ Accept
        end if
      end if
    else
      ▷ Drop
    end if
  end for
procedure `departure`(i)**:**
  if $q_i(t) > 0$ **then**
    $q_i(t) \leftarrow q_i(t) - 1$ ▷ Drain one packet
  end if
  updateThreshold$(i, departure)$
procedure `updateThreshold`$(i,\ event)$**:**
  if *event* $=$ *arrival* **then**
    ▷ *Thresholds are treated as LQD queue lengths*
    if $\Gamma(t) = B$ **then** ▷ Sum of thresholds
      Let $T_j(t)$ be the highest threshold
      $T_j(t) \leftarrow T_j(t) - 1$ ▷ Decrease
      $T_i(t) \leftarrow T_i(t) + 1$ ▷ Increase
    else
      $T_i(t) \leftarrow T_i(t) + 1$ ▷ Increase
      $\Gamma(t) \leftarrow \Gamma(t) + 1$
    end if
  end if
  if *event* $=$ *departure* **then**
    if $T_i(t) > 0$ **then**
      $T_i(t) \leftarrow T_i(t) - 1$ ▷ Decrease
      $\Gamma(t) \leftarrow \Gamma(t) - 1$
    end if
  end if

Algorithm 5: FollowLQD

```
Input  : σ(t)
procedure ARRIVAL(σ(t)):
    for each packet p ∈ σ(t) do
        Let i be the destination queue for the packet p
        updateThreshold(i, arrival)
        if q_i(t) < T_i(t) then
            if Q(t) < B then
                q_i(t) ← q_i(t) + 1                          ▷ accept
            end if
        else
                                                             ▷ Drop
        end if
    end for
procedure DEPARTURE(i):
    if q_i(t) > 0 then
        q_i(t) ← q_i(t) − 1                                  ▷ Drain one packet
    end if
    updateThreshold(i, departure)
function updateThreshold(i, event):
    if event = arrival then
        if Γ(t) = B then                                     ▷ Sum of thresholds
            Let T_j(t) be the largest threshold
            T_j(t) ← T_j(t) − 1                              ▷ Decrease
            T_i(t) ← T_i(t) + 1                              ▷ Increase
        else
            T_i(t) ← T_i(t) + 1                              ▷ Increase
            Γ(t) ← Γ(t) + 1
        end if
    end if
    if event = departure then
        if T_i(t) > 0 then
            T_i(t) ← T_i(t) − 1                              ▷ Decrease
            Γ(t) ← Γ(t) − 1
        end if
    end if
```

since the threshold for queue i that follows the corresponding LQD queue also grows up to B. At the end of the departure phase, FollowLQD transmits one packet and the queue length becomes $B-1$. At this point, we send N packets, one packet to each of the N queues. The thresholds are updated based on LQD, which has the following actions: *(i)* preempt $N-1$ packets from queue i and *(ii)* accept all N packets to N queues. Correspondingly, the threshold for queue i of FollowLQD drops to $B-N+1$ but it still has $B-1$ packets in queue i. As a result, it can only accept one packet out of the N incoming packets. At the end of the departure phase during this timeslot, FollowLQD has $B-1$ packets in queue i and has transmitted 1 packet in total. In the next timeslot, we send N packets to the queue i so that LQD's queue i now gets back to size B again.

As the threshold is larger than the queue length $(B-1)$, FollowLQD accepts 1 packet. At the end of the departure phase, FollowLQD transmits 1 packet from the queue i. Overall, FollowLQD transmitted 2 packets but OPT transmitted $N+1$ packets. We then repeat the sequence such that for every $N+1$ packets transmitted by OPT, FollowLQD transmits 2 packets. The competitive ratio is then at least $\frac{N+1}{2}$. □

Although we have so far discussed the prediction error more intuitively, it requires a quantitative measure in order to analyze the performance of an algorithm relying on predictions. There are two important considerations in defining a suitable error function. First, following the literature, an error function must be independent of the state and actions of our algorithm, so that we can train a predictor without considering all possible states of the algorithm [134]. Second, it is desirable that the performance of our algorithm can be related to the error function in an uncomplicated manner. Taking these into consideration, we define our error function in the following.

Recall that our prediction oracle predicts packet drop (or accept) for each packet in the arrival sequence σ, according to the prediction model introduced in §6.1.3.1. We denote the drop sequence of LQD for the arrival sequence σ by $\phi(\sigma)$, and the predicted drop sequence by $\phi'(\sigma)$. We classify prediction for each packet in to four types: true positive, false positive, true negative and false negative (see Figure 6.5). We denote the sequence of true positive predictions by $\phi'_{TP}(\sigma)$, false positive predictions by $\phi'_{FP}(\sigma)$, true negative predictions by $\phi'_{TN}(\sigma)$ and false negative predictions by $\phi'_{TP}(\sigma)$. We drop σ in our notations when the context is clear.

Hereafter, we mainly compare our online non-preemptive algorithm with predictions against online LQD (preemptive). We now formally define the error made by the oracle by the following error function.

Definition 5 (Error function)**.** *Let LQD(σ) and FollowLQD(σ) denote the total number of packets transmitted by the online push-out algorithm LQD and the online drop-tail algorithm FollowLQD over the arrival sequence σ. Let ϕ denote the sequence indicating drop by LQD for each packet in the arrival sequence σ. Let ϕ' denote the sequence of drops predicted by the machine-learned oracle. Let ϕ'_{TP}, ϕ'_{FP}, ϕ'_{TN}, and ϕ'_{FN} denote the sequence of true positive, false positive, true negative and false negative predictions for the arrival sequence σ. We define the error function $\eta(\phi, \phi')$ as follows:*

$$\eta(\phi, \phi') = \frac{LQD(\sigma)}{FollowLQD\left(\sigma - \phi'_{TP} - \phi'_{FP}\right)} \tag{6.2}$$

Our definition captures the prediction error in terms of the performance of LQD (push-out) and the performance of an algorithm $FollowLQD$. Here, $FollowLQD$ (Algorithm 5) is a deterministic drop-tail algorithm (without predictions) with thresholds similar to Credence.

We now analyze Credence that relies on drop predictions ϕ' and takes decisions in pursuit of *following* LQD more accurately. In the following, we study the competitive ratio of Credence as the error grows. We obtain the competitive ratio as a function of the error η: we show that Credence is 1-competitive against LQD with perfect predictions but at most N-competitive when the error is arbitrarily large.

Theorem 12. *The competitive ratio of* CREDENCE *grows linearly from* 1.707 *to* N *based on the prediction error* $\eta(\phi, \phi')$*, where* N *is the number of ports,* ϕ *is the drop sequence of LQD and* ϕ' *is the predicted sequence of drops i.e., the competitive ratio is at most* $\min(1.707\ \eta(\phi, \phi'), N)$.

Using Definition 5, we analyze the throughput of CREDENCE over an entire packet arrival sequence σ based on the predictions ϕ'. In fact, our error function is upper bounded by an intuitive closed form expression, in terms of the number of true and false predictions, as follows, that can be easily computed.

Lemma 3. *Let* ϕ' *denote the sequence of drops predicted by the machine-learned oracle. Let* ϕ'_{TP}, ϕ'_{FP}, ϕ'_{TN}, *and* ϕ'_{FN} *denote the sequence of true positive, false positive, true negative and false negative predictions for the arrival sequence* σ*. The error function* $\eta(\phi, \phi')$ *(Definition 5) is upper upper bounded as follows:*

$$\eta(\phi, \phi') \leq \frac{\phi'_{TN} + \phi'_{FP}}{\phi'_{TN} - \min\left((N-1) \cdot \phi'_{FN}, \phi'_{TN}\right)}$$

Proof. Our proof is based on two arguments: *(i)* $LQD(\sigma) = \phi'_{TN} + \phi'_{FP}$ and *(ii)* $FollowLQD(\sigma - \phi'_{TP} - \phi'_{FP}) \geq \phi'_{TN} - \min\left((N-1) \cdot \phi'_{FN}, \phi'_{TN}\right)$.

First, $LQD(\sigma)$ is the total number of transmitted packets by LQD. Recall that the ground-truth (transmitted by LQD) for a prediction is an accept if and only if the prediction is either true negative or a false positive. Hence, the total number of packets transmitted by LQD i.e., $LQD(\sigma)$ is the sum of true negative predictions and the false positive predictions.

Second, $FollowLQD(\sigma - \phi'_{TP} - \phi'_{FP})$ transmits at least $\phi'_{TN} + \phi'_{FN} - Y$, where Y is the total number of drops caused by false negative predictions. Note that $\sigma = \phi'_{TN} + \phi'_{FN} + \phi'_{FP} + \phi'_{TP}$. The proof follows by showing that each false negative results in at most N extra drops due to the buffer limit. Further, these extra drops *must* be true negative predictions since we have already removed positive predictions from our arrival sequences (i.e., we assume at most all the positive predictions have be dropped). Additionally, since the extra drops are true negative predictions, it implies that LQD transmits those packets but although our prediction is true, we incur additional drop due to the buffer limit. For each false negative, there can be at most one drop in a single timeslot for up to $N-1$ distinct timeslots such drops that LQD accepts and transmits those packets but FollowLQD drops them. Beyond $N-1$ drops, there can only be at most 1 other drop upon which the existence of an additional packet (false negative) in FollowLQD's buffer would be nullified. This is since, during the initial $N-1$ drops, FollowLQD could not accept the incoming packet but after the transmission phase, the queues having false negative predictions decrement their size by 1. This leaves at least $N-1$ packets free space in FollowLQD after $N-1$ drops and LQD also has the same remaining space after those extra $N-1$ accepted by LQD are also transmitted. At this time, both LQD and FollowLQD have the same remaining space and they also transmit the same number of packets in each timeslot. One additional drop by FollowLQD corresponding to a false negative is still possible due the thresholds i.e., if there exists a packet arrival to the queue having false negative, the incoming packet is dropped since the existence of false negatives implies that the queue length is large than the threshold. As a result, there are at most N drops by FollowLQD for each false negative prediction.

The proof follows by the above two arguments. □

The upper bound of our error function indicates intuitively that *(i)* the error decreases as the total number of true negative predictions dominate the total false predictions, *(ii)* the error increases with each false positive prediction and *(iii)* the error increases with each false negative with a larger weight.

Next, Lemma 4 states the relation between the throughput of CREDENCE, throughput of *LQD* and the prediction error.

Lemma 4. *The total number of packets transmitted by CREDENCE for an arrival sequence σ, a drop sequence ϕ by LQD and the predicted drop sequence ϕ' is given by*

$$\textsc{Credence}(\sigma) \geq \frac{LQD(\sigma)}{\underbrace{\eta(\phi, \phi')}_{error}} \tag{6.3}$$

Proof. For simplicity, we refer to CREDENCE as ALG in the following. We prove our claim by analyzing the drops of ALG and relating the transmitted packets by $ALG(\sigma)$ to $LQD(\sigma')$. Every drop of *ALG* arises from three types of situations. First, ALG can drop a packet due to the thresholds. Note that the thresholds used by *ALG* correspond to the queue lengths of preemptive LQD over the same arrival sequence σ. As a result, both ALG and FollowLQD algorithm have the same thresholds at any time instance. Second, ALG drops a packet if the prediction is either true positive or false positive if and only if the queue length satisfies the corresponding thresholds. This type of drops are at most the total number of true positive and false positive predictions. Third, ALG drops a packet when the buffer is full which is the same condition for FollowLQD. In essence, ALG drops at most all the positive predictions and drops at most the number of packets dropped by FollowLQD serving the arrival sequence $\sigma - \phi'_{TP} - \phi'_{FP}$ i.e., the arrival sequence in which all the packets predicted as positive are removed from σ. In order to prove our main claim, it remains to argue that the extra packets accepted by ALG due to the safeguard condition do not result in additional drops compared to FollowLQD with the arrival sequence $\sigma - \phi'_{TP} - \phi'_{FP}$. For every packet that fails to satisfy the threshold but gets accepted due to the safeguard condition by ALG, could cause at most one extra drop due to the thresholds before the buffer full again compared to FollowLQD. This is since, if FollowLQD accepts a packet, then its queue length is certainly less that the corresponding threshold (that is same for ALG). However, the queue length of ALG may have some extra packets that are accepted due to the safeguard condition. As a result, each such extra packet (dropped by FollowLQD) contributes to at most one drop compared compared to FollowLQD and the transmitted packets remains equivalent. Further, by the time the buffer is full in ALG, all the extra packets accepted due to the safeguard condition would have been drained out of the buffer. This is due to the fact that any such extra packet is at a queue length of at most $\frac{B}{N}$ (the safeguard condition) that drains out before the buffer fills up i.e., it takes at least $\frac{B}{N}$ timeslots to fill the buffer (only N packets can arrive in each timeslot). As a result, ALG transmits at least the total number of packets transmitted by FollowLQD over the arrival sequence $\sigma - \phi'_{TP} - \phi'_{FP}$ i.e.,

$$ALG(\sigma) \geq FollowLQD(\sigma - \phi'_{TP} - \phi'_{FP})$$

Using Definition 5, we express FollowLQD in terms of LQD and the error function $\eta(\phi, \phi')$, and obtain Equation 6.3. □

Equation 6.3 shows that the throughput of CREDENCE reaches closer to (moves away from) *LQD* as the prediction error becomes smaller (larger). Recall that CREDENCE bypasses the drop criterion and accepts packets through a safeguard condition under certain cases (see §6.2.2). Based on this, we obtain another bound for the throughput of CREDENCE in Lemma 5, which is independent of the prediction error.

Lemma 5. *CREDENCE transmits at least $\frac{1}{N}$ times the number of packets transmitted by an offline optimal algorithm OPT i.e., $\textsc{Credence}(\sigma) \geq \frac{1}{N} \cdot OPT(\sigma)$.*

Lemma 5 shows that irrespective of the prediction error (even under large error), CREDENCE can always transmit at least $\frac{1}{N}$ fraction of the packets transmitted by an optimal solution. Our proof of Lemma 5 is based on the fact that upon a drop by CREDENCE, there is at least one queue with length $\frac{B}{N}$ (the safeguard condition). As a result, for every B packets transmitted by OPT, there are at least $\frac{B}{N}$ number of packets transmitted by CREDENCE over the arrival sequence σ. This leads us to the bound expressed in Lemma 5.

Proof. Irrespective of the predictions, CREDENCE always accepts an incoming packet if the longest queue is less than or equal to $\frac{B}{N}$. When CREDENCE drops a packet, there is at least one queue that has at least $\frac{B}{N}$ number of packets. Hence, every packet in OPT can be matched to at least $\frac{B}{N}$ number of packets. Consequently, the competitive ratio is at most N. □

We are now ready to prove our main claim (Theorem 12) using the above results.

Proof of Theorem 12. From Definition 4, in order to prove the competitive ratio of our CREDENCE, we are mainly concerned with the upper bound of $\frac{OPT(\sigma)}{\textsc{Credence}(\sigma)}$ for any arrival sequence σ. Since $\frac{OPT(\sigma)}{LQD(\sigma)} \leq 1.707$ is known from literature [44, 119], we use this result to compare CREDENCE and LQD in order to argue about the competitive ratio i.e, $\frac{OPT(\sigma)}{\textsc{Credence}(\sigma)} \leq 1.707 \cdot \frac{LQD(\sigma)}{\textsc{Credence}(\sigma)}$ for any request sequence σ. From Lemma 4, we have the following:

$$\frac{LQD(\sigma)}{\textsc{Credence}(\sigma)} \leq \eta(\phi, \phi')$$

From Lemma 5, irrespective of the predicted sequence, we have that $\frac{OPT(\sigma)}{\textsc{Credence}(\sigma)} \leq N$. Finally, since $\frac{OPT(\sigma)}{\textsc{Credence}(\sigma)} \leq 1.707 \cdot \frac{LQD(\sigma)}{\textsc{Credence}(\sigma)}$, the competitive ratio of CREDENCE is given by:

$$\frac{OPT(\sigma)}{\textsc{Credence}(\sigma)} \leq \min\left(1.707\ \eta(\phi, \phi'),\ N\right)$$

The proof follows by Definition 4. □

Theorem 12 essentially shows how CREDENCE's competitive ratio in terms of throughput improves from N to 1.707 as the prediction error (Definition 5) decreases. Interestingly, CREDENCE's competitive ratio is independent of the buffer size B i.e., CREDENCE is compatible for shallow buffers as well as deep buffers. We note that our analysis compares an algorithm against an optimal offline algorithm

over a fixed packet arrival sequence. This allows us to analyze the competitive ratio via an error function defined over the corresponding arrival sequence. However, real-world traffic is responsive in nature due to congestion control and packet retransmissions. Although we have used η as our error function to express the competitiveness of CREDENCE, in our evaluation (§6.3), we compare CREDENCE with state-of-the-art approaches under realistic datacenter workloads and we also present the quality of our predictions using more natural error functions that are widely used for machine learning models.

6.2.4 Practicality of Credence

CREDENCE's algorithm itself is simple and close to complexity of the longest queue drop (push-out). However, the machine-learned oracle producing the predictions adds additional complexity in order to deploy CREDENCE on switches. Overall, there are three main parts of CREDENCE that contribute to additional complexity in terms of memory and computation: *(i)* finding the longest queue (and its threshold), *(ii)* remembering thresholds and *(iii)* obtaining predictions.

Finding the longest queue (and its threshold): For every packet arrival, CREDENCE requires finding the longest queue for the safeguard condition described in §6.2.2. Additionally, CREDENCE requires finding the largest threshold during the threshold updates upon every packet arrival. The maximum value search operation has a run-time complexity of $\mathcal{O}(N)$, where N is the number of ports. Note that typical datacenter switches have a relatively small number of ports e.g., 64 ports in Broadcom Tomahawk4 [75]. Prior work in the context of LQD proposes an approximation to further reduce the complexity of finding the longest queue [253] to $\mathcal{O}(1)$. The average case complexity can further be reduced by only maintaining the list of queue lengths (and their thresholds) that are larger than $\frac{B}{N}$. This is sufficient since the safeguard condition checks whether the longest queue is less than $\frac{B}{N}$, which is the same as checking that no queue is longer than $\frac{B}{N}$. Similarly, the largest threshold search during the threshold updates is only triggered when the buffer is full. In this case, the longest queue must be at least $\frac{B}{N}$. Given that switches are becoming more and more computationally capable, we believe that a basic function such as finding the maximum value in a small list is feasible to implement within the available resources.

Thresholds memory: In contrast to existing threshold-based algorithms, CREDENCE's thresholds depend on their previous value i.e., thresholds must be remembered. As a result, CREDENCE adds a small memory overhead of $\mathcal{O}(N)$ for the thresholds. The threshold calculations are in fact much simpler than existing schemes and do not add any further computational complexity since CREDENCE only requires adding and subtracting the threshold values by the packet size.

Predictions: Our prediction model (drop or accept) essentially boils down to binary classification problem. To this end, numerous ML techniques exist ranging from linear classifiers to more advanced neural networks. In view of practicality, we consider random forests as they are implementable in programmable hardware [30, 76, 140]. In order to reduce the prediction latency, we also limit the number of trees and the maximum depth of our trained random forest model. We find that, even a model trained with a maximum depth of four, and as low as four to eight trees achieves reasonable prediction error (precision ≈ 0.65). Further, to reduce the complexity of the model, we also limit the number of features to four: queue

length, total shared buffer occupancy and their corresponding moving averages (exponentially weighted) over one round-trip time (baseRTT).

The fundamental blocks required for CREDENCE are all individually practical in today's hardware. Unfortunately, modifying the buffer sharing algorithm and integrating it with predictions requires switch vendor support. Even in programmable switches, the traffic manager is merely a blackbox that implements Dynamic Thresholds with a single parameter exposed to the user [38, 45]. Given the superior performance of CREDENCE (§6.3), we wish to gain attention from switch vendors to discuss further on the implementation of CREDENCE.

6.3 Evaluation

We evaluate the performance of CREDENCE and compare it against state-of-the-art buffer sharing algorithms in the context of datacenter networks. Our evaluation aims at answering three main questions:

(Q1) *Does* CREDENCE *improve the burst absorption?*
Our evaluation shows that CREDENCE significantly improves the burst absorption capabilities of switches. We find that CREDENCE improves the 95-percentile flow completion times for incast flows by up to 95.4% compared to Dynamic Thresholds (DT) and by up to 96.9% compared to ABM.

(Q2) *Can* CREDENCE *improve the flow completion times for short flows as well as long flows?*
We find that CREDENCE performs similar to existing approaches in terms of 95-percentile flow completion times for short flows and improves upon ABM by up to 22% correspondingly for long flows.

(Q3) *How does prediction error impact the performance of* CREDENCE *in terms of flow completion times?*
We increase the error of our prediction by artificially flipping the predictions with a probability. As the probability increases (error increases), we find that CREDENCE sustains performance up to 0.01 probability and smoothly degrades in performance beyond 0.01.

6.3.1 Setup

Our evaluation is based on packet-level simulations in NS3 [209]. We embed a Python interpreter within NS3 using pybind11 [224] in order to obtain predictions from a random forest model trained with scikit-learn [236].

Topology: We consider a leaf-spine topology with 256 servers organized into 4 spines and 16 leaves. Each link has a propagation delay of 3μs leading to a round-trip-time of 25.2μs. The capacity is set to 10Gbps for all the links leading to $4:1$ oversubscription similar to prior works [16, 21, 232]. All the switches in our topology have 5.12KB buffer-per-port-per-Gbps similar to Broadcom Tomahawk [75].

Workloads: We generate traffic using websearch [35] flow size distribution that is based on measurements from real-world datacenter workloads. We vary the load on the network in the range 20-80%. We additionally generate traffic using a synthetic incast workload similar to prior work [16]. Our incast workload mimics the query-response behavior of a distributed file storage system where each query results in a bursty response from multiple servers. We set the query request rate

to 2 per second from each server, and we vary the burst size in the range 10-100%[4] of the switch buffer size. We use DCTCP [35] and PowerTCP [21] as transport protocols.

Comparisons & metrics: We compare CREDENCE with Dynamic Thresholds [86] (the default algorithm in datacenter switches), ABM [16] and LQD (push-out). Hereafter, we refer to Dynamic Thresholds as DT. We report four performance metrics: 95-percentile flow completion times for short flows ($\leq$ 100KB), incast flows (incast workload), long flows ($\geq$ 1MB), and the 99-percentile shared buffer occupancy. We present the CDFs of flow completion times in §6.4.

Predictions: We collect packet-level traces from each switch in our topology when using LQD (push-out) as the buffer sharing algorithm. Each trace consists of five values corresponding to each packet arrival: *(i)* queue length, *(ii)* average queue length, *(iii)* shared buffer occupancy, *(iv)* average shared buffer occupancy and *(v)* accept (or drop). We train a random forest classifier using queue length and shared buffer occupancy as features and the model predicts packet drops. We set the maximum depth for each tree to 4 in view of practicality. At a train-test split 0.6 of our LQD trace, based on our parameter sweep across the number of trees used for our classifier (Figure 6.15 in §6.4), we set the number of trees to 4. We observe that the quality of our predictions does not improve significantly beyond four trees in our datasets. This gives us an accuracy of 0.99, error score $\frac{1}{\eta}$ 0.99[5] (inverse of our error function based on Definition 5), precision of 0.65, recall of 0.35 and F1 score of 0.45. We train our model with an LQD trace corresponding to websearch workload at 80% load, and a burst size of 75% buffer size for the incast workload, using DCTCP as the transport protocol. We use the same trained model in all our evaluations. We ensure that our test scenarios are different from the training dataset by using different random seeds in addition to different traffic conditions (different loads and different burst sizes) in each experiment in our evaluation.

For completeness, although well-known in the literature, we define accuracy, precision, recall and f1 score below.

$$Accuracy = \frac{\phi'_{TP} + \phi'_{TN}}{\phi'_{TP} + \phi'_{TN} + \phi'_{FP} + \phi'_{FN}}$$

$$Precision = \frac{\phi'_{TP}}{\phi'_{TP} + \phi'_{FP}}$$

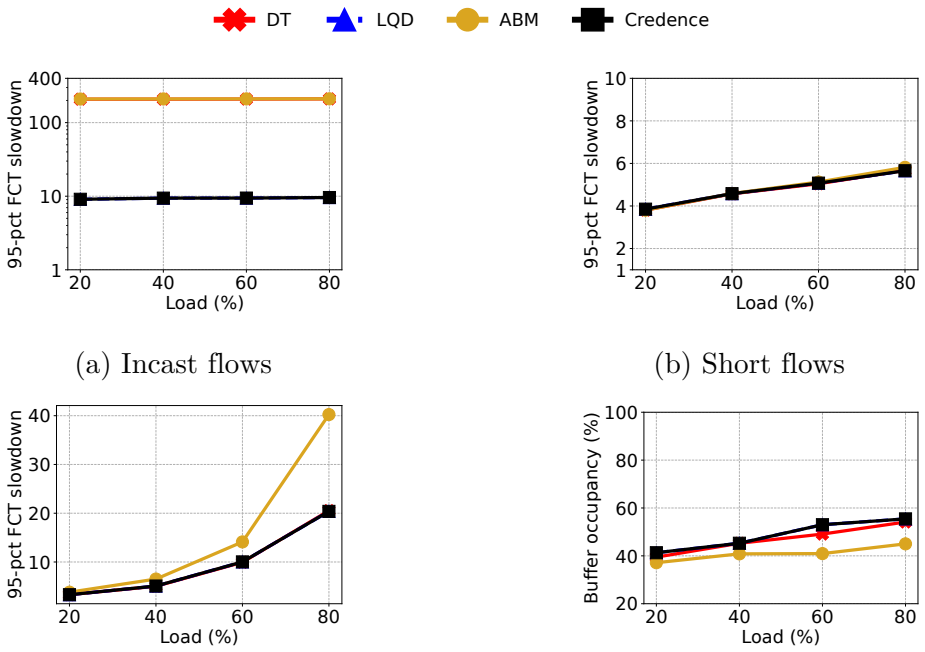
$$Recall = \frac{\phi'_{TP}}{\phi'_{TP} + \phi'_{FN}}$$

$$F1\ score = \frac{2 \cdot \phi'_{TP}}{2 \cdot \phi'_{TP} + \phi'_{FP} + \phi'_{FN}}$$

Configuration: CREDENCE is parameter-less, and it takes input from an oracle (described above) that predicts packet drops. We set $\alpha = 0.5$ for DT and ABM similar to prior work [16]. ABM uses $\alpha = 64$ for all the packets which arrive

[4]We note that if the burst size exceeds the buffer size (> 100%), then no buffer sharing algorithm can prevent excessive packet drops. As such, controlling and mitigating the extent of incast scenarios can be better addressed by congestion control and scheduling techniques.

[5]The high values of accuracy and our error score $\frac{1}{\eta}$ are attributed to the dataset being skewed i.e., congestion is not persistent.



(a) Incast flows

(b) Short flows

(c) Long flows

(d) Shared buffer occupancy

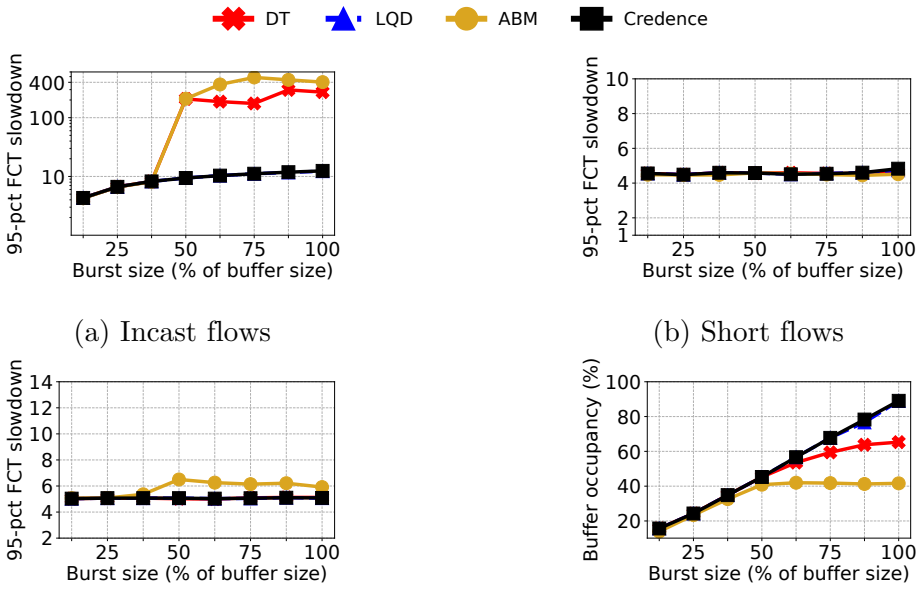
Figure 6.6: Performance of CREDENCE across various loads of websearch workload and incast workload at a burst size 50% of the buffer size, with DCTCP as the transport protocol. As the load increases, ABM penalizes long flows. DT and ABM are unable to absorb bursts of size 50% of the buffer size. CREDENCE achieves superior burst absorption and does not penalize long flows.

during the first round-trip-time [16]. We configure DCTCP according to [35] and PowerTCP according to [21].

6.3.2 Results

Credence significantly improves burst absorption: In Figure 6.6a, using DCTCP as the transport protocol, we generate websearch traffic across various loads in the range 20-80% and generate incast traffic with a burst size of 50% buffer size. We observe that CREDENCE performs close to the optimal performance of LQD. CREDENCE improves the 95-percentile flow complete times for incast flows by 95.50% compared to DT, and by 95.53% compared to ABM. In Figure 6.7a, we set the load of websearch traffic at 40% and vary the burst size for incast workload in the range 10-100% buffer size. CREDENCE performs similar to DT and ABM for small burst sizes. As the burst size increases, CREDENCE improves the 95-percentile flow completion times for incast workload by 95% on average compared to DT, and by 96.92% on average compared to ABM. In Figure 6.8a, even when using PowerTCP as the transport protocol, we see that CREDENCE improves the 95-percentile flow completion times for incast flow by 93.27% on average compared to DT and by 93.36% on average compared to ABM. In terms of burst absorption, both DT and ABM are drop-tail algorithms, hence they face the drawbacks discussed in §6.1.2. CREDENCE relies on predictions and unlocks the performance of LQD (push-out) as shown by our results in Figure 6.6a and Figure 6.7a.

Credence improves long flows FCTs: CREDENCE not only improves the



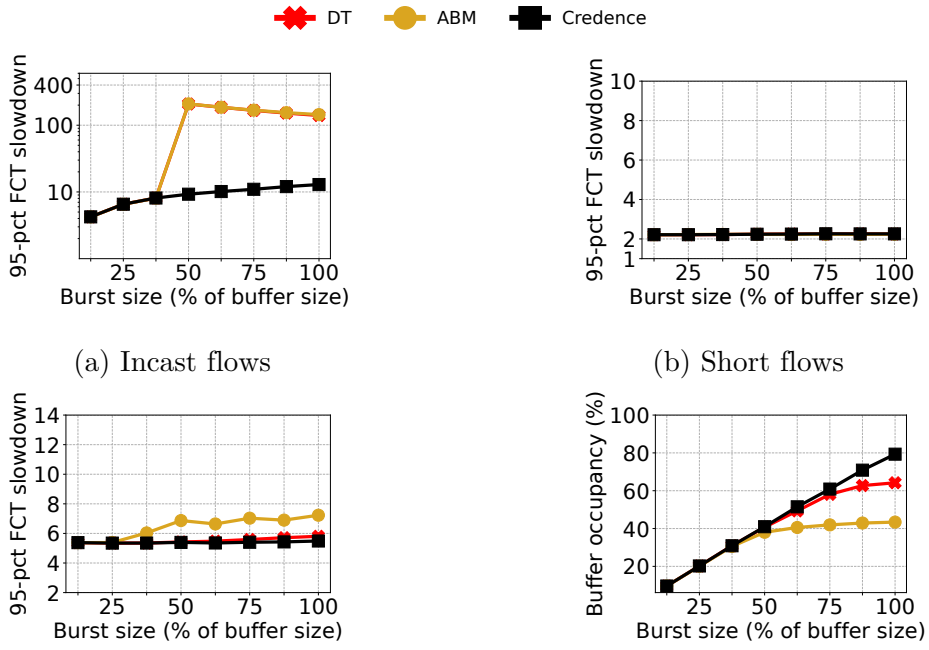
(a) Incast flows

(b) Short flows

(c) Long flows

(d) Shared buffer occupancy

Figure 6.7: Performance of CREDENCE across various burst sizes of incast workload and websearch workload at 40% load, with DCTCP as the transport protocol. At small burst sizes, DT and ABM achieve similar performance compared to CREDENCE but as the burst size increases, CREDENCE outperforms DT and ABM in terms of FCTs for incast flows (burst absorption).



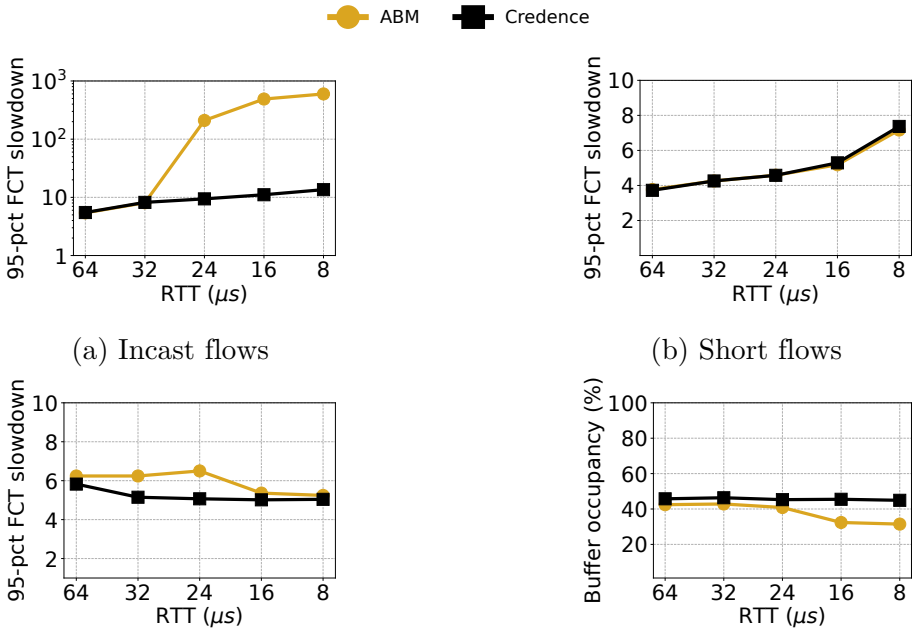
(a) Incast flows

(b) Short flows

(c) Long flows

(d) Shared buffer occupancy

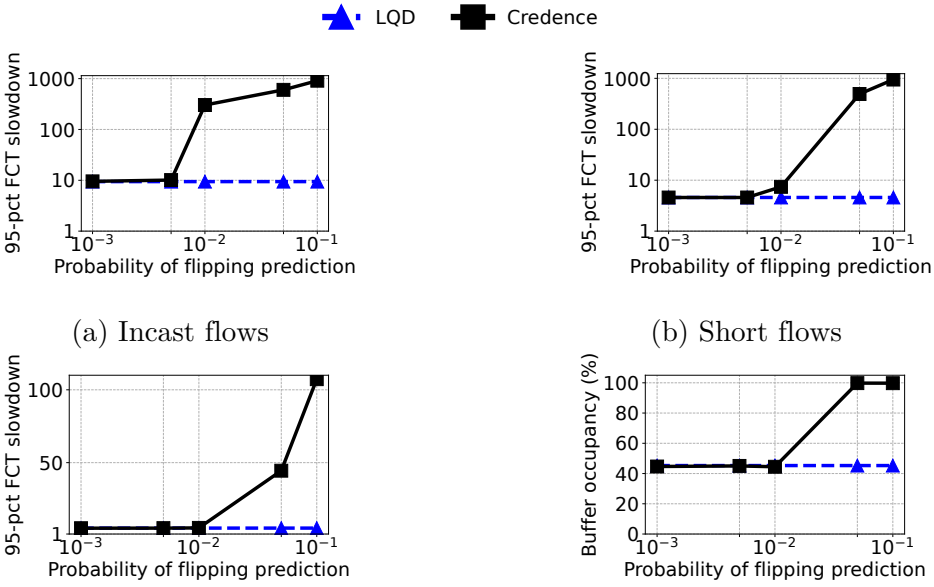
Figure 6.8: Performance of CREDENCE across various burst sizes of incast workload and websearch workload at 40% load, with PowerTCP as the transport protocol. Even with advanced congestion control, DT and ABM only benefit in terms of FCTs for long flows, but CREDENCE outperforms in terms of FCTs for incast flows (burst absorption) as well as FCTs for long flows.



(a) Incast flows (b) Short flows

(c) Long flows (d) Shared buffer occupancy

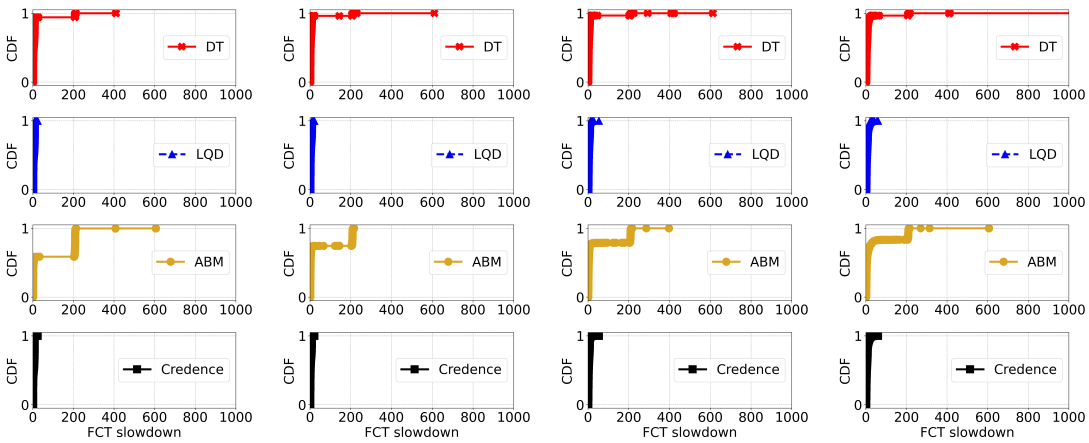
Figure 6.9: ABM is sensitive to RTT and performs significantly worse compared to CREDENCE at low RTTs. At high RTTs, ABM performs similar to CREDENCE.



(a) Incast flows (b) Short flows

(c) Long flows (d) Shared buffer occupancy

Figure 6.10: Even though the predictions from our random forest classifier are intentionally flipped (to increase error), CREDENCE performs close to LQD up to 0.005 flipping probability but smoothly diverges from LQD at 0.01 flipping probability.



(a) Load = 20% (b) Load = 40% (c) Load = 60% (d) Load = 80%

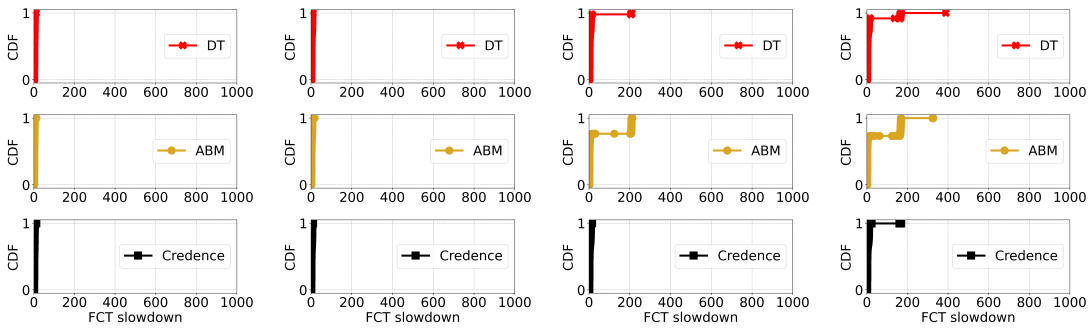
Figure 6.11: CDF of flow completion times (slowdown) for CREDENCE, DT, ABM and LQD across various loads of websearch workload and incast workload at a burst size 50% of the buffer size, with DCTCP as the transport protocol.

burst absorption but also improves the flow completion times for long flows. In Figure 6.6c, at 50% burst size for incast workload and across various loads of websearch workload, we observe that CREDENCE performs similar to DT in terms of 95-percentile flow completion times for long flows and improves upon ABM on average by 28.49%. At 80% load, CREDENCE improves upon ABM by 49.34%. Across various burst sizes, and at 40% load of websearch workload, we observe from Figure 6.7c that CREDENCE improves upon ABM by up to 22.02% and by 12.02% on average. With PowerTCP as the transport protocol (Figure 6.8c), CREDENCE improves the 95-percentile flow completion times for long flows by 3.31% on average compared to DT and by 17.35% compared to ABM. At a burst size of 100% buffer size, CREDENCE improves the flow completion times by 5.49% compared to DT and by 24.09% compared to ABM. As described in §6.1.2, drop-tail algorithms such as DT and ABM cannot effectively navigate proactive and reactive drops, resulting in throughput loss i.e., high flow completion times for long flows. In contrast, predictions guide CREDENCE to effectively navigate proactive and reactive drops. This allows CREDENCE to achieve better flow completion times even for long flows.

Credence does not waste buffer resources: In anticipation of future burst arrivals, both DT and ABM buffer resources. We show the 99.99-percentile buffer occupancies[6] in Figure 6.6d for various loads of websearch workload and at a burst size of 50% buffer size for incast workload. We observe that, DT (ABM) utilizes 3.77% (18.68%) lower buffer space on average compared to CREDENCE, at the cost of increased flow completion times even for long flows. Even as the burst size increases (Figure 6.7d), DT and ABM are unable to efficiently utilize the buffer space. In contrast, CREDENCE efficiently utilizes the available buffer space as the burst size increases, improving burst absorption without sacrificing flow completion times for long flows.

ABM is sensitive to RTT: Although ABM is expected to outperform DT, our evaluation results especially in terms of flow completion times for incast flows

[6]DT, ABM and CREDENCE have similar tail occupancies (100-percentile) that occurs at rare congestion events in our simulations.



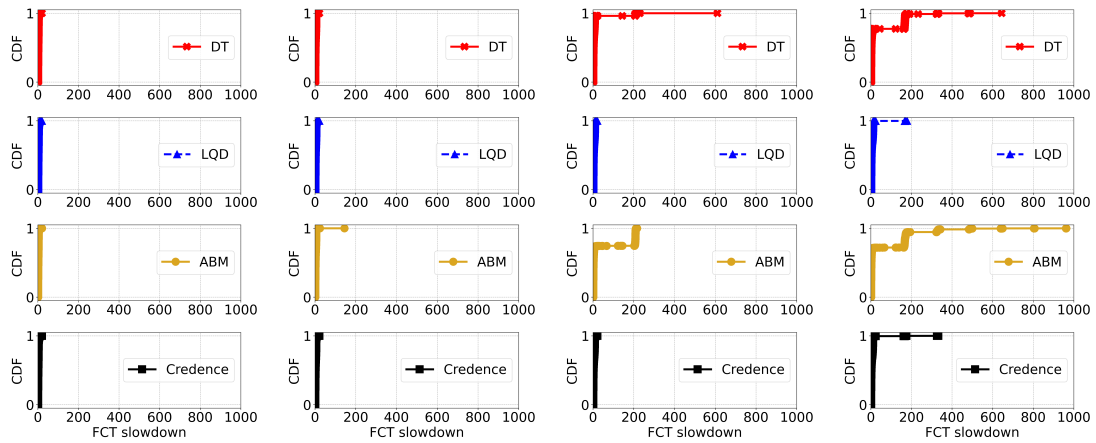
(a) Burst size = 12.5% (b) Burst size = 25% (c) Burst size = 50% (d) Burst size = 75%

Figure 6.12: CDF of flow completion times (slowdown) for CREDENCE, DT, ABM and LQD across various burst sizes of incast workload and websearch workload at 40% load, with PowerTCP as the transport protocol. Burst size is expressed as a percentage of the buffer size.

contradict the results presented in [16]. We ran several simulations varying all the parameters in our setup in order to better understand the performance of ABM. We found that ABM is in fact sensitive to round-trip-time (RTT). We vary the base RTT of our topology in Figure 6.9 and compare CREDENCE with ABM. At high RTTs, we observe that ABM performs close to CREDENCE, but degrades in performance as RTT decreases. Specifically, at 8μs RTT, ABM performs 97.73% worse compared to CREDENCE in terms of flow completion times for incast flows. Although ABM achieves on-par flow completion times for short flows, we observe that ABM degrades in flow completion times for long times as well as under-utilizes the buffer as RTT decreases. The poor performance of ABM at low RTTs is due to the fact that ABM prioritizes the first RTT packets and considers the rest of the traffic as steady-state traffic. However, it is not uncommon that datacenter switches experience bursts for several RTTs [108]. Further, congestion control algorithms require multiples RTTs to converge to steady-state. In contrast, CREDENCE is parameter-less and does not make such assumptions. CREDENCE performs significantly better than existing approaches even with an off-the-shelf machine-learned predictor with a simple model.

Credence gradually degrades with prediction error: Our random forest classifier that we used in our evaluations so far, has a precision close to 0.65. In order to evaluate the performance of CREDENCE with even worse prediction error, we artificially introduce error by flipping every prediction obtained from our random forest classifier with a certain probability. We consider LQD (push-out) as a baseline since CREDENCE is expected to perform close to LQD and degrade as the prediction error grows large. Figure 6.10 presents our evaluation results, under websearch workload at 40% and burst size 50% of the buffer size for incast workload. At 0.001 flipping probability, CREDENCE performs close to LQD. However, at 0.01 flipping probability CREDENCE starts to diverge[7] from LQD and gets significantly worse at 0.1 flipping probability. Figure 6.10 gives practical insights into smoothness of CREDENCE in addition to our analysis.

[7]The extent of divergence in FCT slowdown relates to minRTO (set to 10 ms) due to packet drops that result in timeouts, and is not explicitly bounded by $O(N)$ similar to throughput. Yet, incast and short flows can be protected by incorporating packet priorities (discussed in §6.6).



(a) Burst size = 12.5% (b) Burst size = 25% (c) Burst size = 50% (d) Burst size = 75%

Figure 6.13: CDF of flow completion times (slowdown) for Credence, DT, ABM and LQD across various burst sizes of incast workload and websearch workload at 40% load, with DCTCP as the transport protocol. Burst size is expressed as a percentage of the buffer size.

6.4 Additional Results

In this section, we present additional results from our evaluations. Figures 6.13, 6.11, 6.12 present the CDF of flow completion times for each experiment in our evaluations (§6.3), showing the complete performance profile of each algorithm.

Figure 6.14 presents our numerical results based on a custom simulator in discrete time. Note that Figure 6.14 shows the throughput *ratio* of an algorithm vs LQD. We perform this experiment using custom simulator in order to fully control the prediction error (artificially).

We generate large bursts of the size of the total buffer, where each such burst arrives according to a poisson process (which is fixed in subsequent runs). We then collect a trace of per-packet drop (or accept) trace using LQD as the buffer sharing algorithm. This trace serves as the ground-truth as well as the case for perfect predictions for Credence. We then run Credence over the same packet arrival sequence from above, and use the drop trace of LQD as predictions. With full access to this trace i.e., perfect predictions case, Credence performs exactly as LQD as expected. However, in order to study the performance of Credence with increasing error, in a controlled manner, we flip each packet drop (or accept) from our LQD's drop trace i.e., each flip becomes a false prediction. We control the error via the flipping probability i.e., the false prediction rate. We observe from Figure 6.14 that Credence degrade in throughput as the probability of false predictions increases i.e., as the prediction error increases. However, even at as high as 0.7 probability of false predictions, Credence still out-performs DT.

In Figure 6.15, we present our results obtained from a parameter sweep across the number of trees used for random forest model vs prediction scores.

6.5 Related Work

The buffer sharing problem has been widely studied for many decades. Research works in the literature range from push-out as well as drop-tail algorithms tailored

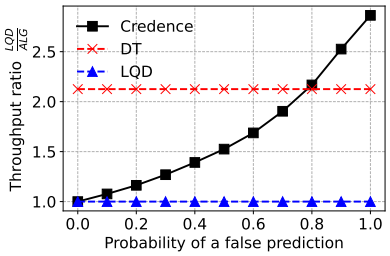


Figure 6.14: As the probability of false predictions increases, CREDENCE's throughput compared to LQD (push-out) i.e., the ratio $\frac{LQD}{ALG}$ increases from 1 to 2.9 (lower values are better). CREDENCE performs significantly better than DT even when the probability of false predictions is as high as 0.7.

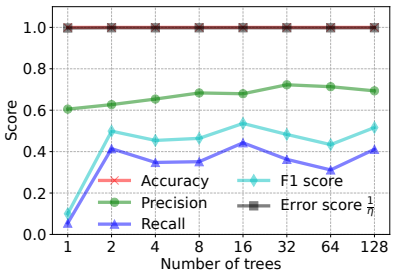


Figure 6.15: The quality of our predictions does not improve significantly beyond 4 trees in our random forest classifier.

for ATM networks [78, 86, 161, 253, 272] to more recent drop-tail algorithms tailored for datacenter networks [16, 19, 38, 46, 133, 240]. While we focus on the buffer sharing problem in this work, several related but orthogonal approaches also tackle buffer problems in datacenter networks e.g., end-to-end congestion control [21, 35, 77, 112, 164, 173], AQM [102, 139, 215], scheduling [37, 130], packet deflection [278] and load-balancing [34, 110, 152]. These approaches aim at reducing congestion events and the overall buffer requirements, but they cannot fundamentally address buffer contention across multiple switch ports sharing the same buffer. Research on algorithms with predictions for various problems has recently been an active field of research [135, 195--198, 223] but ours is the first approach tackling the buffer sharing problem with predictions. Ongoing research efforts show the feasibility of deploying machine-learned predictions in the network data plane [30, 76, 140, 274]. A multitude of recent works have focused on verifying neural networks [174, 233, 243, 263, 275, 280] and learning-augmented systems [96, 153, 228]. In contrast, the analysis framework [39, 198] used in our work assumes the prediction oracle to be a black box and focuses on the algorithmic aspects in order to bound the performance of a prediction-augmented system.

6.6 Future Work

CREDENCE is the first approach showing the performance benefits and guarantees by augmenting buffer sharing algorithms with predictions. This work barely scratches the surface and leaves intriguing open questions: *(i)* practically training the prediction oracle in the real-world, *(ii)* accounting for packet priorities in taking buffering decisions, *(iii)* integrating predictions with buffer sharing in hardware. We believe that this chapter opens several interesting avenues for future work both in systems and theory. In this section, we discuss some of the future work directions to push approaches such as CREDENCE to be deployed in the real-world (§6.6.1), as well as to improve the performance guarantees offered by such approaches (§6.6.2).

6.6.1 Systems for In-Network Predictions

In this chapter, we show how predictions can improve the performance of drop-tail buffer sharing. Many interesting systems research questions remain in order to integrate buffer sharing and predictions in the network data plane.

Training the model: Training a prediction oracle based on the model described in §6.1.3.1 involves collecting the buffering decisions (ground truth) of Longest Queue Drop algorithm (LQD), i.e., push-out, along with a set of features e.g., queue lengths, overall buffer occupancy. We envision two approaches to collect the training data in practice. First, similar to our approach in §6.3, the training data can be obtained from packet-level simulations that implement LQD on the switches. While this approach is simple, it has the limitation that the training data does not necessarily reflect real-world traffic arrivals and the corresponding buffering decisions of LQD. However, simulation-based training data could still be suitable for traffic patterns of a datacenter that can be explicitly modeled e.g., collective communications in GPU clusters [266, 284]. Second, the buffering decisions of the LQD algorithm can be exported by each switch in a real-world datacenter. This can be achieved by implementing LQD *virtually* without taking any buffering decisions on packets, along side any underlying buffer sharing algorithm i.e., maintaining per-queue *counters* that are incremented and decremented upon packet arrival, departure and drop (virtually based on LQD) events, similar to Credence's thresholds. Any feature values exported must also correspond to these counter values. Exporting LQD's events even in a coarse-grained manner would suffice if the trace is sufficiently large. We believe that future work on implementing a virtual LQD (Credence's thresholds) in hardware would not only strengthen Credence's practicality but also improves the accuracy of the trained model, thereby also improving the performance. For simplicity, our random forest model in § 6.3 uses only four features, yet exploring the tradeoff between prediction error and model complexity in terms of space and time would be valuable. Further, the trained model must be simple enough that fits within the resources available in the data plane. Developing such trained models is an important step forward.

Deploying the model: Recent works propose practical implementations for in-network machine-learning models e.g., in the context of traffic classification [76]. P4 implementation of a model that predicts drops would enhance not only the practicality but also stimulate further research to design algorithms with performance guarantees better than Credence.

Alternative predictions: As mentioned in §6.1.3, there are several different prediction models that can be considered for the buffer sharing problem. For instance, instead of predicting the drops, an oracle could predict packet arrivals just for a tiny window of the near future. Alternatively, techniques such as online reinforcement learning can be adopted, where rewards are based on how closely the queue lengths track their thresholds[8]. Systems research on studying the practicality and deployability of different prediction models is a valuable future direction that would better guide the design of algorithms with predictions for the buffer sharing problem.

Understanding push-out complexity: While push-out algorithms raised much interest initially, over the last years, research on this approach has been less active.

[8] Credence's thresholds are equivalent to LQD's (push-out) queue lengths for the same packet arrivals (see §6.2.2).

We believe this is partly due to the lack of support from switch vendors. It is an open question how the complexity of obtaining drop predictions and the complexity of push-out fare against each other. Although we focused on augmenting drop-tail algorithms with predictions, we believe that our approach of using predictions has much potential also in other types of buffer algorithms. While switch vendors may be better informed about the complexity of push-out buffers, an understanding of this complexity in the scientific community is much needed in order to navigate the complexity vs performance spectrum.

6.6.2 Theory for Performance Guarantees

We believe the performance guarantees offered by CREDENCE can be improved in the future. Further, considering packet priorities and traffic classes in the competitive analysis is an open question.

Improving consistency and robustness: An open question is whether an algorithm could be designed to improve the competitive ratio under perfect predictions (consistency) better than 1.707, while also improving the ratio under large error (robustness) better than N. Further research in this direction would enable a better understanding whether a consistency-robustness tradeoff exists for the buffer sharing problem.

Competitive analysis with packet priorities: Literature in theory considers that all packets are of the same priority in the context of competitive analysis. One of the current limitations of CREDENCE is its obliviousness to packet priorities. It is well-known that preferential treatment of packets has various performance benefits, especially in terms of flow completion times when short flow packets are prioritized. The performance degradation of CREDENCE for short flows and incast flows (Figure 6.10a, 6.10b) can potentially be shielded from prediction errors by employing packet priorities. We believe that defining throughput (objective function) as the weighted sum of the number of transmitted packets of different priorities would enable the design of online algorithms that prioritize higher-priority packets, e.g., bursts or short flows, in order to be competitive. For instance, throughput can be defined as $\sum \alpha_p \cdot n_p$, where α_p is the relative importance of a priority p and n_p is the number of transmitted packets of priority p by a buffer sharing algorithm. To this end, developing analysis techniques for such a setup is an interesting future research direction.

6.7 Summary

We presented CREDENCE, the first buffer sharing algorithm augmented with predictions that not only reaches close to optimal performance given low prediction error but also guarantees performance with arbitrarily large prediction error, while maintaining smoothness. We analytically proved our claims and our evaluations show the superior performance of CREDENCE even with an off-the-shelf machine-learned predictor, compared to the state-of-the-art buffer sharing algorithms. The building blocks required for CREDENCE are all individually practical in today's hardware. In future, we plan to pursue switch vendors to further discuss the integration of predictions with buffer sharing algorithm in hardware.

Part III

Reconfigurable Datacenter Networks

7

Oblivious Optical Interconnects

With the popularity of data-centric and distributed applications, the traffic in datacenters is growing explosively. Dealing with this traffic however becomes increasingly challenging: while cloud traffic roughly doubles each year [246], the capacity increase provided by electrical switches for a given power and cost starts to lag behind. This gap is expected to worsen with the current trend to hardware-driven workloads such as distributed machine learning training [60].

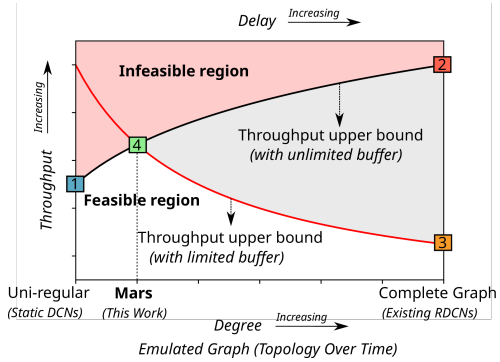
As the *throughput* of datacenter networks is becoming more and more critical for application performance, over the last years, great efforts have been made to increase the capacity of datacenter topologies. A particularly innovative architecture to meet the stringent bandwidth requirements of modern datacenters, are *reconfigurable* (optical) datacenter networks (RDCNs) [60, 79, 98, 109, 121, 187, 189, 234, 249, 261], e.g., based on optical circuit switches, tunable lasers, and simple passive gratings [205]. By quickly cycling through a sequence of different topologies---typically matchings between top-of-rack (ToR) switches---RDCNs such as RotorNet [189], Opera [187] or Sirius [60] can provide periodic direct connectivity between rack pairs, at microsecond or even nanosecond granularity. A common property of these systems is that they emulate a *complete graph*, and so avoid the "bandwidth tax" of multi-hop forwarding [114, 187]. Indeed, empirical studies show that periodic reconfigurable datacenter topologies can achieve significantly higher throughput compared to cost-equivalent traditional datacenters based on *static* topologies [60, 187, 189].

This chapter is motivated by the observation that the existing approach of using reconfigurable technologies to emulate complete graphs comes at a price: long delays and large buffer requirements.

First, at scale, emulating a complete graph can entail *long delays*: the denser the emulated network, the longer the periodic reconfiguration cycle and hence the longer a given rack pair has to wait to be connected again. Second, as we show analytically in this chapter, the resulting long reconfiguration cycles require excessive buffering at the ToR switches and end-hosts. The required buffer can intuitively be viewed as *bandwidth-delay* product of dynamic topologies similar to the corresponding notion for static topologies in TCP literature. In practice though, datacenter switches are equipped with shallow buffers. Further, several studies in the recent past show an increasing gap between switch capacity and buffer sizes [59, 112].

Our main insight in this chapter is that accounting for buffer constraints can significantly change the design considerations of reconfigurable datacenter networks.

In particular, we initiate the study of — and make the case for — reconfigurable



Topo.	Throughput	Delay	Buffer
1	Low (Optimal)	Low	Low
2	High (Optimal)	High	High
3	Low (Not optimal)	High	Low
4	**High (Optimal)**	**Moderate**	**Any**

Figure 7.1: Periodic reconfigurable datacenter topologies pose fundamental tradeoffs across throughput, delay and buffer requirements. Existing designs are at the extremes of a spectrum of optimal designs with lower delay and buffer requirements.

networks which emulate graphs *of lower node degree*, uncovering an entire spectrum of possible topology designs. We present a systematic and formal analysis of the design spectrum and the tradeoffs that these topologies introduce in terms of throughput, delay, and buffer requirements. Our perspective and the main tradeoffs are intuitively visualized in Figure 7.1 (left), where on the x-axis (at the bottom) we show the topologies according to the node degree of the graph they emulate. The x-axis (at the top) also represents increasing delay from left to right. The y-axis shows the throughput of the system from low to high. Let us elaborate on the important points in the figure (summarizes in the table on the right):

1 **Static DCNs:** On the very left of the design space are traditional static datacenter networks referred as uni-regular topologies [204] e.g., expander based DCNs [65, 248, 256][1]. Such a design in principle incurs the lowest delay and requires the least amount of buffer to achieve its ideal throughput, i.e., the optimal throughput under low delay tolerance. However, since the topology remains static, for scalability reasons, each ToR switch can only connect to a limited number of other ToR switches. This results in long multi-hop paths and hence a high "bandwidth tax" [114], and lower throughput.

2 **Existing periodic RDCNs:** To reduce the "bandwidth tax" and to achieve high throughput, existing designs resort to emulating a complete graph where each rack-pair is connected directly once in every matching cycle [60, 189]. Such designs achieve high throughput, in fact maximum across all topologies in our design space, i.e., the optimal throughput. However, as we will show in this chapter, the high throughput offered by emulating a complete graph comes at the cost of high delay and is subject to the availability of large buffers.

3 **Existing periodic RDCNs under resource constraints:** Given the large buffer requirements of existing designs (emulating a complete graph), we study their throughput under limited buffer. Interestingly, we find that, existing periodic reconfigurable datacenter topologies may perform equally or worse compared to a static uni-regular topology in terms of throughput, when buffer sizes are bounded.

4 **Mars:** Exploiting the fundamental tradeoffs across the design space, we

[1]These works also claimed that static expander-based topologies are similar or better designs (in terms of performance and cost) compared to Clos-based topologies. We henceforth focus on static expanders. A detailed discussion can be found in §7.6.

propose MARS, a periodic reconfigurable topology that provides near optimal, high throughput with the limited amount of available buffer. Specifically, we parametrize our design based on the delay tolerance and available buffer. We systematically determine the optimal degree d which depending on the resource constraints lies between a static topology and a complete graph.

It is interesting to observe from Figure 7.1 that the throughput-delay relation implies an *infeasible* region for topology design (shown in red shade). The available buffer space further restricts the design space (shown in gray shade), imposing a fundamental tradeoff across the topologies within the feasible region in terms of throughput, delay and buffer.

Our analytical approach in this chapter is novel and relies on a reduction of a periodic evolving graph to a specific static graph (Theorem 13). This enables us to study the throughput maximization problem using well-known graph analysis techniques for static graphs and to analytically evaluate the throughput of both existing dynamic topologies (RotorNet, Opera, and Sirius) as well as possible alternatives (Mars). We believe that this reduction technique may be of independent interest and could be potentially used to study other properties of dynamic topologies.

In summary, our key contributions in this chapter are:

- We provide a throughput-centric view and formal model of the performance of periodic reconfigurable datacenter topologies. In particular, we analytically derive the relation between throughput, delay, and the required amount of buffer in the network which reveals a non-trivial tradeoff in the design of periodic reconfigurable topologies.
- We present MARS, a novel reconfigurable datacenter design that maximizes throughput under limited buffer and delay requirements.
- We report on an extensive evaluation showing that MARS improves throughput by up to 4x compared to existing approaches when buffer sizes are bounded. Our evaluation also shows that MARS improves the 99-percentile flow completion time for short flows by up to 75% compared to the state-of-the-art approach [187] and by up to 87% compared to Sirius and RotorNet.

7.1 Preliminaries: Throughput of Static Topologies

Our goal in this section is to formally introduce throughput for static topologies and the relevant definitions, including the limitations of existing bounds. This section builds the intuition and motivation for our throughput analysis in the context of periodic reconfigurable topologies (§7.3.1). Much of the prior work [145, 204, 247] focused on static topologies and it remains unclear how the existing methodologies can be used to study the throughput problem in the context of reconfigurable topologies which is the focus of this chapter. Initial studies on reconfigurable topologies [114] informally define throughput of specific existing systems (e.g., RotorNet) which emulate a complete graph. We emphasize that our definitions hold for *any* periodic reconfigurable topology.

The predominant throughput metrics for static topologies are defined via demand matrices (Definition 6, below), and in particular, the worst-case demand matrix. Following the definition of Jyothi et al. [145], we formally introduce throughput given a demand matrix for static topologies. In a nutshell, given a demand matrix $\mathcal{M}$, throughput is the highest scaling factor $\theta(\mathcal{M})$ such that the scaled demand matrix $\theta(\mathcal{M}) \cdot \mathcal{M}$ is feasible in the topology i.e., there exists a feasible flow that satisfies the demand. The throughput θ^* under a worst case demand matrix is the minimum $\theta(\mathcal{M})$ over the set of all saturated demand matrices i.e., $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$. Before formally defining throughput, we first define certain preliminaries which build the intuition for our definition of throughput in dynamic topologies (§7.3.1).

Consider any static topology (at switch level) represented as a static directed multigraph $G = (V, E)$ where V is the set of vertices (switches) and E is the set of **labelled** edges (links). Labelled edges allow for distinguishing parallel links. Let $c(e, \ell)$ denote the capacity of an edge $(e, \ell) \in E$ where ℓ is the corresponding label and e is directed edge of the form (u, v) connecting two vertex u to vertex v. We denote by $c(u)$, the sum of capacities of all outgoing (correspondingly incoming) edges of a node u. We assume that each node has equal incoming and outgoing total capacity. A demand matrix specifies the demand in bps (bits/second) between every pair of vertices.

Definition 6 (Demand matrix)**.** *Given a set of vertices V, a demand matrix specifies the demand rate between every pair of vertices in bps defined as $\mathcal{M} = \{m_{u,v} \mid u \in V, v \in V\}$ where $m_{u,v}$ is the demand between the pair u, v. A saturated demand matrix is such that the total demand originating at a source s equals its outgoing capacity and the total demand terminating at a destination d equals its incoming capacity i.e., $\sum_{u \in V} m_{s,u} = c(s)$ and $\sum_{u \in V} m_{u,d} = c(d)$.*

We consider *saturated* demand matrices since we focus on the maximum achievable throughput. Specifically, saturated demand matrices allow for studying the maximum demand that can be routed in a topology within the capacity constraints.

7.1.1 Paths and Flow in Static Graphs

Given a demand matrix $\mathcal{M}$, the graph G has a set of s-d paths for transmitting the demand $m_{s,d}$ from source s to destination d for all s-d pairs. We consider simple paths i.e., without cycles.

Definition 7 (Simple paths in static graphs (Standard))**.** *Given a static directed graph $G = (V, E)$ with set of vertices V and the set of all labelled edges E, a path p^* of length n connecting a source s and a destination d is a sequence of n labelled edges $\langle (e_1, \ell_1), (e_2, \ell_2) \ldots (e_n, \ell_n) \rangle$ where $(e_i, \ell_i) \in E$; the corresponding sequence of vertices is $\langle v_1, v_2 \ldots v_n, v_{n+1} \rangle$ where $v_i \in V$, v_1 is the source s, v_{n+1} is the destination d and e_i is an edge between v_i and v_{i+1} i.e., $e_i = (v_i, v_{i+1})$ for all $i \in [1, n]$. We consider* simple *paths where each vertex appears only once in the path, so there are no cycles.*

Similar to edge labels in a multigraph, we generalize the definition of paths to facilitate distinguishing between two paths with the same sequence of labelled edges (crucial for our analysis later). From here on, we will use an extended definition of

paths in static graphs, formally defined below. We associate each path p with a unique identifier. As a result, notice that by associating each path with a unique identifier, the set of all extended paths P may contain more than one path with the same sequence of labelled edges and vertices. For example, (p^*, i) and (p^*, j) are treated as two different paths unless $i = j$ even though p^* from Definition 7 has the same sequence of labelled edges and vertices. We define paths with unique identifiers deliberately for ease of analysis later. Given a set of (extended) paths P and a path $p \in P$, we denote by $I(p)$, the set of all (extended) paths which have the identical sequence of labelled edges and vertices as that of p.

Definition 8 (Extended paths in static graphs)**.** *Given a static directed graph $G = (V, E)$, an extended path p is a pair $p = (p^*, i)$, where p^* is a standard path (Definition 7) and i is a unique identifier. An extended set of paths P is a set of extended paths. Given such P, the set of all s-d (extended) paths in P is denoted by $P_{s,d}$.*

Intuitively, the transmission of *s-d* demand from s to d in a graph is called an *s-d flow*. The set of all *s-d* flows is called a *flow*. We consider that *s-d* flow is splittable over multiple edges along all *s-d* paths. A legal (or feasible) flow in a static graph must be conserved along the path and must obey capacity constraints on every edge. Given a static graph and the set of *all* paths (Definition 7), there are infinite possibilities for an extended set of paths from Definition 8. Note that, flow is a map from a *finite* set of extended paths. We explicitly assume that for each path $p \in P$, and any $p' \in I(p)$, we have $F(p) = F(p')$.

Definition 9 (Flow in static graphs)**.** *Given static graph $G = (V, E)$ and a finite set of (extended) paths P, a Flow F is a map $F : P \to \mathbb{R}^+$ where $\mathbb{R}^+$ denotes non-negative real numbers. A* legal *flow further has the following constraints:* (i) *flow is conserved along the path and* (ii) *it obeys the capacity constraints. The amount of flow on a path $p \in P$ is denoted by $F(p)$, namely $F(p)$ is the amount of flow in each edge $e \in p$, denoted by $F(p, e)$. Formally, $F(p, e) = F(p)$ for all $e \in p$. Additionally, w.l.o.g, we assume that for each path $p \in P$, and any $p' \in I(p)$, we have $F(p) = F(p')$. The capacity constraint for a legal flow is given by the following inequality, where $\mathbb{I}(\cdot)$ is the indicator function.*

$$\sum_{p \in P} F(p) \cdot \mathbb{I}((e, \ell) \in p) \leq c(e, \ell) \qquad \forall e,\ \forall \ell \tag{7.1}$$

7.1.2 Throughput of Static Graphs

Following the definition of Jyothi et al. [145], throughput given a demand matrix $\mathcal{M}$ is the highest scaling factor θ such that the scaled demand matrix $\theta \cdot \mathcal{M}$ is feasible in the topology i.e., there exists a legal flow which can serve the scaled demand matrix $\theta \cdot \mathcal{M}$. The Throughput θ^* of a static topology is defined as the throughput under a worst-case demand matrix. Importantly, the following formal definition builds intuition for our context of periodic reconfigurable topologies.

Definition 10 (Throughput of static graphs)**.** *Given a demand matrix $\mathcal{M}$ and a graph G, a legal flow F has a throughput $\theta(\mathcal{M}, F)$ if it satisfies the scaled demand matrix $\theta(\mathcal{M}, F) \cdot \mathcal{M}$. Formally, for each s, d pair, the s-d flow sent from the source*

s to destination d is greater than the scaled demand $\theta(\mathcal{M}, F) \cdot m_{s,d}$.

$$\sum_{p \in P_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V \tag{7.2}$$

The graph G has throughput $\theta(\mathcal{M}) = \max_F \theta(\mathcal{M}, F)$ *and the throughput* θ^* *under a worst-case demand matrix is the minimum* $\theta(\mathcal{M})$ *i.e.,* $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$.

Given a demand matrix $\mathcal{M}$, the throughput maximization problem has the objective to maximize θ, subject to Equation 7.1 (capacity constraint) and Equation 7.2 (demand constraint). Unfortunately, a linear program approach does not scale well to large topologies.

7.1.3 TUB and its Limitations

Recently, Namyar et al., proposed TUB [204], a scalable throughput upper bound for static topologies. The focus of this chapter is not on the throughput of static topologies but rather on periodic reconfigurable topologies. However, even the most recently proposed throughput upper bound for static topologies has key limitations. Specifically, Namyar et al. propose the following throughput upper bound (Theorem 2.2 in [204]),

$$\theta^* \leq \min_{\mathcal{M}} \frac{\sum_e c(e)}{\sum_{s \in V} \sum_{d \in V} m_{s,d} \cdot L_{s,d}}$$

where $c(e)$ is the capacity of an edge $e \in E$ and $L_{s,d}$ is the *shortest path length* from s to d.

Key limitations of TUB: Consider a topology represented as a complete graph K_n with edge capacity c. Every node connects to every other node and hence the shortest path length for all pairs is one. In this case, TUB suggests that the topology has full throughput under a saturated permutation demand matrix, i.e., $\theta^* \leq 1$. However, the actual throughput is only $\frac{n}{2 \cdot n - 1} \approx \frac{1}{2}$. This is since the saturated permutation demand matrix specifies a demand of $(n-1) \times c$ but the shortest path can only accommodate c demand. The remaining demand must be routed through multi hop paths resulting in lower throughput. Unfortunately, TUB does not capture this effect. In general we observe that TUB does not converge even for large scale topologies when the shortest path length significantly differs from the average route length. We show in our analysis that the throughput upper bound for any static graph is indeed related to the *average route length* (Theorem 14) which gives a much tighter bound $\theta^* \leq \frac{1}{2}$ in the above example.

7.2 Background: Periodic Reconfigurable Networks

We first give a brief background on the periodic reconfigurable topologies and introduce our model.

7.2.1 Periodic Reconfigurable Topologies

The key enablers and building blocks for fast periodic reconfigurations are optical circuit switching technologies such as rotor-switches [189], AWGR gratings [84] and tunable lasers [60, 88]. Such technologies are oblivious to the demand, thus avoiding the overhead of measuring and estimating the demand. For instance, the overhead is in the order of microseconds for RotorNet and nanoseconds for Sirius which periodically reconfigure the network topology.

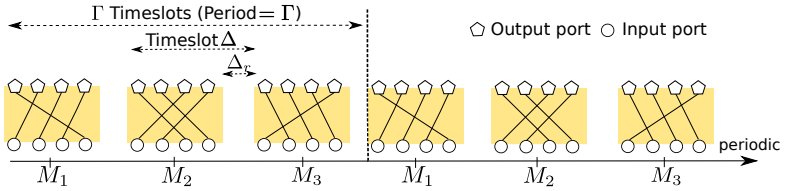


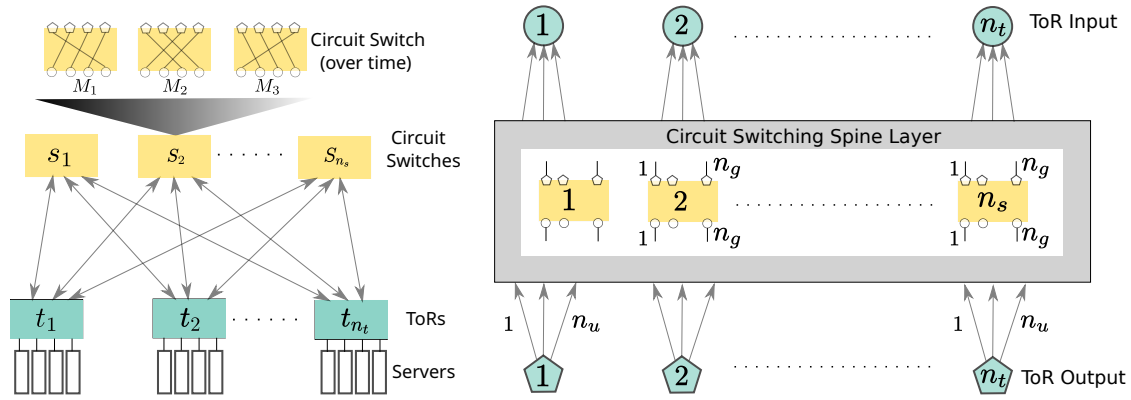
Figure 7.2: The periodic sequence of matchings of an optical circuit switch.

Circuit switch model: As a unified model of periodic circuit switching technologies, a *rotor-switch* serves as a building block for us in the rest of the chapter. Rotor-switches perform *circuit switching* and implement a sequence of matchings in a periodic schedule with period Γ where a matching is a map from the switch's input ports to its output ports. The packets arriving at an input port of a rotor-switch are forwarded to the matched output port. Specifically, rotor-switches do not process packets and the forwarding is dictated by the matching at any given instance of time. Figure 7.2 shows an example of a rotor-switch with four input and output ports with a sequence of three matchings. We use the term *timeslot* denoted by Δ to refer to the total time spent by rotor-switch in each matching including the reconfiguration time. We denote the reconfiguration time by Δ_r. The utilization time (when traffic is sent) is then $\Delta - \Delta_r$. In essence, a rotor-switch pays a "latency tax" of $\Delta_u = \frac{\Delta_r}{\Delta}$.

Note that, newer proposals such as Sirius [60] use optical gratings and tunable lasers to achieve fast periodic reconfigurations at nanosecond scale. In contrast to rotor-switches, Sirius uses optical gratings which are the building blocks and the ToR switches (or end-hosts) connect to the gratings via tunable lasers which tune the wavelength of emitted light periodically. However, both RotorNet and Sirius logically capture a rotor-switch that rotates periodically across a fixed number of matchings. To this end, Sirius can be abstracted as a topology with rotor-switches and the difference arises in the system level parameters such as reconfiguration time Δ_r. We do not model system level parameters in this work and only focus on the topological aspects. In the following, we will sometimes use the more general term *optical switch* when referring to the *rotor-switch*.

Topology: Following the literature on existing proposals [60, 187, 189], we consider 2-tier reconfigurable datacenter topologies with a set of ToR switches (leaf layer) and a set of circuit switches (spine layer) as shown in Figure 7.3a, where each circuit switch functions as a rotor-switch. The uplinks of the ToR switches connect to the spine layer which interconnects the datacenter[2]. Concretely, as illustrated in Figure 7.3b, we consider a periodic reconfigurable datacenter with n_t ToR switches

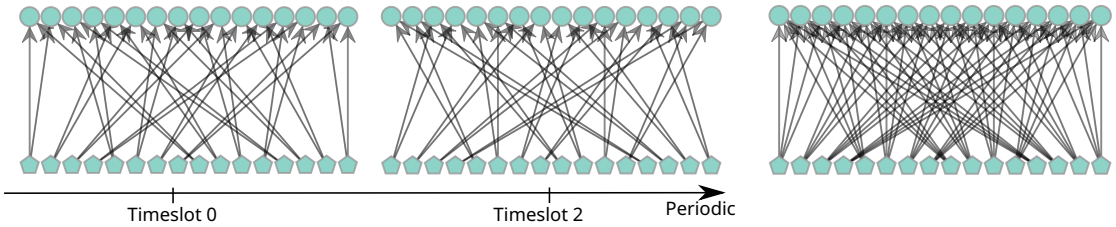
[2]Each ToR uplink is a set of two SERDESes [4, 60] which are uni-directional corresponding to ToR output (pentagons) and input (circles) as depicted in Figure 7.3b.



(a) A 2-tier periodic RCDN.

(b) An abstract view of periodic RCDN topologies.

Figure 7.3: Illustration of periodic RCDN model we consider in this work, consisting of n_u ToR switches each with n_u uplinks. The datacenter is interconnected by a circuit switching spine layer consisting of n_s circuit switches, each with n_p input and output ports.



(a) Example of a periodic evolving graph with period $\Gamma = 2$ depicting the ToR-to-ToR connectivity over time; $n_t = 16$, $n_u = 2$, $n_s = 2$ and $n_p = 16$.

(b) The corresponding *Emulated* graph (static) over time.

Figure 7.4: Given the number of ToR switches n_t with uplinks n_u and a periodic sequence of matchings corresponding to each circuit switch, the topology is a periodic evolving graph which *emulates* a static graph over time.

each with n_u uplinks; n_s circuit switches each with n_p input and output ports. We assume that a link e in the topology has capacity $c(e)$. We will later show that this generalized view reveals an entire spectrum of topologies where the existing systems (e.g., RotorNet and Sirius) are special instances of the spectrum which emulate a complete graph i.e., each ToR connects to every other ToR in a period.

7.2.2 Graph Theoretic Model of Periodic ToR-to-ToR Connectivity

In contrast to packet switched networks, circuit switched networks do not buffer packets. To this end, since periodic reconfigurable topologies are circuit switched, we are mainly interested in the connectivity between the end points of the circuit switched network, namely the ToR-to-ToR connectivity. Hence, circuit availability between a pair of ToR switches can be considered as a direct link between the pair. In the following, we model the periodically reconfigurable ToR-to-ToR connectivity over time as a *periodic evolving graph*.

Periodic evolving graph: Consider the periodic reconfigurable topology described in §7.2.1 (shown in Figure 7.3b). ToR switches connect to the optical switches via uplinks. The optical spine layer in turn establishes a circuit between ToR

pairs *periodically* as shown in Figure 7.4a since the optical switches reconfigure periodically according to a fixed schedule. To this end, we represent the ToR-to-ToR connectivity as *periodic evolving graph* denoted by $\mathcal{G} = (V, \mathcal{E})$. Specifically, $\mathcal{G}$ is a periodic sequence of directed graphs defined for timeslots $t \in [0, \infty)$ where each timeslot is of Δ duration. We denote the graph at time t as $\mathcal{G}_t = (V, \mathcal{E}_t)$, where V is the set of all ToR switches and the edge set $\mathcal{E}_t$ represents the circuit availability between the ToR switches at time t. The sequence of edge sets $\mathcal{E}_t$ and consequently the evolving graph $\mathcal{G}$ are periodic with period of Γ timeslots. The period in turn relates to the periodic switching schedule of the optical switch. We denote the capacity of circuit between ToR pairs at any time t as $c_t(e)$ for all $e \in \mathcal{E}_t$. We explicitly set $c_t(e) = 0$ if $e \notin \mathcal{E}_t$. Further, we model the ''latency tax'' Δ_u due to reconfigurations by limiting the access to an edge $e \in \mathcal{E}_t$ to only a $(1 - \Delta_u)$ fraction of a timeslot (from the start of every timeslot). This model serves as an entry point for our formal analysis of throughput, delay, and buffer requirements of reconfigurable topologies in the next section.

7.3 Motivation: Fundamental Tradeoffs of Periodic RDCNs

We now provide a more detailed motivation for our work. Specifically, we analytically derive the throughput of periodic RDCNs (§7.3.1) and its relation to delay (§7.3.2). We further study the buffer requirements (§7.3.3) and highlight that the topologies introduce a fundamental tradeoff between throughput, delay and buffer. We then present remarks and practical implications of our results (§7.3.4). Finally, we discuss the optimization opportunities in the design of periodic reconfigurable topologies (§7.3.5).

7.3.1 Throughput of Periodic RDCNs

We now study the throughput of periodic reconfigurable topologies. Similar to static topologies, we first need to formally define paths and flow in the context of periodic reconfigurable topologies. In this section, we follow the periodic evolving graph model introduced in §7.2.2. Recall that the period of the periodic evolving graph is denoted by Γ, timeslot is denoted by Δ and the reconfiguration time is denoted by Δ_r. Since circuit switches do not route traffic during reconfiguration, links remain idle for Δ_r amount of time in every timeslot Δ. We denote this fraction of time spent in reconfiguration by Δ_u. We next formally define periodic evolving graph.

Definition 11 (Periodic Evolving Graph)**.** *A periodic evolving graph (henceforth for simplicity referred to as evolving graph) denoted by $\mathcal{G} = (V, \mathcal{E})$ is a periodic sequence of directed graphs with a period of Γ timeslots, defined for each timeslot $t \in \mathbb{W}$. The directed graph at time $t \in \mathbb{W}$ is defined as $\mathcal{G}_t = (V, \mathcal{E}_t)$, where V is the set of vertices and $\mathcal{E}_t \subseteq V \times V$ is the set of directed edges at time t and $\mathbb{W}$ is the set of whole numbers. Note that an edge $e \in V \times V$ may appear in multiple edge sets $\mathcal{E}_t$ at different times t. The evolving graph $\mathcal{G}$ has the following properties:*

- *The sequence of graphs starts at $t = 0$ and is defined for each timeslot in the interval $t \in [0, \infty)$.*

- *The edge set $\mathcal{E}_t$ at time t is periodic with period Γ i.e., $\mathcal{E}_{t+\Gamma} = \mathcal{E}_t$ and consequently the graph $\mathcal{G}_t$ is periodic with period Γ.*

- *An edge $e \in V \times V$ has a capacity $c_t(e)$ at time t. Since the graph is periodic, the capacity of an edge is also periodic: $c_{t+\Gamma}(e) = c_t(e)$. For notational convenience we set $c_t(e) = 0$ whenever $e \notin \mathcal{E}_t$.*

In this section, we first define the notions of path and flow in the periodic evolving graph. Using these definitions, we then formally define throughput.

7.3.1.1 Temporal Paths and Temporal Flow

Recall from Definition 11 that a periodic evolving graph $\mathcal{G}$ is a periodic sequence of directed graphs with period Γ. The graph at time t is denoted by $\mathcal{G}_t = (V, \mathcal{E}_t)$ where V is the set of vertices (switches) and $\mathcal{E}_t$ is the set of edges (links) at time t. Note that the edge set evolves over time. As a result, paths are formed over time as opposed to static paths. To this end, we define a temporal path in the evolving graph as a sequence of (e, t) edge-time pairs, where an edge e is accessed at time t. Consecutive edges along the path are accessed in non-decreasing order of time. We assume that the next edge along a path is accessed within Γ timeslots (the period). Further, temporal paths are periodic. We define the set of all temporal paths which start within the first period as the *foundation set* of temporal paths denoted by $\mathcal{P}^0$. The set of *all* temporal paths is denoted by $\mathcal{P}^*$. The foundation set is a crucial part of our throughput analysis, especially since temporal paths are periodic. We define temporal paths formally in the following.

Definition 12 (Temporal paths in evolving graphs)**.** *Given a periodic evolving graph $\mathcal{G}$ with set of vertices V, a temporal path δ of length n connecting a source s and a destination d is a sequence of n edges and corresponding time values $\langle (e_1, t_1), (e_2, t_2) \ldots (e_n, t_n) \rangle$ where t_i is the time when the edge e_i is accessed along the path and e_i is an edge in $\mathcal{E}_{t_i}$; the corresponding sequence of vertices is $\langle v_1, v_2 \ldots v_n, v_{n+1} \rangle$ where v_1 is the source s, v_{n+1} is the destination d and e_i is an edge between v_i and v_{i+1} i.e., $e_i = (v_i, v_{i+1})$ for all $i \in [1, n]$. A* legal *temporal path additionally has the following properties:*

- *An edge e_{i+1} is accessed at time t_{i+1} within a period time (Γ) after accessing e_i at time t_i i.e., $t_i < t_{i+1} \leq t_i + \Gamma$*

- *$v_i \neq v_j$ if $i \neq j$ i.e., the temporal path is simple and has no cycles*

The foundation set *of temporal paths, denoted as $\mathcal{P}^0$ includes* all *legal temporal paths in $\mathcal{G}$ that start within the first period (i.e., $t_1 \in [0, \Gamma)$). For the* foundation set *$\mathcal{P}^0$, we define the set $\mathcal{P}^*$ of all* legal *temporal paths in $\mathcal{G}$ as follows:*

- *$\mathcal{P}^0 \subset \mathcal{P}^*$ and temporal paths are periodic with period Γ.*

- *A function periodic(δ) is defined as follows for every temporal path δ: the set of all temporal paths which have the same sequence of edges with the time sequence shifted by an integer multiple of Γ. Precisely, for any temporal path $\delta \in \mathcal{P}$, periodic(δ) $= \{\langle (e_1, t_1 + k \cdot \Gamma), (e_2, t_2 + k \cdot \Gamma) \ldots (e_\ell, t_\ell + k \cdot \Gamma) \rangle \mid k \in \mathbb{W}\}$, where $\delta = \langle (e_1, t_1), (e_2, t_2) \ldots (e_\ell, t_\ell) \rangle$ and $\mathbb{W}$ is the set of whole numbers.*

- $\mathcal{P}^* = \bigcup_{\delta \in \mathcal{P}^0} periodic(\delta)$ *i.e., the set of all temporal paths* $\mathcal{P}^*$ *is the union of periodic temporal paths for each temporal path in the foundation set* $\mathcal{P}^0$*. For simplicity of notation, we say* $\mathcal{P}^* = periodic(\mathcal{P}^0)$*.*

The sets $\mathcal{P}^0$ *and* $\mathcal{P}^*$ *are unique given* $\mathcal{G}$*. Let the set of all s-d temporal paths in* $\mathcal{P}^0$ *and* $\mathcal{P}^*$ *be denoted by* $\mathcal{P}^0_{s,d}$ *and* $\mathcal{P}^*_{s,d}$*, respectively.*

Similar to paths and flow in static graphs (§7.1), in our context, the evolving graph and the set of *s-d* paths (Definition 12) in the evolving graph allow for transmitting the demand of all *s-d* pairs. Intuitively, the transmission of *s-d* demand along *s-d* paths is called flow. In contrast to the flow in static graphs, the evolving graph $\mathcal{G}$ forces for the flow (Definition 9) to be additionally split in time since temporal paths are spread over time. Similar to the flow in static graphs, in the following we define a *Temporal-Flow* and its constraints in periodic evolving graphs. Specifically, a legal temporal flow must obey capacity constraints i.e., the sum of all flows on an edge at time t is upper bounded by the edge capacity $c_t(e)$.

Definition 13 (Temporal flow in evolving graphs)**.** *Given a periodic evolving graph* $\mathcal{G}$ *a legal set of temporal paths* $\mathcal{P}$*, a temporal flow* $\mathcal{F}$ *is a map* $\mathcal{F} : \mathcal{P} \to \mathbb{R}^+$*. The amount of flow on a path* $\delta \in \mathcal{P}$ *is denoted by* $\mathcal{F}(\delta)$*, namely* $\mathcal{F}(\delta)$ *is the amount of flow in each edge* $(e,t) \in \delta$*, denoted by* $\mathcal{F}(\delta, e, t)$*. Formally,* $\mathcal{F}(\delta, e, t) = \mathcal{F}(\delta)$ *for all* $(e,t) \in \delta$ *and otherwise* $\mathcal{F}(\delta, e, t) = 0$*. Temporal-Flow is periodic on periodic paths i.e.,* $\mathcal{F}(\delta) = \mathcal{F}(\delta')$ *if* $\delta' \in periodic(\delta)$*. A temporal flow is said to be* legal *if it obeys the capacity constraints at any time* t*, formally*

$$\sum_{\delta \in \mathcal{P}} \mathcal{F}(\delta, e, t) \leq c_t(e) \qquad \forall e,\ \forall t \tag{7.3}$$

where $c_t(e)$ *is the capacity of an edge* e *at time* t*.*

7.3.1.2 Throughput of Evolving Graphs

Based on the definition of temporal-paths and temporal-flow, we are now ready to define throughput in evolving graphs. Intuitively, given a demand matrix $\mathcal{M}$, we define throughput as the highest scaling factor $\theta(\mathcal{M})$ such that $\theta(\mathcal{M}) \cdot \mathcal{M}$ amount of demand can be routed in the evolving graph *over one period* on average. Specifically, a temporal path δ sends $\mathcal{F}(\delta) \cdot (\Delta - \Delta_r)$ amount of demands (in bits) over $\Gamma \cdot \Delta$ time before its next periodic flow begins. Hence, $\frac{(\Delta - \Delta_r)}{\Gamma \cdot \Delta} \cdot \mathcal{F}(\delta) = (\frac{1 - \Delta_u}{\Gamma}) \cdot \mathcal{F}(\delta)$ is the average bits per second transmitted on a temporal path δ. Following this intuition, we define throughput in the evolving graph.

Definition 14 (Throughput in the evolving graph)**.** *Given a demand matrix* $\mathcal{M}$ *and an evolving graph* $\mathcal{G}$*, a legal temporal flow* $\mathcal{F}$ *has throughput* $\theta(\mathcal{M}, \mathcal{F})$ *if it satisfies the scaled demand matrix over* Γ *timeslots. For each* s, d *pair, the s-d temporal flow sent from the source* s *to destination* d *over* Γ *timeslots is greater than the scaled demand* $\theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d}$*. Formally, let* $\mathcal{P}^0$ *be the foundation set of* $\mathcal{G}$ *and* $\mathcal{P}^0_{s,d}$ *is the set of all s-d temporal flows starting in the first period.*

$$\left(\frac{1 - \Delta_u}{\Gamma}\right) \cdot \sum_{\delta \in \mathcal{P}^0_{s,d}} \mathcal{F}(\delta) \geq \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

The graph $\mathcal{G}$ *has throughput* $\theta(\mathcal{M}) = \max_{\mathcal{F}} \theta(\mathcal{M}, \mathcal{F})$ *and the throughput* θ^* *under a worst-case demand matrix is the minimum* $\theta(\mathcal{M})$ *i.e.,* $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$*.*

Throughput in the evolving graph can be calculated over *any* one period time-interval. For simplicity, we consider the first period and the foundation set of temporal paths $\mathcal{P}^0$ especially since the summation of *s-d* flows can be converted to the first period using the periodic property of temporal paths and flow i.e., for every $\delta \in \mathcal{P}^0_{s,d}$, $\mathcal{F}(\delta) = \mathcal{F}(\delta')$ for $\delta' \in \textit{periodic}(\delta)$. However, note that it takes at most the duration of the longest temporal path, in order for Definition 14 to hold at the *destination*. In essence, Definition 14 for periodic evolving graphs, follows the reasoning behind throughput of static topologies i.e., the maximum scaling factor θ such that the scaled demand matrix is feasible to route continuously in the periodic topology.

We now begin to analyze the throughput maximization problem in periodic evolving graphs. Notice that the problem is much more complicated compared to static graphs. First, the set of all temporal paths is infinite. Second, even the foundation set of temporal paths is exponential in size and grows with larger Γ (period).

To this end, our technique involves a static graph which we call the *Emulated graph* corresponding to a periodic evolving graph. We then show that a periodic evolving graph and its corresponding *static* emulated graph have the same throughput.

7.3.1.3 Emulated Graph: An Equivalent Static Topology

Given a periodic evolving graph $\mathcal{G}$, we define the corresponding emulated graph $G(\mathcal{G})$ as a function of $\mathcal{G}$. Specifically, edge set of the emulated graph is obtained from the union of edge sets of the periodic evolving graph taken over any *one period* time interval and the edges are **labelled** with the corresponding time of the edge in the evolving graph. The edge capacities are such that the emulated graph has the same amount of average capacity between all pairs compared to the evolving graph, including the overhead of reconfiguration (Δ_r). In the following, we formally define the emulated graph.

Definition 15 (Emulated graph)**.** *The emulated graph $G(\mathcal{G})$ of periodic graph $\mathcal{G}$ is defined as a static directed multigraph $G = (V, E)$, where V is the set of vertices of $\mathcal{G}$ and $E = \{(e, \ell) \mid e \in \mathcal{E}_\ell,\ \ell \in [0, \Gamma)\}$ is the set of directed edges obtained from the union of edges of the evolving graph over the $[0, \Gamma)$ time interval and every edge (e, ℓ) is labelled with ℓ if $e \in \mathcal{E}_\ell$. An edge $(e, \ell) \in E$ has capacity $\hat{c}(e, \ell)$ where the relation between $\hat{c}(e, \ell)$ and the original capacity $c_\ell(e)$ in the evolving graph is given by,*

$$\hat{c}(e, \ell) = \left(\frac{1 - \Delta_u}{\Gamma}\right) \cdot c_\ell(e) \qquad \forall e,\ \forall \ell \in [0, \Gamma) \tag{7.4}$$

We now precisely specify the extended set of paths (see Definition 8) in the emulated graph. Specifically, the extended set of paths with unique identifiers allow for a relation between a path in the emulated graph to a temporal path in the evolving graph. We define a function *static* which converts a temporal path to static extended path and a function temporal which converts a static extended path to a temporal path. In essence, the set of all extended paths in the emulated graph is $P^* = \text{static}(\mathcal{P}^0)$. The temporal function allows for backward conversion i.e., $\mathcal{P}^0 = \text{temporal}(P^*)$.

Definition 16 (Extended paths in Emulated graph)**.** *Given a periodic evolving graph $\mathcal{G}$ with the foundation set of paths $\mathcal{P}^0$, the corresponding emulated graph $G(\mathcal{G})$ has a set of extended paths P^* given by* static$(\mathcal{P}^0)$*. The* static *function is defined as follows: for every temporal path δ, where $\delta = \langle (e_1, t_1), (e_2, t_2) \ldots (e_n, t_n) \rangle$, the extended static path $p =$ static(δ) is given by $p = (\langle (e_1, \ell_1), (e_2, \ell_2) \ldots (e_n, \ell_n) \rangle, \delta)$, where $\ell_i = t_i$* **mod** *Γ and δ is the unique identifier of the extended static path; recall that δ is unique. The inverse of* static *function is defined as:* temporal$(p) =$ static$^{-1}(p) = \delta$*, for all $p \in P^*$, where δ is the unique identifier associated with the extended path p.*

$$
\begin{aligned}
P^* &= \text{static}(\mathcal{P}^0) = \{\text{static}(\delta) \mid \delta \in \mathcal{P}^0\} \\
\mathcal{P}^0 &= \text{temporal}(P^*) = \{\text{temporal}(p) \mid p \in P^*\}
\end{aligned} \tag{7.5}
$$

Note that, given a periodic evolving graph $\mathcal{G}$, its foundation set of temporal paths $\mathcal{P}^0$ is unique and includes *all* temporal paths which start in the first period. Since the emulated graph is a function of the periodic evolving graph, the set of extended static paths (Definition 16) is also unique. The extended set of paths and the static, temporal functions are a crucial part of our throughput analysis in the next section.

7.3.1.4 Throughput Analysis of Periodic RDCNs

Before presenting our analysis, we first state our main result in this section. Consider a periodic periodic evolving graph (see Definition 11) $\mathcal{G} = (V, \mathcal{E})$ with edge capacities $c_t(e)$ for every edge e at time t. Consider a static graph which is a function of the evolving graph $\mathcal{G}$ obtained from Definition 15 represented as $G(\mathcal{G}) = (V, E)$. We prove that the throughput of a periodic evolving graph is equivalent to the corresponding emulated graph.

Theorem 13 (Relation to Emulated Graph)**.** *Given a demand matrix $\mathcal{M}$, any legal temporal flow $\mathcal{F}$ in the periodic evolving graph $\mathcal{G}$ with throughput $\theta(\mathcal{M}, \mathcal{F})$ can be converted to a legal flow F in the emulated graph $G(\mathcal{G})$ with the same throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$ and vice versa. Consequently the periodic evolving graph and the emulated graph have the same throughput $\theta(\mathcal{M}) = \max_F \theta(\mathcal{M}, F)$: the maximum scaling factor given a demand matrix $\mathcal{M}$; they further have the same throughput θ^* for a worst-case demand matrix, where $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$.*

A sketch of our approach is as follows: *(i)* we prove in Lemma 6 that if a legal flow in the emulated graph has throughput θ over set of extended paths $P^* =$ static$(\mathcal{P}^0)$, then there exists a legal temporal flow in the periodic evolving graph with the same throughput; *(ii)* we prove in Lemma 7 that if a legal temporal flow achieves throughput θ in the evolving graph, then there exists a legal flow in the emulated graph with the same throughput over the set of extended paths $P^* =$ static$(\mathcal{P}^0)$; *(iii)* finally, we prove in Lemma 8 that a flow in emulated graph over the set of *all paths* P has the same throughput upper bound as that of a flow over the set of extended paths $P^* =$ static$(\mathcal{P}^0)$. Using the above three results, we prove our main claim.

Lemma 6. *Let $\mathcal{M}$ be a demand matrix, $\mathcal{G}$ be a periodic evolving graph, $G(\mathcal{G})$ be its emulated graph and $P^* =$ static$(\mathcal{P}^0)$. If $F : P^* \to \mathbb{R}^+$ is a legal flow in $G(\mathcal{G})$ with throughput $\theta(\mathcal{M}, F)$ then there exists a legal temporal flow $\mathcal{F}$ in the evolving*

graph $\mathcal{G}$ with the same throughput $\theta(\mathcal{M}, \mathcal{F}) = \theta(\mathcal{M}, F)$. The temporal flow $\mathcal{F}$ in the evolving graph can be constructed as follows:

$$\mathcal{F}(\delta) = \left(\frac{\Gamma}{1-\Delta_u}\right) \cdot F(p)$$
$$\forall p \in P^*;\ \delta \in \textit{periodic}(\text{temporal}(p)) \tag{7.6}$$

where $\mathcal{F}$ is periodic; for any temporal path $\delta \in \text{temporal}(P^*)$, $\mathcal{F}(\delta, e, t) = \mathcal{F}(\delta)$ for all $(e, t) \in \delta$ and $\mathcal{F}(\delta, e, t) = 0$ if $(e, t) \notin \delta$. $\mathcal{F}$ is then a map $\mathcal{F} : \mathcal{P}^* \to \mathbb{R}^+$, where $\mathcal{P}^* = \textit{periodic}(\text{temporal}(P^*))$.

Proof. From Definition 9, since F is a legal flow in the emulated graph, F obeys capacity constraints.

$$\sum_{p \in P^*} F(p) \cdot \mathbb{I}((e, \ell) \in p) \leq \hat{c}(e, \ell) \qquad \forall e,\ \forall \ell \in [0, \Gamma)$$

We first substitute $\hat{c}(e, \ell) = \frac{(1-\Delta_u)}{\Gamma} \cdot c_\ell(e)$ and change ℓ to ℓ **mod** Γ without changing the value of both sides of the above inequality.

$$\sum_{p \in P^*} F(p) \cdot \mathbb{I}((e, \ell \textbf{ mod } \Gamma) \in p) \leq \left(\frac{1-\Delta_u}{\Gamma}\right) \cdot c_{(\ell \textbf{ mod } \Gamma)}(e)$$
$$\forall e,\ \forall \ell \in [0, \Gamma)$$

We now expand the above inequality as follows for all $\ell \in [0, \infty)$, without changing the value on both sides of the inequality. Recall that $c_\ell(e) = c_{\ell + k \cdot \Gamma}$ for any integer $k \geq 0$ ($c_\ell(e)$ is periodic).

$$\sum_{p \in P^*} F(p) \cdot \mathbb{I}((e, \ell \textbf{ mod } \Gamma) \in p) \leq \left(\frac{1-\Delta_u}{\Gamma}\right) \cdot c_\ell(e)$$
$$\forall e,\ \forall \ell \in [0, \infty)$$

The above inequality holds since $c_\ell(e)$ is periodic and ℓ **mod** Γ always ranges between $[0, \Gamma)$. Substituting the temporal flow $\mathcal{F}$ for the static flow F, using Equation 7.6,

$$\sum_{p \in P^*} \mathcal{F}(\text{temporal}(p)) \cdot \mathbb{I}((e, \ell \textbf{ mod } \Gamma) \in p) \leq c_\ell(e)$$
$$\forall e,\ \forall \ell \in [0, \infty)$$

From Definition 16 and from the periodic property of temporal paths (Definition 12), for every path $p \in P^*$ and for every $(e, \ell \textbf{ mod } \Gamma) \in p$, there exists exactly one path $\delta \in \textit{periodic}(\text{temporal}(p))$ such that $(e, \ell) \in \delta$. Using this relation, we convert the above inequality as follows,

$$\sum_{\delta \in \textit{periodic}(\text{temporal}(P^*))} \mathcal{F}(\delta) \cdot \mathbb{I}((e, t) \in \delta) \leq c_t(e)$$
$$\forall e,\ \forall t \in [0, \infty)$$

Since $\textit{periodic}(\text{temporal}(P^*)) = \mathcal{P}^*$, and since $c_t(e) = 0$ if $e \notin \mathcal{E}_t$, we obtain the following relation.

$$\sum_{\delta \in \mathcal{P}^*} \mathcal{F}(\delta, e, t) \leq \begin{cases} c_t(e) & e \in \mathcal{E}_t \\ 0 & e \notin \mathcal{E}_t \end{cases}$$

From Definition 13, we conclude that $\mathcal{F}$ (from Equation 7.6) obeys capacity constraints. Further since temporal flow is constant and equal for all $(e,t) \in \delta$ for all temporal paths, it obeys flow conservation rules. It remains to prove that $\mathcal{F}$ also achieves throughput θ. Since F has a throughput $\theta(\mathcal{M})$, from Definition 23 we have that,

$$\sum_{p \in P^*_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

using the relation between F and $\mathcal{F}$ from Equation 7.6, we obtain the following inequality. From Definition 14, we conclude that $\mathcal{F}$ also achieves throughput $\theta(\mathcal{M}, \mathcal{F}) = \theta(\mathcal{M}, F)$.

$$\left(\frac{1-\Delta_u}{\Gamma}\right) \cdot \sum_{\delta \in \mathcal{P}^0_{s,d}} \mathcal{F}(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

□

From Lemma 6, we have that, given a demand matrix $\mathcal{M}$, if the static emulated graph has throughput $\theta(\mathcal{M}, F)$ for a specific flow F, then the scaled demand matrix $\theta(\mathcal{M}, F) \cdot \mathcal{M}$ is feasible in the periodic evolving graph. In the following, we state the reverse i.e., given a demand matrix $\mathcal{M}$, if the periodic evolving graph has throughput $\theta(\mathcal{M}, \mathcal{F})$ for a specific temporal flow $\mathcal{F}$, then the scaled demand $\theta(\mathcal{M}, \mathcal{F}) \cdot \mathcal{M}$ is feasible in the static emulated graph.

Lemma 7. *Given a demand matrix $\mathcal{M}$, if $\mathcal{F} : \mathcal{P}^* \to \mathbb{R}^+$ is a legal temporal flow in the evolving graph $\mathcal{G}$ with throughput $\theta(\mathcal{M}, \mathcal{F})$ then there exists a legal flow F in the emulated graph $G(\mathcal{G})$ with same throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$. F is obtained as follows:*

$$F(\text{static}(\delta)) = \left(\frac{1-\Delta_u}{\Gamma}\right) \cdot \mathcal{F}(\delta) \quad \forall \delta \in \mathcal{P}^0 \tag{7.7}$$

F is then a map $F : P^* \to \mathbb{R}^+$, where $P^* = \text{static}(\mathcal{P}^0)$.

Proof. From Definition 9, for F to be a legal flow in the emulated graph G, F must obey capacity and flow conservation constraints. Further in order to achieve throughput $\theta(\mathcal{M}, F)$, we require that $F \geq \theta(\mathcal{M}, F) \cdot \mathcal{M}$. Since $\mathcal{F}$ is a legal flow in the evolving graph, $\mathcal{F}$ obeys capacity constraints. From Definition 13, for all $e \in \mathcal{E}_t$ at any time t we have that,

$$\sum_{\delta \in \mathcal{P}^*} \mathcal{F}(\delta) \cdot \mathbb{I}((e,t) \in \delta) \leq c_t(e) \qquad \forall e,\ \forall t \in [0, \infty)$$

The remainder of the proof follows similar logic as that of the proof of Lemma 6. Since for any path $\delta \in \mathcal{P}^*$, $\mathcal{P}^* = \textit{periodic}(\mathcal{P}^0) = \textit{periodic}(\text{temporal}(P^*))$, if $(e,t) \in \delta$, then there exists no other path $\delta' \in \textit{periodic}(\delta)$ where $(e,t) \in \delta'$, except for the path δ itself. For every such path δ, there exists exactly one path $p \in P^*$ where $(e,\ell) \in p$ and $\ell = t \bmod \Gamma$. Using the relation between $\mathcal{F}$ and F from Equation 7.7 and substituting the capacity relation $\hat{c}(e,\ell) = \frac{(1-\Delta_u)}{\Gamma} \cdot c_\ell(e)$, we obtain the following:

$$\sum_{p \in P^*} F(p) \cdot \mathbb{I}((e,\ell) \in p) \leq \hat{c}(e,\ell) \qquad \forall e,\ \forall \ell \in [0, \Gamma)$$

From Definition 9, the above inequality implies that the flow F obeys capacity constraints. Further, flow is conserved since $F(p)$ is constant and equal for all $e \in p$ for all paths. It remains to prove that F achieves throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$. Since $\mathcal{F}$ has a throughput $\theta(\mathcal{M}, \mathcal{F})$ for a demand matrix $\mathcal{M}$, from Definition 14 we have that,

$$\left(\frac{1-\Delta_u}{\Gamma}\right) \cdot \sum_{\delta \in \mathcal{P}^0_{s,d}} \mathcal{F}(\delta) \geq \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

We now substitute static variables using Equation 7.7 and obtain the throughput relation,

$$\sum_{p \in \text{static}(\mathcal{P}^0_{s,d})} F(p) \geq \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

Using the above inequality and since $static(\mathcal{P}^0_{s,d}) = P^*_{s,d}$, from Definition 23, we conclude that F also achieves throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$. □

From Lemma 6 and Lemma 7, given a demand matrix $\mathcal{M}$, the periodic evolving graph and the emulated graph with extended set of paths $P^* = \text{static}(\mathcal{P}^0)$, have the same throughput. It remains to show that the emulated graph with set of extended paths $P^* = \text{static}(\mathcal{P}^0)$ has the same throughput compared to the emulated graph with set of all possible paths (not extended).

Lemma 8. *Let $\mathcal{M}$ be a demand matrix, $\mathcal{G}$ be a periodic evolving graph, $G(\mathcal{G})$ be its emulated graph and $P^* = \text{static}(\mathcal{P}^0)$. If $F : P \to \mathbb{R}^+$ is a legal flow in $G(\mathcal{G})$ with throughput $\theta(\mathcal{M}, F)$ where P is the set of all simple paths then there exists a legal flow $F' : P^* \to \mathbb{R}^+$ in the emulated graph $\mathcal{G}$ with the same throughput $\theta(\mathcal{M}, F') = \theta(\mathcal{M}, F)$ where F' is as follows:*

$$F'(p') = \frac{F(p)}{|I(p)|} \quad \forall p' \in I(p) \tag{7.8}$$

where $I(p)$ is the set of all extended paths which have the same sequence of labelled edges as that of a path $p \in P$.

Proof. We have that $F : P \to \mathbb{R}^+$ is a legal flow in the emulated graph and has a throughput $\theta(\mathcal{M}, F)$. The proof follows by showing that $F' : P^* \to \mathbb{R}^+$ also obeys capacity constraints and has throughput $\theta(\mathcal{M}, F') = \theta(\mathcal{M}, F)$. We will first prove that the following capacity constraint (from Definition 9) for F' holds. We prove this using the relation between F and F' from Equation 7.8 and the fact that F obeys capacity constraints. Since F obeys capacity constraints, the following inequality holds.

$$\sum_{p \in P} F(p) \cdot \mathbb{I}((e, \ell) \in p) \leq c(e) \qquad \forall e,\ \forall \ell$$

For every path $p \in P$, the set of extended paths P^* consists of a set $I(p)$ of extended paths. Note that $I(p)$ is strictly greater than zero i.e., $I(p) > 0$. The argument is that *(i)* P is the set of all paths; *(ii)* P^* is obtained from the foundation set of all temporal paths $\mathcal{P}^0$ as $P^* = static(\mathcal{P}^0)$; *(iii)* for any path $p \in P$ with the sequence of edges $\langle e_1, e_2, ...e_n \rangle$, there exists at least one legal temporal path

$\delta \in \mathcal{P}^0$ where $\delta = \langle (e_1, t_1), (e_2, t_2) \ldots (e_n, t_n) \rangle$ such that t_1 is the time when the edge e_1 appears for the first time in the first period, t_2 is the time when the edge e_2 appears for the first time after t_1 and so on; each edge appears within Γ timeslots due to the periodicity of the edge set and such a temporal path δ belongs to $\mathcal{P}^0$ by definition. Further, for any path $p \in P$ and a labelled edge $(e, \ell) \in p$, by definition, the labelled edge (e, ℓ) also belongs to any path $p' \in I(p)$. Using this relation, we expand the summation in the above inequality as follows:

$$\sum_{p \in P} \sum_{p' \in I(p)} \frac{F(p)}{|I(p)|} \cdot \mathbb{I}((e, \ell) \in p') \leq c(e) \qquad \forall e, \ \forall \ell$$

Substituting F' using Equation 7.8, we obtain the following relation.

$$\sum_{p \in P^*} F'(p') \cdot \mathbb{I}((e, \ell) \in p') \leq c(e) \qquad \forall e, \ \forall \ell$$

From Definition 9, the above inequality suggests that F' obeys capacity constraints. Similarly, it is easy to show that $\sum_{p' \in P^*_{s,d}} F'(p') \geq \theta(\mathcal{M}, F) \cdot m_{s,d}$ using the fact that F has a throughput $\theta(\mathcal{M})$ i.e., $\sum_{p \in P_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d}$, for all s, d source-destination pairs. This concludes that the flow F' has throughput $\theta(\mathcal{M}, F') = \theta(\mathcal{M}, F)$. □

Using Lemma 6, Lemma 7 and Lemma 8, it is now straight-forward that our claim in Theorem 13 holds.

***Proof of Theorem** 13.* From Lemma 6 and Lemma 7, for any feasible temporal flow $\mathcal{F}$ in the evolving graph with throughput $\theta(\mathcal{M}, \mathcal{F})$, there exists a feasible flow F in the emulated graph with the same throughput $\theta(\mathcal{M}, F) = \theta(\mathcal{M}, \mathcal{F})$ and vice versa. From Lemma 8, we have that, our definition of extended paths and its use in our analysis does not impact the throughput of emulated graph i.e., throughput with our definition of extended set of paths is same as the throughput with the standard definition of paths. This concludes that the evolving graph and the corresponding emulated graph have the same throughput $\theta(\mathcal{M})$: the maximum scaling factor given a demand matrix and the same throughput θ^* under a worst-case demand matrix. □

Corollary 1 (Reduction to simple graph)**.** *Throughput of a periodic evolving graph $\mathcal{G} = (V, \mathcal{E})$ is equivalent to the simple static graph $G(\mathcal{G}) = (V, E)$, where V is the set of vertices (same as the evolving graph) and $E = \bigcup_{t \in [0,\Gamma)} \mathcal{E}_t$ is the union of edges over one period of time (without any labels). The capacity of an edge $e \in E$ is given by,*

$$\hat{c}(e) = \left(\frac{1 - \Delta_u}{\Gamma} \right) \cdot \sum_{t \in [0,\Gamma)} c_t(e)$$

where $c_t(e)$ is the capacity of an edge $e \in \mathcal{E}_t$ at time t in the evolving graph.

Notice that the simple static graph in Corollary 1 is a weighted simple graph corresponding to the emulated graph (Definition 15) with the same amount of capacity between any $u, v \in V$ for every $(u, v) \in E$. It is standard in the literature that a multigraph and the corresponding weighted graph have the same max-flow [208].

■ ***Discussion.*** Theorem 13 enables us to study the throughput of periodic reconfigurable topologies with the techniques used in static graphs. We emphasize that our result is *general* for periodic reconfigurable topologies and is not specific to the existing systems; for instance the emulated graph could indeed be an expander graph, or even a non-regular graph, but our result still holds. We believe that our proof renders useful not only to study throughput maximization problem but also other variants of flow problems. For example, by using our technique to convert flows from the static to the periodic graph and vice versa one could easily derive a relation between the metric of interest in each graph. We leave this exploration for future work.

7.3.1.5 Revisiting Throughput of Static Topologies

We now shift our focus to obtain the throughput upper bound of static topologies for two reasons. First, from Theorem 13, the throughput of a periodic reconfigurable topology is equivalent to its corresponding emulated graph (a *static graph*). Second, the best known throughput upper bound [204] for static topologies is specific to certain datacenter topologies and is not general enough (see §7.1). To this end, we revisit the problem and analyze it again using the formal definitions typically used in the literature. We find that, the throughput of a static topology is directly proportional to the total capacity and inversely proportional to the *average route length*, which generalizes the recently derived throughput upper bound [204] for shortest paths.

Definition 17 (Average route length)**.** *Given a demand matrix $\mathcal{M}$, a static graph $G = (V, E)$ and a flow F that satisfies the demand, the average route length denoted by $\mathrm{ARL}(\mathcal{M}, F)$ is defined by,*

$$\mathrm{ARL}(\mathcal{M}, F) = \sum_{s,d \in V} \sum_{p \in P_{s,d}} \frac{m_{s,d}}{M} \cdot r_p \cdot \mathrm{len}(p)$$

where $M = \sum_{s \in V} \sum_{d \in V} m_{s,d}$; r_p is the fraction of s-d demand transmitted on the path p and $\mathrm{len}(p)$ is the length of the path p.

Theorem 14 (Throughput)**.** *Given a demand matrix $\mathcal{M}$ and a flow F, the throughput $\theta(\mathcal{M}, F)$ of a static graph represented as $G = (V, E)$ is given by Equation 7.9. The graph G has throughput $\theta(\mathcal{M}) = \max_F \theta(\mathcal{M}, F)$: the maximum scaling factor given a demand matrix $\mathcal{M}$; and has throughput $\theta^* = \min_{\mathcal{M}} \theta(\mathcal{M})$ for a worst-case demand matrix.*

$$\theta(\mathcal{M}, F) \leq \frac{\hat{C}}{M \cdot \mathrm{ARL}(\mathcal{M}, F)} \tag{7.9}$$

where $\hat{C} = \sum_{e \in E} \hat{c}(e)$ is the total capacity of the network, $M = \sum_{s,d \in V} m_{s,d}$ is the total demand for the network and $\mathrm{ARL}(\mathcal{M}, F) = \sum_{s,d \in V} \sum_{p \in P_{s,d}} \frac{m_{s,d}}{M} \cdot r_p \cdot \mathrm{len}(p)$ is the average route length *for $\mathcal{M}$ and F, where r_p is the fraction of demand transmitted on the path p.*

Proof. We restrict our analysis to saturated demand matrices. Let the static topology under consideration be a graph $G = (V', E)$, where V' is the set of all vertices and E is the set of all edges. A set $V \subseteq V'$ denotes the set of all vertices

generating traffic (demand). Specifically, given any set of vertices V, a demand matrix $\mathcal{M}$ from Definition 6 has elements $m_{u,v}$ such that $\sum_{v \in V} m_{u,v} = c(u)$ for all $u \in V$ where $c(u)$ is the total *physical* bandwidth of the vertex u. We denote the capacity of an edge e by $\hat{c}(e)$.

Given a graph $G = (V', E)$, the set of vertices generating traffic (demand) $V \subseteq V'$, a demand matrix $\mathcal{M}$ for V, and a flow F, the relation between the throughput as a function of demand matrix $\mathcal{M}$ is given by Definition 23. From Definition 9, for F to be legal we have that,

$$\sum_{p \in P} F(p) \cdot \mathbb{I}(e \in p) \leq \hat{c}(e) \quad \forall e$$

summing over all the edges $e \in E$, we obtain the following:

$$\sum_{e \in E} \sum_{p \in P} F(p) \cdot \mathbb{I}(e \in p) \leq \sum_{e \in E} \hat{c}(e) \tag{7.10}$$

From Definition 23, a flow F achieves throughput $\theta(\mathcal{M}, F)$ if it obeys the following constraint:

$$\sum_{p \in P_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d} \quad \forall s \in V,\ d \in V$$

Let $r_p = \frac{F(p)}{\sum_{p \in P_{s,d}} F(p)}$ for all $p \in P_{s,d}$ for each s, d pair. We now expand the inequality in Equation 7.10 as follows:

$$\sum_{s \in V} \sum_{d \in V} \sum_{e \in E} \sum_{p \in P_{s,d}} F(p) \cdot \mathbb{I}(e \in p) \leq \sum_{e \in E} \hat{c}(e)$$

we convert the summation over all edges $e \in E$ to a summation over only edges of each path $e \in p$ and drop the identifier $\mathbb{I}$ without changing the value of the LHS in the above inequality:

$$\sum_{s \in V} \sum_{d \in V} \sum_{p \in P_{s,d}} \sum_{e \in p} F(p) \leq \sum_{e \in E} \hat{c}(e)$$

substituting $F(p) = r_p \cdot \sum_{p \in P_{s,d}} F(p)$ and since $\sum_{p \in P_{s,d}} F(p) \geq \theta(\mathcal{M}, F) \cdot m_{s,d}$:

$$\sum_{s \in V} \sum_{d \in V} \sum_{p \in P_{s,d}} \sum_{e \in p} \theta(\mathcal{M}, F) \cdot m_{s,d} \cdot r_p \leq \sum_{e \in E} \hat{c}(e)$$

Since $\theta(\mathcal{M}, F) \cdot m_{s,d} \cdot r_p$ is constant for all edges of a given path, the summation over all edges $e \in p$ gives $\text{len}(p) \cdot \theta(\mathcal{M}, F) \cdot m_{s,d} \cdot r_p$ where $\text{len}(p)$ denotes the number of edges in the path p or simply the length of the path.

$$\sum_{s \in V} \sum_{d \in V} \sum_{p \in P_{s,d}} \text{len}(p) \cdot \theta(\mathcal{M}, F) \cdot m_{s,d} \cdot r_p \leq \sum_{e \in E} \hat{c}(e)$$

Finally from the inequality, we obtain the throughput upper bound given a demand matrix as follows,

$$\theta(\mathcal{M}, F) \leq \frac{\sum_{e \in E} \hat{c}(e)}{\sum_{s \in V} \sum_{d \in V} m_{s,d} \cdot \left(\sum_{p \in P_{s,d}} \text{len}(p) \cdot r_p \right)}$$

where $\sum_{p \in P_{s,d}} \text{len}(p) \cdot r_p$ is conceptually the average route length from s to d. Let $M = \sum_{s \in V} \sum_{d \in V} m_{s,d}$ and $\hat{C} = \sum_{e \in E} \hat{c}(e)$,

$$\theta(\mathcal{M}, F) \leq \frac{\hat{C}}{M \cdot \text{ARL}(\mathcal{M}, F)} \tag{7.11}$$

where $\text{ARL}(\mathcal{M}, F) = \sum_{s,d \in V} \sum_{p \in P_{s,d}} \frac{m_{s,d}}{M} \cdot r_p \cdot \text{len}(p)$ is the *average route length* for $\mathcal{M}$ and F.
Given a demand matrix, the above inequality varies based on the flow F. In order to maximize throughput, we rewrite the above inequality as a maximum over possible flow F.

$$\theta(\mathcal{M}) = \max_{F} \frac{\hat{C}}{M \cdot \text{ARL}(\mathcal{M}, F)} \tag{7.12}$$

The throughput under a worst-case demand matrix is then the minimum $\theta(\mathcal{M})$ over the set of all saturated demand matrices.

$$\theta^* = \min_{\mathcal{M}} \max_{F} \frac{\hat{C}}{M \cdot \text{ARL}(\mathcal{M}, F)} \tag{7.13}$$

□

We note that providing a *lower bound* for the $\text{ARL}(\mathcal{M}, F)$, even when M and/or F are not known would provide an upper bound for θ^*. One such example is to take the shortest paths in the topology, as was done in [204]. But this will not always lead to a tight bound (see §7.1.3), for example in the complete graph K_n. While efficiently computing the throughput upper bound is an on-going research [145, 204, 247], Theorem 14 shows that computing throughput essentially boils down to computing average route lengths.

■ ***Discussion.*** Theorem 14 indeed expresses that the throughput maximization problem essentially boils down to minimizing *average route length* as opposed to shortest path lengths [204] or average shortest paths [247]. Both Theorem 13 and Theorem 14 provide a concrete understanding on the achievable throughput of periodic reconfigurable topologies. First, Theorem 13 provides a relation between the throughput of periodic reconfigurable topologies and the throughput of static topologies. Second, Theorem 14 provides the throughput of static topologies more concretely. Finally, for the worst-case demand matrix, we rely on an important result stated in [204] (and even earlier informally in [145, 247]) and claims that the worst-case demand matrix in a static topology is a specific permutation matrix (longest-matching). Consequently the worst-case demand matrix for *any* periodic reconfigurable topology, due to Theorem 13, is a specific permutation matrix in the corresponding emulated graph[3].

However, so far we still assumed that *(i)* a certain amount of buffer is available at each intermediate node and *(ii)* any amount of delay is tolerable. Hence, we study two important questions in the remainder of this section. We first seek to understand whether high throughput provided by periodic reconfigurations comes at the cost of inflated delay.

[3] A similar but less general result was recently claimed for specific RDCN designs [114]. In particular, it holds only for designs which emulate a complete graph, while our result holds for *any* emulated graph due to our new result in Theorem 13.

(Q1) Delay: *What is the relation between throughput and delay in a periodic reconfigurable topology?*

Given the increasing gap between capacity growth and switch buffers, we are further interested in investigating whether periodic reconfigurable topologies address the near-end of Moore's law w.r.t buffer requirements.

(Q2) Buffer: *How much buffer is required at each node to achieve the throughput upper bound stated in Theorem 14?*

7.3.2 Delay of Periodic RDCNs

The periodic reconfigurations naturally introduce a certain delay. In our context, delay is the time it takes for a packet sent from a source to reach its destination in the periodic evolving graph, *without experiencing congestion from other packets.* The maximum delay for a path δ is bounded by $\text{len}(\delta) \cdot \Gamma$ since every consecutive edge in the path is available within Γ timeslots (period). We are mainly interested in the maximum delay incurred by feasible paths which typically reflects in the tail latencies observed in a datacenter. In the following, we state the relation between throughput θ^* and delay. We first formally define the delay of a temporal path.

Definition 18 (Temporal path delay)**.** *Given a periodic evolving graph $\mathcal{G}$ and the set of all temporal paths $\mathcal{P}$, the delay of a temporal path $\delta \in \mathcal{P}$ denoted by $L(\delta)$ is time it takes when the first edge $e_1 \in \delta$ is accessed until the time the flow reaches the destination on the last edge $e_n \in \delta$. Formally, for a path $\delta \in \mathcal{P}$, if $\langle (e_1, t_1), (e_2, t_2) \ldots (e_n, t_n) \rangle$ is the sequence of edge-time values, then $L(\delta) = (t_n - t_1 + 1) \cdot \Delta + (\Gamma - 1) \cdot \Delta$ where Δ is the absolute time of each timeslot and Γ is the period of the evolving graph.*

Clearly, the delay is lower bounded by $\text{len}(\delta) \cdot \Delta + (\Gamma - 1) \cdot \Delta$ since it takes Δ (timeslot) amount of time to send over each edge $e \in \delta$ and there is always an inherent delay of $(\Gamma - 1)$ timeslots due to the periodic nature. Similarly, the delay is upper bounded by $\text{len}(\delta) \cdot \Gamma \cdot \Delta$ since the delay between accessing each edge along a path can be at most Γ timeslots.

We define *average route delay* similar to average route length (Definition 17, which plays a key role in our analysis of the maximum latency.

Definition 19 (Average route delay)**.** *Given a demand matrix $\mathcal{M}$, a periodic evolving graph $\mathcal{G}$ and a flow $\mathcal{F}$ that satisfies the demand, the average route delay denoted by* $\text{ARD}(\mathcal{M}, \mathcal{F})$ *is defined by,*

$$\text{ARD}(\mathcal{M}, \mathcal{F}) = \sum_{s,d \in V} \frac{m_{s,d}}{M} \cdot L_{s,d}$$

where $M = \sum_{s \in V} \sum_{d \in V} m_{s,d}$; $L_{s,d} = \sum_{\delta \in \mathcal{P}^0_{s,d}} r_\delta \cdot L(\delta)$ is the s-d delay; r_δ is the fraction of s-d demand transmitted on the temporal path δ and $L(\delta)$ is the delay of the temporal path δ.

We now state the lower bound for the maximum delay in a periodic evolving graph.

Theorem 15 (Delay)**.** *Given a demand matrix $\mathcal{M}$, a n_u-regular periodic evolving graph $\mathcal{G}$ emulating a d-regular graph with a period of Γ timeslots each of duration Δ and a flow $\mathcal{F}$ that achieves throughput $\theta(\mathcal{M},\mathcal{F})$, the average route delay $\mathrm{ARD}(\mathcal{M},\mathcal{F})$ and the maximum delay L_{max} are given by*[4]*,*

$$\begin{aligned} L_{max} \geq \mathrm{ARD}(\mathcal{M},\mathcal{F}) &= \mathrm{ARL}(\mathcal{M},\mathcal{F}) \cdot \Gamma \cdot \Delta \\ &\geq \Omega\left(\frac{d \cdot \Delta}{n_u \cdot \theta(\mathcal{M},\mathcal{F})}\right) \end{aligned} \tag{7.14}$$

where $\mathrm{ARD}(\mathcal{M},\mathcal{F}) = \sum_{s,d\in V}\sum_{\delta}\frac{m_{s,d}}{M}\cdot r_\delta \cdot L(\delta)$ *is the* average route delay *for $\mathcal{M}$ and $\mathcal{F}$, where r_δ is the fraction of demand transmitted on the legal temporal path δ, and $L(\delta)$ is the delay of the path δ, $M = \sum_{s\in V}\sum_{d\in V} m_{s,d}$ is the total demand. In particular, for a worst-case demand matrix, the maximum latency is bounded by*

$$L_{max} \geq \Omega\left(\frac{d \cdot \Delta}{n_u \cdot \theta^*}\right) \tag{7.15}$$

Proof. Based on Definition 14, for every s-d source-destination pair, the source generates $\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d}$ flow on average in bits per second in every period. From Definition 19, $L_{s,d}$ denote the average delay between a source s and destination d. Then the destination only receives its first data at $L_{s,d}$ time later, implying that the source has already generated $\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d} \cdot (L_{s,d})$ amount of data. From then on, the source transmits $\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d}$ average flow in bits per second and the destination receives $\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d}$ average flow in bits per second in every period. Due to conservation of flow, for every source-destination pair, the data generated until $L_{s,d}$ delay i.e., $\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d} \cdot (L_{s,d})$ amount of data, circulates (moves between source-destination) in the network in every period. As a result, the capacity consumed by each s-d flow is $\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d} \cdot \frac{L_{s,d}}{\Gamma\cdot\Delta}$. The total capacity utilized is then $\sum_{s\in V}\sum_{d\in V}\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d} \cdot \frac{L_{s,d}}{\Gamma\cdot\Delta}$. However, the total utilized capacity can also be written as $\theta(\mathcal{M},\mathcal{F}) \cdot M \cdot \mathrm{ARL}(\mathcal{M},\mathcal{F})$, where $\theta(\mathcal{M},\mathcal{F}) \cdot M$ is the total bits per second generated by the sources and $\mathrm{ARL}(\mathcal{M},\mathcal{F})$ is the average route length. Equating the above two, we have that,

$$\sum_{s\in V}\sum_{d\in V}\theta(\mathcal{M},\mathcal{F}) \cdot m_{s,d} \cdot \frac{L_{s,d}}{\Gamma\cdot\Delta} = \theta(\mathcal{M},\mathcal{F}) \cdot M \cdot \mathrm{ARL}(\mathcal{M},\mathcal{F})$$

$$\frac{1}{M}\sum_{s\in V}\sum_{d\in V} m_{s,d} \cdot L_{s,d} = \mathrm{ARD}(\mathcal{M},\mathcal{F}) = \mathrm{ARL}(\mathcal{M},\mathcal{F}) \cdot \Gamma \cdot \Delta$$

Since the maximum delay L_{max} is atleast the average delay,

$$L_{max} \geq \mathrm{ARD}(\mathcal{M},\mathcal{F}) = \mathrm{ARL}(\mathcal{M},\mathcal{F}) \cdot \Gamma \cdot \Delta$$

The period Γ is at least $\frac{d}{n_u}$ timeslots since the node degree is limited to the number of uplinks n_u in each timeslot. Further, since the total capacity $\hat{C}$ equals the total demand M for saturated demand matrices; and using the relation between ARL and θ^* from Theorem 14, for a worst-case demand matrix we obtain the following:

$$L_{max} \geq \Omega\left(\frac{d \cdot \Delta}{n_u \cdot \theta^*}\right)$$

□

[4] Ω is the asymptotic lower bound notation.

■ ***Discussion.*** Using Theorem 15, we illustrate the throughput and delay relation for different values of d in Figure 7.1. Here d is the degree of the graph emulated by a periodic reconfigurable topology. Note that the worst-case demand matrix (a permutation matrix) specifies non-zero demands between ToR pairs at maximum distance in the graph [204] and hence the average route lengths ARL is close to the diameter which is bounded by $\log_d(n_t)$ for d-regular graphs. Hence the throughput θ^* is inversely proportional to $\log_d(n_t)$ whereas the delay based on Eq. (7.15) is proportional to $d \cdot \log_d(n_t)$. Specifically, we notice that the existing designs which emulate a complete graph achieve the highest throughput but at the cost of high delay.

7.3.3 Buffer Requirements of Periodic RDCNs

With the understanding of the relation between throughput and delay of periodic reconfigurable topologies, we now study their buffer requirements. We assume that each node is equipped with a memory region shared across the entire device and hence buffer is an aggregate value per node in our analysis. This model is similar to shared memory architectures with complete sharing [16]. Notice from Definition 12 and Definition 13 that it requires to ''store'' flow for a certain time and ''forward'' the flow when the next edge along a path is available. As a result, every node in the evolving graph requires certain amount of buffer space to store and forward flow on temporal paths in the evolving graph. Our analysis reveals an interesting inequality for the required buffer, which is conceptually of the well-known ''bandwidth-delay product'' form.

Theorem 16 (Buffer)**.** *Given a demand matrix $\mathcal{M}$, a periodic evolving graph $\mathcal{G}$ requires at least $\hat{B}$ total amount of buffer in the network in order to achieve throughput $\theta(\mathcal{M}, \mathcal{F})$.*

$$\hat{B} \geq (\theta(\mathcal{M}, \mathcal{F}) \cdot M) \cdot \mathrm{ARD}(\mathcal{M}, \mathcal{F}) \tag{7.16}$$

where $\hat{B} = \sum_{u \in V} B(u)$ is the total buffer of the network and $B(u)$ is the available buffer at a node u; $M = \sum_{s,d \in V} m_{s,d}$ is the total demand for the network; $\mathrm{ARD}(\mathcal{M}, \mathcal{F}) = \sum_{s,d \in V} \sum_{\delta} \frac{m_{s,d}}{M} \cdot r_\delta \cdot L(\delta)$ *is the* average route delay *for $\mathcal{M}$ and $\mathcal{F}$, where r_δ is the fraction of demand transmitted on the legal temporal path δ.*

Proof. Let $B(u)$ be the amount of available (and used) buffer at a node $u \in V$. Then at any time $T \geq 0$, the difference between the total flow arrived and departed from node u (except the flow originating and terminating at u) is the amount of flow stored at u, formally expressed below. For simplicity, we set $\Delta_t = \Delta - \Delta_r$ and $\frac{\Delta_t}{\Delta} = 1 - \frac{\Delta_r}{\Delta} = 1 - \Delta_u$. We denote the set of incoming (outgoing) edges of a node u at any time t by $\mathcal{E}_t^-(u)$ ($\mathcal{E}_t^+(u)$).

$$B(u) \geq \sum_{t=0}^{T} \sum_{s,d \in V \setminus \{u\}} \sum_{\delta \in \mathcal{P}_{s,d}} \left(\sum_{e \in \mathcal{E}_t^-(u)} \mathcal{F}(\delta, e, t) - \sum_{e \in \mathcal{E}_t^+(u)} \mathcal{F}(\delta, e, t) \right) \cdot \Delta \tag{7.17}$$

For simplicity, we define the sum of all flows arriving and departing from a node u at any time t as follows:

$$R_t^-(u) = \sum_{s,d \in V} \sum_{\delta \in \mathcal{P}_{s,d}} \sum_{e \in \mathcal{E}_t^-(u)} \mathcal{F}(\delta, e, t)$$

$$R_t^+(u) = \sum_{s,d \in V} \sum_{\delta \in \mathcal{P}_{s,d}} \sum_{e \in \mathcal{E}_t^+(u)} \mathcal{F}(\delta, e, t)$$

Notice that at any time t, $\sum_{u \in V} R_t^-(u) = \sum_{u \in V} R_t^+(u)$. Since we take the summation over all nodes, for every outgoing flow on an edge, there is a corresponding equal flow incoming at the other end of the edge.
We now simplify Equation 7.17 using the above notation. Note that $\sum_{s \in V \setminus \{u\}} \sum_{\delta \in \mathcal{P}_{s,u}} \sum_{e \in E^-(u)} \mathcal{F}(\delta, e, t)$ is same as $\sum_{s \in V} \sum_{\delta \in \mathcal{P}_{s,u}} \sum_{e \in E^-(u)} \mathcal{F}(\delta, e, t)$ since $\mathcal{P}_{u,u}$ is a null set.

$$B(u) \geq \sum_{t \in [0,T]} \left(R_t^-(u) - \sum_{s \in V} \sum_{\delta \in \mathcal{P}_{s,u}} \sum_{e \in E^-(u)} \mathcal{F}(\delta, e, t) \right) \cdot \Delta$$
$$- \sum_{t \in [0,T]} \left(R_t^+(u) - \sum_{d \in V} \sum_{\delta \in \mathcal{P}_{u,d}} \sum_{e \in E^+(u)} \mathcal{F}(\delta, e, t) \right) \cdot \Delta$$

Summing over all the nodes $u \in V$, we obtain the following,

$$\sum_{u \in V} B(u) \geq \overbrace{\sum_{u \in V} \left(\sum_{t \in [0,T]} \sum_{d \in V} \sum_{\delta \in \mathcal{P}_{u,d}} \sum_{e \in E^+(u)} \mathcal{F}(\delta, e, t) \right) \cdot \Delta}^{\text{Data sent out from each source } u \in V}$$
$$\underbrace{- \sum_{u \in V} \left(\sum_{t \in [0,T]} \sum_{s \in V} \sum_{\delta \in \mathcal{P}_{s,u}} \sum_{e \in E^-(u)} \mathcal{F}(\delta, e, t) \right) \cdot \Delta}_{\text{Data received at each destination } u \in V}$$

Let $L(\delta)$ denote the temporal path delay. $\mathcal{F}(\delta) \cdot \frac{L(\delta) \cdot (1 - \Delta_u)}{\Gamma}$ is the amount of flow sent on a periodic path before the first data arrives at the destination i.e., $\mathcal{F}(\delta) \cdot \Delta \cdot (1 - \Delta_u)$ is the data sent over Γ timeslots ($\Gamma \cdot \Delta$ time) and $\mathcal{F}(\delta) \cdot \frac{L(\delta) \cdot (1 - \Delta_u)}{\Gamma}$ is the data sent over $L(\delta)$ time. For each s-d source-destination, the total data received at the destination d until time T is the total data sent from source s until time T minus the total data sent from source s before the destination d receives its first data.

$$\sum_{u \in V} B(u) \geq \sum_{u \in V} \left(T \cdot \Delta \cdot \sum_{d \in V} \theta(\mathcal{M}, \mathcal{F}) \cdot m_{u,d} \right)$$
$$- \sum_{u \in V} \left(T \cdot \Delta \cdot \sum_{s \in V} \theta(\mathcal{M}, \mathcal{F}) \cdot m_{s,u} \right)$$
$$+ \sum_{u \in V} \sum_{s \in V} \sum_{\delta \in \mathcal{P}_{s,u}^0} \frac{\mathcal{F}(\delta) \cdot L(\delta) \cdot (1 - \Delta_u)}{\Gamma}$$

Finally, the above inequality reduces to,

$$\sum_{u \in V} B(u) \geq \sum_{s \in V} \sum_{d \in V} \sum_{\delta \in \mathcal{P}_{s,d}^0} \mathcal{F}(\delta) \cdot \frac{L(\delta) \cdot (1 - \Delta_u)}{\Gamma}$$

We expand the above summation $\sum_{\delta \in \mathcal{P}_{s,d}^0} \mathcal{F}(\delta)$ by multiplying and dividing by

$\theta(\mathcal{M},\mathcal{F})\cdot m_{s,d}$ where $\theta(\mathcal{M},\mathcal{F})\cdot m_{s,d} = \sum_{\delta\in\mathcal{P}^0_{s,d}} \frac{\mathcal{F}(\delta)\cdot(1-\Delta_u)}{\Gamma}$ from Definition 14.

$$\sum_{u\in V} B(u) \geq$$

$$\sum_{s\in V}\sum_{d\in V}\left(\frac{\theta(\mathcal{M},\mathcal{F})\cdot m_{s,d}\cdot\Gamma\cdot\sum\limits_{\delta\in\mathcal{P}^0_{s,d}}\mathcal{F}(\delta)\cdot\frac{L(\delta)\cdot(1-\Delta_u)}{\Gamma}}{\sum\limits_{\delta\in\mathcal{P}^0_{s,d}}\mathcal{F}(\delta)\cdot(1-\Delta_u)}\right)$$

Let r_δ denote the fraction of temporal flow on a path $\delta\in P^0_{s,d}$ i.e., $r_\delta = \frac{\mathcal{F}(\delta)}{\sum_{\delta\in\mathcal{P}^0_{s,d}}\mathcal{F}(\delta)}$.

$$\sum_{u\in V} B(u) \geq \theta(\mathcal{M},\mathcal{F})\cdot\sum_{s\in V}\sum_{d\in V}\left(m_{s,d}\cdot\sum_{\delta\in\mathcal{P}^0_{s,d}} r_\delta\cdot L(\delta)\right)$$

Let $M=\sum_{s\in V}\sum_{d\in V} m_{s,d}$ and the *average route delay* for $\mathcal{M}$ and $\mathcal{F}$ is denoted by $\mathrm{ARD}(\mathcal{M},\mathcal{F}) = \sum_{s,d\in V}\sum_{\delta\in\mathcal{P}_{s,d}}\frac{m_{s,d}}{M}\cdot r_\delta\cdot L(\delta)$. The total buffer in the network is $\hat{B}=\sum_{u\in V}B(u)$. We obtain the following,

$$\hat{B}\geq(\theta(\mathcal{M},\mathcal{F})\cdot M)\cdot\mathrm{ARD}(\mathcal{M},\mathcal{F})$$

□

■ ***Discussion.*** Theorem 16 reveals a non-trivial tradeoff across the spectrum of periodic reconfigurable topologies which we discuss in detail in §7.3.5. Our result for the required buffer in Theorem 16 intuitively resembles the ‘‘bandwidth-delay product’’ commonly used in the TCP literature. A ‘‘pipe’’ must have at least a bandwidth-delay product amount of bytes in transit (or inflight) in order to achieve full utilization. Similarly, our result shows that the total buffer in a periodic reconfigurable network must be at least the product of total demand (in bits per second) and the average route delay. The required buffer stems from the waiting times at intermediate nodes along a path due to the periodic reconfigurations. We believe this analogy may also render useful for instance to predict the overall graph throughput based on the configuration of transport protocols. For example, the window size[5] of a transport protocol typically relates to the amount of buffering, and the maximum window size may relate to the overall periodic graph throughput. We leave it for future work to study such relations in detail.

7.3.4 Remarks and Discussion on Theorems 13-16

Our results in the previous sections allow us to answer and shed light on several basic questions also discussed in the literature [60, 175, 189].

Can buffering at the end-hosts instead of ToRs alleviate the problems of excessive buffering?

Our result in Theorem 16 expresses the *total* amount of buffering required in the network. We note that this buffering can either be done at the ToR switches or can be moved to the end-hosts. Several works in the past [175, 189] informally discussed the need for in-network buffers but outside the circuit switching layer[6].

[5] The window size of a transport protocol is typically the maximum amount of bytes allowed to be in transit at any point in time.

[6] Recall that circuit switches are bufferless.

Two-hop and in general multi-hop routing further worsens the need for additional in-network buffers in periodic reconfigurable networks [189]. To this end, prior works proposed to move the buffering from ToR switches to the end-hosts with specialized synchronization and packet pause/unpause techniques between the end-hosts and the ToR switches [175]. These techniques are motivated based on the fact that end-host DRAM memory is cheap and abundant while on-chip ToR buffers are costly and limited in size. Nevertheless, excessive buffering at the end-hosts still poses the well-known problems of queuing delay, although it alleviates the problems of scarce buffer resource at the ToRs. The question of buffering at the end-hosts vs ToR switches poses an inherent tradeoff between *a)* simplicity by buffering at the ToR but at the cost of large on-chip costly buffers and *b)* cost-effectiveness by buffering at the end-hosts but at the cost of complex techniques that consume additional CPU resources and potentially added processing delays.

Can nanosecond reconfiguration mask the problems of delay and buffer needs?

Notice that delay (from Theorem 15) is directly proportional to the timeslot duration Δ. Naturally, if the reconfiguration delay Δ_r is small, the timeslot duration Δ can also be reduced. As a result, the maximum delay and the average route delay can be significantly reduced. However, notice that the buffer requirements (from Theorem 16) are directly proportional to the timeslot duration Δ as well as the total capacity e.g., the buffer requirements would remain the same if all the links in the network are increased in capacity by 2x even if the reconfiguration time is reduced by 2x.

What is the difference between the degree of the emulated graph and the physical topology?

We emphasize that the emulated graph is obtained by the union of edges of the periodic evolving graph (ToR-to-ToR connectivity) over one period of a time-interval. Hence, the node degree of the emulated graph is the number of ToR switches that are directly connected to a given ToR switch (over one period). In contrast, the node degree of the physical topology is the number of circuit switches that are directly connected to a given ToR switch at any time. In particular, the node degree in the physical topology is given by n_u: the number of uplinks. However, for example, both RotorNet and Sirius emulate a complete graph and hence have the same node degree in the emulated graph given by n_t: the total number of ToR switches. Hereafter in this chapter, the emulated graph and its degree play an important role (§7.4) especially as seen in Theorem 13 and Theorem 15 where d is the degree.

Are the results applicable to Valiant routing?

We stress that our results in Theorem 13, 14, 15, 16 are general for an ideal routing scheme and are *not* based on Valiant load balancing. We only assume the specific case of Valiant load balancing later in §7.4.

7.3.5 Tradeoffs & Optimization Opportunity

Interestingly, the relation between throughput and delay is a concave function without buffer constraints (Theorem 14) but is a convex function for a fixed amount of buffer (Theorem 16). Figure 7.1 illustrates the tradeoffs that arise

across the design space of periodic reconfigurable topologies. Specifically, for a certain fixed amount of buffer, existing designs emulating a complete graph are no longer throughput optimal as the available buffer simply cannot hold enough traffic to achieve the ideally achievable maximum. Further, throughput and delay pose a fundamental tradeoff in terms of performance.

Although periodic reconfigurable datacenter designs were developed in view of bandwidth scaling problem posed by the gradual end of Moore's law, the required buffer with linear relation to throughput does not indicate high scalability in future. Several studies in the recent past show an increasing gap between the capacity growth and switch buffers [59, 112]. Specifically, the top-of-the-rack buffers are extremely shallow and the buffer per port per Gbps has been gradually decreasing over the recent years. In the following, we summarize our key motivation to explore the design space of topologies.

- **Throughput:** If buffer is not a concern, then emulating a complete graph (existing designs) results in high throughput but at the cost of high delay.
- **Delay:** If delay is a concern, even without buffer concerns, emulating a d-regular graph where $d < n_t$ results in lower delay but at the cost of throughput.
- **Buffer:** Finally, if buffer is a concern, there exists a d-regular graph that maximizes throughput within the buffer limits.

Our goal in this work is to design a periodic reconfigurable topology that maximizes throughput given the buffer and delay requirements.

7.4 Mars: Near-Optimal Throughput RDCN with Shallow Buffers

Reflecting on our observations in §7.3, we study the family of periodic reconfigurable topologies that emulate a d-regular graph. Specifically, our aim is to systematically determine the high throughput topology within the underlying buffer limits.

7.4.1 Overview

We first give an intuition on our design. Recall that the network consists of n_t ToR switches each with n_u in/out ports interconnected via optical circuit switches.

Mars emulates a d-regular graph with near-optimal throughput: Given the importance of the graph emulated by the topology, MARS emulates a "good" d-regular graph with the degree d optimized for throughput with the limited available buffer. Specifically, the degree d of the emulated graph influences two factors *(i)* average path lengths which relates to throughput and *(ii)* delay which relates to the buffer requirements. Among the set of all d-regular graphs, we are interested in the "good" graphs with diameter close to $\log_d(n_t)$, where n_t is the number of ToR switches. Note that the lower bound for diameter of any d-regular graph is $\log_d(n_t)$. Following prior work [60], we assume Valiant load balancing [258] for the routing scheme for simplicity. This inflates the average route length by a factor

of two i.e., $2 \cdot \log_d(n_t)$. Notice that emulating a K_{n_t} complete graph with large degree would result in shorter average route length. However, as we will see later, emulating a degree $d \leq n_t$ results in better delay and better throughput under limited buffer.

A parametrized approach: We parametrize our design based on two values *(i)* delay requirement L and *(ii)* buffer limit B. The degree d of the emulated graph of MARS depends on the delay and buffer requirements.

Delay and buffer requirements: MARS finds a balance between throughput, delay and buffer. Specifically in Figure 7.1, MARS is the topology at the intersection of throughput upper bound with and without buffer restrictions. To this end, MARS maximizes the throughput while finding a balance in delay and buffer requirements.

7.4.2 Properties of Mars

We first express the throughput of MARS without delay and buffer constraints which gives an intuition on the ideally achievable throughput.

Theorem 17 (Unconstrained Throughput Upper Bound)**.** *The throughput of MARS connecting n_t ToR switches each with n_u in/out ports, emulating a d-regular graph without any delay and buffer constraints, under Valiant load balancing, is given by:*

$$\theta^* = \frac{1}{ARL} \approx \frac{1}{2 \cdot \log_d(n_t)}$$

First the diameter of MARS's emulated graph is close to $\log_d(n_t)$. Second, we then argue that the worst-case permutation demand matrix specifies non-zero demands between ToR pairs which are separated by a distance close to diameter. Further, Valiant load balancing inflates the *route lengths* by a factor of 2 e.g., similar to existing designs [60] i.e., ARL $= 2 \cdot \log_d(n_t)$ under worst-case demand matrix and hence $\theta^* = \frac{1}{2 \cdot \log_d(n_t)}$. For instance, in the case of $d = n_t$ (emulating a complete graph), we obtain $\theta^* = \frac{1}{2}$.

With the understanding of the throughput of MARS, we can now relate its throughput to delay. Using our result in Theorem 15, the delay L_{max} of MARS is related to throughput as $\Omega(\frac{d \cdot \Delta}{n_u \cdot \theta^*}) = \Omega(\frac{2 \cdot (\log_d n_t) \cdot d \cdot \Delta}{n_u})$. Given a constraint on delay i.e., the topology must ideally incur a delay less than L, we state the optimal degree for MARS in the following.

Theorem 18 (Optimal degree d with delay constraints)**.** *The optimal d-regular graph emulated by MARS that maximizes throughput (given the delay requirement L and under Valiant load balancing) has a degree d given by,*

$$d = \lfloor e^{-\mathcal{W}(k)} \rfloor$$

where $k = \frac{-2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$; $\mathcal{W}$ is the Lambert W function [89]; $\ln(.)$ is the natural logarithm and e is the Euler's number.

Proof. We begin by equating the delay of MARS and the desired delay L.

$$L = \frac{2 \cdot \log_d(n_t) \cdot d \cdot \Delta}{n_u}$$

Rearranging the terms in the above equation, we obtain the following,

$$\frac{\ln(d)}{d} = \frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$$

$$\frac{1}{d} \cdot \ln\left(\frac{1}{d}\right) = -\frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$$

For simplicity, let $-\frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}$ be some constant k.

$$\frac{1}{d} \cdot \ln\left(\frac{1}{d}\right) = k$$

The above equation is of the form $y \cdot \ln(y) = k$ whose solution is $y = e^{\mathcal{W}(k)}$ where $\mathcal{W}$ is the lambert W function [89].

$$\frac{1}{d} = e^{\mathcal{W}(k)}$$

Substituting k in the above, we obtain the following.

$$d = e^{-\mathcal{W}(k)} = e^{-\mathcal{W}\left(-\frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L}\right)}$$

Here, if $k < -\frac{1}{e}$, there exists no real solution. However, we show that k is always greater than $-\frac{1}{e}$ i.e., $k > -\frac{1}{e}$

$$k = -\frac{2 \cdot \ln(n_t) \cdot \Delta}{n_u \cdot L} \geq -\frac{2 \cdot \ln(n_t)}{n_u} \geq -2 \cdot \ln(n_t) > -\frac{1}{e}$$

The above inequalities hold since the latency L is at least one timeslot i.e., $L \geq \Delta$; the topology consists of at least two ToRs i.e., $n_t \geq 2$ and $\ln(2) > \frac{1}{e}$.

If $-\frac{1}{e} \leq k < 0$, there exists two real solutions for the degree d. In this case, we take the the highest value and round it to the nearest integer. Here, choosing the highest value d lowers the average route length $2 \cdot \log_d(n_t)$ and consequently maximizes throughput within the latency requirement. □

Our observations in §7.3 reveal that a key tradeoff arises across the buffer requirements and the achievable throughput. We are now interested in determining the optimal degree d of MARS for a limited buffer B at each node. As an intuition, a topology requires more buffer as the product of the scaled demand and the average route delay increases.

Theorem 19 (Optimal degree d with buffer constraints)**.** *The optimal degree d for the emulated graph of MARS independent of the specific flow that maximizes throughput, given a limited buffer $B \leq n_t \cdot \Delta$ at each node, is given by,*

$$d = \lfloor \frac{B}{c \cdot \Delta} \rfloor$$

where c is the capacity of every edge in the topology and Δ is the timeslot value.

Proof. From Theorem 16, the required buffer is $(\theta(\mathcal{M}, \mathcal{F}) \cdot M \cdot ARD(\mathcal{M}, \mathcal{F}))$. Further, from Theorem 15 we have that the delay is bounded by $\frac{d \cdot \Delta}{n_u \cdot \theta(\mathcal{M}, \mathcal{F})}$. Since $M = n_t \cdot n_u \cdot c$ (total demand and capacity), this gives us the buffer requirement of $n_t \cdot c \cdot d \cdot \Delta$ in the whole network. Given the regularity of the MARS, each node would then require $c \cdot d \cdot \Delta$ amount of buffer to hold the demand. Now, given the constraint that only B amount of buffer is available at each node, the optimal degree d is then $\frac{B}{c \cdot \Delta}$ rounded to the nearest integer. □

From Theorem 19, emulating a complete graph like the existing designs would require a buffer of $\frac{n_t}{n_u} \cdot (n_u \cdot c) \cdot \Delta = n_t \cdot c \cdot \Delta$ where $n_u \cdot c \cdot \Delta$ is the amount of data that can be sent out or received in a single timeslot and $\frac{n_t}{n_u}$ is the period for a complete graph. In contrast, a d-regular graph would only require $d \cdot c \cdot \Delta$ amount of buffer and achieves better throughput under limited buffer compared to a periodic topology emulating complete graph.

7.4.3 Interconnect

Switch size: Given n_t number of ToR switches each with n_u uplinks, MARS requires a switch size of at least $n_p = n_t$ similar to prior work [187, 189]. We need n_u such circuit switches to interconnect the datacenter. We leave it for future work to study the feasibility of using circuit switches with lower port count.

Wiring: Each ToR uses its n_u uplinks to connect to each of the n_u circuit switches.

Matchings: We first generate a d-regular directed graph. Specifically, in generating the emulated graph, we choose the d-regular graph for which the diameter approaches the lower bound of $\lceil \log_d(n_t) \rceil$ [70, 136, 137] e.g., deBruijn digraphs [93]. A straight-forward argument shows that a d-regular directed graph can be decomposed into d perfect matchings. The idea is to recursively find a 1-factor and delete it from the graph where each 1-factor by definition is a perfect matching [203, 218]. We shuffle the resulting d matchings and randomly assign $\frac{d}{n_u}$ matchings to each of the n_u circuit switches in the topology. Each circuit switch then cycles through the $\frac{d}{n_u}$ matchings periodically. We note that decomposing the d-regular emulated graph into matchings can be computationally expensive. However, this is performed only once at the time of deployment and the circuit switches cycle through the installed matchings periodically thereafter.

Note that the interconnect for MARS can be implemented using the same hardware as in prior work [60, 187, 189]; in fact, the only change required in MARS compared to these systems concerns the matchings configurations.

7.4.4 Example: deBruijn-based Emulated Graph

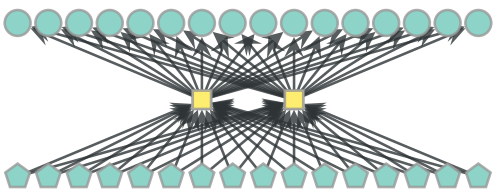
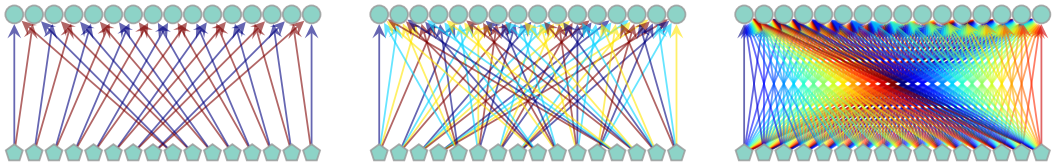


Figure 7.5: An example of a periodic RDCN interconnect with 16 ToR switches each with 2 in/out ports; 2 circuit switches each with 16 in/out ports. For better visualization, the ToR switch outputs are depicted as pentagons and inputs are depicted as circles. Circuit switches are depicted as squares.

We walk through an example to better illustrate the topology construction and the inherent tradeoffs. We consider an example with $n_t = 16$ ToR switches each with $n_u = 2$ in/out ports. The interconnect requires $n_s = n_u = 2$ optical circuit switches each of size $n_p = n_t = 16$. Each ToR uses its uplinks to connect to each



(a) Static 2-regular Emulated Graph (2 Matchings)

(b) 4-regular deBruijn Emulated Graph (4 Matchings)

(c) Complete Emulated Graph (16 Matchings)

Figure 7.6: A periodic reconfigurable topology (shown in Figure 7.5) can emulate a spectrum of graphs depending on the matchings schedule in the circuit switches. **(a)** A static emulated graph has the least delay but at the cost of throughput. **(b)** MARS finds the best degree of the emulated graph in order to achieve near-optimal throughput based on the available buffer and delay tolerance. **(c)** Existing designs emulate a complete graph trading delay for throughput and suffer in terms of throughput with shallow buffers. Each color in the figure corresponds to a matching. Since there are two circuit switches (shown in Figure 7.5), at any point in time, the union of two matchings corresponds to the ToR-to-ToR connectivity.

of the two circuit switches as shown in Figure 7.5. Circuits are set up for $90\mu s$ duration and it takes $10\mu s$ to reconfigure (timeslot $\Delta = 100\mu s$ for each matching) similar to RotorNet switches [189]. All the links have a capacity of $c = 400$Gbps.

Before constructing the sequence of matchings, we take one of the factors into consideration *(i)* buffer size of ToR switches and *(ii)* latency tolerance. Using Theorem 18 and Theorem 19, we determine the optimal degree d to maximize throughput. It remains to generate a d-regular emulated graph with optimal diameter and decompose it to a sequence of matchings which are then deployed in the circuit switches.

We consider the spectrum of deBruijn digraphs in generating the emulated graph. Specifically, for n_t ToR switches (vertices), $G = (V, E)$ is a deBruijn graph (emulated graph), where $V = \{0, 1, ..., n_t - 1\}$ is the set of vertices and $E = \{(u, v) \mid v \equiv (u \cdot d + a) \textbf{ mod } n_t, a \in \{0, 1, ..., d - 1\}\}$ is the edge set [137]. It is known that deBruijn graphs have a low diameter of $\lceil \log_d(n_t) \rceil$ which is close to Moore's bound for diameter. Our emphasis on the graph diameter is due to Theorem 14 as the average route length gets closer to diameter under a worst-case demand matrix which specifies non-zero demand between pairs at maximum distance in the graph.

Using the optimal degree d based on Theorem 18 and Theorem 19, we generate a d-regular directed deBruijn graph and decompose the edge set into d matchings. We shuffle the d-matchings and assign $\frac{d}{2}$ number of matchings to each of the two circuit switches. In Figure 7.6, we show three emulated graph instances which differ in the degree d for $n_t = 16$ ToR switches each with $n_u = 2$ in/out ports. We note that the standard complete graph K_{n_t} has a degree of $n_t - 1$ without self-loops. For simplicity, we consider that the optical circuit switches match every ToR to every ToR including one self-loop i.e., degree is n_t instead of $n_t - 1$. Table 7.1 summarizes our example. We next walkthrough the throughput, delay and buffer requirements of different design choices including MARS.

1 **Static uni-regular DCNs:** With two circuit switches in the topology, each

Topology	Throughput	Delay	Buffer
1	0.125	≈ 0	≈ 0
2	0.5	$1600\mu s$	80MB
3	0.125	$1600\mu s$	20MB
4	0.25	$850\mu s$	20MB

Table 7.1: Tradeoffs across different design choices based on the degree of the emulated graph for the example topology shown in Figure 7.6, 7.5.

deployed with only one matching results in a static topology[7] as shown in Figure 7.6a. While this is an extreme choice given the reconfigurable circuit switches, it is interesting to observe the throughput vs buffer requirements. Specifically, the diameter is 4 resulting in a throughput of $\frac{1}{8}$ (from Theorem 17) for a worst-case permutation demand matrix. However, the delay is nearly zero theoretically since the topology remains static assuming negligible transmission and propagation delays. Since the delay is near-zero, the topology also requires near-zero buffers at each node to achieve the ideal throughput of $\frac{1}{8}$.

2 **Existing designs:** Using 16 matchings in total, with 8 matchings deployed in each of the two circuit switches, the topology emulates a complete graph as shown in Figure 7.6c which is similar to existing designs [60, 189]. Emulating a complete graph results in optimal throughput of $\frac{1}{2}$ but at the expense of high delay as much as $16 \cdot \Delta = 1600\mu s$ (from Theorem 15). In order to achieve the optimal throughput of $\frac{1}{2}$, each ToR switch would require significantly large buffers as much as 80MB in our small scale example topology.

3 **Existing designs under resource constraints:** While emulating a complete graph as shown in Figure 7.6c may result in optimal throughput, the achievable throughput critically depends on the available buffer at each ToR switch. For instance, if each ToR switch is equipped with only 20MB buffer, then the throughput drops to $\frac{1}{8}$ (from Theorem 16). Notice that even with 10MB buffer, a simple static topology achieves similar throughput with significantly lower delay (see above).

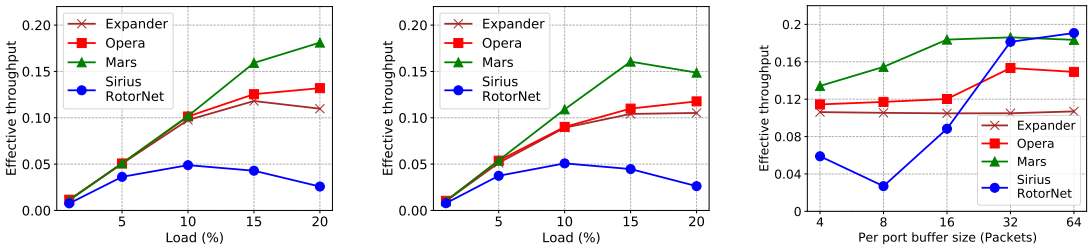
4 **Mars:** Our approach leverages the insights from §7.3. Based on the available buffer (say $B = 20$MB) and the delay requirements (say $L = 850\mu s$), we systematically determine the degree d of the emulated graph to be emulated by the reconfigurable network. In this case, the optimal degree turns out to be $d = 4$ both in terms of delay L (from Theorem 18) and available buffer B (from Theorem 19). The topology has a diameter of $\log_4(16) = 2$ and achieves much higher throughput: $\frac{1}{4}$ in our example.

7.5 Evaluation

We evaluate Mars and compare against the existing approaches in the design space. Our evaluation aims at answering four main questions.

***(Q1)* Can Mars improve the throughput in datacenters?** Our evaluation shows that, Mars improves the throughput of existing datacenter designs by up

[7] In this case, since there are no reconfigurations, we set the timeslot value as $\Delta \approx 0$, the latency tax due to reconfigurations $\Delta_u = 0$ and the period $\Gamma = 1$.



(a) Effective throughput under all-to-all demand matrix and websearch workload with 8 packets per port buffer size.

(b) Effective throughput under random permutation demand matrix and websearch workload with 8 packets per port buffer size.

(c) Effective throughput across buffer sizes under random permutation demand matrix and websearch workload.

Figure 7.7: MARS achieves the best throughput under shallow buffers. Existing approaches which emulate a complete graph (Sirius/RotorNet) suffer significantly under limited buffer (a,b) but can gain throughput if buffers are large (c).

to 64% compared to an expander DCN, by up to 37% compared to Opera [187] and by more that 4x compared to Sirius [60] and RotorNet [189] under Valiant load balancing.

***(Q2)* How does Mars perform under shallow buffers?** Even under extremely shallow buffers, MARS significantly outperforms existing approaches by improving the throughput by up to 6x at moderate load conditions. At low loads, MARS performs similar to existing approaches.

***(Q3)* Can Mars improve the FCTs of short flows?**
We find that MARS does not trade latency for throughput. Indeed MARS's low buffer requirements to achieve high throughput also contribute to better latency even under permutation demand matrices. Our evaluation shows that MARS can improve the FCTs of short flows by up to 96% compared to expander DCN, by up to 75% compared to Opera and by up to 87% compared to RotorNet and Sirius under Valiant load balancing.

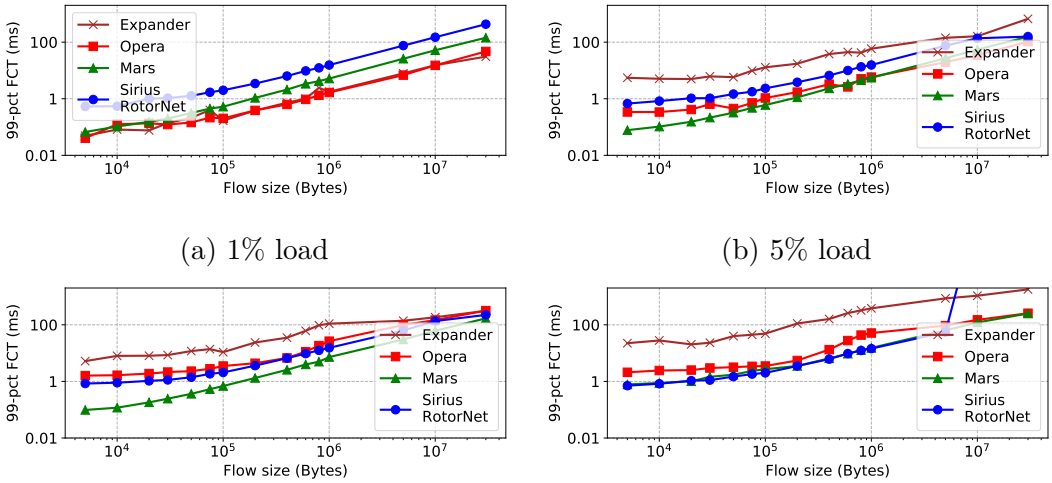
***(Q4)* Do long flows benefit from Mars?** Our results show that MARS reduces the FCTs of long flows for various workloads by up to 4% on average compared to existing approaches under various loads and demand matrices.

7.5.1 Setup

Our evaluation is based on packet-level simulations using HTSIM used in prior work [124, 187].

Interconnect: We consider a datacenter network with 256 servers organized into 64 ToR switches. Each ToR switch has 4 uplinks that connect to 4 rotor-switches [189] with a reconfiguration delay of $1\mu s$. All the links have a capacity of 10Gbps and 500 nanoseconds delay.

Workload: We generate traffic using websearch [35] workload. We evaluate across various loads on the server-ToR links in the range $1-20\%$ for all-to-all and random permutation demand matrices. Note that 20% load is already close to the



(a) 1% load

(b) 5% load

(c) 10% load

(d) 20% load

Figure 7.8: For the websearch workload under a random permutation demand matrix, MARS improves the 99-percentile flow completion times for short flows without sacrificing the flow completion times of long flows. Note the log scale on both axes.

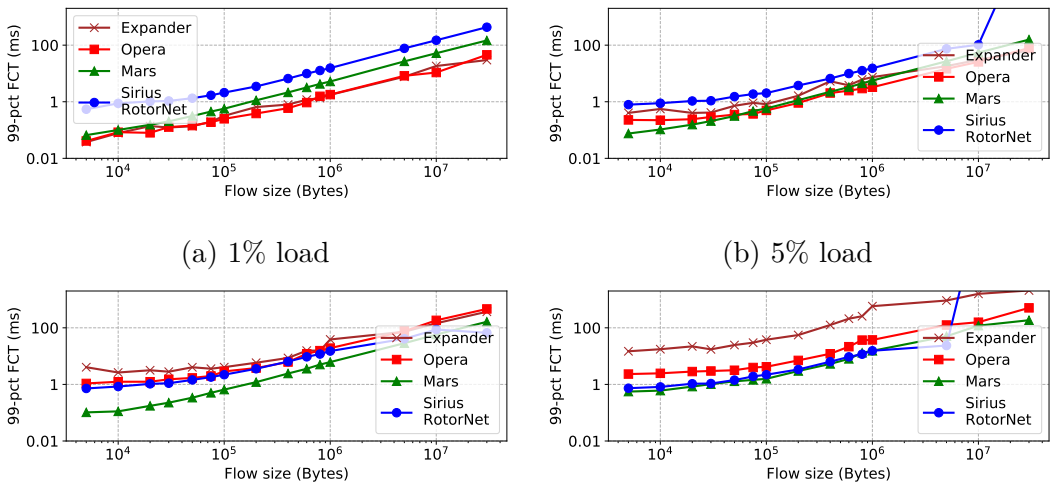
maximum load that an expander topology can sustain[8]. Flows arrive according to a Poisson process and we control the mean inter-arrival time to achieve the desired load.

Comparisons: We compare MARS to Opera [187], Sirius [60], RotorNet [189] and static expander networks. For expander, we generate $d = 4$ (number of uplinks) random regular graphs G with 64 ToRs as vertices such that $\lambda(G) \leq 2\sqrt{u-1}$ where λ is the second eigen value and $2\sqrt{u-1}$ is the Ramanujan constant. This gives us a Ramanujan graph which is known to be excellent expander.

Metrics: We report server downlink utilization indicating the effective throughput. We also report 99-percentile flow completion times (FCTs) and buffer occupancies.

Configuration: We construct MARS which emulates a deBruijn directed graph of degree 8 (see §7.4.3) optimized for a shallow buffer size of 8 packets per port in the interconnect described above. Switches are configured with routing information statically at initialization time and packets are sprayed across all equal cost paths. MARS, Sirius [60] and RotorNet [189] use Valiant load balanced paths; expander uses all (edge-disjoint) shortest paths; and Opera uses all (edge-disjoint) shortest paths in each topology slice [187] for short flows and single hop paths for long flows. The long flow cutoff is set to 15MB based on [187]. We use NDP [124] as the transport protocol and set the trimming threshold per port to 8 packets by default based on [124] unless explicitly stated. We vary the packet trimming threshold to vary the maximum buffer size values in our evaluation. Finally, we set minRTO to 1ms.

[8]Atleast $\frac{1}{\text{ARL}}$ capacity is sacrificed due to multi-hop routing; where ARL is the average route length. An alternative in order to further increase throughput is to simply"undersubscribe" i.e., with oversubscription < 1. We leave it for future work to analyze such designs which requires a comprehensive cost-analysis for a fair comparison.



(a) 1% load

(b) 5% load

(c) 10% load

(d) 20% load

Figure 7.9: For the websearch workload under all-to-all demand matrix, as the load increases, Mars outperforms in flow completion times for both short and long flows. Note the log scale on both the axes.

7.5.2 Results

Mars significantly improves the throughput: In Figure 7.7a and Figure 7.7b, we show the effective throughput achieved by Mars and existing approaches across various loads of websearch workload. Specifically, in Figure 7.7a for the All-to-All demand matrix, we see that Mars improves the effective throughput at 20% load by 1.37x compared to Opera, by 1.64x compared to expander and by 7.02x compared to Sirius and RotorNet under Valiant load balancing. In Figure 7.7b, for random permutation demand matrix, we see that Mars improves the effective throughput by up to 1.33x on average compared to Opera and expander; and by 5.66x compared to Sirius and RotorNet. At low loads, Mars achieves similar throughput compared to existing approaches.

Mars outperforms under shallow buffers: Given the high throughput of Mars, we evaluate its performance under various buffer sizes at the ToR switches and compare against existing approaches. In Figure 7.7c, we see that even at extremely shallow buffers such as 4 packets per port, Mars achieves 1.57x better throughput on average compared to Opera, expander, Sirius and RotorNet. However, as we show in Figure 7.1, existing systems require significantly more buffer to achieve high throughput. As expected, in Figure 7.7c, we see that Opera, Sirius and RotorNet increase in throughput with large buffers. Indeed, Sirius and RotorNet can sustain even higher loads up to 50% since they emulate a complete graph with shorter path lengths but this requires extremely large buffers. We omit these results for brevity, given that Sirius and RotorNet require buffer sizes as large as 64 packets per port to achieve 20% throughput in Figure 7.7c.

Mars does not require large buffers: In Figure 7.10, we show the CDF of buffer occupancies at the ToR switches for 20% websearch load and 64 packet per port buffers. We see that Mars and Opera require significantly lower buffer compared to expander, Sirius and RotorNet (under Valiant load balancing). Opera selectively buffers packets at the end-hosts based on their flow size and only routes short flows across multi-hops which contributes its very low buffer requirements as

seen in Figure 7.10. However, Opera achieves 1.22x *lower* throughput compared to MARS as seen in Figure 7.7c. We also observe that Sirius and RotorNet emulating a complete graph, consume significantly more buffer compared to MARS: 2.09x higher at the tail.

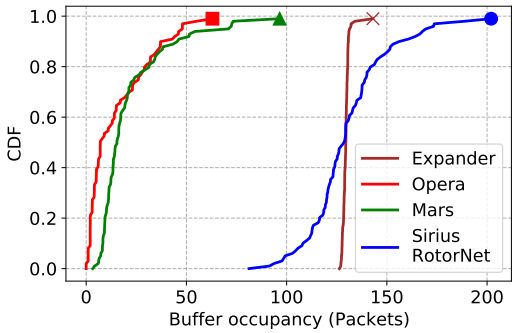


Figure 7.10: MARS achieves high throughput but requires significantly less buffer compared to Expander, Sirius and RotorNet; and similar buffers as Opera.

Mars significantly improves FCTs of short flows: The high throughput as well as low buffer requirements of MARS significantly improve the FCTs of short flows. Figure 7.8 shows the 99-percentile FCTs across various loads for random permutation demand matrix under websearch workload. At 1% load (Figure 7.8a), we see that MARS achieves similar FCTs for short flows compared to Opera and expander networks and improves upon Sirius and RotorNet by 87.37%. As the load increases, MARS outperforms existing approaches. Specifically, at 5% load (Figure 7.8b), MARS reduces the FCTs for short flows by 77% compared to Opera, by 98.11% compared to expander and by 88.43% compared to Sirius and RotorNet. Further, at 10% load (Figure 7.8c), MARS improves the FCTs for short flows by 93.88% compared to Opera, by 98.11% compared to expander and by 83.47% compared to Sirius and RotorNet.

We observe similar improvements under an all-to-all demand matrix as shown in Figure 7.9. At 5% (10%) load, MARS reduces the FCTs for short flows by 66.96% (90.15%) compared to Opera, by 81.35% (97.46%) compared to expander and by 87.6% (83.90%) compared to Sirius and RotorNet.

It is interesting to observe that the 99%-percentile flow completion times for short flows at 1% load follows the delay trend shown in Figure 7.1: a static expander has the least path delay and similarly Opera since it routes all the short flows over an expander; Sirius and RotorNet which emulate a complete graph experience the highest delays. However, as the load increases, multi-hop routing and queueing delays also impact the FCTs of short flows.

Mars does not trade long flow FCTs for short flows: MARS not only improves the FCTs for short flows but also achieves on-par 99-percentile FCTs for long flows compared to existing approaches. Under random permutation demand matrix (Figure 7.8), at 20% load, MARS reduces the 99-percentile FCTs for long flows by 2.37% compared to Opera, by 85.98% compared to expander and by 99.9% compared to Sirius and RotorNet. Further, under an all-to-all demand matrix (Figure 7.9), at 20% load, MARS reduces the FCTs for long flows by 63.42% compared to Opera, by 81.96% compared to expander and by 99.2% compared to Sirius and RotorNet. Interestingly, at 20% load, under random permutation

(Figure 7.8d) and all-to-all (Figure 7.9d) demand matrices, Sirius and RotorNet cannot sustain long flow FCTs given the shallow buffers used in our setup. This result further strengthens our motivation on the performance of existing approaches under resource constraints.

Overall, our evaluation confirms that MARS outperforms existing approaches by improving throughput and reducing the flow completion times for short flows as well as long flows.

7.6 Discussion

While our model and choice of topologies are in line with the assumptions and conclusions in the literature, they have certain practical implications which we discuss below.

Demand Matrix: Our model assumes that a demand matrix is fixed and does not change over time. In contrast, real-world demand matrices evolve. This would indeed complicate our analysis framework by introducing time variables to the demand. However, our analysis still provides insights within the duration in which the demand matrix remains constant. As demand matrices do not change rapidly over time [222], one could use our analysis to find the throughput between two time instances when the demand matrix changes significantly.

Congestion control and load balancing: Both congestion control and load balancing significantly impact the achievable throughput. Our main results in this work rely on the theoretical definition of flow and throughput maximization problem. As a result, we make a simplifying assumption on the underlying system: congestion control, load balancing and routing have a central view of the entire network and perform ideally. Our theoretical results can be useful in understanding the ideal scenario and provides insights into the performance gap of a deployed system.

Worst-case analysis: Literature defines throughput of topologies based on worst-case demand matrices and corresponding maximum flow. On one hand, worst-case demand matrices help in understanding the performance bound of a topology under *any* demand matrix (within the scope of our definition). In other words, a topology optimized for the worst-case demand matrix achieves strictly greater throughput under any other demand matrix. On the other hand, the theoretical definition of ‘‘flow’’ relates to fluid transmissions with ideal congestion control, load balancing and routing (see above). In essence, our analysis captures the performance bounds of an ideal transport under worst-case demand. Our key insights from theory (summarized in §7.3.5) drove the design of MARS which is optimized for the worst-case. However, our evaluation of MARS incorporates stochastic flow arrival process (not the worst-case) based on real-world flow size distributions [35, 37] at a given load. The significantly better performance of MARS (optimized for the worst-case) in our evaluation setup (not the worst-case) shows that our analysis indeed finds itself useful even under realistic settings. While we consider static (fluid) flow definitions in this work, variable flow on an edge and flows over time have been studied in the past [48, 100, 101, 120, 250]. The analytical model for flows over time is motivated by the propagation delay over edges and appears to have

an interesting connection to periodic networks, where the delay is introduced along paths due to changes in the topology. Extending our analysis to variable flows and studying their impact on throughput would be a valuable direction for future research.

Cost-equivalent Clos topologies: We emphasize that our results in Theorem 13, 14, 15, 16 and our evaluation concerns uni-regular topologies i.e., every switch is connected to servers and generates traffic. In contrast, traditional datacenters built on Clos topologies add additional layers of switches (additional capacity) that do not generate traffic in order to maximize throughput. This work *does not* argue that uni-regular topologies are better than Clos topologies. However, most recent works [60, 109, 150, 187, 189] demonstrate the significant benefits of reconfigurable (uni-regular) topologies over Clos. Furthermore, comparing any other topologies outside the spectrum shown in Figure 7.1 requires a cost analysis which is volatile in nature abd . We hence focus strictly on the spectrum of topologies with periodic reconfigurations and their tradeoffs. It would be interesting to determine the buffer size at which the performance of a throughput-optimal periodic reconfigurable topology drops below a cost-equivalent Clos topology.

We leave it for future work to theoretically study the throughput of periodic reconfigurable topologies under dynamic demand matrices and more practical design considerations.

7.7 Related Work

Indeed, only little is known today about how existing reconfigurable topologies fare against each other, and whether alternative designs could improve the throughput further. In the following, we discuss the related work in network throughput both in the context of traditional static datacenter topologies [31, 65, 113, 246, 248, 256, 281] and emerging reconfigurable datacenter topologies [60, 69, 79, 80, 91, 98, 109, 121, 123, 148, 182, 187, 189, 234, 235, 249, 261, 279, 286].

Throughput as a metric: Bisection bandwidth has been used extensively as a metric in the networking community. A classic result in graph theory suggests that the max-flow can be an $\mathcal{O}(\log(n))$ factor lower than the sparsest cut [168, 169]. The *throughput* maximization problem has been studied in the context of max-flow multicommodity flow and maximum concurrent flow problems [52, 106, 238]. A fully polynomial-time approximation scheme exists for arbitrary demands and uniform capacity [238]. Recently, Jyothi et al. [145] revisited the relation between cut-based metrics and max-flow in our networking context and proposed throughput under a worst-case demand matrix as metric. While it still remains an active area of research to efficiently compute throughput of a static topology [145, 204, 247], initial studies on dynamic and demand-aware topologies attempt to characterize throughput in terms of demand-completion times [114]. We are not aware of a formal definition and analysis of throughput in the context of periodic reconfigurable topologies.

Static topologies: Clos-based topologies [31, 113] have been shown to provide optimal throughput [204]. Bcube [115] proposes a server-centric architecture and provides high capacity for all-to-all traffic patterns but may not be optimal in terms of the throughput metric. JellyFish [248] argues that random graphs are

highly flexible for datacenters in-terms of heterogeneous expansion and fault-tolerance while achieving high throughput. SlimFly [65] optimizes for diameter of the topology (consequently throughput) but imposes strict conditions on the size of switches. Xpander [256] focuses on incremental deployability while achieving high throughput. F10 [179] on fault-tolerance, and FatClique [281] on the cost, incremental expansion and management in a datacenter.

Reconfigurable topologies: In contrast to static topologies based on costly, power-intensive electrical packet switches, reconfigurable topologies rely on cost-effective technologies such as optical circuit switches and tunable lasers. Reconfigurable topologies can be broadly classified into two types *(i)* demand-aware and *(ii)* demand-oblivious. Demand-aware topologies such as Duo [279], ReNets [55] and others [109, 163, 234, 261] adjust the topology based on the traffic patterns. However, such networks incur high ''latency tax'' due to the added complexity of measuring and calculating the demand via control-plane. Demand-oblivious topologies such as RotorNet [60, 187, 189] rely on a pre-defined schedule for circuit setup and have been shown to provide high-throughput with low ''latency tax'' due to reconfigurations. More recently in a parallel work, the relation between throughput and delay of periodic reconfigurable networks has been studied in detail [42]. Our main focus in this work has been on demand-oblivious designs under resource constraints. In particular, we reveal the fundamental tradeoffs across throughput, delay *and* buffer requirements for such designs. We further propose Mars, a throughput-optimal periodic (demand-oblivious) reconfigurable topology.

Reducing the buffer requirements: A vast literature in buffer management [16, 46, 86] and active queue management [102, 139, 215, 267], scheduling [37, 130, 242] and end-host congestion control [21, 35, 124, 173, 193, 199] focuses on reducing the queueing at a bottleneck link specifically in static datacenter topologies. However, as we rigorously discussed in this work (§7.3, §7.4), periodic reconfigurable topologies fundamentally require certain amount of buffering and incur significant throughput loss under limited amount of buffer. Our approach exploits this tradeoff to systematically find the throughput-optimal design for any given delay and buffer constraints.

7.8 Summary

This work was motivated by the observation that while dynamically reconfigurable datacenter networks can greatly improve throughput, existing RCDN designs which emulate complete graphs can entail high delays and buffer requirements. Based on our analysis of the underlying performance tradeoffs, we presented a more scalable network design, Mars, which achieves near-optimal throughput by emulating a d-regular graph and ensuring shallow buffers.

We understand our work as a first step and believe that it opens several interesting avenues for future research. In particular, naturally, the optimal RDCN topology will also depend on the price, and it will be interesting to conduct an economic study of the viability of different architectures. Furthermore, we have so far focused on flat networks; while such networks are common in the literature and have several advantages, it will be interesting to extend our study of buffer-aware

RDCN designs to multi-tier networks as well.

8

Demand-Aware Optical Interconnects

Datacenters have experienced explosive growth in overall network traffic volume over the past decade [246]. With the recent introduction of high-bandwidth Machine Learning workloads into datacenters, the peak network traffic is expected to increase even more rapidly [222]. Unfortunately, traditional networks, which are built using electrical packet switches, struggle to keep up with this growing demand [60]. Further, the rapid evolution of datacenter applications and their changing bandwidth requirements implies: "the best laid plans quickly become outdated and inefficient, making incremental and adaptive evolution a necessity" [222]. This led to the emergence of novel technologies based on reconfigurable optical circuit switches [60, 98, 109, 189, 222]. Two prominent types of reconfigurable datacenter networks have emerged recently: demand-oblivious [18, 40, 60, 189] and demand-aware [98, 109, 279] networks. These networks are optically circuit-switched and feature bufferless switches. The circuits can be reconfigured, enabling the network topology to adapt dynamically to evolving communication patterns in datacenter workloads, which can potentially improve performance. The reconfiguration schedule varies between designs: some achieve high performance but are impractical for large-scale deployment, while others offer more practical solutions but at the cost of moderate performance.

From a performance standpoint, demand-oblivious networks, such as RotorNet [189], Sirius [60], and Opera [187], offer low reconfiguration overheads (in the range of nanoseconds) but sacrifice throughput due to their fixed and periodic switching schedules, which are independent of the underlying communication patterns. Only recently have the throughput bounds of demand-oblivious networks been established [18, 42], showing that they are tightly bounded by $\frac{1}{2}$ i.e., a sustainable load of at most 50% under worst-case traffic patterns (e.g., ring communication), even with ideal routing and congestion control. In contrast, demand-aware networks such as Mordia [221], Helios [98], and ProjecToR [109] are capable of achieving higher throughput because their switching schedules are optimized for the underlying communication patterns. Unfortunately, formal bounds on the achievable throughput of demand-aware networks remains an open question in the literature.

From a practicality standpoint, periodic fixed-duration reconfigurations have emerged as a promising design choice for reconfigurable datacenter networks [186, 188]. However, existing periodic networks are demand-oblivious and require non-trivial additional support from hardware (NIC and switch) in terms of routing, packet reordering, congestion control and buffer architecture [40, 60, 189]. This complexity hinders the large-scale deployment of these networks in practice. In

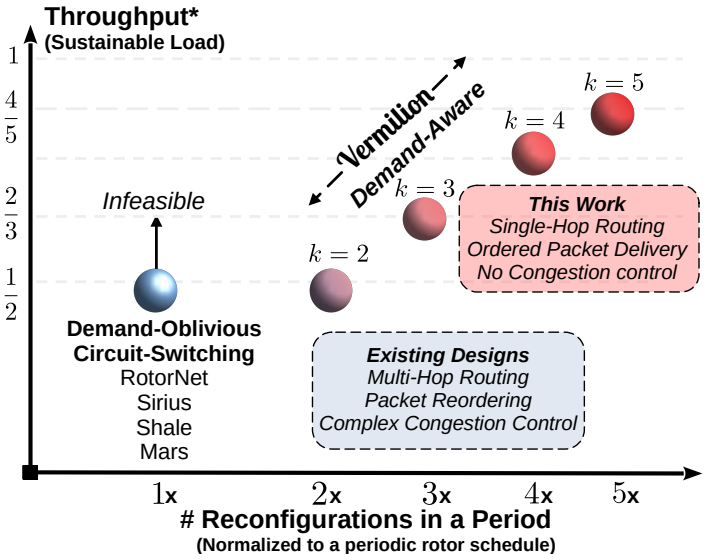


Figure 8.1: Existing designs based on periodic optical circuit-switching are oblivious to traffic patterns, requiring complex multi-hop routing and congestion control, which reduces throughput. *Vermilion* overcomes this limitation by introducing a few additional fixed-duration reconfigurations per period in a demand-aware manner, while significantly simplifying both routing and congestion control.

contrast, demand-aware networks typically only require single-hop (direct) routing and do not depend on in-network congestion control mechanisms. However, this comes at the cost of complex reconfiguration schedules with variable durations, and they often rely on an additional packet-switched network [98, 109, 221].

In view of both performance and practicality, we explore a new direction in this chapter: *Can a high-throughput network be designed using periodic fixed-duration reconfigurations, without relying on multi-hop routing?*

Figure 8.1 illustrates our perspective. Not only are existing designs limited in throughput, the worst-case throughput of *any* demand-oblivious network is bounded by $\frac{1}{2}$ [18, 42, 114], making it infeasible to achieve higher throughput. However, it remains unexplored so far, whether and to what extent a demand-aware approach to periodic reconfigurable networks can improve throughput. Intuitively, if the reconfiguration delay is negligible, a demand-aware network can ideally achieve full-throughput. For instance, most prior works follow this intuition and use Birkhoff–von Neumann (BvN) [66] decomposition technique to devise a circuit switching schedule that perfectly matches the underlying traffic pattern [176, 221]. However, this not only results in a schedule with variable duration for each reconfiguration, it can also result in significantly low throughput due to reconfiguration overheads. Designing a demand-aware network using only fixed-duration periodic reconfigurations, while surpassing the throughput limits of oblivious networks, requires new techniques and remains a challenging open problem.

We present *Vermilion*, a first demand-aware reconfigurable network design that breaks the throughput barrier of existing designs, using only fixed-duration periodic reconfigurations. *Vermilion* not only achieves higher throughput but also greatly simplifies the protocol stack by eliminating multi-hop routing, congestion control and packet reordering. A recent work reports the following regarding single-hop routing in periodic circuit-switched network:

"Notably, none of this requires any modifications to the Linux application, TCP, or the Linux networking stack." [188]

Vermilion relies solely on direct communication. As a result, *Vermilion* is more practical for deployment, within the available hardware capabilities e.g., using Rotor switches [188] and server-grade NICs. We discuss *Vermilion*'s practicality as well as future research directions in more detail later in this chapter.

The key innovation behind *Vermilion* is the use of a matrix rounding technique [57] to derive a switching schedule that matches the underlying traffic pattern. This is in contrast to BvN decomposition technique [66] and greedy approximations [68] that have been largely used in the literature for designing demand-aware networks [176, 221]. Importantly, our rounding technique allows decomposing a demand matrix (after rounding) into a set of permutation matrices that directly serve as the switching schedule with *fixed-duration* for each circuit, in a periodic manner. Interestingly, the schedules produced by rounding technique are capable of serving a large portion of the demand but may leave certain residual demand. To this end, we introduce one extra cycle that is similar to that of existing designs, providing direct connections between all communicating nodes. Our rounding technique is simple and efficient to compute, making it practical for quick updates based on the changes in the traffic patterns. We establish the throughput bounds of *Vermilion*, marking the first theoretical result on the achievable throughput of demand-aware networks, while accounting for reconfiguration delays.

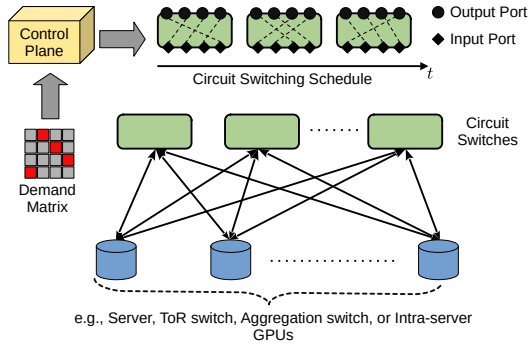
We evaluate using packet-level simulations and show that *Vermilion* significantly improves throughput by up to 2.04x and reduces the flow completion times by up to 99.88% compared to existing approaches.

Our main contributions in this work are:

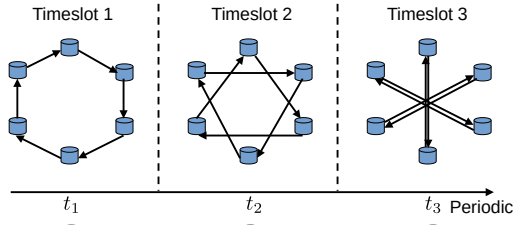
- A first separation result proving that demand-aware reconfigurable datacenter networks are strictly superior to demand-oblivious networks in terms of throughput.
- *Vermilion*, an innovative, yet simple, demand-aware network design based on periodic fixed-duration reconfigurations. *Vermilion* achieves a throughput of at least $\frac{2}{3}$ (lower bound): a significant improvement over existing designs.
- A simplification of the requirements from network protocol stack. *Vermilion* relies only on direct communication without multi-hop routing and does not need additional congestion control mechanisms to be deployed.
- Evaluations, highlighting the performance benefits of *Vermilion* compared to demand-oblivious counterparts. Our results show that *Vermilion* significantly improves flow completion times for both short flows and long flows.

8.1 Motivation

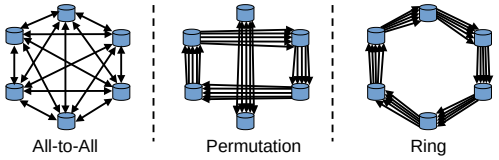
In this section, we motivate our work by outlining the limitations of current periodic reconfigurable networks and the opportunities presented by adopting a demand-aware approach within these systems. Our primary focus in the rest of this chapter is on *periodic* reconfigurable networks, given the recent technological



(a) The physical topology of a reconfigurable network consists of a set of nodes connected by optical circuit-switches arranged in a hierarchical Clos topology (e.g., leaf-spine). The circuit-switches are time-synchronized and rapidly reconfigure their circuits providing direct links between pairs of nodes in a periodic manner. We assume a control plane that defines the switching schedule for each switch.



(b) The physical topology establishes a node-to-node network that evolves over time periodically, allocating bandwidth between pairs of nodes in each timeslot.



(c) Examples of topologies that can be emulated over a *period* using periodic circuit-switching

Figure 8.2: The periodic circuit-switched network (8.2a) rapidly reconfigures its circuits, forming a dynamic node-to-node topology that evolves over time (8.2b). Over time, this topology emulates a specific static network (8.2c), for example, splitting bandwidth evenly across all pairs to create an all-to-all mesh topology (as in RotorNet [189] and Sirius [60]) or concentrating bandwidth between pairs with high demand to form a permutation or ring topology. The throughput of the network is heavily influenced by the choice of switching schedule, the emulated topology, and the underlying demand matrix: *an optimization opportunity.*

advancements showing their practicality and scalability [60, 188]. In the following, we briefly describe the network architecture that we consider in this work.

Network model: Figure 8.2a illustrates the physical topology. A set of nodes are interconnected by optical circuit-switches such that at any time instance the network provides pair-wise direct connections across the nodes. For generality, we say ''nodes'' to refer to either servers, or ToR switches, or Aggregation switches, or intra-server components such as GPUs, that may be interconnected by an optical circuit-switched network. The optical circuit-switches reconfigure according to a schedule in a synchronized and periodic manner. Specifically, the circuit-switches reconfigure at *fixed-duration* intervals (timeslots) and each reconfiguration takes a specific amount of time (reconfiguration delay). This aligns with existing periodic circuit-switching technologies such as RotorNet [186, 188, 189] (Sirius [60]) with a reconfiguration delay of $7\mu s$ ($3.84ns$). As a result, the network topology evolves over time as shown in Figure 8.2b. In each timeslot, the degree of the topology is limited to the number of physical links. However, as the topology evolves over time, it can emulate a variety of topologies with high-degree, as shown in Figure 8.2c. For example, the topology can emulate an all-to-all mesh, a ring, or a permutation topology, depending on the periodic circuit-switching schedule. So far, in the literature, periodic reconfigurable networks were only studied in a demand-oblivious setup i.e., the switching schedule is independent of the underlying traffic patterns, emulating an all-to-all mesh topology [60, 187, 189] or a d-regular topology [18, 40, 42] over time.

We first discuss our formal approach to optimizing the throughput of reconfigurable networks (§8.1.1), followed by the drawbacks of existing demand-oblivious designs (§8.1.2), and later, we make a case for demand-aware networks (§8.1.3). We conclude by summarizing the design challenges (§8.1.5).

8.1.1 Throughput of Periodic Networks

Throughput offered by an interconnect is a crucial metric for assessing the sustainable load a network can handle, especially under highly concurrent communication patterns. Chapter 7 provides a detailed formal model on the throughput of periodic networks. In this section, we refresh the definitions introduced in Chapter 7 as they are crucial to our design and analysis of *Vermilion*.

In order to quantify the throughput, we first define the communication pattern i.e., the demand matrix (Definition 20). The demand matrix specifies the demand in bits per second between each pair of nodes i.e., the total demand originating from a source towards a destination. Following prior work [204], we consider the hose model [94] such that the total demand originating from (and destined to) each node is less than its corresponding capacity limits.

Definition 20 (Demand matrix)**.** *Given a set of nodes $\mathcal{N}$, each with d outgoing and incoming links of capacity c, a demand matrix specifies the demand rate between every pair of nodes in bits per second defined as $\mathcal{M} = \{m_{u,v} \mid u \in \mathcal{N}, v \in \mathcal{N}\}$ where $m_{u,v}$ is the demand between the pair u, v. The demand matrix is such that the total demand originating at a source s is less than its outgoing capacity and the total demand terminating at a destination t is less than its incoming capacity i.e., $\sum_{u \in \mathcal{N}} m_{s,u} \leq c \cdot d$ and $\sum_{u \in \mathcal{N}} m_{u,t} \leq c \cdot d$.*

For a given communication pattern and the corresponding demand matrix

(Definition 20), we define throughput as the maximum scaling factor such that there exists a feasible flow that can satisfy the scaled demand subject to flow conservation and capacity constraints. We denote flow by $F : P \mapsto \mathbb{R}^+$, a map from the set of all paths P (static or temporal) to the set of non-negative real numbers. This mapping naturally ensures that the flow transmitted from a source eventually reaches the destination along a path $p \in P$. To obey capacity constraints, a feasible mapping is such that the sum of all flows traversing a link do not exceed the link capacity. We are now ready to define throughput formally.

Definition 21 (Throughput)**.** *Given a demand matrix $\mathcal{M}$ and a reconfigurable network, throughput denoted by $\theta(\mathcal{M})$ is the highest scaling factor such that there exists a feasible flow for the scaled demand matrix $\theta(\mathcal{M}) \cdot \mathcal{M}$. Throughput θ^* is the highest scaling factor for a worst-case demand matrix i.e., $\theta^* = \min_{\mathcal{M} \in \hat{M}} \theta(\mathcal{M})$, where $\hat{M}$ is the set of all demand matrices.*

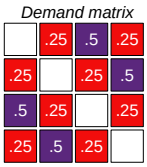
Intuitively, throughput for a specific communication pattern captures the maximum sustainable load by the underlying topology. Based on Definition 21, similar to prior works [18, 114, 145, 204], throughput of a topology is the minimum throughput across the set of all saturated demand matrices i.e., if a topology has throughput θ^*, then it can achieve at least throughput θ^* for *any* demand matrix and at most throughput θ^* for a worst-case demand matrix.

Several variants of the throughput problem have been studied over the last decades, especially in the context of the maximum concurrent flow problem [238]. However, in contrast to static networks, the fundamental challenge to study throughput in the context of reconfigurable networks is that the topology changes over time and can even be a function of the demand matrix in the case of demand-aware networks.

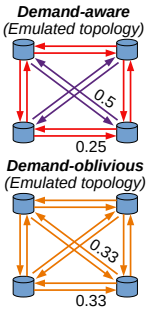
In the specific case of periodic networks, our recent work (Chapter 7) establishes an equivalence between the throughput of any periodic network and a corresponding emulated topology (as static graph), allowing the study of throughput in periodic networks using existing techniques [18]. We refer the reader to Chapter 7 for a formal definition of the emulated graph. Essentially, the emulated graph is a time-collapsed view over an entire period of the periodic network as shown in Figure 8.2c.

For example, in RotorNet, the length of the period is $\Gamma = \frac{n}{d}$ timeslots. If a link appears once between every pair of nodes over a period, then the emulated graph has one link between every pair of nodes (complete graph), where each link has a capacity of $c \cdot \frac{1}{\Gamma} = c \cdot \frac{d}{n}$. As we will discuss later, emulating a complete graph results in a drop in throughput by a factor 2 under any permutation (e.g., ring) communication patterns. Intuitively, in order to maximize throughput, the emulated graph must provide high bandwidth between specific pairs of nodes with high-demand i.e., by emulating a topology that closely matches the underlying communication pattern.

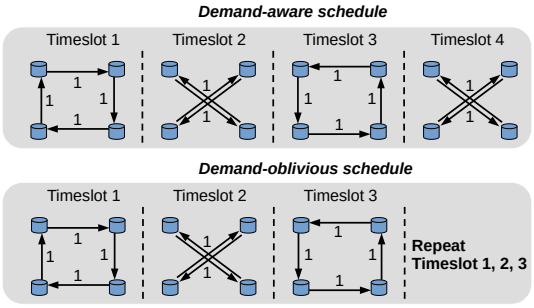
■ ***Takeaway.*** *Optimizing periodic networks entails finding the best static graph that can be emulated over a period, that provides the highest throughput for an (any) underlying communication pattern.*



(a) Example demand matrix for a topology with 4 nodes, each with 1 physical link.



(b) Target emulated topology for demand-aware vs oblivious periodic networks.



(c) Derived periodic schedule for a demand-aware vs oblivious periodic network.

Figure 8.3: Interpreting the demand matrix (Figure 8.3a) as a target emulated graph (Figure 8.3b) allows deriving a periodic schedule (Figure 8.3c) that can achieve full throughput for demand-aware periodic networks, even with single-hop routing. In contrast, the schedule for an oblivious periodic network often does not match the underlying demand matrix, necessitating multi-hop routing and resulting in reduced throughput.

8.1.2 Drawbacks of Oblivious Periodic Networks

Oblivious periodic circuit-switching has an obvious advantage of simplicity without any control plane involvement for optimizing the topology. However, this leads to certain drawbacks that we elaborate in the following.

Factor of 2 reduction in throughput: Recent works have established a tight bound of $\frac{1}{2}$ on the throughput of periodic networks [18, 42, 114]. We emphasize that this reduction in throughput is not solely attributed to the use of valiant load-balancing; rather, the network fundamentally cannot sustain beyond 50% load, even with an ideal routing scheme, under a worst-case communication pattern such as a ring. Specifically, in ring communication, only specific node pairs exchange data, but emulating a complete graph only offers a capacity of $c \cdot \frac{d}{n}$ (as discussed above), falling short of the ideal bandwidth provisioning of $c \cdot d$ (leveraging all available links) for these pairs. Even an optimal routing scheme is thus forced to route traffic over 2-hop paths, in order to fully utilize the network capacity, leading to a throughput of $\frac{1}{2}$.

Multi-hop routing: The use of indirect paths is a necessity to improve utilization in oblivious periodic networks. However, since the network is evolving over time, this implies that intermediate nodes need to buffer (hold) traffic until the next link along the path becomes available. This results in additional latency and buffer requirements. In addition, the use of multi-hop routing requires the use of in-network congestion control mechanisms to prevent packet loss due to buffer overflows [40]. This further complicates the network stack and can lead to performance degradation under high load. In fact, recent works even suggest sacrificing throughput further in order to satisfy buffer constraints [18].

Packet reordering: Not only do existing oblivious networks require multi-hop routing, but they also require multi-path transmission in order to fully utilize the network. This results in packets arriving out-of-order at the destination, necessitating additional support such as reorder buffers [188]. This can be particularly challenging in the context of RDMA since RoCE implementations typically react

to packet reordering at the receiver with NACK that triggers retransmissions immediately at the sender.

■ ***Takeaway.*** *While oblivious periodic networks simplify switching schedule selection, they rely on complex protocols such as multi-hop routing, multi-path transmission, and reorder buffers, which can significantly reduce the maximum achievable throughput.*

8.1.3 A Case for Demand-Aware Networks

In contrast to oblivious reconfigurable networks, demand-aware networks can potentially achieve higher throughput by optimizing the network topology for the underlying communication pattern. Demand-aware networks have been studied in the literature [98, 109, 221] and have been empirically shown to achieve better performance compared to oblivious networks. In fact, demand-aware networks can satisfy any demand within the hose model, if the reconfiguration delay is hypothetically near-zero or negligible. This result has been intuitively known in the literature [176, 221], which relies on Birkhoff-von-Neumann (BvN) matrix decomposition technique [66]. For completeness, we formally state it here.

Theorem 20 (Ideal throughput of demand-aware network)**.** *The throughput of an ideal demand-aware reconfigurable network is* 1 *i.e., full-throughput for any demand matrix if the reconfiguration delay is negligible.*

Proof. Within the hose model set of demand matrices, we consider saturated demand matrices i.e., the sum of every row (column) equals the outgoing (incoming) capacity of each node. If a topology can achieve throughput θ for all saturated demand matrices, then the topology can achieve throughput θ for any demand matrix [204]. Given that saturated demand matrices are doubly stochastic, we first decompose the matrix using Birkhoff–von Neumann (BvN) decomposition technique [66] into k permutation matrices, where k can be up to n^2. Let $\mathcal{M}$ be any saturated demand matrix, where the sum of every row and column is $c \cdot u$ (total capacity of each node). Let the corresponding BvN decomposition be $\mathcal{M} = \lambda_1 \cdot P_1 + \lambda_2 \cdot P_2 ... + \lambda_k P_k$, where P_i is a permutation matrix and the coefficients λ are such that $\sum_{i=1}^{k} \lambda = c \cdot u$. Using this decomposition, we configure the topology such that each permutation P_i is executed using full node capacity $c \cdot u$ for $\frac{\lambda_i}{c \cdot u} \cdot \Delta$ units of time over a period of one unit of time Δ. Over Δ amount of time, $\lambda_i \cdot P_i$ portion of the demand matrix generates $\lambda_i \cdot P_i \cdot \Delta$ demand in volume. As a result, during $\frac{\lambda_i}{c \cdot u} \cdot \Delta$ amount of time, by executing the corresponding permutation P_i using full capacity $c \cdot u$, the topology can fully satisfy $\lambda_i \cdot P_i$ portion of the demand matrix. As a result, the topology can fully satisfy the demand matrix $\mathcal{M}$ over each period of one unit of time Δ and achieves full throughput. □

The core intuition behind BvN-based demand-aware network design is to allocate bandwidth between node-pairs as a convex combination (over time) of permutations derived from BvN matrix decomposition. However, this approach suffers from two key limitations: *(i)*, the durations between reconfigurations in the switching schedule are variable, and *(ii)*, the time between reconfigurations can be shorter than the reconfiguration delay, leading to a significant drop in throughput. Consequently, BvN-based designs are incompatible with current technologies that rely on fixed-duration periodic reconfigurations.

Interestingly, it is possible to design periodic reconfigurable networks that achieve full throughput for specific types of communication patterns. For instance, consider a ring demand matrix. Even with fixed-duration periodic reconfigurations, a simple switching schedule with a period of one timeslot --- where the direct links between communicating nodes in the ring are maintained --- can achieve full throughput. In contrast, oblivious periodic networks can only achieve a throughput of $\frac{1}{2}$ for the same communication pattern. Similarly, if the demand matrix consists of integer multiples of link capacity, it is straightforward to design a switching schedule that achieves almost full throughput. Simply establishing direct links between communicating nodes is sufficient to achieve full throughput for such demand matrices.

Our key insight is that when the periodic schedule is constrained to Γ timeslots, any demand matrix $\mathcal{M}$ with non-zero demand represented as an integer multiple of $\frac{c}{\Gamma}$ can be satisfied with full throughput. For such communication patterns, a feasible periodic schedule with fixed-duration reconfigurations always exists and achieves full throughput. Intuitively, the demand matrix can be visualized as an edge-weighted graph, where the weights represent the demand between node pairs. This graph corresponds to the target emulated topology for the periodic schedule, with weights now representing link capacities. A periodic schedule can then be derived to allocate bandwidth between source-destination pairs according to the demand matrix, ensuring full throughput.

Theorem 21 (Throughput under integer demand matrices)**.** *There exists a periodic reconfigurable network with a period of Γ timeslots with Δ_r fraction of time spent in reconfiguration, that can achieve nearly full throughput of $1 - \Delta_r$ using only single-hop routing for demand matrices where any non-zero demand is an integer multiple of $c \cdot \frac{1}{\Gamma}$, where c is the link capacity in the physical topology.*

Proof. A demand matrix $\mathcal{M}$ in the hose model has the property that the sum of every row and column is at most $\hat{d}$, where $\hat{d}$ is the degree (physical links). Further, we have that every entry in the matrix is an integer multiple of $c \cdot \frac{1}{\Gamma}$, where c is the link capacity and Γ is the period of the periodic schedule. We assume, without loss of generality, that the capacity of each physical link is 1. The emulated multigraph G has a degree of $\Gamma \cdot \hat{d}$, and each link has capacity $\frac{1}{\Gamma}$. We now upscale both the demand matrix $\mathcal{M}$ and each link capacity of the emulated multigraph by Γ. Thus, it is equivalent to find the throughput of the emulated multigraph G with degree $\Gamma \cdot \hat{d}$ and each link having capacity 1, under the scaled demand matrix $\mathcal{M}' = \Gamma \cdot \mathcal{M}$, where the sum of each row and column is at most $\Gamma \cdot \hat{d}$. Since every entry in the matrix $\mathcal{M}$ is an integer multiple of $\frac{1}{\Gamma}$, the scaled demand matrix $\mathcal{M}'$ is an integer matrix with sum of every row and column at most $\Gamma \cdot \hat{d}$. Further, we have a degree of $\Gamma \cdot \hat{d}$ with link capacity of 1. It is now straight-forward that constructing a graph by adding links between each pair based on the value of the demand in the scaled matrix $\mathcal{M}'$ can fully satisfy the demand, and this requires only single-hop routing. □

Figure 8.3 illustrates a demand matrix for which a demand-aware periodic schedule can be trivially derived. The demand matrix consists of integer multiples of $\frac{1}{4}$, and the corresponding switching schedule, with a length of 4 timeslots, is depicted in Figure 8.3. This schedule enables the network to fully satisfy the demand matrix within each period, achieving full throughput. In contrast, the

schedule of an oblivious network contains only 3 timeslots, omitting the additional 4^{th} timeslot used in the demand-aware schedule. The oblivious schedule distributes uniform bandwidth across all node-pairs but mismatches the underlying demand matrix, as demonstrated in Figure 8.3. Notably, the inclusion of just a few extra timeslots in a demand-aware manner can significantly enhance the network's throughput.

Single-hop routing: Notice that the demand-aware schedule in Figure 8.3 achieves high throughput even with single-hop routing. This is because the emulated topology provides capacity between each node pair that precisely matches the demand matrix specifications. Generally, any schedule derived from Theorem 21 maintains this single-hop routing advantage, simplifying the protocol stack by eliminating the need for multi-hop routing, congestion control, and packet reordering.

However, a caveat is that increasing the period Γ can lead to an excessively long periodic schedule to achieve full throughput for any demand matrix, as indicated by Theorem 21. This can result in unacceptable delays. Theorem 21 suggests potential throughput gains by deriving periodic schedules tailored to the underlying communication patterns. The challenge remains in deriving *short* schedules while still achieving high throughput.

■ ***Takeaway.*** *Specific communication patterns showcase the substantial throughput gains that demand-aware periodic networks can achieve over oblivious designs. The main challenge — and opportunity — lies in deriving compact schedules that can achieve high throughput for any communication pattern.*

8.1.4 Linear Program Formulation

Throughput maximization is a variant of multi-commodity maximum flow problem, commonly referred as maximum concurrent flow problem [238]. In the case of demand-aware periodic networks, our goal is to find the best emulated graph. In the following we present an integer linear program formulation. Given a network of n nodes, each with $\hat{d}$ physical links (incoming and outgoing), the LP takes demand matrix $\mathcal{M}$ and the length of the desired schedule Γ as input. The capacity of the physical links is denoted by c. The LP has to find the number of links $\hat{c}^{i,j}$ between each node pair (i, j). We set the link capacities to 1 and interpret $\hat{c}^{i,j}$ as the capacity between i, j. We use $f_{i,j}^{s,d}$ to refer to the flow on edge (i, j) corresponding to (s, d) demand. Our objective is to maximize throughput θ such that the scaled demand matrix $\theta \cdot \mathcal{M}$ satisfies source-destination demands, flow conservation and capacity constraints. We consider a degree constraint $\hat{d}$ for each node. Consequently, the demand matrices of interest are those with the sum of each row and column limited to $\hat{d}$.

Input:

$$\begin{aligned} \text{Demand matrix} \quad & \mathcal{M} = \{m_{s,d} \mid s \in V,\ d \in V\} \\ \text{In-out degree} \quad & \hat{d} \end{aligned}$$

Objective Function:

$$\text{Maximize} \quad \theta$$

Subject to the constraints:

$$\text{Source demand: } \sum_{i \in V \setminus \{s\}} f_{s,i}^{s,d} \geq \theta \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

$$\text{Destination demand: } \sum_{i \in V \setminus \{d\}} f_{i,d}^{s,d} \geq \theta \cdot m_{s,d} \qquad \forall s \in V,\ \forall d \in V$$

$$\text{Flow conservation: } \sum_{i \in V \setminus \{j\}} f_{i,j}^{s,d} - \sum_{k \in V \setminus \{j\}} f_{j,k}^{s,d} = 0 \qquad \forall j \in V \setminus \{s,d\} \quad \forall s \in V,\ \forall d \in V$$

$$\text{Capacity constraints: } \sum_{s \in V} \sum_{d \in V} f_{i,j}^{s,d} \leq \hat{c}^{i,j} \qquad \forall i \in V,\ \forall j \in V$$

$$\text{In-degree constraints: } \sum_{s \in V} c^{s,d} \leq \hat{d} \qquad \forall d \in V$$

$$\text{Out-degree constraints: } \sum_{s \in V} c^{s,d} \leq \hat{d} \qquad \forall d \in V$$

Variables:

$$\text{Flow: } f_{i,j}^{s,d} \geq 0\ ,\ f_{i,j}^{s,d} \in \mathbb{R} \qquad \forall i \in V, j \in V, s \in V,\ \forall d \in V$$

$$\text{Throughput: } \theta \geq 0\ ,\ \theta \in \mathbb{R}$$

$$\text{Number of links: } \hat{c}^{i,j} \geq 0\ ,\ \hat{c}^{i,j} \in \mathbb{Z}$$

8.1.5 Design Challenges & Roadmap

Our vision to leverage fixed-duration periodic circuit-switching in a demand-aware manner requires tackling several fundamental questions that remain open and unanswered in the literature. We outline the key challenges and our roadmap to address them in the following.

(Q1) *Is it computationally feasible to derive optimal demand-aware periodic schedules for large topologies?*

As mentioned before, the underlying problem is to construct an optimal emulated topology within a degree constraint. § 8.1.4 presents the linear program formulation. However, in our initial experiments, the solver (Gurobi [117]) did not terminate after 30 minutes even for a 16 node topology. Our roadmap to address this challenge is to develop an algorithm that can quickly derive near-optimal schedules for large topologies and demand matrices. ▷ §8.2.1

(Q2) *Given a degree constraint, what is the best topology that maximizes throughput for a given demand matrix?*

Answering this question not only allows us to then derive a periodic schedule that emulates an optimal topology but also provides insights into more constrained networks with slow reconfigurations such as those with patch-panels [266]. While some recent works focus on specific communication patterns in distributed training under a domain-specific cost model [285], the throughput-optimization problem largely remains open for general communication patterns. Our roadmap to solve this problem builds upon our observations in §8.1.3. Specifically, building upon Theorem 21, our main intuition is to serve the bulk portion of the demand matrix in a demand-aware manner using a subset of the available links within the degree constraints and tackle the residual demand with a demand-oblivious topology, while guaranteeing high throughput. We address this question for specific degree constraints relevant for periodic networks but under general demand matrices within the hose model, providing a first step towards a general solution. ▷ §8.2.2

(Q3) *Is it fundamentally feasible to achieve high throughput for any demand matrix using only single-hop routing?*

Under constrained length of the schedule (and degree), answering this question not only requires finding optimal topology under an "ideal routing" obtained from solving the concurrent flow problem, we further need to restrict the paths to direct communication. Our roadmap is to construct *short* periodic schedules that provide bandwidth between communicating pairs such that the gap between bandwidth and demand between any pair is bounded by a certain ratio, ensuring high throughput with single-hop routing. ▷ §8.2.2

8.2 Towards Demand-Aware Periodic Networks

Based on our observations in §8.1, we seek to design a *simple* demand-aware periodic network, within the practical capabilities of existing optical circuit-switching technologies [60, 186, 188] and the end-host networking stack. Our goal is to achieve high throughput for any demand matrix using only single-hop routing i.e., direct communication. We first present our network design (§8.2.1), followed by the throughput guarantees of our design (§8.2.2) and its practicality (§8.2.3).

8.2.1 Vermilion

We present *Vermilion*, a first demand-aware periodic network design that can probably achieve high throughput compared to existing oblivious designs. We walk through each component of *Vermilion* in the following.

Physical topology: Our network model remains the same as described in §8.1, with a set of nodes interconnected by optical circuit-switches in a hierarchical CLOS topology. The circuit-switches are synchronized in time and reconfigure at fixed-duration intervals, forming a dynamic node-to-node topology that evolves over time. Each node in the topology has $\hat{d}$ physical links that connect to the optical interconnect, hence at any time instance, each node can connect to at most $\hat{d}$ other nodes. The physical links have a capacity of c.

Parameters: *Vermilion* has one parameter k, that controls the degree of the target emulated topology. A higher k leads to higher throughput but also increases the schedule length. The resulting schedule can achieve at least $\frac{k-1}{k}$ throughput for a given demand matrix $\mathcal{M}$. For example, even with $k = 3$, *Vermilion* can achieve a

Algorithm 6: *Vermilion*

Input : Demand matrix $\mathcal{M}$, number of nodes n, degree $\hat{d}$, link capacity c, parameter k

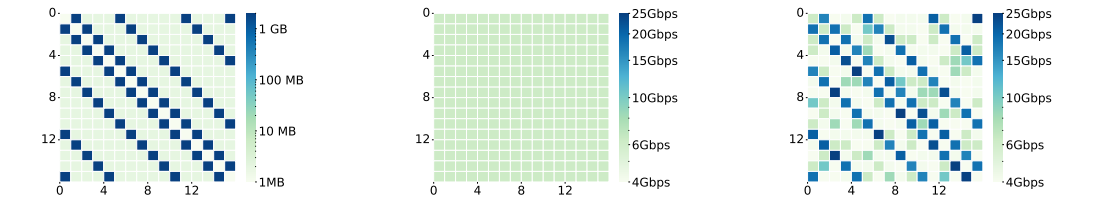
procedure `generateSchedule`$(\mathcal{M})$**:**
  G = emulatedTopology($\mathcal{M}$)
  ▷ *Sequence of matchings for the periodic schedule*
  for $i = 1$ **to** $k \cdot n$ **do**
    remove one perfect matching M_i in G
    add M_i to schedule
  end for
  return M
procedure `emulatedTopology`$(\mathcal{M})$**:**
  ▷ *Initialize a multigraph*
  $G = (V, E)$, $V = \{1, ..., n\}$, $E : V \times V \to \mathbb{N}$
  Normalize the demand matrix $\mathcal{M}$
  $\mathcal{M} \leftarrow (k-1) \cdot n \cdot \mathcal{M}$
  $\mathcal{R}$ = Round($\mathcal{M}$) ▷ *Matrix rounding*
  for *each node pair* (u, v) **do**
    ▷ *Allocate bandwidth for bulk demand*
    $E((u,v)) \leftarrow \mathcal{R}(u,v)$ ▷ *# Edges between u, v*
    ▷ *Allocate bandwidth for residual demand*
    $E((u,v)) \leftarrow E((u,v)) + 1$
  end for
  ▷ *Ensure that the final graph is regular*
  $x^{in} \leftarrow k \cdot n - x^{in}$ ▷ *Remaining in-degree*
  $x^{out} \leftarrow k \cdot n - x^{out}$ ▷ *Remaining out-degree*
  $G' = (V, E')$ = ConfigurationModel(x^{in}, x^{out})
  $E \leftarrow E \uplus E'$
  return G

throughput of $\frac{2}{3}$, breaking the throughput bounds of oblivious periodic networks. We discuss the choice of k in §8.2.2.

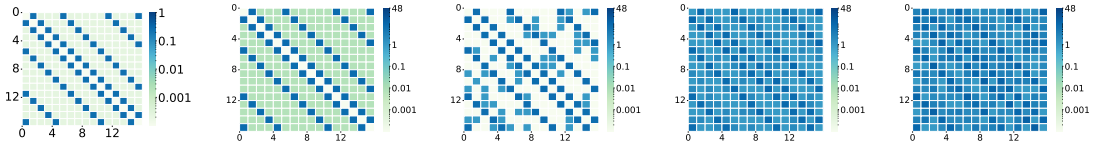
Periodic schedule: The key innovation in *Vermilion* is its demand-aware periodic schedule. Algorithm 6 outlines the steps to derive the periodic schedule. Figure 8.4 shows an example workflow of *Vermilion* for $k = 3$. Given a demand matrix $\mathcal{M}$, we first generate an emulated topology that can achieve high throughput for the given demand matrix. Our emulated topology is always regular and allows us to then decompose it into a periodic schedule. We construct the emulated topology as follows:

- **Matrix rounding:** We first normalize the demand matrix such that the maximum sum of any row and column is at most 1. We then upscale the demand matrix by $(k-1) \cdot n$, where k is a parameter to *Vermilion* and n is the number of nodes. We round entries of the scaled matrix such that the sum of each row and column remains the same, a technique known as matrix rounding [57].

- **Demand-aware multigraph:** Based on the rounded matrix, we construct a multigraph by adding edges between each node pair based on the rounded



(a) DLRM data parallelism demand matrix (b) Capacity provided by Oblivious network (c) Capacity provided by *Vermilion*



(d) Normalized matrix (e) Scaled by $(k-1)\cdot n$ (f) Rounded matrix (g) $\uplus$ complete graph (h) $\uplus$ configuration model

Figure 8.4: An example workflow of *Vermilion* with (8.4a) DLRM data parallel demand matrix. An oblivious periodic network (8.4b) provides uniform capacity between all node pairs irrespective of the demand matrix. *Vermilion* (8.4c) provides capacity between node pairs based on the underlying demand matrix. *Vermilion* first (8.4d) normalizes the given demand matrix, upscales it by $(k-1)\cdot n$ (8.4e), rounds it (8.4f), augments it with a complete graph (8.4g) for any-to-any connectivity, and finally augments it with additional links to ensure regularity (8.4h). The resulting matrix (a multigraph) is the target emulated topology (8.4c), which is then decomposed into a sequence of matchings for the periodic schedule.

matrix. For instance if the rounded matrix specifies 2 between s-t node pair, then we add 2 edges between s and t. This ensures that majority of the demand matrix is served efficiently in a demand-aware manner.

- **Demand-oblivious residual graph:** Matrix rounding may not exactly match the original demand matrix and can leave some residual demand. To address this, we ensure that any residual demand can be routed by adding one additional edge between each node pair in the multigraph. This step guarantees any-to-any connectivity and is demand-oblivious, meaning it is independent of the specific demand matrix.

- **Augmenting the regularity of the graph:** At this point, say each node has in-degree x_i^{in} and out-degree x_i^{out}, we then take the degree sequences $\langle k\cdot n - x_1^{in}, ..., k\cdot n - x_n^{in}\rangle$ and $\langle k\cdot n - x_1^{out}, ..., k\cdot n - x_n^{out}\rangle$, and add additional links to our graph based on configuration model using the above degree sequence.

The resulting graph is our target emulated topology. The above construction always leads to a multigraph with degree $k\cdot n$, a directed regular graph, that can be decomposed into $k\cdot n$ number of perfect matchings. These perfect matchings are then executed in round-robin periodically, using optical circuit switching.

Routing: *Vermilion* relies solely on single-hop routing, as the topology provides sufficient direct links between communicating nodes over time, ensuring high throughput even with direct communication.

Congestion control: *Vermilion* does not explicitly require any congestion control algorithm in the network. If the network is all-optical i.e., when servers connect

directly to circuit switches, then *Vermilion* does not require even end-host congestion control (except for reliability) since every packet is transmitted directly to the destination. The direct communication paths ensure that the network is not congested.

Flow scheduling: Given that packets reach destinations directly, flow scheduling dominates in determining per-flow performance. We consider that the packets of all active flows at the end-host are scheduled in a round-robin manner and all transmissions are paused during the synchronized reconfiguration events, similar to prior work [188]. Scheduling algorithms such as shortest remaining processing time (SRPT) based on remaining flow size could potentially improve flow completion times. We leave the design of tailored scheduling algorithms for future work.

8.2.2 Throughput Guarantees of Vermilion

Vermilion offers attractive theoretical guarantees on throughput and consequently completion time for a given demand matrix. We discuss the key properties of *Vermilion* in the following.

Theorem 22 (Throughput lower bound)**.** *Vermilion achieves a throughput of at least* $\frac{k-1}{k} \cdot (1 - \Delta_r)$ *using single-hop routing for any given demand matrix within the hose model, where* Δ_r *is the fraction of time spent in reconfiguration and* k *is a parameter to Vermilion.*

Proof. We assume, without loss of generality, that the capacity of each physical link is 1 and the degree of the physical topology is $\hat{d}$. A demand matrix $\mathcal{M}$ in the hose model has the property that the sum of every row and column is at most $\hat{d}$. Let the period (number of timeslots) of the periodic schedule be $k \cdot \frac{n}{\hat{d}}$, i.e., k times the period of an oblivious periodic schedule. The emulated multigraph G has a degree of $k \cdot n$, and each link has capacity $\frac{1}{k \cdot \frac{n}{\hat{d}}} = \frac{\hat{d}}{k \cdot n}$. We now upscale both the demand matrix $\mathcal{M}$ and each link capacity of the emulated multigraph by $\frac{k \cdot n}{\hat{d}}$. Thus, it is equivalent to find the throughput of the emulated multigraph G with degree $k \cdot n$ and each link having capacity 1, under the scaled demand matrix $\mathcal{M}' = \frac{k \cdot n}{\hat{d}} \cdot \mathcal{M}$, where the sum of each row and column is at most $k \cdot n$. We seek to find the edge multiset of the emulated multigraph G that maximizes throughput. In order to show that *Vermilion* achieves a throughput of at least $\frac{k-1}{k}$, it suffices to show that *Vermilion* can satisfy $\frac{k-1}{k} \cdot \mathcal{M}'$ within the capacity constraints. According to *Vermilion*, edges are added based on matrix rounding using $\frac{(k-1) \cdot n}{\hat{d}} \mathcal{M} = \frac{k-1}{k} \cdot \mathcal{M}'$. These edges satisfy all the demand in the demand matrix $\frac{k-1}{k} \cdot \mathcal{M}'$ except for entries that were rounded down during the rounding process. This is because, the sum of every row and column in the demand matrix (and the rounded matrix) is at most $(k-1) \cdot n$, entries are either rounder up or down, and links are added only to non-zero entries. The rounding process utilizes at most $(k-1) \cdot n$ incoming and outgoing links from each node. We are left with at least n incoming and outgoing links for each node. The residual demand is then served by adding additional edges between each node pair. The residual demand between any node pair is strictly less than 1 (due to rounding), and a single additional link between the pair can fully satisfy the residual demand. Finally, each link loses $(1 - \Delta_r)$ fraction of capacity due to reconfigurations and hence the overall throughput is $\frac{k-1}{k} \cdot (1 - \Delta_r)$. □

Theorem 22 suggests that a throughput of $\frac{k-1}{k}$ is achievable for any given demand matrix, with k acting as a control parameter for throughput. Essentially, k represents the factor by which the schedule of *Vermilion* is elongated compared to an oblivious schedule that provides periodic any-to-any connectivity. For instance, with $k = 3$, *Vermilion* guarantees a throughput of $\frac{2}{3}$, and this can be further increased by increasing k, resulting in higher throughput. However, this comes at the cost of longer schedules and increased delay. As a result, the throughput guarantee can only be achieved over extended periods of time, making higher k values potentially unsuitable for workloads with stringent latency requirements. In our evaluations, we use $k = 3$ by default, as it strikes a good balance between throughput and schedule length.

8.2.3 Practicality of Vermilion

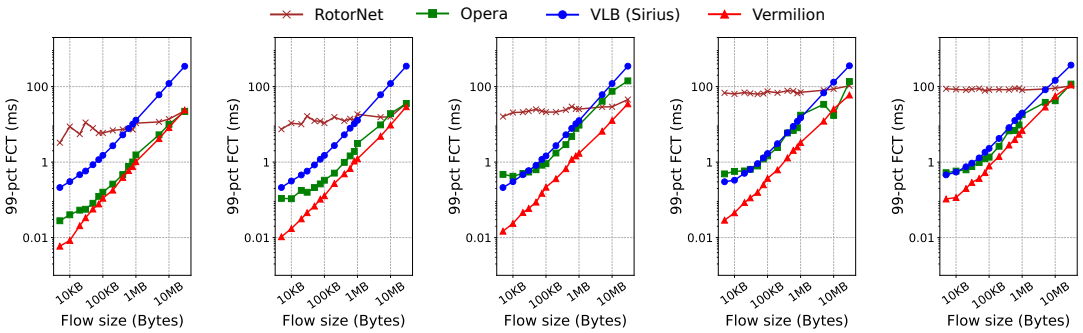
We discuss the practicality of *Vermilion* in the context of modern datacenter infrastructure and optical circuit-switching technologies.

Scalability: Periodic circuit-switched networks in general exhibit excellent scalability properties [40, 60, 186, 189]. The required switch size is arguably the most important scalability concern in these networks. A simple leaf-spine topology may quickly become limited in terms of the required switch size. *Vermilion* can scale to large topologies with thousands of nodes interconnected by circuit switches arranged in a non-oversubscribed k-ary fattree topology [31]. Fattree allows any permutation to be executed. For example, even a greedy algorithm can find edge-disjoint paths corresponding to a required matching. These edge-disjoint paths then reveal the required circuits at each switch in the network, consequently, the sequence of matchings for each switch can be obtained. Further, *Vermilion* is much more scalable than oblivious periodic networks in terms of protocol stack as it only requires single-hop routing and does not rely on complex in-network congestion control algorithms.

Complexity: Deriving a schedule based on *Vermilion* is solvable in polynomial time. Specifically, matrix rounding is polynomial time solvable [57], and all other transformations have a complexity of $O(n^2)$, which is inherent to any demand-aware approach due to the need to traverse the demand matrix at least once. In contrast, approaches that rely on Birkhoff decomposition not only produce schedules with variable reconfiguration durations but also face the challenge that finding a schedule of minimum length using Birkhoff decomposition is known to be NP-hard [162].

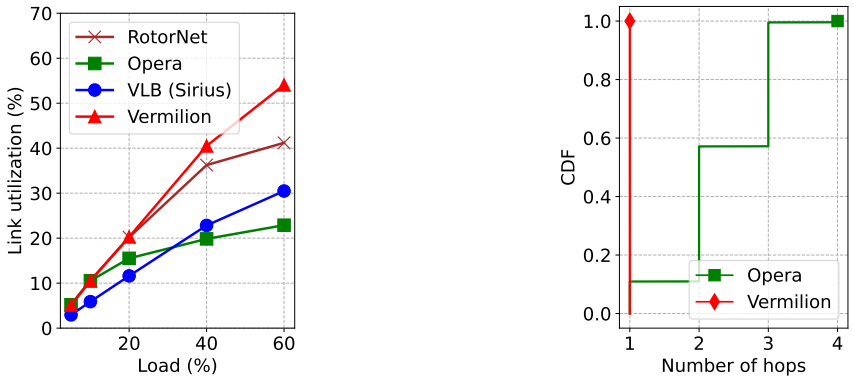
Updating the schedule: *Vermilion* being a demand-aware design, it requires that the switches can be updated with a new periodic schedule when the communication patterns change. Efficiently updating the circuit-switching schedule is an active area of research, particularly for fast-reconfigurable periodic circuit switches operating on microsecond (μs) or nanosecond (ns) timescales [188, 190]. Commercially available optical switches that can reconfigure at millisecond scale already allow for updating the switch with arbitrary matchings via control plane [220].

Demand estimation: Modern datacenters are capable of accurately estimating the demand matrix at scale [222]. Further, more recent distributed training workloads in GPU clusters have a predictable demand matrix that is also periodic in nature [172, 265, 266].



(a) Load=5% (b) Load=10% (c) Load=20% (d) Load=40% (e) Load=60%

Figure 8.5: Flow completion times for the websearch workload. *Vermilion* significantly improves the 99-percentile FCTs compared to existing designs, both for short flows and long flows.



(a) Link utilization (b) Path lengths

Figure 8.6: Opera uses k-shortest paths for short flows, leading to longer path lengths and loss in link utilization, whereas *Vermilion* uses exclusively direct paths and achieves higher link utilization compared to other designs.

Overall, we believe *Vermilion* does not fundamentally require novel hardware components, and is well within the practical capabilities of existing technologies.

8.3 Evaluation

We evaluate the performance of *Vermilion* in terms of flow completion times and link utilization using real-world datacenter workloads (§8.3.1), and in terms of throughput with commonly observed demand patterns in datacenters (§8.3.2). We compare *Vermilion* with existing periodic network designs, namely, RotorNet [186], Opera [187] and Sirius [60].

8.3.1 Flow Completion Times & Link Utilization

Our evaluation in this section is based on packet-level simulations using htsim [11].

Topology: We consider a datacenter consisting of 512 servers arranged across 64 top-of-rack (ToR) switches. These ToR switches are interconnected by a layer of 8 optical circuit switches, with all link capacities set to 25 Gbps. For all systems compared, including *Vermilion*, the circuit switches have a reconfiguration delay of

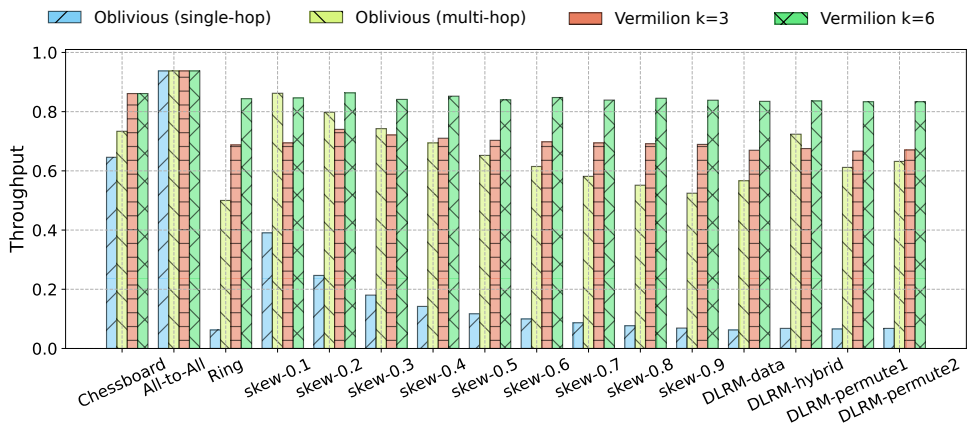


Figure 8.7: *Vermilion* achieves higher throughput even with single-hop direct communication compared to oblivious networks with multi-hop routing, across real-world demand matrices as well as synthetic demands. The throughput of oblivious periodic networks is severely low when restricted to single-hop routing.

$0.5\mu s$, which is well within the practical capabilities of existing technologies [60].

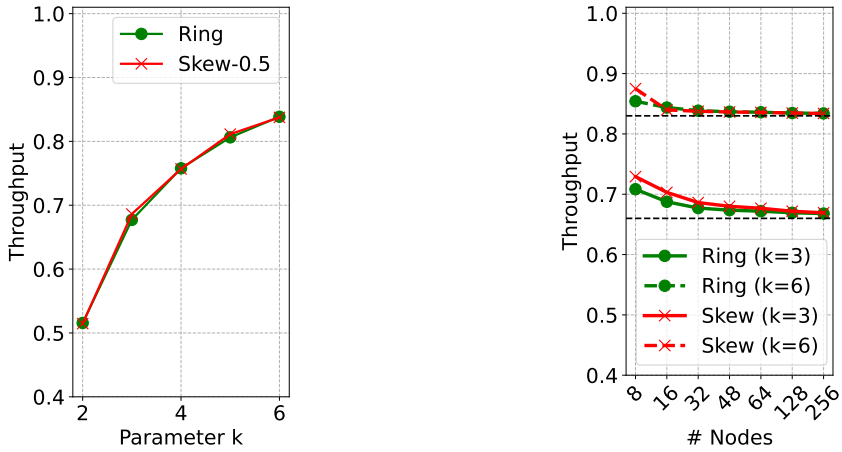
Comparisons & Configurations: We set $k = 3$ for *Vermilion*, and compare it with RotorNet [189], Opera [187] and Sirius [60]. We use the recommended configurations of these systems as provided in their respective papers. Specifically, RotorNet uses RotorLB load-balancing algorithm for managing congestion between the ToRs; Opera uses k-shortest paths for short flows within the same timeslot to reduce FCTs; Sirius uses valiant load balancing (VLB) [60, 257]; and *Vermilion* uses single-hop routing without any further congestion control mechanisms. All systems use NDP [124] as the transport protocol. In the case of *Vermilion*, we turn off all actions of NDP and set a constant congestion window size.

Workload: We launch the websearch [35] workload, a widely-used datacenter benchmark from prior work. We simulate a pair-wise communication pattern between racks and vary the load between 5% and 60% of the server link capacity. Flows arrive according to a poisson process such that an average load is achieved on each server's outgoing link. We construct a periodic schedule for *Vermilion* based on the average load[1], while the other systems use an oblivious schedule (emulating an all-to-all mesh topology) [60, 187, 189] for all loads. We report the 99-percentile flow completion times (FCTs) and the average link utilization across server downlinks.

***Vermilion* significantly improves short flow FCTs:** Figure 8.5 shows the flow completion times for the websearch workload. *Vermilion* significantly improves the 99-percentile FCTs compared to existing designs, even at low loads. Figure 8.5a, at 5% load, shows that *Vermilion* improves the 99-percentile FCTs by 99.81% compared to RotorNet, by 78.8% compared to Opera, and by 97.2% compared to VLB. This is because *Vermilion* provides direct communication links between communicating pairs, reducing the FCTs for short flows. As the load increases, *Vermilion* improves the FCTs for short flows on average by 99.88% compared to RotorNet, by 88.01% compared to Opera, and by 90.55% compared to VLB.

***Vermilion* does not compromise long flow FCTs:** While *Vermilion* significantly reduces short flow FCTs, it also improves the FCTs for long flows. Figure 8.5

[1]The instantaneous demand matrix is in fact variable but we optimize based on the average load.



(a) Impact of parameter k

(b) Variation with network size

Figure 8.8: *Vermilion*'s throughput follows its theoretical lower bound with increasing k and converges to the bound as the network size increases.

shows that *Vermilion* outperforms alternative approaches in terms of long flow FCTs. This is primarily due to *Vermilion*'s demand-aware schedule, which provides direct links between communicating pairs. At 5% load (Figure 8.5a), *Vermilion* achieves comparable FCTs to other systems. At 10% load (Figure 8.5b), *Vermilion* improves long flow FCTs by 17.68% compared to RotorNet, 18% compared to Opera, and 91.5% compared to VLB. On average, as the load increases, *Vermilion* achieves a 43% improvement in long flow FCTs compared to existing methods. *Vermilion*'s high-throughput capability is especially beneficial for long flows, as they are bandwidth-intensive and require a robust interconnect to improve FCTs.

***Vermilion* improves average link utilization:** We sample the link utilization of each server at $1ms$ intervals and report the average link utilization across all downlinks in the network. Figure 8.6 highlights significant improvements in link utilization with *Vermilion*. Up to 10% load, *Vermilion* achieves similar link utilization to RotorNet and Opera. However, as the load increases, Opera's utilization declines beyond 20%. At 40% load, *Vermilion* improves average link utilization by 1.11× compared to RotorNet, 2.04× compared to Opera, and 1.77× compared to VLB. Figure 8.6b presents the CDF of the number of hops taken by Opera in the ToR-to-ToR optical interconnect. Opera employs k-shortest paths to opportunistically reduce FCTs for short flows, but this results in longer paths, lower throughput, and consequently, reduced link utilization. In contrast, *Vermilion* provides direct single-hop communication between ToR pairs, significantly enhancing link utilization.

8.3.2 Throughput

We now evaluate the throughput capabilities of *Vermilion* in comparison to existing approaches by directly analyzing throughput using a linear programming approach, eliminating protocol-level interference. We use Gurobi [117] to solve the linear program for throughput maximization.

Topology: We consider a 16 node topology with degree 4 (incoming and outgoing links), interconnected by a layer of optical circuit switches. We set the link capacity to 25Gbps and the reconfiguration delay to $0.5\mu s$ as before. We obtain similar results even for a larger network with 48 nodes (Figure 8.9).

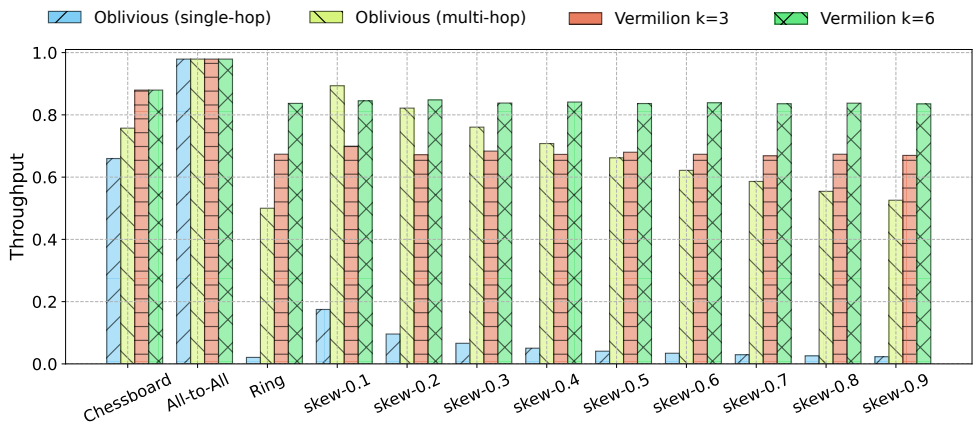


Figure 8.9: Throughput of oblivious periodic network and *Vermilion* for a 48 node topology. *Vermilion* achieves higher throughput compared to oblivious networks, even with single-hop direct communication.

Demand matrices: We evaluate across a variety of demand matrices gathered from a 16 node GPU cluster running distributed training workload of a deep learning recommendation model, under data-parallelism, hybrid parallelism and permutations of the data-parallel workload. We further consider synthetic demand matrices to stress the throughput capabilities of each system; parametrized by a skew parameter that combines a permutation matrix with an all-to-all uniform matrix. For instance, skew-0.1 indicates a 10% skew towards a permutation matrix.

Comparisons: We compare *Vermilion* with an ideal oblivious periodic network that emulates an all-to-all mesh topology, using an ideal routing algorithm that maximizes throughput. We call this system *Oblivious (multi-hop)*. We also compare *Vermilion* with oblivious systems restricted to single-hop. *Vermilion* explicitly uses single-hop routing and we compare $k = 3$ and $k = 6$.

***Vermilion* consistently achieves high throughput:** From Figure 8.7, we see that *Vermilion* achieves high throughput across a wide range of communication patterns. Specifically for distributed training workloads, we see that *Vermilion* achieves 6.64% better throughput compared to oblivious periodic networks using an ideal multi-hop routing. Figure 8.7 shows the clear advantage (and the need) for multi-hop routing in the case of oblivious networks, with significantly lower throughput under single-hop routing. *Vermilion*, on the other hand, achieves high throughput with single-hop routing. Oblivious networks with an ideal routing scheme, however, outperform *Vermilion* when the demand matrix is close to uniform. This is expected since with $k = 3$, *Vermilion*'s lower bound is $\frac{2}{3}$. As the skew increase, oblivious network design drops to a throughput of $\frac{1}{2}$ as discussed in §8.1.2. In contrast, *Vermilion* maintains a throughput greater than $\frac{2}{3}$ even with skewed demand matrices.

***Vermilion*'s throughput converges to the lower bound:** Figure 8.8 confirms our theoretical bounds established in §8.2.2. With increasing k, Figure 8.8a shows that *Vermilion*'s throughput closely tracks its lower bound of $\frac{k-1}{k}$. Further, even with increasing size of the network, Figure 8.8b shows that *Vermilion*'s throughput gradually converges to the theoretical lower bound of $\frac{2}{3}$ for $k = 3$ and $\frac{5}{6}$ for $k = 6$, respectively. This demonstrates the robustness of *Vermilion*'s throughput guarantees across different network sizes and demand matrices.

8.4 Limitations and Future Work

Vermilion represents an initial step toward demand-aware periodic networks capable of achieving high throughput for any demand matrix using only single-hop routing. However, several challenges remain and open avenues for future research.

Temporal dependencies in communication patterns: *Vermilion* assumes that the demand matrix, whether defined by rate (bits per second) or volume (bits), is available and that the demands between source-destination pairs are independent. However, certain workloads, such as distributed training, exhibit temporal dependencies in their communication patterns. For example, while the demand matrix may accurately represent traffic, specific portions of the demand (e.g., from the backward pass) may only become available after the completion of other parts (e.g., from the forward pass) [172, 265]. Addressing these temporal dependencies presents a significant research challenge and offers opportunities to further optimize topologies like *Vermilion* for such workloads.

Structured communication patterns: *Vermilion* is designed to achieve high throughput for any arbitrary demand matrix. However, some communication patterns have inherent structure that could be exploited for further optimization. For example, the ring-allReduce collective communication, commonly used in distributed training, can be efficiently supported by a simple ring-emulated topology with periodic schedules. Optimizing for specific communication patterns is complementary to our approach. In principle, *Vermilion* could be extended to recognize and leverage these structured communication patterns using existing solutions [285], potentially achieving even higher throughput and faster completion times. Exploring these optimizations is an avenue for future work.

Fault-tolerance and resilience: *Vermilion* does not explicitly address fault-tolerance or resilience. While the periodic nature of the network may offer some inherent resilience to failures, designing fault-tolerant periodic schedules remains an open challenge. Failures in optical networks can be particularly difficult to detect, as they often manifest as packet corruption due to optical collisions. Recent work discusses techniques to mitigate link-layer and physical-layer errors [188]. Future work could explore the design of fault-tolerant schedules that can quickly adapt to failures while still maintaining high throughput.

Heterogeneous link capacities: In this work, we assume that all physical links in the topology have uniform capacity. However, datacenter topologies often include links with heterogeneous capacities. For example, faulty auto-negotiation between two NICs can result in a link operating at a lower capacity than expected. In practice, these capacities are often multiples of a base rate. We believe *Vermilion* can be generalized to accommodate heterogeneous link capacities by selecting an appropriate base capacity and adjusting the topology to handle varying multiples of that capacity. We leave the generalization of *Vermilion* for heterogeneous link capacities to future work.

8.5 Related Work

Datacenter topologies have been widely studied in the literature both in the context of traditional packet-switched networks [31, 65, 113, 115, 179, 246, 248, 256, 281] and emerging reconfigurable optically circuit-switched networks [60, 68, 79, 81, 98, 104,

109,114,121,148,182,187,189,234,249,261,264,279,286]. In the design of topologies, various metrics of interest have been considered. For instance, uniformly high bandwidth availability [31,115], expansion [248,256], fault-tolerance [179], and even the life cycle management of a datacenter [281]. In the context of reconfigurable networks, typically, the goal has been either to minimize the reconfiguration overhead [60,189] or to minimize the bandwidth tax [98,109,266,279].

Recent works argue for a new measure i.e., ''throughput'', to understand the maximum load supported by a topology [18,114,145,204,247]. In fact, the max-flow that relates to the throughput of a topology, can be $\mathcal{O}(\log n)$ factor lower than the sparsest cut [145,168,169]. Namyar et al. study the throughput upper bound for static datacenter topologies and show a separation between Clos (i.e., fat-trees) and expander-based networks in terms of throughput [204]. In the context of reconfigurable networks, only recently have the throughput bounds of demand-oblivious networks been established [18, 42, 114]. In this chapter, we show the first separation result between demand-aware and demand-oblivious networks and present *Vermilion* that breaks the throughput bounds of existing demand-oblivious periodic networks.

While throughput of a datacenter topology is interesting from a theory standpoint, a vast majority of the literature focuses on practically achieving the ideal throughput of a topology. For instance, congestion control [21,35,112,164,173,193,232,269], buffer management [16,19,24,38,46,133], scheduling [32,37,225], load-balancing [34,110,152]. In fact, the underlying protocols can turn out to be the key enablers (or limiters) of system performance in the datacenter [164]. Only recently, congestion control [41,82,201] and routing [171] algorithms tailored for reconfigurable networks have been considered. Interestingly, if *Vermilion* is deployed in an all-optical setting, it does not require any additional congestion control, buffer management and load-balancing mechanisms, since it relies solely on direct communication.

8.6 Summary

We introduced *Vermilion*, a simple demand-aware optical interconnect that achieves high throughput using only periodic circuit-switching and direct communication. Through formal analysis, we established throughput bounds for *Vermilion*, marking the first formal separation result that demonstrates demand-aware reconfigurable networks' superiority over oblivious counterparts in terms of throughput. We believe that *Vermilion* offers a practical solution for datacenter networks with predictable communication patterns. In the future, we plan to explore the temporal dependencies in communication patterns that arise in distributed training workloads and investigate how *Vermilion* can be further optimized for such scenarios.

9

Augmenting Demand-Aware Interconnects with Predictions

The rapid growth of datacenter traffic, accompanied with a slowdown in Moore's law for the capacity scaling of electrical switches has led to a surge in research on reconfigurable optical interconnects, which offer high bandwidth and low latency [60]. The diverse applications hosted by datacenters and their changing communication patterns further necessitate a reconfigurable network architecture that satisfies the network demands of the applications [222]. Prior work shows the feasibility of such a network architecture: It has been established that optical circuit-switching is practical [40, 60, 109, 188, 189, 221, 266]. Moreover, adapting the network topology to the underlying communication patterns can improve the performance of datacenter workloads [68, 98, 163, 176, 221, 266]. Optical circuit switches are already commercially available that can reconfigure at millisecond scale [220]. Recently, optical circuit switches have also been deployed in Google's Jupiter network [222]. Significant research efforts have been made to reduce the reconfiguration time to a few microseconds [186, 188, 189] or nanoseconds [60].

Notably, *periodic and fixed-duration* reconfigurable optical circuit-switching technologies have emerged as viable solutions for datacenter networks [60, 188]. In these networks, circuits are established periodically and remain active for a fixed duration, enabling fast reconfigurations. Given their practicality, recent work has focused on the theoretical throughput guarantees of such interconnects. Demand-oblivious periodic networks have been shown to be tightly bounded by $\frac{1}{2}$ in terms of throughput i.e., irrespective of the underlying communication pattern, a demand-oblivious periodic network can guarantee a throughput of at least $\frac{1}{2}$ and in the worst case (a ring communication pattern), the throughput is at most $\frac{1}{2}$ [18, 42]. Although demand-oblivious approaches simplify the schedule and ideally achieve a throughput of $\frac{1}{2}$, they require highly complex protocols, often non-standard, in order to achieve this throughput. For instance, RotorNet [189], Sirius [60], Opera [187], Mars [18], Shale [40], all require non-standard multi-hop routing and congestion control with additional support from the network switches (ToRs). Furthermore, due to the periodic nature of the topology, multi-hop routing results in packet reordering that may have undesirable performance issues with modern RDMA network cards.

In contrast, *Vermilion* breaks this bound, achieving a throughput of $\frac{2}{3}$ by leveraging a demand-aware approach, while relying solely on single-hop routing and fixed-duration reconfigurations. However, the demand-aware approach of *Vermilion* is based on the assumption that the demand matrix is known in advance. In practice though, the demand matrix is not known a priori, and the network must adapt to evolving traffic patterns. This begs the question:

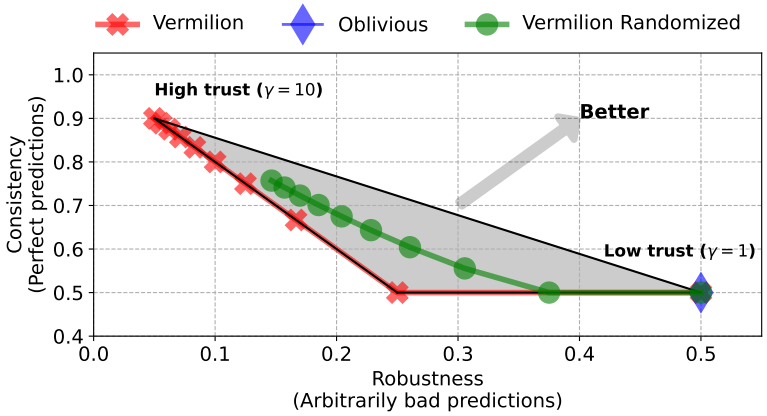


Figure 9.1: We present the first throughput analysis of demand-aware networks from an online perspective by augmenting *Vermilion* with machine-learned predictions of the underlying demand matrix. Our analysis demonstrates that *Vermilion* achieves $\frac{\gamma-1}{\gamma}$-consistency under perfect predictions and $\frac{1}{2\cdot\gamma}$-robustness under arbitrarily inaccurate predictions, where k serves as a trust factor. Our results reveal a tradeoff between consistency and robustness, highlighting the cost of prediction (estimation) errors when employing a demand-aware approach. The grey shaded region indicates the convex hull of the consistency-robustness tradeoff curve with our randomized algorithm.

How does the accuracy of demand estimation impact the potential throughput gains in demand-aware networks?

Unfortunately, little is known in the literature about the throughput of demand-aware networks when adding demand uncertainty to the mix. The majority of works [98, 109, 176, 221, 266] assume that the demand matrix is known a priori and derive the circuit-switching schedule in a demand-aware manner, often using Birkhoff-von-Neumann (BvN) [66] or matrix rounding [57] techniques. However, optimizing the network topology based on an estimated matrix is susceptible to prediction errors, which can result in significantly lower throughput than even a simple demand-oblivious topology. For instance, if the estimated demand matrix resembles a ring, the throughput of a demand-aware network can reach as high as 1 with perfect predictions, but drop to as low as $\mathcal{O}(\frac{1}{n})$ if the actual communication pattern is all-to-all. In contrast, a demand-oblivious periodic network guarantees a throughput of at least $\frac{1}{2}$ in both cases. Given the practical challenges of accurately predicting or estimating the demand matrix, it is crucial to understand the performance of demand-aware networks under demand uncertainty.

In this work, we initiate the study of demand-aware networks under demand uncertainty, focusing on the design of periodic and fixed-duration reconfigurable optical interconnects. We develop an analytical framework from an online perspective, where an algorithm only has access to an estimated demand matrix from a prediction oracle, without foresight into the actual demand matrix or future traffic. Building upon existing work, we augment *Vermilion* with predictions and analyze its performance using competitive analysis, comparing our online algorithm with an offline optimal algorithm that has full visibility of the underlying demand matrix. Consistent with prior work, we focus on two metrics: *consistency*, which measures the performance under perfect predictions, and *robustness*, which assesses performance under arbitrarily poor predictions. Additionally, we define an error

function to capture the differences between the estimated and actual demand matrices, allowing us to parameterize the competitive ratio based on prediction errors.

We present the first throughput analysis of demand-aware networks from an online perspective by augmenting *Vermilion* with machine-learned predictions of the underlying demand matrix, as illustrated in Figure 9.1. Our analysis shows that *Vermilion* achieves $\frac{\gamma-1}{\gamma}$-consistency under perfect predictions and $\frac{1}{2\gamma}$-robustness under arbitrarily poor predictions, where $\gamma \geq 1$ is a user-specified trust factor which can be chosen depending on how much one trusts the prediction. Our results reveal a tradeoff between consistency and robustness, emphasizing the cost of prediction (estimation) errors when adopting a demand-aware approach. We further present a randomized algorithm building upon *Vermilion*, revealing an entire spectrum of consistency-robustness tradeoff curves depending on the trust factor. For instance, with high trust, our randomized algorithm has a consistency of $1 + \frac{1}{2\cdot\gamma} - \frac{H_\gamma}{\gamma}$ and an improved robustness of $\frac{H_\gamma}{2\cdot\gamma}$, where H_γ denotes the γ^{th} harmonic number. In contrast, demand-aware networks based on the Birkhoff-von Neumann decomposition (e.g., Helios [98]) exhibit a competitive ratio of 1 under perfect predictions but have unbounded robustness (∞) under arbitrarily poor predictions. These results highlight the practical challenges of applying demand-aware techniques to periodic networks in the presence of prediction errors.

We evaluate the performance of *Vermilion* with predictions through extensive simulations on both real-world and synthetic traffic matrices under varying prediction accuracies. Our results demonstrate the potential of machine-learned predictions to improve the throughput of reconfigurable optical interconnects, while also highlighting the challenges posed by prediction inaccuracies. We believe this work presents several opportunities for future research in both systems and theory, which we discuss at the end of this chapter. Notably, improving robustness and identifying the Pareto-optimal frontier of the consistency-robustness tradeoff remain open challenges for future investigation.

In summary, our main contributions are as follows:

- We formalize demand-aware network design in an online setting with the aim to maximize throughput.
- We take *Vermilion* into the online setting with predictions and analyze its performance through competitive analysis. We demonstrate that the augmented *Vermilion* achieves $\frac{\gamma-1}{\gamma}$-consistency under perfect predictions and exhibits $\frac{1}{2\gamma}$-robustness under arbitrarily poor predictions. Here, γ serves as a trust factor.
- We present a randomized algorithm building upon *Vermilion*, revealing an entire spectrum of consistency-robustness tradeoff curves depending on the trust factor γ.
- We evaluate the performance of *Vermilion* with predictions through extensive simulations on both real-world and synthetic traffic matrices under varying prediction accuracies, empirically confirming our theoretical results.

9.1 Background & Motivation

To motivate our work, we first describe the physical network architecture and the demand-aware network design problem for throughput maximization. We then discuss the limitations of demand-aware networks under demand uncertainty and the challenges of integrating predictions into the design of demand-aware networks.

Physical network architecture. This chapter focuses on periodic, fixed-duration reconfigurable networks, which have gained significant practical relevance in recent years [60, 188]. Following previous work [40, 60, 186, 189], we consider a network consisting of n nodes, each connected by d directed physical links with capacity c. The network is interconnected via optical circuit switches, which execute a sequence of matchings in a periodic manner. As a result, the node-to-node connectivity can be represented as a periodic graph that evolves over time. Figure 9.2 shows an example of a periodic network with 6 nodes and degree 1 in each timeslot. Specifically, a periodic graph consists of a sequence of Γ directed graphs (with degree bound d), where Γ represents the period of the graph. Each graph is active for transmission over a fixed duration of 1 time unit, after which the network topology is reconfigured. The reconfiguration process requires Δ_r time units, during which transmissions are paused. This implies that the network spends a fraction Δ_r of its time in reconfiguration. Here, $(1 - \Delta_r)$ can be interpreted as the duty cycle of the network. Additionally, we assume that the network operates synchronously, with all nodes synchronized to a common clock [188]. Recent technologies are capable of reconfiguration delays as low as $3.84ns$, allowing the timeslot durations to be chosen such that a duty cycle of $(1 - \Delta_r) = 0.99$ can be achieved [60].

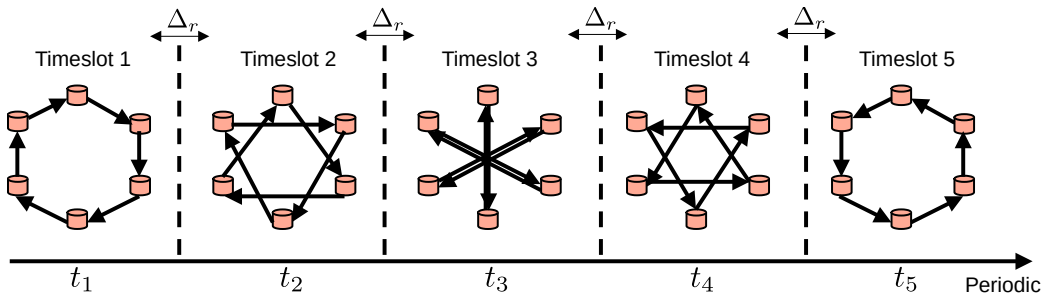


Figure 9.2: A periodic fixed-duration reconfigurable network operates by switching between different configurations (graphs) over time. Each configuration remains active for a fixed duration (timeslot), during which data transmission occurs. After each active timeslot, the network undergoes a reconfiguration phase, which consumes a fraction Δ_r of time, during which transmissions are paused. The entire network is defined by a repeating sequence of these configurations, forming the periodic network.

Demand-aware network design. Two primary types of network designs have been explored using the architecture described above. The first type, demand-oblivious networks, selects a sequence of graphs that effectively emulates a complete graph over time. This type guarantees a throughput of $\frac{1}{2}$ regardless of the communication pattern [18, 42]. The second type, demand-aware periodic networks, optimizes the topology by selecting graphs based on an estimated demand matrix, aiming to enhance throughput. However, this demand-aware design is

typically addressed from an offline perspective, assuming a perfect demand matrix [98, 109, 176, 221]. In practice, networks operate in an online manner, where demand matrices are subject to prediction errors, which can result in a significantly lower throughput than demand-oblivious networks. For instance, if the estimated demand matrix represents a permutation matrix (ring communication pattern), similar to the graph of timeslot t_1 in Figure 9.2, a demand-aware periodic network will aim to maximize throughput by selecting only the graph depicted in timeslot t_1 in its periodic schedule. This would yield a throughput of 1, provided that there are no estimation errors. However, if the actual demand matrix resembles a different pattern, like the graph in timeslot t_5, then the chosen schedule (which only includes t_1) would result in a throughput of $\frac{1}{5}$ since it requires transmitting over paths of length 5, which is significantly worse than the throughput of $\frac{1}{2}$ guaranteed by an oblivious periodic network that emulates a complete graph using all five timeslots shown in Figure 9.2. This work focuses on the design and analysis of demand-aware periodic networks in an online setting, aiming to maximize throughput while accounting for imperfect demand predictions.

Challenges. Integrating predictions into demand-aware network design presents several key challenges. First, the network must perform (as close as possible to) optimally when the demand predictions are accurate. Second, it must be robust to prediction errors, i.e., maintain reasonable performance even when predictions deviate significantly from actual demands. Lastly, the network's performance should degrade gracefully as prediction errors increase. These challenges raise a critical question: *To what extent can demand-aware networks outperform demand-oblivious networks under demand uncertainty?* Understanding these tradeoffs requires a formal analysis of demand-aware networks in an online setting, which we initiate in this work.

9.2 Preliminaries and Problem Formulation

We begin by defining the throughput of periodic networks and then introduce the online periodic network design problem.

9.2.1 Throughput of Periodic Networks

The throughput serves as an essential metric for the assessment of the sustainable load a network can handle. We rely on the definitions introduced in the previous chapters. We briefly discuss the demand matrix, throughput and the emulated graph concepts here for completeness.

The demand matrix specifies the demand in bits per second for each source--destination pair of nodes in the network. As established in prior work [204], we consider the hose model [94], in which one assumes that the total demand arriving and leaving each node is less than its corresponding capacity limits.

Definition 22 (Demand matrix)**.** *Given a set V of n nodes, each with d incoming and outgoing links with capacity c, a* demand matrix $\mathcal{M}$ *has size $n \times n$, wherein for each pair u, v of nodes, the entry $m_{u,v}$ in $\mathcal{M}$ specifies the demand from u to v in* bits per second*. Any demand matrix has the property that for every node u, the incoming and the outgoing demand do not exceed the overall capacity, i.e., $\sum_{v \in V} m_{v,u} \leq c \cdot d$ and $\sum_{v \in V} m_{u,v} \leq c \cdot d$.*

Periodic and static graphs. By default, our graphs are directed and may have parallel edges. We work both with static and periodic graphs. Herein, a periodic graph with period length Γ is defined as $G = (V, E_1, \ldots, E_\Gamma)$, where E_i is the (directed multi-) edge set of the i-th period. Moreover, each edge $e \in E_i$, $i \in \{1, \ldots, \Gamma\}$, has a capacity $c(e)$. Denote by $\mathcal{P}$ the set of all (periodic) paths in G and by $\mathcal{P}_{u,v}$ the set of all paths from u to v in G. A (feasible) flow in a (static or periodic) G is a function $F \colon \mathcal{P} \to \mathbb{R}^+$ such that for every edge $e \in E_i$, $i \in \Gamma$, the sum of flows traversing that edge does not exceed its capacity $c(e)$.

Throghput. Given a demand matrix $\mathcal{M}$, and a reconfigurable network modeled as a (static or periodic) directed graph $G = (V, E)$, we define the throughput as the maximum scaling factor such that, for each pair u, v of nodes, there exists a feasible flow that satisfies the scaled demand for each node pair u, v.

Definition 23 (Throughput)**.** *Let $\mathcal{M}$ be a demand matrix and $G = (V, E)$ be a (static or periodic) graph. For a feasible flow F in G, the throughput is the greatest number $\vartheta \in \mathbb{R}^+$ such that for each node pair $u, v \in \mathcal{N}$, the sum of flows on paths from u to v is at least $\vartheta \cdot m_{u,v}$, where $m_{u,v}$ is the demand. We denote by $\theta(\mathcal{M}, G)$ the maximum throughput achievable by a feasible flow. We write $\theta(\mathcal{M})$ or θ for $\theta(\mathcal{M}, G)$ whenever G and/or $\mathcal{M}$ are clear from context.*

Emulating periodic graphs. Central to our work is a result regarding the throughput-preserving conversion from static into periodic graphs (see Chapter 7). Specifically, for a given integer d, we can convert any static regular graph G into a periodic graph G_Γ of period Γ timeslots that satisfies the following properties:

- The edge set E_i, in a timeslot $i \in \{1, \ldots, \Gamma\}$, has maximum in- and out-degree d;
- there are $\Gamma = \lceil \frac{n}{d} \rceil$ timeslots in total; and
- for any static feasible flow F for G, one can compute a periodic feasible flow F_Γ for G_Γ with the same throughput.

Recall from Chapter 7 that the converse holds as well (Lemma 6 and Lemma 7), i.e., given a periodic graph, one can compute a throughput-preserving static graph. The emulated graph plays a crucial role in our throughput analysis.

9.2.2 Online Periodic Network Design

In practice, we cannot expect to know the exact demand in advance. Thus, we need to reconfigure the network before the actual demand matrix is revealed. In the *online periodic network design problem*, the algorithm is tasked with designing a periodic graph that serves the traffic demand while maximizing throughput. The algorithm can only observe prior demand and does not know the demand to be expected in advance.

Typically, the problem is intended to repeat over many rounds, and in each round the algorithm has to return a new periodic network. In this work however, we focus on the case where the problem lasts only for one round. This is well motivated in the setting where changing from one to the next periodic network comes at a fixed cost that does not depend on the amount of change between

the last and the new network. In the one-round setting (without predictions) our algorithm thus has to outpur a periodic network without any prior information. This exactly resembles the demand-oblivious setting.

Competitive ratio. A rigorous framework for designing algorithm that deal with uncertainty is the framework of *competitive analysis* [72]. In this setting, the performance of such online strategies is typically analyzed using competitive ratios, which compare the performance of an online algorithm (operating with uncertainty) to an optimal offline solution (which has full knowledge of the future demand). The goal is to minimize the worst case ratio between the online and offline solutions among all input sequences. In competitive analysis of maximization problems (recall that our objective is to maximize throughput), the goal is to maximize the competitive ratio, defined as $ALG(\sigma)/OPT(\sigma)$ for the worst-case input σ, where OPT is the performance of an optimal offline algorithm and ALG is the performance of the (deterministic) online algorithm. This definition can be extended to randomized algorithms by considering the expected performance of the algorithm, and the adversary that generates the input sequences cannot see the random choices made by the algorithm. For the online periodic network design problem (running for one round), the algorithm receives no input and is compared to the optimum. Let $\hat{\mathcal{M}}$ be the actual demand matrix, let $OPT(\hat{\mathcal{M}})$ be the periodic network that achieves the highest throughput for $\hat{\mathcal{M}}$, and let $ALG_{\hat{\mathcal{M}}}$ be the throughput achieved by the algorithm on the demand matrix (recall that the algorithm does not receive $\hat{\mathcal{M}}$ as input). Then, the algorithm is c-competitive, if, for any demand matrix $\hat{\mathcal{M}}$,

$$\frac{ALG_{\hat{\mathcal{M}}}}{OPT(\hat{\mathcal{M}})} \geq c.$$

As previously mentioned, a throughput of 1/2 can be achieved by simply outputting a complete graph (Chapter 7). Since the optimal offline solution (OPT) achieves at most a throughput of 1, this implies that a simple static online algorithm that always outputs a complete graph is 1/2-competitive. However, without additional information about the demand matrix, no deterministic online algorithm can achieve a competitive ratio better than 1/2. This limitation arises from the inherent uncertainty in online scenarios, where the network configuration is fixed before the actual traffic demand is revealed, preventing the algorithm from adapting to the specific traffic needs. Next, we show that online algorithms can indeed improve over the competitive ratio of 1/2, given that they are augmented with accurate predictions.

9.2.3 Online Periodic Network Design with Predictions

In practice, the reconfiguration of optical networks must happen without knowing the exact traffic demand in advance. However, predictions based on historical data or learned patterns can be used to improve the network's performance, particularly its throughput. The challenge then becomes how to design an algorithm that can leverage these predictions while remaining robust to potential inaccuracies.

In the offline setting, where the traffic matrix is fully known beforehand, it is possible to design a periodic network that achieves a throughput of 2/3 [18], which significantly improves upon the throughput 1/2 solution from the fully

oblivious online setting. This improvement demonstrates the value of incorporating additional information, like predictions, into the network design process.

Competitive ratio of prediction-augmented algorithms. Building on this idea, we explore online algorithms that are augmented with predictions. The predictions model [198] allows online algorithms to receive advice in the form of traffic predictions. However, these predictions can be faulty, and the performance of the algorithm depends on how well it balances reliance on this advice. The key element of this model is that the errors made by the predictor are quantified by an error function. The competitive ratio of the algorithm is then analyzed in terms of this error: when predictions are accurate, the algorithm should approach the performance of the optimal offline solution (consistency), while for large errors, it ideally should return a solution that is very close to the guarantees of a non-augmented online algorithm (robustness).

Robustness. Robustness refers to the ability of an online algorithm to maintain a satisfactory level of performance even when the predictions about the traffic demand are significantly flawed or inaccurate. In practical scenarios, the predictions can deviate substantially from the actual demand due to unforeseen changes in user behavior, network conditions, or other external factors. A robust algorithm is designed to handle such discrepancies, ensuring that its performance does not degrade excessively under adverse prediction scenarios. Ideally, a robust algorithm will still meet or exceed the performance guarantees provided by standard online algorithms, thus demonstrating its capability to operate effectively regardless of the accuracy of the predictions.

Consistency. Consistency, on the other hand, measures how closely the performance of the prediction-augmented algorithm can align with that of an optimal offline solution when the predictions are accurate. When the predicted traffic matrix closely resembles the actual demand, a consistent algorithm should achieve throughput that approaches the optimal level, maximizing resource utilization and minimizing latency. The aim of consistency is to ensure that the algorithm can leverage accurate predictions effectively, resulting in performance that is competitive with the best possible outcome that could be achieved with complete knowledge of future demands. High consistency indicates that the algorithm is well-designed to capitalize on reliable predictions, thus enhancing its overall effectiveness in network design.

In the context of online periodic network design, the algorithm receives a predicted traffic matrix before outputting the network configuration. The competitive analysis in this setting is generalized to account for the prediction error, with the aim of achieving high throughput when the predictions are accurate while maintaining a robust performance when they are not. Let $\hat{\mathcal{M}}$ be the ground truth demand matrix and let $\mathcal{M}$ be the predicted demand matrix, which our algorithm receives as input. Moreover, denote by $OPT(\hat{\mathcal{M}})$ the best achievable throughput for the actual demand matrix, and let $ALG_{\hat{\mathcal{M}}}(\mathcal{M})$ be the throughput that the algorithm achieves for the actual demand matrix $\hat{\mathcal{M}}$, when given the predicted matrix $\mathcal{M}$ as input. Then, we define the competitive ratio c of our prediction-augmented algorithm is

$$\max_{\mathcal{M}} \frac{ALG_{\hat{\mathcal{M}}}(\mathcal{M})}{OPT(\hat{\mathcal{M}})}.$$

9.3 Analysis of BvN Decomposition-Based Demand-Aware Networks

A vast majority of the literature on demand-aware networks [68, 176, 221] --- not strictly limited to periodic fixed-duration architecture --- relies on the Birkhoff-von Neumann (BvN) decomposition [66] to optimize the network topology based on an estimated demand matrix. The BvN decomposition is a well-known technique for decomposing a non-negative doubly-stochastic matrix into a convex combination of permutation matrices. In the context of demand-aware network design, the BvN decomposition (or an approximation [68]) is used to define the schedule for reconfigurations of a demand-aware network. In this section, we analyze the performance of BvN decomposition-based demand-aware networks under demand uncertainty, focusing on the robustness of the network to prediction errors.

BvN decomposition-based demand-aware networks are near-optimal under prefect predictions [68], but unsurprisingly, their performance is in fact arbitrarily inaccurate even with a small deviation from the estimated matrix.

Theorem 23 (Competitive ratio of BvN-based Demand-Aware Networks)**.** *The competitive ratio of a BvN decomposition-based demand-aware network is unbounded under imperfect predictions (robustness) i.e., there exist estimated demand matrices for which a prediction error $\epsilon > 0$ results in a competitive ratio of ∞.*

Proof. We prove our claim using a simple example. Consider the following predicted demand matrix $\mathcal{M}$ and the actual demand matrix $\hat{\mathcal{M}}$. Without loss of generality, we consider the physical network degree to be $\hat{d} = 1$ in this example.

$$\mathcal{M} = \begin{bmatrix} 0 & \mathbf{1} & 0 & 0 & \cdots & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \cdots & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & \cdots & \mathbf{1} & 0 \end{bmatrix}$$

$$\hat{\mathcal{M}} = \begin{bmatrix} 0 & 1-n\epsilon & \epsilon & \epsilon & \cdots & \epsilon & \epsilon \\ 1-n\epsilon & 0 & \epsilon & \epsilon & \cdots & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & 1-n\epsilon & \cdots & \epsilon & \epsilon \\ \epsilon & \epsilon & 1-n\epsilon & 0 & \cdots & \epsilon & \epsilon \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \epsilon & \epsilon & \epsilon & \epsilon & \cdots & 0 & 1-n\epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \cdots & 1-n\epsilon & 0 \end{bmatrix}$$

Notice that the predicted demand matrix $\mathcal{M}$ is a permutation matrix, and thus, BvN decomposition results in a single permutation matrix. Approaches that rely on BvN decomposition (e.g., Eclipse [68]) construct a graph directly from this decomposition, which, in this case, is a single permutation (matching). The resulting graph forms pairwise cycles: between node 1 and node 2, between node 3 and node 4, and so on. Consequently, the graph becomes disconnected.

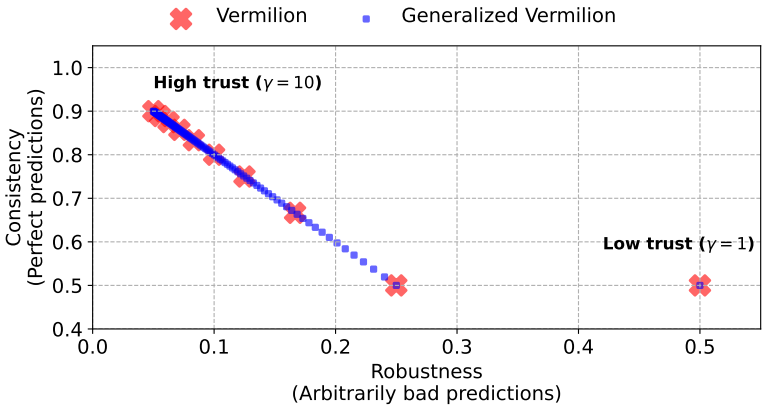


Figure 9.3: The generalization of *Vermilion* opens several points along the consistency-robustness tradeoff space, as indicated in blue. Our generalization allows γ to be a real value, while the original work restricts (necessitates) γ to be an integer value.

While this construction provides optimal throughput for a perfectly predicted matrix, the actual demand matrix $\hat{\mathcal{M}}$ may differ slightly from the predicted matrix, as illustrated above. Since the constructed graph is disconnected, any ϵ deviation in demand specified by $\hat{\mathcal{M}}$ cannot be routed, and the throughput drops to zero. As a result, the competitive ratio becomes unbounded (i.e., ∞). □

We believe that this simple insight provides a strong motivation for exploring alternative approaches to demand-aware network design that are more robust to prediction errors. Theorem 23 suggests that optimal BvN decompositions do not necessarily result in optimal demand-aware network designs and implies that sacrificing consistency for robustness is inevitable. We analyze alternative approaches in the subsequent sections.

9.4 Analysis of Vermilion with Predictions

A recent work proposes *Vermilion* as a simple alternative to BvN-based approaches for periodic and fixed-duration reconfigurable network architectures (see §9.1). The authors demonstrate a lower bound of $\frac{2}{3}$ for the throughput of *Vermilion*, surpassing the throughput bound of demand-oblivious networks. However, this result assumes perfect knowledge of the demand matrix, which is rarely the case in practice. Notably, *Vermilion* includes a parameter that balances throughput and delay, even in the offline setting. As we will show in this section, this parameter allows *Vermilion* to remain throughput-competitive under prediction errors, unlike BvN-based approaches. In this section, we analyze the performance of *Vermilion* under demand uncertainty, focusing on its consistency and robustness in the presence of prediction errors.

We generalize *Vermilion* to include a trust factor γ, relevant to the context of prediction errors. The trust factor $\gamma = \frac{k}{\ell}$ represents the ratio between the bandwidth used in a demand-aware (k) and demand-oblivious (ℓ) manner; where k, ℓ are integers and $\ell \leq k$. The previous chapter (§8) sets $\ell = 1$, but as we show later, the choice of ℓ reveals additional points along the consistency-robustness tradeoff space. We present the generalized *Vermilion* algorithm in Algorithm 7. For clarity, we highlight our changes (for generalization) in blue.

Algorithm 7: Generalized *Vermilion*

Input : Demand matrix $\mathcal{M}$, number of nodes n, degree $\hat{d}$, link capacity c, parameter k, and **parameter ℓ (generalization)**
$\gamma = \frac{k}{\ell}$ is the trust factor

```
procedure generateSchedule(M, k, ℓ):
    if γ < 2 then
        G = completeGraph(n)                          ▷ No trust
    else
        G = emulatedTopology(M, k, ℓ)                 ▷ Positive trust in predictions
    end if
    for i = 1 to k · n do                             ▷ Sequence of matchings for the periodic schedule
        remove one perfect matching M_i in G
        add M_i to schedule
    end for
    return M
procedure emulatedTopology(M, k, ℓ):
    G = (V, E), V = {1, ..., n}, E : V × V → ℕ        ▷ Initialize a multigraph
    Normalize the demand matrix M
    M ← (k − ℓ) · n · M
    R = Round(M)                                      ▷ Matrix rounding
    for each node pair (u, v) do                      ▷ # Edges between u, v
        E((u, v)) ← R(u, v)                           ▷ Allocate bandwidth for bulk demand
        E((u, v)) ← E((u, v)) + ℓ                     ▷ Allocate bandwidth for residual demand
    end for
                                                      ▷ Ensure that the final graph is regular
    x^in ← k · n − x^in                               ▷ Remaining in-degree
    x^out ← k · n − x^out                             ▷ Remaining out-degree
    G' = (V, E') = ConfigurationModel(x^in, x^out)
    E ← E ⊎ E'
    return G
```

Theorem 24 (Consistency of *Vermilion*)**.** *Suppose that the predicted input $\mathcal{M}$ perfectly matches the true demand $\hat{\mathcal{M}}$, i.e., $\mathcal{M} = \hat{\mathcal{M}}$. Then the competitive ratio of Vermilion with trust factor $\gamma \geq 1$ is lower-bounded by* $\max\left(\frac{1}{2}, \frac{\gamma-1}{\gamma}\right)$.

Proof. We prove the result by showing that the demand matrix $\mathcal{M}$ scaled by a factor of $\frac{\gamma-1}{\gamma}$ is feasible under the capacity limits of the multigraph G constructed by *Vermilion*. Without loss of generality, assume that each physical link has capacity 1 and the physical topology has a maximum degree $\hat{d}$. The emulated graph G constructed by *Vermilion* achieves a degree of $k \cdot n$, with each link having a capacity of $\frac{1}{k \cdot n}$.

In order to emulate a degree of $k \cdot n$, *Vermilion* requires a periodic schedule of length $k \cdot \frac{n}{\hat{d}}$ timeslots, as the degree in each timeslot is $\hat{d}$. Moreover, the demand matrix $\mathcal{M}$, constrained by the hose model, has the property that the sum of every row and column is at most $\hat{d}$.

We now upscale both the demand matrix $\mathcal{M}$ and the link capacities of the emulated multigraph by the factor $\frac{k \cdot n}{\hat{d}}$. Consequently, it is equivalent to analyze

the throughput of the emulated graph G, in which each node has a degree of at most $k \cdot n$ and each edge has capacity 1, under the scaled demand matrix $\mathcal{M}' = \frac{k \cdot n}{\hat{d}} \cdot \mathcal{M}$. In this scaled matrix, the sum of each row and column is at most $k \cdot n$.

Let us analyze the throughput of *Vermilion*. If $\gamma < 2$, *Vermilion* effectively constructs a complete graph, which has a throughput of $\frac{1}{2} \cdot (1 - \Delta_r)$ [18,42]. When $\gamma \geq 2$, edges are added according to matrix rounding, where the rounding process uses $\frac{(k-\ell) \cdot n}{\hat{d}} \mathcal{M} = \frac{k-\ell}{k} \mathcal{M}'$. These edges satisfy all demand except for entries that were rounded down. Since the sum of each row and column in the demand matrix (and the rounded matrix) is at most $(k - \ell) \cdot n$, entries are either rounded up or down, and links are added to non-zero entries.

The rounding procedure uses at most $(k - \ell) \cdot n$ incoming and outgoing links per node, leaving at least $\ell \cdot n$ links available for each node. The residual demand is served by adding extra edges between node pairs. Since the residual demand between any node pair is less than 1, a single additional link can fully satisfy the demand.

Due to reconfiguration, each link loses a fraction $(1 - \Delta_r)$ of its capacity, resulting in an overall throughput of $\frac{k-\ell}{k} \cdot (1 - \Delta_r) = \frac{\gamma-1}{\gamma} \cdot (1 - \Delta_r)$. An optimal offline algorithm achieves a throughput of at most $(1 - \Delta_r)$. Therefore, for $\gamma < 2$, *Vermilion* achieves a throughput of $\frac{1}{2} \cdot (1 - \Delta_r)$, yielding a competitive ratio of $\frac{1}{2}$. For $\gamma \geq 2$, *Vermilion* achieves a throughput of $\frac{\gamma-1}{\gamma} \cdot (1 - \Delta_r)$, with a competitive ratio of $\frac{\gamma-1}{\gamma}$.

Thus, the overall competitive ratio is $\max\left(\frac{1}{2}, \frac{\gamma-1}{\gamma}\right)$. □

We next turn to the robustness of *Vermilion*, which is the main concern under prediction errors. While Theorem 24 shows that the competitive ratio under perfect predictions (consistency) can be improved by increasing the trust factor γ, our next result reveals a tradeoff between consistency and robustness.

Theorem 25 (Robustness of *Vermilion*)**.** *The competitive ratio of Vermilion with trust factor $\gamma \geq 1$ is at least $\frac{1}{2 \cdot \gamma}$, no matter the quality of the prediction.*

Proof. We show that *Vermilion* achieves a throughput of at least $\frac{1}{2 \cdot \gamma} \cdot (1 - \Delta_r)$ for any predicted demand matrix $\mathcal{M}$ and actual demand matrix $\hat{\mathcal{M}}$. Consequently, the competitive ratio will be at least $\frac{1}{2 \cdot \gamma}$. First, recall that the parameter $\gamma = \frac{k}{\ell}$ represents the ratio between the number of links used in a demand-aware (k) and demand-oblivious (ℓ) manner. For any given demand matrix, *Vermilion* adds at least ℓ links between every pair of nodes, ensuring a minimum level of connectivity regardless of the specific demand pattern. Next, consider the capacity of each of these links. Since *Vermilion* operates over a periodic schedule of length $\frac{k \cdot n}{\hat{d}}$, the capacity of each link in the emulated graph is $\frac{\hat{d}}{k \cdot n}$. This implies that the ℓ links added between each pair of nodes form a complete graph, where the total capacity between each pair is $\frac{\ell}{k} \cdot \frac{\hat{d}}{n}$.

Now, observe that in a complete graph where the capacity between each pair of nodes is $\frac{\hat{d}}{n}$, it is well known that the network achieves a throughput of at least $\frac{1}{2} \cdot (1 - \Delta_r)$ for any demand matrix [18,42]. In our case, since the capacity between each pair of nodes is scaled by a factor of $\frac{\ell}{k}$ compared to this complete graph, the throughput achieved by *Vermilion* is scaled accordingly.

Thus, using the ℓ links, *Vermilion* can guarantee a throughput of at least

$$\frac{1}{2} \cdot \frac{\ell}{k} \cdot (1 - \Delta_r) = \frac{1}{2 \cdot \gamma} \cdot (1 - \Delta_r)$$

As the throughput of an optimal offline algorithm is at most $1 - \Delta_r$, *Vermilion*'s competitive ratio is

$$\frac{\frac{1}{2 \cdot \gamma} \cdot (1 - \Delta_r)}{1 - \Delta_r} = \frac{1}{2 \cdot \gamma}.$$ □

Figure 9.3 summarizes our results. In essence, the trust factor γ allows for various choices along the consistency-robusntess tradeoff space. An open question is how smoothly the competitive ratio degrades as the prediction error increases. Devising a suitable error function is particularly challenging due to the non-trivial objective of maximizing throughput. We leave it for future work to study the smoothness.

9.5 Randomized Algorithm and Tradeoff Curves

Vermilion is a deterministic algorithm and our analysis so far reveals a specific curve in the consistency-robustness tradeoff space. We now extend *Vermilion* to a randomized algorithm by introducing a probability mass function $P(R)$ that samples the trust factor r from the set $\{1, 2, \ldots, \gamma\}$. Algorithm 8 presents the randomized *Vermilion* algorithm. We deliberately set $\ell = 1$ in the randomized version of *Vermilion* and hence $\gamma = k$ is restricted to integer values. We show that the competitive ratio of the randomized *Vermilion* algorithm is a convex combination of the competitive ratios of the deterministic *Vermilion* algorithm with different trust factors, revealing a entire region of points in the consistency-robustness tradeoff space.

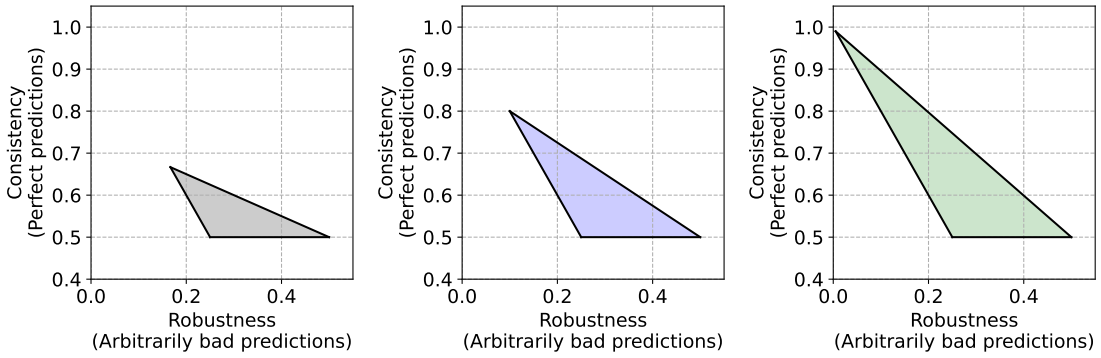
Algorithm 8: Randomized *Vermilion*

Input : Demand matrix $\mathcal{M}$, number of nodes n, degree $\hat{d}$, link capacity c, parameter $\gamma = k$, probability mass function $P(R)$

```
procedure generateSchedule(M):
    Sample r from the probability mass function P(R), where 1 ≤ r ≤ k
    G = emulatedTopology(M, r, 1)
    for i = 1 to k · n do ▷ Sequence of matchings for the periodic schedule
        remove one perfect matching M_i in G
        add M_i to schedule
    end for
    return M
```

We first analyze the consistency of the randomized *Vermilion* algorithm i.e., the competitive ratio under perfect predictions of the demand matrix $\mathcal{M} = \hat{\mathcal{M}}$.

Theorem 26 (Consistency of randomized *Vermilion*)**.** *The competitive ratio of the randomized Vermilion algorithm (RAND) under perfect predictions, for a chosen*



(a) Convex hull for $\gamma = 3$ (b) Convex hull for $\gamma = 5$ (c) Convex hull for $\gamma = 100$

Figure 9.4: The convex hulls of the competitive ratios for the randomized *Vermilion* algorithm, corresponding to various trust factors γ, reveal the entire tradeoff space. The probability mass function $P(R)$ employed by the randomized algorithm defines the points within each convex hull.

and fixed parameter γ, a probability mass function $P(R)$, is given by,

$$\min_{\mathcal{M}=\hat{\mathcal{M}}} \frac{\mathbb{E}(RAND_{\hat{\mathcal{M}}}(\mathcal{M}))}{OPT(\hat{\mathcal{M}})} \geq \sum_{r=1}^{\gamma} \Pr(R=r) \cdot \max\left(\frac{1}{2}, \frac{r-1}{r}\right)$$

Proof. We refer to the randomized *Vermilion* (Algorithm 8) as $RAND$. Our algorithm samples the trust factor r from the probability mass function $P(R)$, where $r \in \{1, \ldots, \gamma\}$. Given that the predicted demand matrix is perfect, the throughput of $RAND$ for a sampled trust factor r would then be at least $\max(\frac{1}{2}, \frac{r-1}{r}) \cdot (1 - \Delta_r)$, from Theorem 24. The expected throughput of $RAND$ can now be expressed as,

$$\mathbb{E}(RAND(\hat{\mathcal{M}})) \geq (1 - \Delta_r) \cdot \sum_{r=1}^{\gamma} \Pr(R=r) \cdot \max\left(\frac{1}{2}, \frac{r-1}{r}\right)$$

The optimal offline algorithm has a throughput of $1 - \Delta_r$. The competitive ratio of $RAND$ i.e., $\frac{\mathbb{E}(RAND)}{OPT}$ follows from the above inequality. □

Theorem 26 shows a degradation in the consistency of *Vermilion* for a chosen trust factor γ, compared to the deterministic algorithm. However, this degradation is compensated by an improvement in the robustness of the randomized algorithm, as we show next.

Theorem 27 (Robustness of randomized *Vermilion*)**.** *The competitive ratio of the randomized Vermilion algorithm (RAND) under prediction errors, for a chosen and fixed parameter γ, a probability mass function $P(R)$, is given by,*

$$\min_{\mathcal{M},\hat{\mathcal{M}}} \frac{\mathbb{E}(RAND_{\hat{\mathcal{M}}}(\mathcal{M}))}{OPT(\hat{\mathcal{M}})} \geq \frac{1}{2} \cdot \sum_{r=1}^{\gamma} \Pr(R=r) \cdot \frac{1}{r}$$

Proof. The randomized algorithm $RAND$ samples the trust factor r from the probability mass function $P(R)$, where $r \in \{1, \ldots, \gamma\}$. Given that the predicted demand matrix is imperfect, the throughput of $RAND$ for a sampled trust factor r would then be at least $\frac{1}{2 \cdot \gamma} \cdot (1 - \Delta_r)$, from Theorem 25. The expected throughput of $RAND$ can be expressed as,

$$\mathbb{E}(RAND(\hat{\mathcal{M}})) \geq (1 - \Delta_r) \cdot \sum_{r=1}^{\gamma} \Pr(R=r) \cdot \frac{1}{2 \cdot r}$$

The optimal offline algorithm has a throughput of $1 - \Delta_r$. The competitive ratio of $RAND$ i.e., $\frac{\mathbb{E}(RAND)}{OPT}$ follows from the above inequality. □

Figure 9.4 illustrates our results on the consistency and robustness of the randomized *Vermilion* algorithm. For each trust factor γ, the set of all possible probability mass functions defines a convex hull that represents the tradeoff space between consistency and robustness. Importantly, the convex hull for any parameter γ is contained within the convex hull for any larger parameter $\gamma' > \gamma$. This indicates that the entire tradeoff space can be accessed by increasing γ and appropriately selecting the probability mass function $P(R)$. We state without proof the following corollary that states the consistency and robustness of the randomized *Vermilion* algorithm for the special case of uniform probability mass function. The proof is a direct consequence of Theorem 26 and Theorem 27. Figure 9.1 illustrates the following result.

Corollary 2. *The competitive ratio of the randomized Vermilion algorithm using uniform probability mass function, i.e.,* $\Pr(R = r) = \frac{1}{\gamma}$*, for a chosen and fixed parameter* γ*, is given by,*

$$\begin{cases} 1 + \frac{1}{2\gamma} - \frac{H_\gamma}{\gamma} & \textit{Perfect predictions (consistency)} \\ \frac{H_\gamma}{2 \cdot \gamma} & \textit{Large prediction error (robustness)} \end{cases}$$

where H_γ *is the* γ*-th harmonic number.*

Overall, the randomized *Vermilion* algorithm offers a wide range of points in the consistency-robustness tradeoff space, providing a flexible solution for demand-aware networks with prediction errors.

9.6 Evaluation

We evaluate the throughput of the *Vermilion* algorithm under prediction errors. Our evaluation aims at answering the following main questions:

(Q1) *How does the throughput of Vermilion vary with prediction errors?*

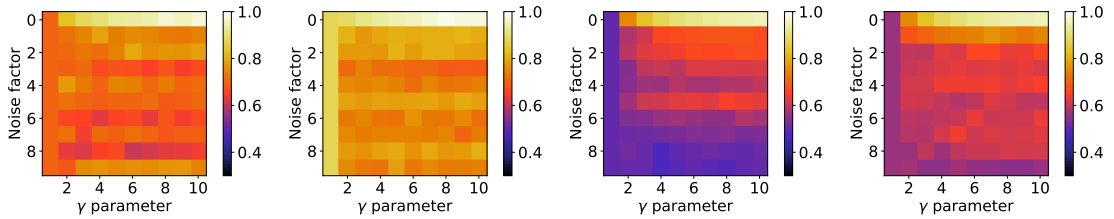
We observe that the throughput of *Vermilion* indeed smoothly degrades as the error in the predicted demand matrix increases. The throughput of *Vermilion* is robust to prediction errors, as the competitive ratio remains bounded by $\frac{1}{2\gamma}$, as suggested by Theorem 27.

(Q2) *Does the throughput of Vermilion match its theoretical bounds for consistency and robustness?*

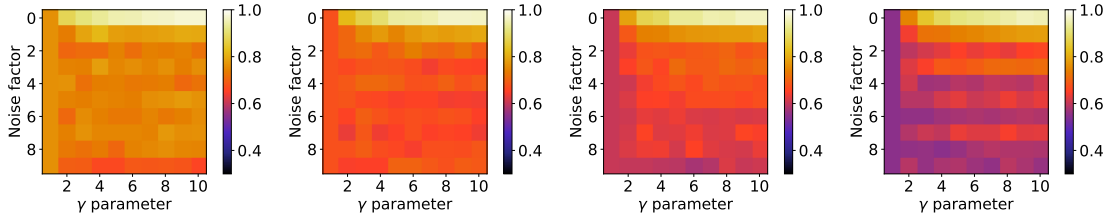
We find that the throughput of *Vermilion* is within the theoretical bounds for both consistency and robustness. Across various demand matrices and errors, we find that the throughput of *Vermilion* is significantly better than the theoretical lower bounds we obtained in the previous sections, indicating that our bounds can potentially be improved.

(Q3) *What is the impact of the trust factor* γ *on the throughput of Vermilion?*

We observe that the trust factor γ indeed plays a crucial role in determining the throughput of *Vermilion*. A higher trust factor γ results in a higher throughput,

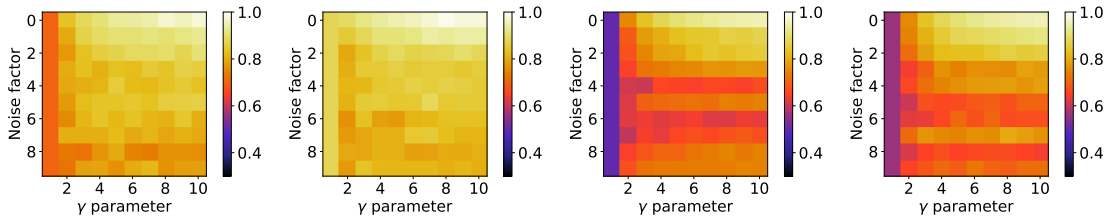


(a) Chessboard (b) All-to-All (c) Ring (d) Random Skewed

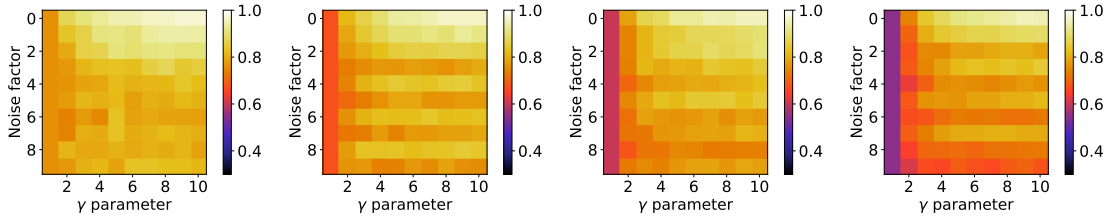


(e) Skewed $\alpha = 0.2$ (f) Skewed $\alpha = 0.4$ (g) Skewed $\alpha = 0.6$ (h) Skewed $\alpha = 0.8$

Figure 9.5: Throughput of deterministic *Vermilion* under additive noise for the predicted demand matrix across various demand matrices and trust factors γ. Additive noise changes the structure as well as the magnitude of demands in the demand matrix.



(a) Chessboard (b) All-to-All (c) Ring (d) Random Skewed



(e) Skewed $\alpha = 0.2$ (f) Skewed $\alpha = 0.4$ (g) Skewed $\alpha = 0.6$ (h) Skewed $\alpha = 0.8$

Figure 9.6: Throughput of deterministic *Vermilion* under multiplicative noise for the predicted demand matrix across various demand matrices and trust factors γ. Multiplicative noise preserves the sparsity of the real demand matrix.

as suggested by Theorem 26. However, the throughput degrades as the error increases, as indicated by Theorem 27.

(Q4) *How does the randomized Vermilion algorithm perform under prediction errors?*

Our results show that the throughput of the randomized *Vermilion* algorithm is better than the deterministic algorithm by a small margin, and also more robust to errors in the predicted demand matrix.

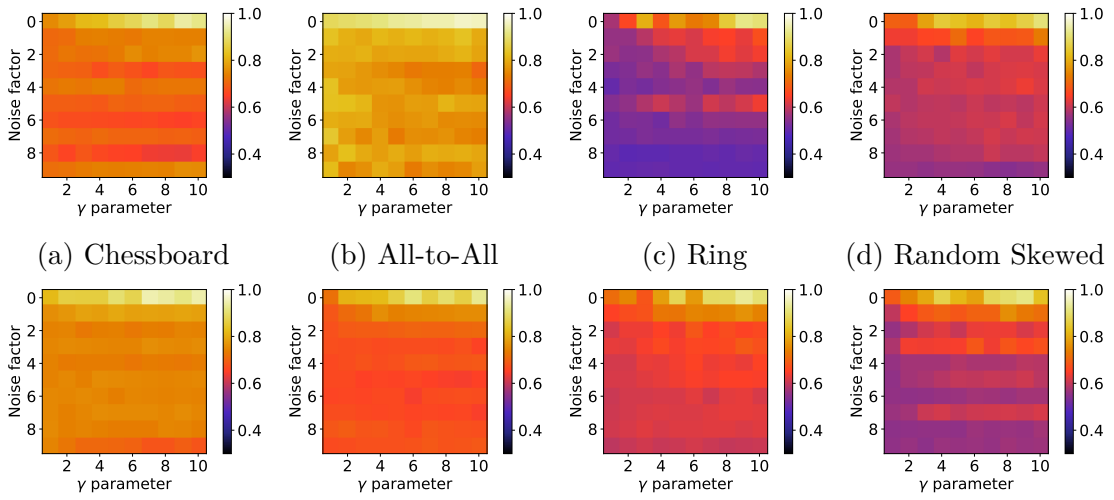
9.6.1 Setup

Our evaluation is based on a linear-programming approach. We use the Gurobi optimizer [117] to solve the linear program for the throughput of *Vermilion*. We consider a topology of 8 nodes with physical degree 4. Our topology matches the scale of a DGX H100 server, consisting of 8 GPUs interconnected together. We set link capacities to 400Gbps and a reconfiguration delay of $1\mu s$ and a timeslot value of $9.9\mu s$ (a duty cycle of 0.99), within the practical capabilities of existing technologies [60]. We consider various demand matrices: Ring and All-to-All demand matrices relevant to collective communication workloads, and Random and Skewed demand matrices that represent a combination of ring and permutation communication patterns. Specifically, we parametrize the skewed matrices with a skew factor α, where $\alpha = 0$ corresponds to a ring matrix and $\alpha = 1$ corresponds to a permutation matrix. Additionally, we consider chessboard demand matrix that specifies alternating 1.5 and 0.5 units of demand between pairs of nodes, that stresses the demand-awareness of the algorithm due to degree constraints. We consider two types of errors: additive and multiplicative errors sampled from a normal distribution. We indicate the standard deviation of the noise by ‘‘Noise factor’’ in our evaluation, ranging from 0 (no error) to 10 (maximum demand). We vary the trust factor γ from 1 to 10 for both the deterministic and randomized algorithms. We consider a uniform probability mass function for the randomized *Vermilion* algorithm. We set the number of samples to 100 for each demand matrix and error type.

9.6.2 Results

Throughput under perfect predictions. Figure 9.5, 9.6 show the throughput of the deterministic *Vermilion* algorithm under additive and multiplicative noise, respectively. Particularly at noise factor 0, the predicted matrix matches the real demand matrix. We observe that the throughput of *Vermilion* is within (better than) the theoretical bounds for consistency. Further, we observe that the throughput gradually increases with the trust factor γ, as expected. Our results show an interesting trend; as the skew in the demand matrix increases (from skew $\alpha = 0.2$ to Ring), the throughput of *Vermilion* decreases close to the lower bound of $\frac{\gamma-1}{\gamma}$ for each γ, even with zero error in the predicted demand matrix. This indicates that the structure of the demand matrix plays a significant role in the throughput of *Vermilion*, beyond its lower bound guarantee.

Throughput under prediction errors. As the prediction error increases, indicated by the noise factor, we observe that the throughput of *Vermilion* gradually decreases. From Figure 9.5, 9.6, we observe that the throughput of *Vermilion* is



(a) Chessboard (b) All-to-All (c) Ring (d) Random Skewed

(e) Skewed $\alpha = 0.2$ (f) Skewed $\alpha = 0.4$ (g) Skewed $\alpha = 0.6$ (h) Skewed $\alpha = 0.8$

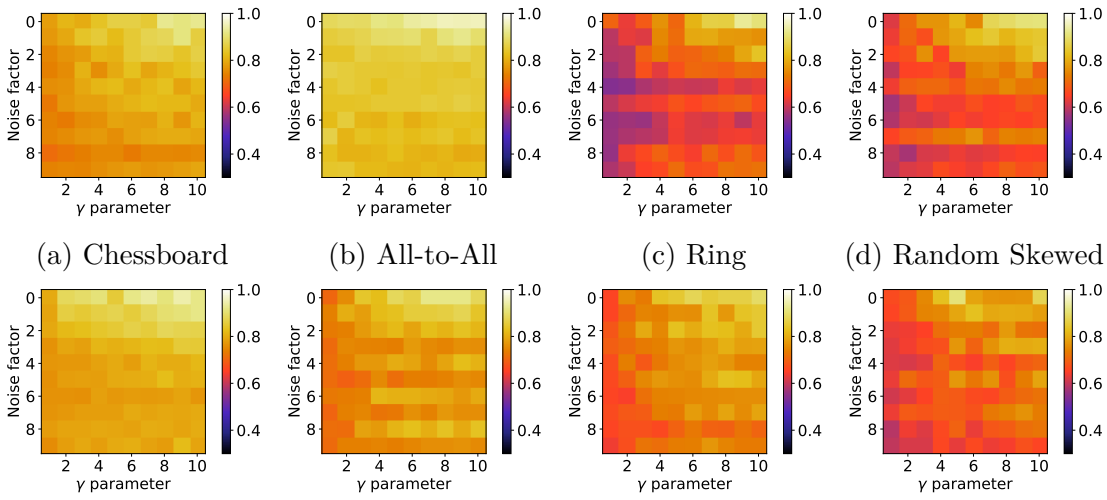
Figure 9.7: Throughput of randomized *Vermilion* under additive noise for the predicted demand matrix across various demand matrices and trust factors γ, using uniform probability mass function. Additive noise changes the structure as well as the magnitude of demands in the demand matrix.

robust to prediction errors, as the throughput is at least 0.47 — significantly better than the lower bound of $\frac{1}{2\cdot\gamma}$ — across various demand matrices, as suggested by Theorem 27. Figure 9.5, 9.6 highlight that structural changes in the demand matrix caused by additive noise has a greater impact on throughput compared to multiplicative noise that remains structurally close to the real demand matrix. In particular, We observe a similar trend for the randomized *Vermilion* algorithm, as shown in Figure 9.7, 9.8.

Impact of the trust factor γ. We observe that the throughput of *Vermilion* increases with the trust factor γ, as expected. Even when the noise increases, we observe from Figure 9.5, 9.6 that the throughput of *Vermilion* does not degrade drastically. Further, the throughput with increasing noise rather smoothly degrades even for high trust factors. Although our theoretical analysis does not include smoothness of *Vermilion*, our empirical results provide evidence for smoothness in the competitive ratio of *Vermilion* as the prediction error increases. Across various noise factors and demand matrices, the throughput of *Vermilion* is within the theoretical bounds for robustness, a minimum of 0.47 under additive noise and a minimum of 0.51 under multiplicative noise, as shown in Figure 9.5, 9.6.

Randomized *Vermilion* algorithm. Compared to the deterministic algorithm (Figure 9.5, 9.6), the randomized algorithm (Figure 9.7, 9.8) performs better under prediction errors. We observe that the throughput of the randomized algorithm is more robust to errors in the predicted demand matrix, balancing the throughput between zero error and large error scenarios. The highest throughput (zero noise) achieved by the randomized algorithm under Ring demand matrix is 0.91 (compared to 0.95 for deterministic), whereas the minimum throughput (large noise) is 0.49 (compared to 0.47 for deterministic).

Overall, our empirical results corroborate our theoretical results on the throughput of *Vermilion* under prediction errors. Unlike BvN decomposition-based approaches (see §9.3), *Vermilion* is robust to prediction errors and provides a flexible solution for demand-aware networks. In the design of demand-aware networks,



(a) Chessboard (b) All-to-All (c) Ring (d) Random Skewed

(e) Skewed $\alpha = 0.2$ (f) Skewed $\alpha = 0.4$ (g) Skewed $\alpha = 0.6$ (h) Skewed $\alpha = 0.8$

Figure 9.8: Throughput of randomized *Vermilion* under multiplicative noise for the predicted demand matrix across various demand matrices and trust factors γ, using uniform probability mass function. Multiplicative noise preserves the sparsity of the real demand matrix.

balancing the throughput between zero error and large error scenarios by allocating a fraction of bandwidth in a demand-oblivious manner is the key to ensure robustness to prediction errors.

9.7 Related Work

This work is situated at the intersection of network topologies, optical circuit-switched networks, online algorithms with predictions, and network flow problems. We provide a brief overview of related work in each of these areas, especially in the context of datacenter and reconfigurable networks.

Topologies. The study of datacenter topologies has received significant attention in the literature [31, 65, 113, 115, 179, 246, 248, 256, 281]. Various key metrics have been explored in the design and evaluation of these topologies, such as ensuring high and uniform bandwidth availability [31, 115], network expansion properties [248, 256], fault tolerance [179], life cycle management [281]. More recently, topologies tailored for collective communication [266, 285], rail-optimized topologies [265], topologies tailored for large-scale distributed training [29, 128] have also been studied.

Optical circuit-switched networks. The near-end of Moore's law led to major research efforts on optical circuit-switched network in the recent past [60, 68, 79, 81, 98, 104, 109, 114, 121, 148, 182, 187, 189, 234, 249, 261, 264, 279, 286]. These networks are reconfigurable, allowing for dynamic allocation of bandwidth between nodes. The design of these networks has focused on minimizing the reconfiguration overhead [60, 188, 189], minimizing the bandwidth tax [98, 109, 266, 279], maximizing throughput [18, 40, 42], minimizing demand completion time [68, 176, 221], expected path lengths [54, 214]. More recently, reconfigurable networks have been shown to improve the performance of distributed training workloads [266].

Network flow problems. Multicommodity flow [106, 168, 169] is a classic problem in graph theory with various applications in real-world networks, even

in the context of emerging distributed training workloads in datacenters [180]. Throughput is a variant of the multicommodity flow problem, referred as maximum concurrent flow problem in the literature [238]. Recent works on topologies have focused on throughput in order to understand the maximum load supported by a topology [18, 114, 145, 204, 247]. Understanding the throughput bounds of network topologies has been an active research area in the recent past, not only for static datacenter topologies [145, 204, 247], but also in the context of optical reconfigurable networks [18, 42, 114]. Majority of works in the literature on demand-aware reconfigurable networks, including *Vermilion*, analyze throughput in an offline setting i.e., the demand is known in advance. Our work is the first to study the throughput of a demand-aware network under prediction errors.

Online algorithms with predictions. The predictions model, introduced by Mitzenmacher and Vassilvitskii [198], enhances traditional online algorithms by incorporating advice that may not always be accurate. The literature on this predictions model covers a wide range of online problems, such as paging [62, 142, 184, 226], metrical task systems [43], and online set cover [61], among others, as aggregated in [39]. More recently, the predictions model has also been applied to networking problems such as buffer sharing [24], and dynamic acknowledgement [135]. The competitive analysis framework for these prediction-augmented algorithms builds on the traditional competitive analysis for online algorithms [72], but is generalized to parameterize the competitive ratio based on the predictor's error. Mitzenmacher and Vassilvitskii [198] formalized the competitive ratio for such prediction-augmented algorithms, specifically for minimization problems. Our work builds on this framework and introduces the demand-aware network design problem under prediction errors, with throughput as the objective, a maximization problem.

9.8 Discussion

This work initiates the study of demand-aware network design under demand uncertainty, focusing on throughput as the key performance metric. We briefly discuss the limitations of our initial results and outline future research directions.

9.8.1 Limitations

Various parts of our model can be further refined to better capture the real-world scenario. Certain well-known limitations are inherent to our model and analysis framework.

Throughput objective. Several works in the past have considered throughput as a metric for network topologies [51, 106, 238]. Implicitly, the throughput maximization problem takes a demand matrix as input that specifies constant demands between node pairs. This model is a simplification of the real-world scenario, where the demands are dynamic and may change over time. Further, demands between node pairs is comprised of multiple flows, each with different size and priorities. Our model does not capture these aspects of the real-world scenario. Despite these limitations, throughput objective provides the worst-case performance guarantee of a topology under an ideal protocol stack and provides a way to quantitatively

compare different network topology designs.

Competitive ratio. By definition, the competitive ratio is the worst-case performance guarantee of an online algorithm compared to an offline algorithm [72]. Competitive analysis is a standard approach to evaluate online algorithms, but worst-case scenarios may not necessarily be of concern in real-world networks. For instance, in datacenter networks, the demand patterns are often predictable and can be learned from historical data. Moreover, the communication patterns are typically not the worst-case ones, making the average-case performance [229, Ch. 8-12] more relevant to the operators. Our model and analysis incorporate predictions (which may be imperfect) [198], taking a step forward in addressing these limitations and providing a more realistic evaluation of network design under both predictable and uncertain communication patterns.

Latency and Buffers. Particularly in the context of dynamic reconfigurable topologies, latency and buffer requirements are crucial aspects that are not considered in our model. Recent works shows the inherent tradeoffs in demand-oblivious periodic networks in terms of throughput, latency and buffer requirements [18]. We leave it for future work to study these tradeoffs in demand-aware periodic networks.

Routing and congestion control. Our throughput analysis assumes ideal routing and congestion control, a typical assumption in network flow problems [145, 169, 204, 247]. In practice, the achievable throughput of a topology critically depends on routing and congestion control algorithms that are not ideal e.g., Equal Cost Multipath Routing (ECMP), TCP congestion control. We leave it for future work to empirically study the impact of routing and congestion control on the performance of demand-aware networks in terms of more practical metrics such as link utilization, flow completion times, buffer requirements and packet loss rates.

9.8.2 Open Questions

Our work opens up several interesting research directions that can be explored in future work. We briefly describe the key open questions below.

Smoothness. We focused primarily on the competitive ratio under perfect predictions (consistency) and arbitrarily inaccurate prediction errors (robustness). Analyzing the smoothness of the competitive ratio with respect to the prediction error is an important open question. However, expressing the competitive ratio as a function of the prediction error is a challenging task in our model, particularly with the complex objective of throughput maximization. Devising a suitable error function that captures the prediction error and quantifies the competitive ratio is a key research direction.

Worst-case demand matrix. While our analysis provides lower bounds on the throughput of *Vermilion*, it remains an open question whether these bounds are tight. Our empirical results indicate that *Vermilion* achieves significantly higher throughput than the theoretical lower bounds suggest. In this context, characterizing the worst-case demand matrix could lead to improved *upper* bounds on throughput and help determine the tightness of the lower bounds. Recent studies have shown that a permutation demand matrix is the worst-case for demand-oblivious networks [18, 42, 114]. However, in demand-aware networks, the permutation demand matrix does not serve as the worst-case scenario, as constructing a ring

topology that aligns with the permutation demand matrix results in optimal throughput. Characterizing the worst-case demand matrices for demand-aware networks is an interesting research direction that could yield better performance bounds.

Switching costs. Updating the switching schedule in a reconfigurable network may incur a cost, which is not considered in our model. In fact, our model boils down to comparing the performance of an algorithm with a single predicted demand matrix and corresponding real demand matrix, mainly because we do not consider switching costs. The cost of reconfiguration is a crucial aspect that can significantly impact the performance of demand-aware networks. In particular, the optimal offline algorithm may also be non-trivial with switching costs, unlike our setting where the optimal offline solution achieves an ideal throughput of close to 1. We leave it for future work to study the impact of the costs incurred in updating the periodic schedule on the competitive ratio of demand-aware networks.

Pareto-optimal solutions. Our analysis of a generalization of *Vermilion* and its randomized version reveal several points in the consistency-robustness tradeoff space. However, it remains an open question whether *Vermilion* is already pareto-optimal in this tradeoff space. In other words, are there other *feasible* points in the tradeoff space that can outperform *Vermilion* in both consistency and robustness? We leave it for future work to explore this question.

9.9 Summary

We analyzed *Vermilion*, a recent demand-aware optical interconnect that delivers high throughput using periodic circuit-switching and direct communication. Our study is the first to consider the demand-aware network design problem under demand uncertainty with throughput maximization objective — specifically, when the network's demands are based on machine-learned predictions that may be erroneous compared to actual demands.

Our theoretical analysis shows that *Vermilion* performs exceptionally well when the predictions align with actual demands, reaching close to the optimal throughput. Even when predictions are imperfect, *Vermilion* stays within a constant factor of the best possible performance. We complement our theoretical results with empirical evidence, suggesting that *Vermilion* is a practical design for optically circuit-switched datacenter networks with predictable traffic patterns.

Looking ahead, we see exciting directions for future research. First, we aim to study how *Vermilion*'s competitive ratio changes as prediction errors increase, and how smoothly performance degrades. Second, we plan to explore how *Vermilion* can adapt dynamically when new demand predictions arrive over time. This is particularly challenging, as frequent changes to the switching schedule can be costly, but addressing these challenges could unlock the throughput potential of demand-aware optical networks.

10

Future Research Directions

The various techniques and methodologies developed in this thesis lay the foundation and strengthen the problem-solving skills required to address some of the most challenging problems emerging in the field of networking, particularly in new use cases such as large-scale distributed training and inference clusters.

The massive scale of investment and the growing economic value of datacenters in recent times demand a paradigm shift in the design of network protocols and algorithms. The traditional best-effort paradigm is no longer sufficient to meet the stringent requirements of this critical infrastructure, especially in the case of dedicated GPU clusters. Traditional datacenters with general-purpose processors served a variety of workloads, often in a best-effort manner, as traffic patterns largely remained unpredictable. Consequently, much of the research — spanning congestion control, topologies, load balancing, and routing — has focused on designing protocols and algorithms to handle this unpredictability, often at the expense of performance. However, two key shifts are now underway. First, emerging GPU clusters are designed for specific purposes, such as AI/ML training workloads, which generate predictable traffic patterns and therefore require a fundamentally different approach. Second, new technologies such as optical circuit switches enable not only traffic engineering but also topology engineering, allowing the network topology itself to be optimized for specific traffic patterns. As a result, network infrastructure is evolving into a collection of specialized clusters, each tailored to the demands of specific workloads.

In the following, we outline the future research directions building upon the material presented in this thesis.

10.1 Systems and Infrastructure for AI/ML Workloads

Leveraging Predictability (and Mutability) of Traffic Patterns to Design Network Protocols

In contrast to traditional datacenters, large-scale GPU clusters exhibit predictable traffic patterns. Workloads are pre-determined, and the traffic patterns are explicit due to the nature of collective communication in distributed training. The traffic matrices generated by common collectives are mutable. For example, an allReduce operation results in a uniform traffic matrix when using an all-to-all algorithm, or a permutation matrix with a ring algorithm. This mutability can be exploited to optimize network performance. Recent research has proposed alternative collective communication algorithms that adapt the traffic matrix

based on various objectives, often without considering the underlying topology or assuming ideal network conditions.

However, the state-of-the-art protocol suite, designed with a best-effort philosophy for commodity hardware, does not provide ideal network conditions; network performance is influenced by topology, routing, and congestion control. For instance, a decade of research on optimizing congestion control, load-balancing, scheduling, buffer-sharing, still leave surprising and unique problems under collective communication in GPU clusters: while some communication patterns (e.g., point-to-point) perform well under specific protocols, others suffer (e.g., all-to-all) i.e., the best-effort philosophy of designing ''one ring to rule them all'' does not hold anymore.

Future research work is needed to bridge this gap by co-optimizing collective communication algorithms with underlying network protocols, particularly routing, load balancing, and congestion control, to maximize performance. This thesis takes the first steps in this direction. Credence (Chapter 6) exploits the predictability of traffic patterns and improves throughput when multiple congestion hotspots contend for commonly shared buffer space. Ethereal (Chapter 3) challenges the need for packet spraying in GPU clusters and shows that the properties of the underlying traffic matrices can be exploited to optimize network performance. Several open questions remain, such as: How can we design network protocols that exploit the mutability of traffic matrices? How can we co-optimize collective communication algorithms with underlying network protocols to maximize performance?

Self-Adjusting Optical Interconnects for GPU Clusters

Emerging technologies using optical circuit switches enable network infrastructures to adapt their topology and capacity provisioning based on observed traffic patterns, marking a shift from traditional datacenter network designs. Conventional topologies like the fat-tree are built to offer high bisection bandwidth and low latency for unpredictable traffic patterns. In contrast, today's GPU clusters generate predictable and often structured traffic, which opens opportunities to optimize network topology and capacity provisioning for performance and cost efficiency. In the context of the post-Moore's law era, developing high-performance optical interconnects is essential, supporting a transition toward optical computing.

Chapters 7, 8, and 9 on reconfigurable networks are foundational steps in understanding the trade-offs between reconfigurability and performance. Several interesting questions remain open: How can we co-optimize dynamic network topologies and collective communication algorithms to maximize performance? What are the strategies for building failure-resilient optical interconnects? How can we achieve fully automated control plane management for optical interconnects, minimizing manual intervention?

Transport Protocol for GPU clusters

Congestion control and load balancing are increasingly critical in the context of AI/ML workloads, as underscored by the focus from hyperscale datacenter operators, including the Ultra Ethernet Consortium [13].

Building upon Chapters 2, 3, future research could develop a transport protocol that can adapt dynamically to AI/ML traffic patterns, without necessitating costly new hardware, enabling high-performance solutions for both hyperscale datacenters and emerging HPC clusters at universities. Further, adaptive routing algorithms

is an interesting direction in order to tackle the challenges of load balancing under link failures.

10.2 Learning-Augmented Systems

Learning-augmented algorithms [39] are a hot-topic within the theory community, demonstrating promising results across various fields. These algorithms offer a hybrid approach: they provide near-optimal performance when predictions are accurate, while still ensuring robust performance even when predictions are far from perfect. This duality is particularly appealing for networked systems, where reliability and explainable performance are essential. However, integrating learning-augmented algorithms into networked systems remains nascent, with current models often requiring high prediction accuracy to deliver strong performance.

Future research work could advance this area by designing learning-augmented networked systems capable of delivering formal guarantees, not only under high prediction accuracy but also when faced with inaccuracies. Chapter 6 represents a foundational step, showing that learning-augmented buffer-sharing algorithms can maintain throughput guarantees despite imperfect predictions. Building upon similar concepts, learning-augmented packet classification, scheduling, and load balancing, are some of the interesting future research directions in the context of networked systems.

11

Conclusion

In this thesis, we explored the design of high-performance protocols and algorithms for datacenter networks, focusing on three key challenges: *(i)* transport protocols, *(ii)* switch buffer sharing, and *(iii)* reconfigurable optical interconnects. For each area, we proposed novel solutions and answered the core research questions raised in Chapter 1.

We addressed the design of congestion control and load balancing schemes for datacenter transport. In response to Question 1, we presented PowerTCP, a congestion control algorithm that leverages in-band network telemetry to react instantaneously to network conditions. PowerTCP maintains near-zero queue lengths while sustaining high throughput, even under bursty workloads and dynamic topologies. To address Question 2, we proposed Ethereal, a load balancing algorithm tailored to distributed training workloads in GPU clusters. Ethereal achieves near-optimal load distribution without requiring hardware changes, by exploiting the structured and predictable nature of collective communication patterns.

We studied how to manage limited buffer resources in shallow-buffered datacenter switches. In response to Question 3, we proposed ABM and Reverie, buffer management schemes that provide fairness, isolation, and predictable burst absorption while remaining compatible with current hardware. To address the more fundamental Question 4, we introduced Credence, a drop-tail buffer sharing algorithm augmented with machine-learned predictions. Credence approximates the performance of an ideal buffer sharing algorithm with full foresight and gracefully degrades under imperfect predictions, thereby bridging the gap between theory and practice.

We examined the design of reconfigurable topologies for optical datacenter networks. In response to Question 5, we introduced Mars, a periodic demand-oblivious topology that explicitly accounts for delay and buffer constraints. Mars uncovers a rich design space beyond complete graph emulation and reveals inherent tradeoffs between throughput, delay, and buffer requirements. To answer Question 6, we developed Vermilion, a demand-aware reconfigurable interconnect that breaks the throughput limitations of traditional periodic designs, achieving higher worst-case throughput while simplifying routing to direct, single-hop communication. Finally, in response to Question 7, we extended Vermilion with predictive capabilities using machine-learned traffic forecasts. We formally characterized its robustness under demand uncertainty and demonstrated empirically that it maintains significant throughput gains even in the presence of prediction errors.

Overall, this thesis shows that key limitations in modern datacenter networks can be addressed by combining systems insights with carefully designed algorithms and, where appropriate, learning-based techniques. Across transport protocols, switch buffer management, and reconfigurable topologies, we presented practical

solutions grounded in theoretical analysis — paving the way for the design of more adaptive, scalable, and high-performance datacenter networks.

References

[1] Arista 7060CX-32 and 7260CX-64. https://people.ucsc.edu/~warner/Bufs/7060CX.html.

[2] Arista lanz overview. https://www.arista.com/assets/data/pdf/Whitepapers/Arista_LANZ_Overview_TechBulletin_0213.pdf.

[3] Braodcom Tomahawk. https://people.ucsc.edu/~warner/Bufs/tomahawk.

[4] Broadcom. 2020. 25.6 tb/s strataxgs tomahawk 4 ethernet switch series. URL: https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series.

[5] Cisco nexus 9000 series switches. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-739134.html.

[6] Gpt-4. https://openai.com/index/gpt-4/.

[7] Introduction to spectrum, the 100gbe switch silicon. https://www.cisco.com/c/en/us/support/docs/switches/nexus-3500-series-switches/118904-technote-nexus-00.html.

[8] Nccl environment variables. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html.

[9] Nexus 3500 output drops and buffer qos. https://community.mellanox.com/s/article/introduction-to-spectrum--the-100gbe-switch-silicon.

[10] Nvidia collective communications library (nccl). https://developer.nvidia.com/nccl.

[11] opera-Sim. https://github.com/TritonNetworking/opera-sim.

[12] Trident2 / BCM56850 Series, High-Capacity StrataXGS® Trident II Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56850-series.

[13] Ultra ethernet consortium. https://ultraethernet.org/.

[14] Advanced Congestion and Flow Control with Programmable Switches, 2011. https://opennetworking.org/wp-content/uploads/2020/04/JK-Lee-Slide-Deck.pdf.

[15] Broadcom. 2020. 25.6 tb/s strataxgs tomahawk 4 ethernet switch series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series.

[16] Vamsi Addanki, Maria Apostolaki, Manya Ghobadi, Stefan Schmid, and Laurent Vanbever. Abm: Active buffer management in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 36–52, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3544216.3544252.

[17] Vamsi Addanki, Chen Avin, Goran Dario Knabe, Giannis Patronas, Dimitris Syrivelis, Nikos Terzenidis, Paraskevas Bakopoulos, Ilias Marinos, and Stefan Schmid. Vermilion: A traffic-aware reconfigurable optical interconnect with formal throughput guarantees. *arXiv preprint arXiv:2504.09892*, 2025. URL: https://arxiv.org/abs/2504.09892.

[18] Vamsi Addanki, Chen Avin, and Stefan Schmid. Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(1), mar 2023. doi:10.1145/3579312.

[19] Vamsi Addanki, Wei Bai, Stefan Schmid, and Maria Apostolaki. Reverie: Low pass Filter-Based switch buffer sharing for datacenters with RDMA and TCP traffic. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 651--668, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/addanki-reverie.

[20] Vamsi Addanki, Prateesh Goyal, Ilias Marinos, and Stefan Schmid. Ethereal: Divide and conquer network load balancing in large-scale distributed training. *arXiv preprint arXiv:2407.00550*, 2025. URL: https://arxiv.org/abs/2407.00550.

[21] Vamsi Addanki, Oliver Michel, and Stefan Schmid. PowerTCP: Pushing the performance limits of datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 51--70, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/addanki.

[22] Vamsi Addanki, Maciej Pacut, Leon Kellerhals, Goran Dario Knabe, and Stefan Schmid. Vermilion, pt. 2: Tradeoffs between throughput and prediction accuracy in reconfigurable optical interconnects. 2025.

[23] Vamsi Addanki, Maciej Pacut, Arash Pourdamghani, Gabor Rétvári, Stefan Schmid, and Juan Vanerio. Self-adjusting partially ordered lists. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pages 1--10, 2023. doi:10.1109/INFOCOM53939.2023.10228937.

[24] Vamsi Addanki, Maciej Pacut, and Stefan Schmid. Credence: Augmenting datacenter switch buffer sharing with ML predictions. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 613--634, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/addanki-credence.

[25] Saksham Agarwal, Rachit Agarwal, Behnam Montazeri, Masoud Moshref, Khaled Elmeleegy, Luigi Rizzo, Marc Asher de Kruijf, Gautam Kumar, Sylvia Ratnasamy, David Culler, and Amin Vahdat. Understanding host

interconnect congestion. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, HotNets '22, page 198–204, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3563766.3564110.

[26] Saksham Agarwal, Qizhe Cai, Rachit Agarwal, David Shmoys, and Amin Vahdat. Harmony: A congestion-free datacenter architecture. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 329--343, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/agarwal-saksham.

[27] Saksham Agarwal, Arvind Krishnamurthy, and Rachit Agarwal. Host congestion control. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 275–287, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3603269.3604878.

[28] Krishna Agrawal, Vamsi Addanki, and Habib Mostafaei. Dequeue rate-agnostic switch buffer sharing through packet queueing delay. In *Proceedings of the CoNEXT on Student Workshop 2024*, CoNEXT-SW '24, page 1–2, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3694812.3699924.

[29] Yuichiro Ajima, Takahiro Kawashima, Takayuki Okamoto, Naoyuki Shida, Kouichi Hirai, Toshiyuki Shimizu, Shinya Hiramoto, Yoshiro Ikeda, Takahide Yoshikawa, Kenji Uchida, and Tomohiro Inoue. The tofu interconnect d. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 646--654, 2018. doi:10.1109/CLUSTER.2018.00090.

[30] Aristide Tanyi-Jong Akem, Michele Gucciardo, and Marco Fiore. Flowrest: Practical flow-level inference in programmable switches with random forests. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pages 1--10, 2023. doi:10.1109/INFOCOM53939.2023.10229100.

[31] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, page 63–74, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1402958.1402967.

[32] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*, San Jose, CA, April 2010. USENIX Association. URL: https://www.usenix.org/conference/nsdi10-0/hedera-dynamic-flow-scheduling-data-center-networks.

[33] Mohammad Alizadeh and Tom Edsall. On the data path performance of leaf-spine datacenter fabrics. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pages 71--74, 2013. doi:10.1109/HOTI.2013.23.

[34] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan,

Navindra Yadav, George Varghese, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 503--514. ACM, 2014.

[35] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 63–74, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1851182.1851192.

[36] Mohammad Alizadeh, Adel Javanmard, and Balaji Prabhakar. Analysis of dctcp: stability, convergence, and fairness. *SIGMETRICS Perform. Eval. Rev.*, 39(1):73–84, June 2011. doi:10.1145/2007116.2007125.

[37] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 435–446, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2486001.2486031.

[38] Hamidreza Almasi, Rohan Vardekar, and Balajee Vamanan. Protean: Adaptive management of shared-memory in datacenter switches. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pages 1--10, 2023. doi:10.1109/INFOCOM53939.2023.10229046.

[39] Algorithms with predictions. https://algorithms-with-predictions.github.io/.

[40] Daniel Amir, Nitika Saran, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. Shale: A practical, scalable oblivious reconfigurable network. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 449–464, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672248.

[41] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, and Robert Kleinberg. Poster: Scalability and congestion control in oblivious reconfigurable networks. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 1138–1140, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3603269.3610862.

[42] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal. Optimal oblivious reconfigurable networks. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1339–1352, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520020.

[43] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 345--355. PMLR, 2020.

[44] Antonios Antoniadis, Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Breaking the Barrier Of 2 for the Competitiveness of Longest Queue Drop. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1--17:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl -- Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14086, doi:10.4230/LIPIcs.ICALP.2021.17.

[45] Maria Apostolaki, Vamsi Addanki, Manya Ghobadi, and Laurent Vanbever. Fb: A flexible buffer management scheme for data center switches. *arXiv preprint arXiv:2105.10553*, 2021. URL: https://arxiv.org/abs/2105.10553.

[46] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. Fab: Toward flow-aware buffer sharing on programmable switches. BS '19, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3375235.3375237.

[47] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. SNAP: Stateful network-wide abstractions for packet processing. In *SIGCOMM '16*. ACM, 2016.

[48] Jay E Aronson. A survey of dynamic network flows. *Annals of Operations Research*, 20(1):1--66, 1989.

[49] M. Arpaci and J. A. Copeland. Buffer management for shared-memory atm switches. *IEEE Communications Surveys Tutorials*, 3(1):2--10, First 2000. doi:10.1109/COMST.2000.5340716.

[50] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. Bolt: Sub-RTT congestion control for Ultra-Low latency. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 219--236, Boston, MA, April 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/arslan.

[51] Yonatan Aumann and Yuval Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291--301, 1998.

[52] Yonatan Aumann and Yuval Rabani. An o(log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291--301, 1998. doi:10.1137/S0097539794285983.

[53] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(1), May 2020. doi:10.1145/3379486.

[54] Chen Avin, Kaushik Mondal, and Stefan Schmid. Demand-aware network design with minimal congestion and route lengths. *IEEE/ACM Transactions on Networking*, 30(4):1838--1848, 2022. doi:10.1109/TNET.2022.3153586.

[55] Chen Avin and Stefan Schmid. *ReNets: Statically-Optimal Demand-Aware Networks*, pages 25--39. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611976489.3, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611976489.3, doi:10.1137/1.9781611976489.3.

[56] James Aweya, Michel Ouellette, and Delfin Y Montuno. Buffer management scheme employing dynamic thresholds, September 7 2004. US Patent 6,788,697.

[57] Michael Bacharach. Matrix rounding problems. *Management Science*, 12(9):732--742, 1966. URL: https://doi.org/10.1287/mnsc.12.9.732.

[58] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 49--67, Boston, MA, April 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/bai.

[59] Wei Bai, Shuihai Hu, Kai Chen, Kun Tan, and Yongqiang Xiong. One more config is enough: Saving (dc)tcp for high-speed extremely shallow-buffered datacenters. *IEEE/ACM Transactions on Networking*, 29(2):489--502, 2021. doi:10.1109/TNET.2020.3032999.

[60] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. 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*, SIGCOMM '20, page 782–797, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3406221.

[61] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

[62] Nikhil Bansal, Christian Coester, Ravi Kumar, Manish Purohit, and Erik Vee. Learning-augmented weighted paging. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 67--89. SIAM, 2022.

[63] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. Multipath tcp: From theory to practice. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *NETWORKING 2011*, pages 444--457, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[64] Andreas V Bechtolsheim and David R Cheriton. Per-flow dynamic buffer management, February 4 2003. US Patent 6,515,963.

[65] Maciej Besta and Torsten 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*, pages 348--359, 2014. doi:10.1109/SC.2014.34.

[66] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucuman, Ser. A*, 5:147--154, 1946.

[67] Ivan A. Bochkov, Alex Davydow, Nikita Gaevoy, and Sergey I. Nikolenko. New competitiveness bounds for the shared memory switch. *CoRR*, abs/1907.04399, 2019. URL: http://arxiv.org/abs/1907.04399.

[68] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS '16, page 75–88, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2896377.2901479.

[69] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. *Queueing Systems*, 88(3):311--347, 2018.

[70] Béla Bollobás and W Fernandez de la Vega. The diameter of random regular graphs. *Combinatorica*, 2(2):125--134, 1982.

[71] Tommaso Bonato, Abdul Kabbani, Daniele De Sensi, Rong Pan, Yanfang Le, Costin Raiciu, Mark Handley, Timo Schneider, Nils Blach, Ahmad Ghalayini, et al. Smartt-reps: Sender-based marked rapidly-adapting trimmed & timed transport with recycled entropies. *arXiv preprint arXiv:2404.01630*, 2024.

[72] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. 1998.

[73] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR'14*, 2014.

[74] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. Tcp vegas: new techniques for congestion detection and avoidance. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, page 24–35, New York, NY, USA, 1994. Association for Computing Machinery. doi:10.1145/190314.190317.

[75] Broadcom. 12.8 tb/s strataxgs tomahawk 3 ethernet switch series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56980-series.

[76] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680*, 2019. URL: http://arxiv.org/abs/1909.05680.

[77] Qizhe Cai, Mina Tahmasbi Arashloo, and Rachit Agarwal. Dcpim: Near-optimal proactive datacenter transport. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 53–65, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3544216.3544235.

[78] J.W. Causey and H.S. Kim. Comparison of buffer allocation schemes in atm switches: complete sharing, partial sharing, and dedicated allocation. In *Proceedings of ICC/SUPERCOMM'94 - 1994 International Conference on Communications*, pages 1164--1168 vol.2, 1994. doi:10.1109/ICC.1994.368919.

[79] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. Osa: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking*, 22(2):498--511, 2014. doi:10.1109/TNET.2013.2253120.

[80] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. Enabling Wide-Spread communications on optical fabric with MegaSwitch. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 577--593, Boston, MA, March 2017. USENIX Association. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/chen.

[81] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. Enabling Wide-Spread communications on optical fabric with MegaSwitch. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 577--593, Boston, MA, March 2017. USENIX Association. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/chen.

[82] Shawn Shuoshuo Chen, Weiyang Wang, Christopher Canel, Srinivasan Seshan, Alex C. Snoeren, and Peter Steenkiste. Time-division tcp for reconfigurable data center networks. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 19–35, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3544216.3544254.

[83] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, page 73–82, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1592681.1592693.

[84] Stanley Cheung, Tiehui Su, Katsunari Okamoto, and SJB Yoo. Ultra-compact silicon photonic 512× 512 25 ghz arrayed waveguide grating router. *IEEE Journal of Selected Topics in Quantum Electronics*, 20(4):310--316, 2013.

[85] Marco Chiesa, Guy Kindler, and Michael Schapira. Traffic engineering with equal-cost-multipath: An algorithmic perspective. *IEEE/ACM Transactions on Networking*, 25(2):779--792, 2017. doi:10.1109/TNET.2016.2614247.

[86] A.K. Choudhury and E.L. Hahne. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions on Networking*, 6(2):130--140, 1998. doi:10.1109/90.664262.

[87] Cisco. Nexus 9000 series switches. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.html.

[88] Larry A Coldren, Gregory Alan Fish, Y Akulova, JS Barton, L Johansson, and CW Coldren. Tunable semiconductor lasers: A tutorial. *Journal of Lightwave Technology*, 22(1):193, 2004.

[89] Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. On the lambertw function. *Advances in Computational mathematics*, 5(1):329--359, 1996.

[90] Intel Corporation. Intel Tofino, 2020. Retrieved Dec. 29, 2020 from https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html.

[91] Fred Douglis, Seth Robertson, Eric van den Berg, Josephine Micallef, Marc Pucci, Alex Aiken, Keren Bergman, Maarten Hattink, and Mingoo Seok. Fleet—fast lanes for expedited execution at 10 terabits: Program overview. *IEEE Internet Computing*, 25(3):79--87, 2021. doi:10.1109/MIC.2021.3075326.

[92] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401--414, Seattle, WA, April 2014. USENIX Association. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi%C4%87.

[93] D. Z. Du and F. K. Hwang. Generalized de bruijn digraphs. *Networks*, 18(1):27--38, 1988. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230180105, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230180105, doi:10.1002/net.3230180105.

[94] Nick G Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, Kadangode K Ramakrishnan, and Jacobus 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*, pages 95--108, 1999.

[95] Nandita Dukkipati and Nick McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, January 2006. doi:10.1145/1111322.1111336.

[96] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 305–318, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3452296.3472936.

[97] F. Ertemalp. Using dynamic buffer limiting to protect against belligerent flows in high-speed networks. In *Proceedings of the Ninth International Conference on Network Protocols*, ICNP '01, pages 230--, Washington, DC, USA, 2001. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=876907.881596.

[98] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, page 339–350, 2010. doi:10.1145/1851182.1851223.

[99] Anja Feldmann, Balakrishnan Chandrasekaran, Seifeddine Fathalli, and Emilia N. Weyulu. P4-enabled network-assisted congestion feedback: A case for nacks. In *Proceedings of the 2019 Workshop on Buffer Sizing*, BS '19, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3375235.3375238.

[100] Lisa Fleischer and Martin Skutella. Minimum cost flows over time without intermediate storage. In *SODA*, volume 3, pages 66--75, 2003.

[101] Lisa Fleischer and Martin Skutella. Quickest flows over time. *SIAM Journal on Computing*, 36(6):1600--1630, 2007. arXiv:https://doi.org/10.1137/S0097539703427215, doi:10.1137/S0097539703427215.

[102] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397--413, Aug 1993. doi:10.1109/90.251892.

[103] Sally Floyd, Ramakrishna Gummadi, Scott Shenker, et al. Adaptive red: An algorithm for increasing the robustness of red's active queue management, 2001.

[104] Klaus-Tycho Foerster and Stefan Schmid. Survey of reconfigurable data center networks: Enablers, algorithms, complexity. In *SIGACT News*, 2019.

[105] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 519--533. USENIX Association, April 2021. URL: https://www.usenix.org/conference/nsdi21/presentation/gao.

[106] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235--251, 1996. doi:10.1137/S0097539793243016.

[107] Jim Gettys. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 15(3):96--96, 2011.

[108] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. A microscopic view of bursts, buffer contention, and loss in data centers. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 567–580, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517745.3561430.

[109] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 216–229, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2934872.2934911.

[110] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 225–238, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3098822.3098839.

[111] Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, mar 2010. doi:10.1145/1753171.1753195.

[112] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E. Anderson. Backpressure flow control. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 779--805, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/goyal.

[113] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51--62. ACM, 2009.

[114] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. Cerberus: The power of choices in datacenter topology design - a throughput perspective. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(3), dec 2021. doi:10.1145/3491050.

[115] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 Conference*, page 63–74, 2009. doi:10.1145/1592568.1592577.

[116] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 202--215, 2016.

[117] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL: https://www.gurobi.com.

[118] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. 42(5):64–74, July 2008. doi:10.1145/1400097.1400105.

[119] Ellen L. Hahne, Alexander Kesselman, and Yishay Mansour. Competitve buffer management for shared-memory switches. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, page 53–58, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/378580.378589.

[120] Alex Hall, Steffen Hippler, and Martin Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 379(3):387--404, 2007. Automata, Languages and Programming. URL: https://www.sciencedirect.com/science/article/pii/S0304397507001508, doi:10.1016/j.tcs.2007.02.046.

[121] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R. Das, Jon P. Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *Proceedings of the ACM SIGCOMM 2014 Conference*, page 319–330, 2014. doi:10.1145/2619239.2626328.

[122] Sarah-Michelle Hammer, Vamsi Addanki, Max Franke, and Stefan Schmid. Starlink performance through the edge router lens. In *Proceedings of the 2nd International Workshop on LEO Networking and Communication*, LEO-NET '24, page 67–72, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3697253.3697273.

[123] Michelle Hampson. Reconfigurable optical networks will move supercomputerdata 100x faster,". *IEEE Spectrum*, 2021.

[124] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin 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, SIGCOMM '17, page 29–42, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3098822.3098825.

[125] David A. Hayes and Grenville Armitage. Revisiting tcp congestion control using delay gradients. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *NETWORKING 2011*, pages 328--341, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[126] Yihua He, Nitin Batta, and Igor Gashinsky. Understanding switch buffer utilization in clos data center fabric.

[127] Jörn-Thorben Hinz, Vamsi Addanki, Csaba Györgyi, Theo Jepsen, and Stefan Schmid. Tcp's third eye: Leveraging ebpf for telemetry-powered congestion control. In *Proceedings of the 1st Workshop on EBPF and Kernel Extensions*, eBPF '23, page 1–7, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3609021.3609295.

[128] Torsten Hoefler, Tommaso Bonato, Daniele De Sensi, Salvatore Di Girolamo, Shigang Li, Marco Heddes, Jon Belk, Deepak Goel, Miguel Castro, and Steve Scott. Hammingmesh: A network topology for large-scale deep learning. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1--18, 2022. doi:10.1109/SC41404.2022.00016.

[129] C.V. Hollot, V. Misra, D. Towsley, and Wei-Bo Gong. A control theoretic analysis of red. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 3, pages 1510--1519 vol.3, 2001. doi:10.1109/INFCOM.2001.916647.

[130] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. Finishing flows quickly with preemptive scheduling. *SIGCOMM Comput. Commun. Rev.*, 42(4):127–138, August 2012. doi:10.1145/2377677.2377710.

[131] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. Aeolus: A building block for proactive transport in datacenters. 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*, SIGCOMM '20, page 422–434, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3405878.

[132] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Tagger: Practical pfc deadlock prevention in data center networks. CoNEXT '17, page 451–463, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3143361.3143382.

[133] Sijiang Huang, Mowei Wang, and Yong Cui. Traffic-aware buffer management in shared memory switches. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1--10. IEEE, 2021.

[134] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 285–294, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3409964.3461790.

[135] Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Online dynamic acknowledgement with learned predictions. pages 1--10, 2023. doi:10.1109/INFOCOM53939.2023.10228882.

[136] Makoto Imase and Masaki Itoh. A design for directed graphs with minimum diameter. *IEEE Transactions on Computers*, 32(08):782--784, 1983.

[137] Makoto Imase, Terunao Soneoka, and Keiji Okada. Connectivity of regular directed graphs with small diameters. *IEEE Transactions on Computers*, 34(03):267--273, 1985.

[138] V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 314–329, New York, NY, USA, 1988. Association for Computing Machinery. doi:10.1145/52324.52356.

[139] V Jacobson and N Kathleen. Controlling queue delay-a modern aqm is just one piece of the solution to bufferbloat. *Asscociation for Computing Machinery (ACM Queue)*, 2012.

[140] Syed Usman Jafri, Sanjay Rao, Vishal Shrivastav, and Mohit Tawarmalani. Leo: Online traffic classification at Multi-Terabit line rate. In *21th USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, 2024. USENIX Association.

[141] Grzegorz Jereczek, Theo Jepsen, Simon Wass, Bimmy Pujari, Jerry Zhen, and Jeongkeun Lee. Tcp-int: lightweight network telemetry with tcp transport. In *Proceedings of the SIGCOMM '22 Poster and Demo Sessions*, SIGCOMM '22, page 58–60, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3546037.3546064.

[142] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 69:1--69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[143] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. MegaScale: Scaling large language model training to more than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745--760, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/jiang-ziheng.

[144] Cheng Jin, D.X. Wei, and S.H. Low. Fast tcp: motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, volume 4, pages 2490--2501 vol.4, 2004. doi:10.1109/INFCOM.2004.1354670.

[145] Sangeetha Abdu Jyothi, Ankit Singla, P. Brighten Godfrey, and Alexandra Kolla. Measuring and understanding throughput of network topologies. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 761--772, 2016. doi:10.1109/SC.2016.64.

[146] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter rpcs can be general and fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 1--16, Boston, MA, February 2019. USENIX Association. URL: https://www.usenix.org/conference/nsdi19/presentation/kalia.

[147] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 437--450, Denver, CO, June 2016. USENIX Association. URL: https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia.

[148] Srikanth Kandula, Jitu Padhye, and Victor Bahl. Flyways to de-congest data center networks. 2009.

[149] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, page 202–208, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1644893.1644918.

[150] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 281--294. ACM, 2017.

[151] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, page 89–102, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/633025.633035.

[152] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '16, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2890955.2890968.

[153] Yafim Kazak, Clark Barrett, Guy Katz, and Michael Schapira. Verifying deep-rl-driven systems. In *Proceedings of the 2019 Workshop on Network Meets AI & ML*, NetAI'19, page 83–89, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3341216.3342218.

[154] Srinivasan Keshav. *Mathematical foundations of computer networking*. Addison-Wesley, 2012.

[155] Alexander Kesselman and Yishay Mansour. Harmonic buffer management policy for shared memory switches. *Theoretical Computer Science*, 324(2):161--182, 2004. Online Algorithms: In Memoriam, Steve Seiden. URL: https://www.sciencedirect.com/science/article/pii/S0304397504003779, doi:10.1016/j.tcs.2004.05.014.

[156] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM '15 Demos*, 2015.

[157] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*, pages 77--88. IEEE, 2008.

[158] Koji Kobayashi, Shuichi Miyazaki, and Yasuo Okabe. A tight bound on online buffer management for two-port shared-memory switches. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, page 358–364, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1248377.1248437.

[159] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. Collie: Finding performance anomalies in RDMA subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 287--305, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/kong.

[160] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14--76, 2015. doi:10.1109/JPROC.2014.2371999.

[161] Santosh Krishnan, Abhijit K Choudhury, and Fabio M Chiussi. Dynamic partitioning: A mechanism for shared memory management. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 1, pages 144--152. IEEE, 1999.

[162] Janardhan Kulkarni, Euiwoong Lee, and Mohit Singh. Minimum birkhoff-von neumann decomposition. In Friedrich Eisenbrand and Jochen Koenemann, editors, *Integer Programming and Combinatorial Optimization*, pages 343--354, Cham, 2017. Springer International Publishing.

[163] Janardhan Kulkarni, Stefan Schmid, and Paweł Schmidt. Scheduling opportunistic links in two-tiered reconfigurable datacenters. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 318–327, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3409964.3461786.

[164] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael 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*, pages 514--528, 2020.

[165] Yanfang Le, Jeongkeun Lee, Jeremias Blendin, Jiayi Chen, Georgios Nikolaidis, Rong Pan, Robert Soule, Aditya Akella, Pedro Yebenes Segura, Arjun singhvi, Yuliang Li, Qingkai Meng, Changhoon Kim, and Serhat Arslan. Sfc: Near-source congestion signaling and flow control, 2023. URL: https://arxiv.org/abs/2305.00538, arXiv:2305.00538.

[166] Changhyun Lee, Chunjong Park, Keon Jang, Sue Moon, and Dongsu Han. Accurate latency-based congestion feedback for datacenters. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 403--415, Santa Clara, CA, July 2015. USENIX Association. URL: https://www.usenix.org/conference/atc15/technical-session/presentation/lee-changhyun.

[167] Jason Lei and Vishal Shrivastav. Seer: Future-Aware caching system for network processors. In *21th USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, 2024. USENIX Association.

[168] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. Technical report, Massachusetts Inst of Tech Cambridge Microsystems Research Center, 1989.

[169] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, nov 1999. doi:10.1145/331524.331526.

[170] Bojie Li, Tianyi Cui, Zibo Wang, Wei Bai, and Lintao Zhang. Socksdirect: Datacenter sockets can be fast and compatible. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 90–103, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3341302.3342071.

[171] Jialong Li, Haotian Gong, Federico De Marchi, Aoyu Gong, Yiming Lei, Wei Bai, and Yiting Xia. Uniform-cost multi-path routing for reconfigurable data center networks. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 433–448, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672245.

[172] Wenxue Li, Xiangzhou Liu, Yuxuan Li, Yilun Jin, Han Tian, Zhizhen Zhong, Guyue Liu, Ying Zhang, and Kai Chen. Understanding communication characteristics of distributed training. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, APNet '24, page 1–8, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3663408.3663409.

[173] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3341302.3342085.

[174] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244--404, 2021. URL: http://dx.doi.org/10.1561/2400000035, doi:10.1561/2400000035.

[175] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. Circuit switching under the radar with REACToR. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 1--15, Seattle, WA, April 2014. USENIX Association. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/liu_he.

[176] He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael Kaminsky, George Porter, and Alex C. Snoeren. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2716281.2836126.

[177] Kexin Liu, Zhaochen Zhang, Chang Liu, Yizhi Wang, Vamsi Addanki, Stefan Schmid, Qingyue Wang, Wei Chen, Xiaoliang Wang, Jiaqi Zheng, Wenhao Sun, Tao Wu, Ke Meng, Fei Chen, Weiguang Wang, Bingyang Liu, Wanchun Dou, Guihai Chen, and Chen Tian. Pyrrha: Congestion-Root-Based flow control to eliminate Head-of-Line blocking in datacenter. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 379--405, Philadelphia, PA, April 2025. USENIX Association. URL: https://www.usenix.org/conference/nsdi25/presentation/liu-kexin.

[178] Shiyu Liu, Ahmad Ghalayini, Mohammad Alizadeh, Balaji Prabhakar, Mendel Rosenblum, and Anirudh Sivaraman. Breaking the transience-equilibrium nexus: A new approach to datacenter packet transport. In *NSDI*, pages 47--63, 2021.

[179] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A Fault-Tolerant engineered network. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 399--412, Lombard, IL, April 2013. USENIX Association. URL: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/liu_vincent.

[180] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. Rethinking machine learning collective communication as a multi-commodity flow

problem. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 16–37, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672249.

[181] Yi Liu, Jiangping Han, Kaiping Xue, Ruidong Li, and Jian Li. L2bm: Switch buffer management for hybrid traffic in data center networks. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 1--11, 2023. doi:10.1109/ICDCS57875.2023.00076.

[182] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. Quartz: A new design element for low-latency dcns. *ACM SIGCOMM Computer Communication Review*, 44(4):283–294, aug 2014. doi:10.1145/2740070.2626332.

[183] S.H. Low, F. Paganini, and J.C. Doyle. Internet congestion control. *IEEE Control Systems Magazine*, 22(1):28--43, 2002. doi:10.1109/37.980245.

[184] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1--24:25, 2021.

[185] Matt Mathis and Andrew McGregor. Buffer sizing: a position paper.

[186] William M. Mellette, Ilya Agurok, Alex Forencich, Spencer Chang, George Papen, and Joseph E. Ford. A scalable, high-speed optical rotor switch. In *Optical Fiber Communication Conference (OFC) 2024*, page Th1A.5. Optica Publishing Group, 2024. URL: https://opg.optica.org/abstract.cfm?URI=OFC-2024-Th1A.5, doi:10.1364/OFC.2024.Th1A.5.

[187] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1--18, Santa Clara, CA, February 2020. USENIX Association. URL: https://www.usenix.org/conference/nsdi20/presentation/mellette.

[188] William M. Mellette, Alex Forencich, Rukshani Athapathu, Alex C. Snoeren, George Papen, and George Porter. Realizing rotornet: Toward practical microsecond scale optical networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 392–414, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672273.

[189] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. SIGCOMM '17, page 267–280, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3098822.3098838.

[190] William Maxwell Mellette, Glenn M. Schuster, George Porter, George Papen, and Joseph E. Ford. A scalable, partially configurable optical switch for data center networks. *Journal of Lightwave Technology*, 35(2):136--144, 2017. doi:10.1109/JLT.2016.2636025.

[191] Rui Miao, Bo Li, Hongqiang Harry Liu, and Ming Zhang. Buffer sizing with hpcc. 2019.

[192] Vishal Misra, Wei-Bo Gong, and Don Towsley. Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, page 151–160, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/347059.347421.

[193] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 537–550, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2785956.2787510.

[194] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for rdma. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 313–326, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3230543.3230557.

[195] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/0f49c89d1e7298bb9930789c8ed59d48-Paper.pdf.

[196] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. *arXiv preprint arXiv:1902.00732*, 2019. URL: http://arxiv.org/abs/1902.00732.

[197] Michael Mitzenmacher. Queues with small advice. In *Proceedings of the 2021 SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, pages 1--12. SIAM, 2021. doi:10.1137/1.9781611976830.1.

[198] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Commun. ACM*, 65(7):33–35, jun 2022. doi:10.1145/3528087.

[199] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: a receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 221–235, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3230543.3230564.

[200] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. AccelTCP: Accelerating network applications with stateful TCP offloading. In *17th USENIX Symposium on Networked Systems Design*

and Implementation (NSDI 20), pages 77--92, Santa Clara, CA, February 2020. USENIX Association. URL: https://www.usenix.org/conference/nsdi20/presentation/moon.

[201] Matthew K. Mukerjee, Christopher Canel, Weiyang Wang, Daehyeok Kim, Srinivasan Seshan, and Alex C. Snoeren. Adapting TCP for reconfigurable datacenter networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 651--666, Santa Clara, CA, February 2020. USENIX Association. URL: https://www.usenix.org/conference/nsdi20/presentation/mukerjee.

[202] Aisha Mushtaq, Asad Khalid Ismail, Abdul Wasay, Bilal Mahmood, Ihsan Ayyub Qazi, and Zartash Afzal Uzmi. Rethinking buffer management in data center networks. *SIGCOMM Comput. Commun. Rev.*, 44(4):575--576, August 2014. URL: http://doi.acm.org/10.1145/2740070.2631462, doi:10.1145/2740070.2631462.

[203] D. Naddef and W.R. Pulleyblank. Matchings in regular graphs. *Discrete Mathematics*, 34(3):283--291, 1981. URL: https://www.sciencedirect.com/science/article/pii/0012365X81900066, doi:10.1016/0012-365X(81)90006-6.

[204] Pooria Namyar, Sucha Supittayapornpong, Mingyang Zhang, Minlan Yu, and Ramesh Govindan. A throughput-centric view of the performance of datacenter topologies. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 349–369, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3452296.3472913.

[205] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. A survey of reconfigurable optical networks. *Optical Switching and Networking*, 41:100621, 2021. URL: https://www.sciencedirect.com/science/article/pii/S1573427721000187, doi:10.1016/j.osn.2021.100621.

[206] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 1–15, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3341301.3359646.

[207] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3458817.3476209.

[208] Mark EJ Newman. Analysis of weighted networks. *Physical review E*, 70(5):056131, 2004.

[209] ns 3. Network simulator. https://www.nsnam.org/.

[210] Vladimir Olteanu, Haggai Eran, Dragos Dumitrescu, Adrian Popa, Cristi Baciu, Mark Silberstein, Georgios Nikolaidis, Mark Handley, and Costin Raiciu. An edge-queued datagram service for all datacenter traffic. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 761--777, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/olteanu.

[211] Eugene Opsasnick. Buffer management and flow control mechanism including packet-based dynamic thresholding. *US patent US7953002B2*. URL: https://patents.google.com/patent/US7953002B2/en.

[212] Oracle. Oracle cloud infrastructure blog: First principles: Building a high-performance network in the public cloud. URL: https://blogs.oracle.com/cloud-infrastructure/post/building-high-performance-network-in-the-cloud.

[213] Christoph Paasch and Olivier Bonaventure. Multipath tcp. *Commun. ACM*, 57(4):51–57, apr 2014. doi:10.1145/2578901.

[214] Maciej Pacut, Wenkai Dai, Alexandre Labbe, Klaus-Tycho Foerster, and Stefan Schmid. Improved scalability of demand-aware datacenter topologies with minimal route lengths and congestion. 49(3):35–36, March 2022. doi:10.1145/3529113.3529125.

[215] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pages 148--155, July 2013. doi:10.1109/HPSR.2013.6602305.

[216] Jonathan Perry, Hari Balakrishnan, and Devavrat Shah. Flowtune: Flowlet control for datacenter networks. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 421--435, Boston, MA, March 2017. USENIX Association. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/perry.

[217] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: a centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 307–318, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2619239.2626309.

[218] Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15(1):193--220, 1891.

[219] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, USA, 2008. USENIX Association.

[220] Polatis. POLATIS® 7000 Optical Circuit Switch. https://www.hubersuhner.com/en/shop/product/other-systems/optical-switches/rack-mount-circuit-switches/85223159/polatis-7000-optical-circuit-switch.

[221] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating microsecond circuit switching into the data center. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 447–458, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2486001.2486007.

[222] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 66–85, 2022. doi:10.1145/3544216.3544265.

[223] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/73a427badebe0e32caa2e1fc7530b7f3-Paper.pdf.

[224] Pybind11. Seamless operability between C++11 and Python. https://pybind11.readthedocs.io/en/stable/.

[225] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. Cassini: Network-aware job scheduling in machine learning clusters. In *21th USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, 2024. USENIX Association.

[226] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834--1845. SIAM, 2020.

[227] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, and Torsten Hoefler. ReDMArk: Bypassing RDMA security mechanisms. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4277--4292. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/rothenberger.

[228] Noga H. Rotman, Michael Schapira, and Aviv Tamar. Online safety assurance for learning-augmented systems. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, page 88–95,

New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3422604.3425940.

[229] Tim Roughgarden. Beyond the worst-case analysis of algorithms. pages 1--24. Cambridge University Press, 2020.

[230] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123--137, 2015.

[231] Ruixue Fan, A. Ishii, B. Mark, G. Ramamurthy, and Qiang Ren. An optimal buffer management scheme with dynamic thresholds. In *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99. (Cat. No.99CH37042)*, volume 1B, pages 631--637 vol. 1b, Dec 1999. doi:10.1109/GLOCOM.1999.830130.

[232] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, and Amin Vahdat. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. SIGCOMM '20, page 735–749, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3405899.

[233] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *Advances in Neural Information Processing Systems*, 32:9835--9846, 2019.

[234] Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Transactions on Networking*, 24(3):1421--1433, 2016. doi:10.1109/TNET.2015.2410313.

[235] Roy Schwartz, Mohit Singh, and Sina Yazdanbod. Online and offline greedy algorithms for routing with switching costs. *CoRR*, abs/1905.02800, 2019. URL: http://arxiv.org/abs/1905.02800, arXiv:arXiv:1905.02800, doi:10.48550/arXiv.1905.02800.

[236] scikit-learn. Machine Learning in Python. https://scikit-learn.org/stable/.

[237] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. TACCL: Guiding collective algorithm synthesis using communication sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 593--612, Boston, MA, April 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/shah.

[238] Farhad Shahrokhi and David W Matula. The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2):318--334, 1990.

[239] Danfeng Shan, Wanchun Jiang, and Fengyuan Ren. Absorbing micro-burst traffic by enhancing dynamic threshold policy of data center switches. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 118--126. IEEE, 2015.

[240] Danfeng Shan, Wanchun Jiang, and Fengyuan Ren. Analyzing and enhancing dynamic threshold policy of data center switches. *IEEE Transactions on Parallel and Distributed Systems*, 28(9):2454--2470, 2017. doi:10.1109/TPDS.2017.2671429.

[241] Danfeng Shan, Yuqi Liu, Tong Zhang, Yifan Liu, Yazhe Tang, Hao Li, and Peng Zhang. Less is more: Dynamic and shared headroom allocation in pfc-enabled datacenter networks. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 591--602, 2023. doi:10.1109/ICDCS57875.2023.00019.

[242] Naveen Kr Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. Programmable calendar queues for high-speed packet scheduling. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 685--699, 2020.

[243] Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities. *arXiv preprint arXiv:2405.21063*, 2024.

[244] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[245] Vishal Shrivastav. Fast, scalable, and programmable packet scheduler in hardware. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 367–379, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3341302.3342090.

[246] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proceedings of the ACM SIGCOMM 2015 Conference*, page 183–197, 2015. doi:10.1145/2785956.2787508.

[247] Ankit Singla, P. Brighten Godfrey, and Alexandra Kolla. High throughput data center topology design. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 29--41, Seattle, WA, April 2014. USENIX Association. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/singla.

[248] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *9th USENIX Symposium on Networked*

Systems Design and Implementation (NSDI 12), pages 225--238, San Jose, CA, April 2012. USENIX Association. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/singla.

[249] Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, and Yueping Zhang. Proteus: A topology malleable data center network. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, 2010. doi:10.1145/1868447.1868455.

[250] Martin Skutella. *An Introduction to Network Flows over Time*, pages 451--482. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-540-76796-1_21.

[251] SONiC. Software for Open Networking in the Cloud. URL: https://sonic-net.github.io/SONiC/.

[252] Tim Stevenson. Nexus 9000 architecture, 2020. URL: https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2020/pdf/BRKDCN-3222.pdf.

[253] B. Suter, T.V. Lakshman, D. Stiliadis, and A.K. Choudhury. Buffer management schemes for supporting tcp in gigabit routers with per-flow queueing. *IEEE Journal on Selected Areas in Communications*, 17(6):1159--1169, 1999. doi:10.1109/49.772451.

[254] Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, and Torsten Hoefler. sRDMA -- efficient NIC-based authentication and encryption for remote direct memory access. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 691--704. USENIX Association, July 2020. URL: https://www.usenix.org/conference/atc20/presentation/taranov.

[255] NVIDIA Onyx User Manual v3.10.4302 (LTS). Shared Buffers. URL: https://docs.nvidia.com/networking/display/Onyxv3104302/Shared+Buffers.

[256] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 205--219, 2016.

[257] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350--361, 1982.

[258] Leslie G Valiant and Gordon J Brebner. Universal schemes for parallel communication. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263--277, 1981.

[259] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *SIGCOMM Comput. Commun. Rev.*, 42(4):115–126, August 2012. doi:10.1145/2377677.2377709.

[260] Midhul Vuppalapati, Saksham Agarwal, Henry Schuh, Baris Kasikci, Arvind Krishnamurthy, and Rachit Agarwal. Understanding the host network. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 581–594, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672271.

[261] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. C-through: Part-time optics in data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):327–338, aug 2010. doi:10.1145/1851275.1851222.

[262] Haiyong Wang. OCPUS18 – SONiC Development and Deployment at Alibaba, 2018. URL: https://www.youtube.com/watch?v=aSd3R3gnQtw.

[263] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 34, 2021.

[264] Weitao Wang, Dingming Wu, Sushovan Das, Afsaneh Rahbar, Ang Chen, and T. S. Eugene Ng. RDC: Energy-Efficient data center network congestion relief with topological reconfigurability at the edge. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1267--1288, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/wang-weitao-rdc.

[265] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. Rail-only: A low-cost high-performance network for training llms with trillion parameters. In *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 1--10, Los Alamitos, CA, USA, aug 2024. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/HOTI63208.2024.00013, doi:10.1109/HOTI63208.2024.00013.

[266] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 739--767, Boston, MA, April 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/wang-weiyang.

[267] Cedric Westphal, Kiran Makhijani, and Richard Li. Packet trimming to reduce buffer sizes and improve round-trip times.

[268] CISCO white paper. Priority flow control: build reliable layer-2 infrastructure. 2009.

[269] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 50–61, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/2018436.2018443.

[270] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, Boston, MA, March 2011. USENIX Association. URL: https://www.usenix.org/conference/nsdi11/design-implementation-and-evaluation-congestion-control-multipath-tcp.

[271] Jackson Woodruff, Andrew W Moore, and Noa Zilberman. Measuring burstiness in data center applications. In *Proceedings of the 2019 Workshop on Buffer Sizing*, BS '19, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3375235.3375240.

[272] Guo-Liang Wu and J.W. Mark. A buffer allocation scheme for atm networks: complete sharing based on virtual partition. *IEEE/ACM Transactions on Networking*, 3(6):660--670, 1995. doi:10.1109/90.477712.

[273] Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu, Ziyang Yang, Hongyi Liu, and Ang Chen. Bedrock: Programmable network support for secure RDMA systems. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2585--2600, Boston, MA, August 2022. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/xing.

[274] Zhaoqi Xiong and Noa Zilberman. Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 25–33, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3365609.3365864.

[275] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021. URL: https://openreview.net/forum?id=nVZtXBI6LNn.

[276] Zhenggen Xu. OCPSummit19 - EW: SONiC - LinkedIn Adoption of OCP SONiC, 2019. URL: https://www.youtube.com/watch?v=skUnjqPOvXs.

[277] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. Axiomatizing congestion control. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(2), June 2019. doi:10.1145/3341617.3326148.

[278] Kyriakos Zarifis, Rui Miao, Matt Calder, Ethan Katz-Bassett, Minlan Yu, and Jitendra Padhye. Dibs: Just-in-time congestion mitigation for data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2592798.2592806.

[279] Johannes Zerwas, Csaba Györgyi, Andreas Blenk, Stefan Schmid, and Chen Avin. Duo: A high-throughput reconfigurable datacenter network using local routing and control. volume 7, New York, NY, USA, mar 2023. Association for Computing Machinery. doi:10.1145/3579449.

[280] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *Advances in Neural Information Processing Systems*, 2022.

[281] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. Understanding lifecycle management complexity of datacenter topologies. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 235--254, Boston, MA, February 2019. USENIX Association. URL: https://www.usenix.org/conference/nsdi19/presentation/zhang.

[282] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 78–85, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3131365.3131375.

[283] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. Justitia: Software Multi-Tenancy in hardware Kernel-Bypass networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1307--1326, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/zhang-yiwen.

[284] Liangyu Zhao, Siddharth Pal, Tapan Chugh, Weiyang Wang, Prithwish Basu, Joud Khoury, and Arvind Krishnamurthy. Optimal direct-connect topologies for collective communications. *CoRR*, abs/2202.03356, 2022. URL: https://arxiv.org/abs/2202.03356, arXiv:2202.03356.

[285] Liangyu Zhao, Siddharth Pal, Tapan Chugh, Weiyang Wang, Jason Fantl, Prithwish Basu, Joud Khoury, and Arvind Krishnamurthy. Efficient direct-connect topologies for collective communications. *CoRR*, abs/2202.03356, 2024. URL: https://arxiv.org/abs/2202.03356, arXiv:2202.03356.

[286] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y. Zhao, and Haitao Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):443–454, aug 2012. doi:10.1145/2377677.2377761.

[287] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 45(4):523--536, 2015.

[288] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. Ecn or delay: Lessons learnt from analysis of dcqcn and timely. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, page 313–327, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2999572.2999593.