TOWARDS SCALABLE AND OPTIMAL OBLIVIOUS RECONFIGURABLE NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Daniel Amir

August 2024

TOWARDS SCALABLE AND OPTIMAL OBLIVIOUS RECONFIGURABLE NETWORKS

Daniel Amir, Ph.D.

Cornell University 2024

Datacenter network demands show explosive growth, doubling nearly every year. Unfortunately, datacenter networks are built primarily using packet switches, which do not scale as quickly as these demands and are expected to scale even slower in the future. Nanosecond-scale optical circuit switches represent a potential alternative: unlike packet switches, they are not limited by semiconductor scaling trends, and unlike previous optical circuit switches, they are fast enough to support all datacenter traffic types, including short flows. To be used to their full potential, however, these switches will require novel network designs.

This dissertation examines how to build datacenter-scale networks using exclusively nanosecond-scale optical circuit switches. We identify the Oblivious Reconfigurable Network (ORN) design paradigm, which is designed to use the capabilities of these switches. We develop Shale, the first ORN to achieve a tunable tradeoff between latency scalability and throughput. We also show how to compose multiple tunings to support multiple traffic classes, a common feature of datacenter network traffic. To enable Shale, we develop a novel congestion control algorithm tailored to Shale's unique environment, which we extend to address node and link failures. Finally, we implement a Field-Programmable Gate Array (FPGA)-based hardware prototype for a Shale end-host. Our designs show that Shale can achieve orders of magnitude better latency and hardware resource requirements than previous ORN designs. Additionally, we investigate the fundamental performance limits of ORNs, and prove that ORNs must grapple with an inherent tradeoff between latency and throughput. The tradeoffs achieved by Shale match this fundamental tradeoff up to a constant factor, meaning that every tuning of Shale is Pareto optimal among ORNs.

Together, Shale and our exploration of the fundamental limits of ORNs represent important steps towards scalable and optimal ORNs.

BIOGRAPHICAL SKETCH

Daniel Amir is a Ph.D. candidate in the department of Computer Science at Cornell University, advised by Hakim Weatherspoon. He holds a Master of Science in Computer Science from Cornell University, as well as a Bachelor in Computer Science and a Bachelor in Engineering Physics from the University of Illinois at Urbana-Champagin.

ACKNOWLEDGEMENTS

First, I would like to express my deepest gratitude to my advisor, Hakim Weatherspoon. When I first came to Cornell during my visit day, I received an important piece of advice for choosing an advisor: You should make sure to pick someone who is not just a good researcher, but also a good person, because students often grow to resemble their advisors. I can only hope this has been true in my case. Throughout my time at Cornell, Hakim has helped me to always keep the bigger picture in mind, and remember how my research will impact and improve the world. I would also like to thank Bobby Kleinberg, who taught me how to take a theoretical approach to my research, giving it much more depth. I've learned so much through our hands-on collaboration. Finally, I would like to thank Andrew Campana, who was willing to be on my committee despite being in a vastly different field. His classes have helped me to be a better communicator, and have provided me with valuable perspectives on the world.

I would also like to thank my research collaborators. Vishal Shrivastav was an early role model for me as a Ph.D. student. Without his ideas and mentorship, my research would not have come as far as it has. Tegan Wilson has been a close collaborator throughout my research and on all of my publications, in addition to being an important friend. We share a number of coincidences between us, from being born two days apart, to growing up in the same region, and finally ending up in the same office as first year students, even before we started working on the same project. I am grateful for the theoretical perspective that she brought to our research, and for the opportunity to grow together over the past six years. Nitika Saran's contributions, as well as her immense drive and perseverance, have been immensely helpful in my recent work. It is wonderful to see her ideas come to life, and I can't wait to see what she will come up with in the future.

Finally, I would like to thank my family for supporting me in my studies. Since my birth, my grandparents have worked to ensure that I would have access to the highest level of education that I wanted. They have always been willing to give me advice, and have

been excited to hear about my accomplishments. My parents helped me to see my own strengths, and worked to give me the best environment to develop them. They continue to push me to use my abilities to achieve real impact in the world. I am extremely grateful for their advice and encouragement. My sister Daphne has been a massive source of support throughout my Ph.D. In addition to reading and editing my work on numerous occasions, she has also been a close friend. I wish her the best of luck as she begins her own Ph.D. studies. Finally, my brother Ethan is an important source of inspiration. He is already so talented, curious, and resilient, and I am excited to see the person he will become.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Since the advent of the Internet, datacenters have exploded in popularity as the most efficient way to store and process large amounts of data. As the number of Internet users has grown exponentially, reaching 5.4 billion people (representing 67% of humanity) by 2023 [74], access to and reliance on datacenters has grown with it. This has placed an immense strain on the networks inside datacenters, which have had to grow exponentially to keep pace. Google has revealed that between 2008 and 2014, bandwidth demand in its datacenters nearly doubled each year [69]. Similarly, between 2010 and 2020, Microsoft Azure's datacenter link rates increased 400× [20], closely matching Google's scaling. The ongoing rise of extremely data-intensive machine learning and generative AI workloads will only continue this growth; some projections indicate that datacenter network demand will again increase by over 400× between 2020 and 2030 [9].

Unfortunately, datacenter networks are built primarily using packet switches, which are limited by semiconductor scaling trends such as Moore's law [9, 63, 54]. These trends are already too slow to keep up with the explosive growth seen in datacenter network demands, and are projected to slow even further. Fortunately, nanosecond-scale optical circuit switches are emerging as a potential alternative to packet switches [7]. These switches are not limited by semiconductor scaling, and unlike previously existing optical circuit switches, they are fast enough to efficiently support the short flows which make up the majority of flows in datacenter networks. To be used to their full potential, however, these switches will require novel network designs.

This dissertation examines how to build networks using exclusively nanosecond-scale optical circuit switches that can operate at datacenter scales (on the order of a hundred thousand nodes). We advance the state of the art by identifying the Oblivious

Reconfigurable Network (ORN) design paradigm, which is tailored to take advantage of the capabilities of these switches. We develop Shale, the first ORN to achieve a tunable tradeoff between latency scalability and throughput. We also show how to compose multiple tunings to support multiple traffic classes, a common feature of datacenter network traffic. To enable Shale, we develop a novel congestion control algorithm tailored to Shale's unique environment, which we extend to address node and link failures. Finally, we implement a Field-Programmable Gate Array (FPGA)-based hardware prototype for a Shale end-host. Our designs show that Shale can achieve orders of magnitude better latency and hardware resource requirements than previous ORN designs, enabling ORNs to achieve datacenter scale for the first time. Additionally, we investigate the fundamental performance limits of ORNs, and prove that ORNs must grapple with an inherent tradeoff between latency and throughput. The tradeoffs achieved by Shale match this fundamental tradeoff up to a constant factor, meaning that every tuning of Shale is Pareto optimal among ORNs.

1.1 Background

We begin by giving a brief overview of packet and circuit switches, the major switching technologies available for datacenter networks. Technology trends suggest that networks based on packet switches will face significant difficulties in scaling, motivating a shift to circuit switches.

1.1.1 Packet switches

At the time of writing, datacenter networks are largely built using exclusively *packet switches* [1] [9, 63]. These switches are designed to route *packets* of data tagged with a

[1] Italicized terms are defined in the glossary

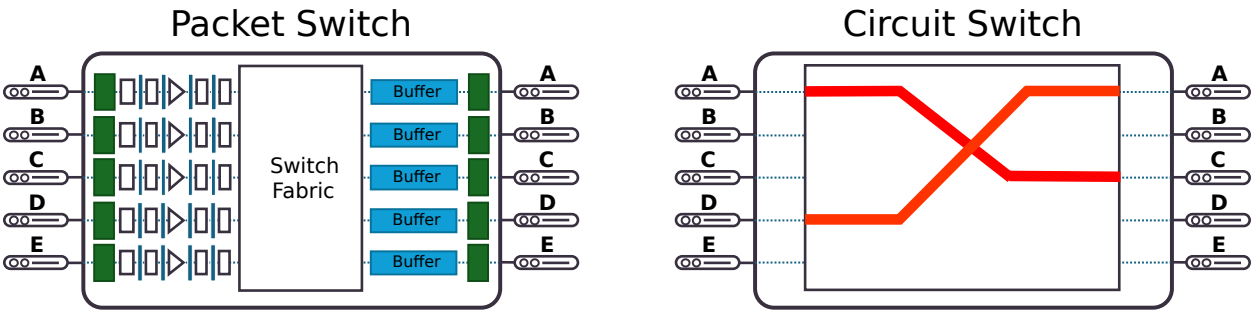
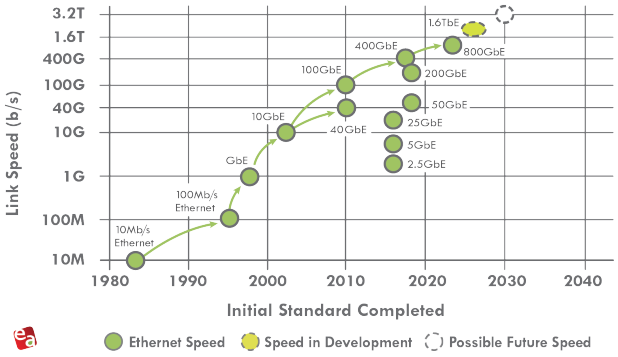


Figure 1.1: Schematics of a packet switch and a circuit switch. Packet switches require complex internal circuitry, including packet processing pipelines, a switch fabric, and buffers, to send individual packets to the correct destination. Circuit switches create fixed circuits between input and output ports, greatly simplifying the switch design.

header indicating the destination as well as other information needed for *routing*. Each time a packet switch receives a packet, it reads this header and autonomously decides which way to forward the packet next. This autonomy within each switch makes packet-switched networks flexible and easy to design. However, reading each packet's header and forwarding it to the correct destination requires complex circuitry, as can be seen in Figure 1.1. As networks increase in speed to meet ever-growing demands, this circuitry causes two issues.

First, as the speed of network links, or *line rate*, rises, this circuitry must also speed up for packet switches to keep pace. However, integrated circuits can only double in processing speed roughly every two years, following *Moore's law*. This is slower than the rate at which network demands are increasing. To keep up, packet-switched networks must grow in size and complexity over time, increasing costs and creating difficulties for network management and debugging. This will only worsen as Moore's law begins to slow.

Second, as link rates have scaled to 100 Gbps and beyond, it is no longer practical to transmit data over long distances using electrical signals over copper wires. Instead, network links have transitioned to optical technology, which use laser light to transmit data over optical fibers. This can be seen in Figure 1.2, which shows the development timeline



(a)

Speed	Electrical Standards: Twinax Cable	Electrical Standards: Twisted Pair	Optical Standards: Multi-Mode Fiber	Optical Standards: Single-Mode Fiber
1 G		100 m	550 m	40 km
10 G		100 m	400 m	40 km
40 G	7 m	30 m	150 m	40 km
100 G	7 m		150 m	40 km
200 G	7 m		100 m	40 km
400 G	3 m		150 m	40 km
800 G	3 m*		150 m*	40 km†
1.6 T	3 m†		150 m*	

* Commercially available while standard in development
† Standard in development

(b)

Figure 1.2: Development timeline (a) and maximum connection length (b) for selected Ethernet standards as of 2024. For speeds above 40 Gbps, while short-range electrical standards have been developed, long-range connections are exclusively optical [3, 80].

and maximum distance of various connection standards for Ethernet, by far the most common networking technology in datacenters. The last long-range electrical standard was developed in 2016 for 40 Gbps connection speeds; subsequent electrical standards are only sufficient for very short links. This poses a problem for packet switches, which can only process data in electronic form. To use high-bandwidth optical links, packet switches must convert data from optical to electronic form, and then back to optical, at every single switch. These conversions require expensive *optical transceivers* and consume significant power, increasing the cost of packet-switched networks.

1.1.2 Optical circuit switches

To continue scaling datacenter networks without paying the increasing costs associated with fully packet-switched networks, some network operators have turned to *optical circuit switches* (OCSes), a form of *circuit switch* designed for networks based on optical links. Unlike packet switches, circuit switches do not make individual decisions on where to forward individual packets. Instead, circuit switches set up *circuits* which directly connect a particular input *port* to a particular output port. For example, in Figure 1.1, all data

received from input port A is forwarded to output port C. To change this, for example so that data from input port A is forwarded to output port B instead, the switch must receive an instruction to *reconfigure* the circuit, which is sent separately from the data traversing the switch. This eliminates the need for the complex circuitry used in packet switches, as well as the need to process the data traversing the switch. In the case of OCSes, data can remain in light form as it traverses the switch, being redirected using optical devices such as mirrors and waveguides.

Despite this promise, at the time of writing, OCSes have only seen limited deployment in datacenters. One reason for this is that current commercially deployed OCSes take several milliseconds to reconfigure [45, 84]. Most *flows* sent in datacenter networks are quite short: in some workloads, half of all flows are under a kilobyte [62], which can be transmitted in under 100 nanoseconds on 100 Gbps links. If a network were built completely out of millisecond-scale OCSes, the long reconfiguration times would make the network too inefficient.

Instead, OCSes have been used to augment packet-switched networks. In these networks, while traffic is largely routed using packet switches, OCSes are used to dynamically add high-bandwidth links, optimizing the network for the observed *traffic demand* [57, 19, 77, 44]. The most successful implementation of this idea so far has been in Google's datacenter networks. In these networks, small sections of the datacenter, known as aggregation blocks, are internally networked using packet switches. Then, the aggregation blocks are connected using OCSes. The network uses these OCSes to optimize the available bandwidth between each pair of aggregation blocks once every few hours [57]. This strategy is able to efficiently allocate bandwidth, eliminating the need for some packet switches and simplifying the network.

Google's deployment experience has also demonstrated an additional benefit of OCSes: the same OCSes can continue to be used even as transceivers at *end-hosts* are upgraded [57].

This means that the same physical OCS can be used across multiple network generations. OCSes thus completely eliminate the switch as a barrier to upgrading network speeds. However, Google's decision to continue to use packet switches throughout their networks shows that their OCSes cannot support the short flows found in datacenter workloads on their own.

The technology of *rapid circuit switches* promises to change this. Several designs have been demonstrated for OCSes that can reconfigure within nanosecond timescales [7, 15, 17]. With such fast reconfiguration speeds, rapid OCSes can efficiently support both long and short flows. However, conventional circuit-switched network designs, which dynamically optimize circuits based on the traffic demand observed in the network, are unable to take advantage of their capabilities. This is because dynamic optimization requires a time-consuming control loop: Traffic demands must first be collected from the entire network, and then an optimized set of circuits must be computed and deployed to all OCSes and end-hosts. It is impractical to run such a control loop at microsecond, let alone nanosecond scale. Realizing the potential of rapid OCSes requires optimized network designs.

1.2 Oblivious Reconfigurable Networks

To take advantage of the capabilities of rapid OCSes, this dissertation identifies the *Oblivious Reconfigurable Network* (ORN) design paradigm. The key feature of ORNs is that they do not dynamically optimize the network based on the observed traffic demand. On the contrary, ORNs design the network to be entirely oblivious with respect to the traffic demand.

ORNs combine two main components: a predetermined traffic-oblivious *connection schedule*, and a coordination-free *oblivious routing scheme*. We begin by describing the traffic

oblivious schedule, which dictates which circuits are deployed at every point in time. Since the schedule is predetermined, it can be run at a very high speed, allowing rapid OCSes to frequently reconfigure in a predictable and useful pattern. The schedule of an ORN can be expressed in terms of *timeslots*. During each timeslot, the schedule dictates which *nodes* are to be connected to which other nodes, and switches remain in a fixed configuration, allowing nodes to send a single fixed-size packet, or *cell*. This schedule is run synchronously, enabled by nanosecond-scale clock synchronization [7, 41, 67] and careful control of the propagation delay of each link. Timeslots are separated by *guard bands* during which no data is sent, allowing switches to reconfigure and absorbing slight clock desynchronization. With nanosecond-scale OCSes, ORNs can iterate through their schedules very quickly: designs have been proposed that begin a new timeslot every 5.6 ns [68].

Since traffic demands cannot be perfectly predicted in advance, it is impossible for a fixed schedule to always connect nodes to the destinations they wish to send to. Instead, ORNs use an *oblivious routing scheme* to route data indirectly, via intermediate nodes. This allows nodes to use their outgoing bandwidth at any time, even if they are connected to a node other than their desired destination. The oblivious routing scheme ensures that eventually, their data will be forwarded to the correct destination.

1.2.1 The Single Round-Robin Design

While we were the first to specifically identify the ORN network design paradigm as such in [4], several designs had previously been proposed following its blueprint, including RotorNet [49, 48], Shoal [68], and Sirius [7]. All of these existing designs used similar, easy-to-express schedules and routing schemes. Here, we present an abstracted version of these designs which we refer to as the Single Round Robin Design (SRRD). As the name

Timeslot	Node A	B	C	D	E	F
1	B	C	D	E	F	A
2	C	D	E	F	A	B
3	D	E	F	A	B	C
4	E	F	A	B	C	D
5	F	A	B	C	D	E

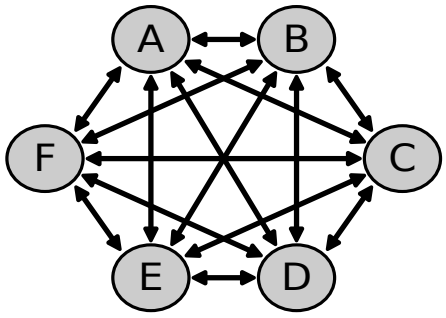


Figure 1.3: A single round-robin schedule for 6 nodes and connections formed over the course of the schedule.

implies, this design uses a schedule based on a single round robin among all nodes in the system: each node is connected to every other node once. We refer to one iteration of the schedule (one round-robin in this case) as an *epoch*.

To support arbitrary workloads, the SRRD uses a routing scheme based on Valiant Load Balancing (VLB) [76]. In VLB, when a node has a cell to send, rather than sending it directly to the destination, it first sends to a random intermediate node in the system[2]. The SRRD accomplishes this in one hop, which we refer to as a *spraying hop* due to its similarity to an existing technique in datacenter networking called *packet spraying*. Next, the cell is forwarded directly to the destination. This is also accomplished in one hop in the SRRD, which we refer to as a *direct hop*.

Latency Ignoring queuing for now, this design achieves a *latency* of N timeslots in the worst case, where N is the number of nodes in the network. The first hop to the intermediate node takes place over the first timeslot. However, in the worst case, the intermediate node may need to wait an entire epoch to be able to forward the cell to the destination, for a total latency of N timeslots.

[2]Both the SRRD and Shale deviate from this slightly to improve latency. When a node has a cell to send, it sends the cell to the first neighbor it can, rather than a random one. While long flows will sample all outgoing links, and flow arrival usually produces sufficient randomness for short flows, additional mechanisms (described in Section 2.2.3) ensure that intermediate nodes are well-distributed in all cases.

Throughput Since half of all hops in the SRRD are spraying hops, while half are direct hops that end at the destination node, a fully-congested SRRD network achieves *throughput* of half of the line rate. The use of VLB ensures that the network is evenly loaded regardless of the workload, preventing any bottleneck links that would reduce throughput below this value.

1.2.2 Challenges

The oblivious nature of ORNs allows them to reconfigure circuits quickly, taking advantage of the capabilities of rapid OCSes. However, the actual performance achieved by an ORN is highly dependent on the specific schedule and routing scheme chosen. Prior to the advances made in this dissertation, all proposed ORN designs had been based on the SRRD. This resulted in significant gaps in the body of knowledge, addressed in this dissertation.

Poor scalability of SRRD-based designs. SRRD-based designs have been demonstrated in several different contexts, showing that the design is fundamentally sound. However, all of these evaluations were performed on systems with only several hundred nodes at most [49, 68, 7]. These small sizes mask scalability issues inherent to the SRRD.

Because the latency of the SRRD is N timeslots, datacenter-scale systems suffer from very high latency. ORNs have been proposed for systems with tens of thousands of nodes [7], reflecting the massive scale of datacenters. Even using the fastest demonstrated nanosecond-scale OCSes, the latency at this scale would be hundreds of microseconds: almost as slow as conventional, non-oblivious circuit-switched network designs.

Additionally, each node must allocate memory to store both network state and cells to be forwarded for each neighboring node it connects to. The SRRD directly connects

every single node to every other node in the system, inflating memory requirements: Some implementations require up to $O(N^2)$ memory [68], which can quickly become overwhelming at datacenter scale. For example, for a 25,000 node network, Shoal requires over 4 GB of memory at each node.

For ORNs to become practical at datacenter scale, ORN designs which overcome these scalability issues must be developed.

Unexplored design space. The SRRD represents only a single point within the ORN design space. However, other schedules and routing schemes are possible, potentially resulting in significantly different performance characteristics. With the design space almost entirely unexplored, it is difficult to evaluate whether any specific ORN design is good or bad relative to other designs. It is also difficult to evaluate the ORN design paradigm as a whole. Methodically exploring the ORN design space is a necessary step to fully understanding this network paradigm.

1.3 Approach

In the previous section, we identified the ORN design paradigm, as well as major challenges and gaps in the surrounding body of knowledge. We now turn our attention to the approach taken by this dissertation to address these gaps and enable datacenter-scale ORNs, here meaning tens to hundreds of thousands of nodes. In particular, we focus on two research questions.

1.3.1 Scalable ORN designs

Research Question 1. *How can one design an ORN that is capable of operating at datacenter scale while supporting both low latency and high throughput?*

To address this research question, we develop a family of schedules and routing schemes which achieve far better scalability than existing SRRD-based designs. Our design uses schedules that are significantly shorter than the SRRD's schedule, dramatically improving latency. This reduction in schedule length also greatly improves hardware resource requirements at large scales.

Routing within a shorter schedule requires using longer paths with multiple intermediate nodes. Unfortunately, using such multi-hop paths reduces the achievable throughput, as each cell consumes bandwidth multiple times along its path. As part of our approach, we develop a novel method to mitigate this tradeoff in the context of datacenter traffic.

An additional challenge is designing an effective congestion control mechanism. As we describe in Section 2.2.3, ORNs are far more sensitive to congestion than existing network types. This makes effective congestion control a necessity for an ORN to be usable in practice. The congestion control designs used by existing ORNs are heavily tied to the two-hop paths used by the SRRD, and would require rethinking from the ground up to work in a multi-hop context. On the other hand, while many current datacenter congestion control mechanisms are designed to work across multiple hops, they are also designed assuming each source-destination pair communicates using either a single path, or a small subset of paths. Our analysis in Section 3.5 implies that ORNs must use many paths between each source-destination pair to achieve good performance across all traffic demands. This requires us to develop novel congestion control mechanisms tailored to our unique context.

An additional challenge created by shorter schedules and multi-hop paths is increased complexity in detecting and reacting to node and link failures. In existing ORNs based on the SRRD, each node directly connects to every other node, allowing each node to independently detect failed nodes. Additionally, because paths are only two hops long, information about failed links can easily be communicated to earlier nodes in any paths that used the failed link. With multi-hop paths, information about failures must traverse more hops. Therefore, we develop new algorithms to properly react to failures in a multi-hop ORN context.

Evaluating these schedules and routing schemes at scale poses an additional challenge. It would be impractically expensive and time-consuming to build a datacenter-scale testbed with tens of thousands of servers in order to fully test our new designs. Even commonly-used network simulators, such as htsim [60], are unable to simulate such a large number of nodes. We use a custom-designed network simulator that can take advantage of the simple design of ORNs to efficiently simulate this large scale.

1.3.2 Fundamental limits of ORNs

Research Question 2. *What are the fundamental limits for the throughput and latency achievable by ORNs?*

To address this research question, we mathematically investigate the ORN design space. Our analysis shows that for ORNs that aim to support arbitrary traffic demands, there is a fundamental tradeoff between throughput and latency: While some ORN designs may achieve optimal throughput for a given network size, they will be limited in the latency they can achieve. Other ORN designs may achieve improved latency, but must sacrifice throughput to do so. An ideal ORN design must therefore be *Pareto optimal*, meaning that while another ORN design might achieve better throughput or better latency, no other

ORN design achieves both at the same time. Any improvement in one must come at the expense of the other. The primary goal of our analysis is to characterize the *Pareto front*, or the latency-throughput tradeoffs achieved by Pareto optimal ORN designs.

To accomplish this, we first develop a mathematical model of an ORN schedule and routing scheme, and show how to evaluate throughput and latency in the context of our model. We carefully choose a model that is both simple enough for us to analyze and prove theorems, and expressive enough to model the full range of ORN designs and accurately predict their achieved throughput and latency. Using our model, we separately prove both an upper bound and a lower bound on the Pareto front.

To prove our upper bound, we construct two families of ORN designs in the context of our mathematical model. Each individual ORN design in our constructions achieves a particular latency-throughput tradeoff at a particular system size. For each system size and target throughput value, we select the particular ORN design from our constructions that achieves the lowest latency while still supporting the target throughput. This provides a maximum value for the Pareto front at every system size and target throughput value. Additionally, one of the two families of designs we construct is a formalization of the designs we develop to answer Research Question 1, allowing us to prove the properties of these designs.

To prove our lower bound, we use the mathematical techniques of *linear programming* and *duality*. In broad strokes, we try to find the maximum throughput that can be supported by an ORN design that respects a given maximum latency. We express this goal in the form of several linear inequalities, each of which describes a *constraint* that must be obeyed by the routing scheme of any potential solution. We combine these constraints with our *objective value* of maximizing throughput to form a *linear program*, or LP. Using a well-known method, we systematically derive a *dual LP* whose solution bounds the solution of our original *primal LP*. Our dual LP can be viewed as attempting to form traffic demands

that overload links in the ORN while requesting as little throughput as possible. We then bound the achievable values for the dual LP, in turn bounding the values achievable by the primal LP. In other words, this produces an upper bound on the throughput achievable by ORNs which respect a given maximum latency. We invert this to get a lower bound on the latency achievable by ORNs while supporting a given throughput.

1.4 Contributions

In this dissertation, we present the following contributions.

1.4.1 Designing a Scalable ORN

We develop **Shale**, the first ORN which is practical at datacenter scale, and take a significant step in addressing Research Question 1. Using a tuning parameter, Shale achieves multiple different tradeoffs between latency and throughput. By properly setting this tradeoff, Shale achieves orders of magnitude better latency and hardware resource requirements compared to existing ORN designs at datacenter scale.

Since datacenter traffic contains some flows which are more latency-sensitive and others which are more throughput-sensitive, we propose **interleaving**. This allows multiple tunings of Shale to be run in parallel, allowing each flow to be sent on the tuning that achieves the best balance for its particular needs. Interleaving helps Shale to mitigate the effects of the latency-throughput tradeoff inherent to ORNs.

We also propose **two congestion control mechanisms**, **`hop-by-hop`** and **`spray-short`**, which are specifically designed to work in the context of Shale. Together, these two strategies outperform several comparison mechanisms, including receiver-

driven and idealized end-to-end mechanisms, in large-scale packet simulations.

We further extend **`hop-by-hop`** to **address node and link failures**. With a simple augmentation, **`hop-by-hop`** can propagate information about failures throughout the system, enabling nodes to avoid routes that traverse failed nodes or links. We show that even under failures, the performance of Shale remains strong.

Finally, we develop an **FPGA-based end-host hardware prototype** which implements Shale with our proposed congestion control mechanisms. To enable this prototype, we develop several strategies to significantly reduce the hardware resources required. We show that at datacenter scales, the hardware resources needed by our prototype are orders of magnitude lower than those required to run existing ORNs, such as Shoal.

To evaluate Shale, we develop a **custom packet-level network simulator** optimized for ORNs. We use our simulator to demonstrate Shale's performance on tens of thousands of nodes, as well as to compare our various tested congestion control mechanisms.

1.4.2 Determining the Fundamental Limits of ORNs

We develop a **formalized model of ORNs** which we use to achieve major advances in addressing Research Question 2. Using our model of ORNs, we prove both an upper and a lower bound on the latency-throughput tradeoffs possible for ORNs. Our upper and lower bound exactly match up to a constant factor for all target throughput values, meaning that we **fully characterize the Pareto front** of the latency-throughput tradeoff for ORNs which support all traffic demands.

Using our model of ORNs, we construct two families of ORN designs, which we call **EBS** (short for Elementary Basis Scheme) and **VBS** (short for Vandermonde Bases Scheme). Together, these two families produce our upper bound. This means that for every

throughput value, EBS or VBS is within a constant factor of the lower bound on latency for all ORNs. Additionally, every single schedule in the EBS family is within a constant factor of the lower bound on latency. Since the EBS family is a formalized version of Shale's schedule and routing scheme, this means that we **prove that Shale is a Pareto optimal ORN design** up to a constant factor in latency.

As a further consequence of our proofs, we **prove the optimality of Valiant load balancing** (VLB) in the context of ORNs. VLB is a commonly used technique within oblivious routing in many network types, and it is used in all ORN designs to date despite imposing a factor-of-two cost on throughput. It is notable that this cost is fundamental to ORNs which support arbitrary traffic patterns.

Finally, we show how to use **EBS with missing nodes**. Specifically, while EBS is only defined for very specific numbers of nodes, we show that arbitrary system sizes can be supported by simply using the next largest valid system size for EBS, and then removing nodes in an evenly spread manner until the intended number of nodes remain. Even when this is done, we show that the resulting system remains within a constant factor of Pareto optimal. This result has important implications for practical deployments of Shale, as well as for the performance of Shale under failures.

1.5 Organization

The rest of this dissertation is organized as follows. In Chapter 2, we describe Shale, the first ORN to be practical at datacenter scale. In Chapter 3, we prove both an upper and a lower bound for the latency-throughput Pareto front for ORNs. We survey related work in Chapter 4, and then discuss directions for future work in Chapter 5. Finally, in Chapter 6, we conclude.

CHAPTER 2

PRACTICAL, SCALABLE OBLIVIOUS RECONFIGURABLE NETWORKS: SHALE

This chapter presents Shale, an oblivious reconfigurable network design which achieves Pareto optimal tradeoffs between latency and throughput, and is practical at datacenter scale. This work first appeared in *Shale: A Practical, Scalable Oblivious Reconfigurable Network*, published in SIGCOMM 2024 with co-authors Tegan Wilson, Nitika Saran, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon.

2.1 Introduction

Oblivious Reconfigurable Networks (ORNs) are a network design paradigm that can realize the potential of rapid circuit switches. In ORNs, circuit switches are reconfigured oblivious to traffic using a predetermined schedule. This eliminates the latency inherent in classical circuit-switched designs. To support arbitrary traffic, ORNs use an oblivious routing scheme to route data indirectly to its destination. By co-designing the schedule and routing scheme, good performance can be achieved regardless of the traffic demand without dynamically adapting any part of the network. RotorNet [49, 48], Shoal [68], and Sirius [7] are three network designs following the ORN concept that have already been demonstrated on physical test-beds. However, these designs all use a similar schedule and routing scheme based on a Single Round Robin Design (SRRD), maximizing throughput at the cost of poor latency at large scales. This chapter investigates how one can design an ORN capable of operating at datacenter scale while supporting both low latency and high throughput.

We develop Shale, an ORN that generalizes this design to achieve a tunable tradeoff between throughput and latency scaling. For datacenter-scale networks, Shale can be tuned to achieve an intrinsic latency multiple orders of magnitude lower than that of

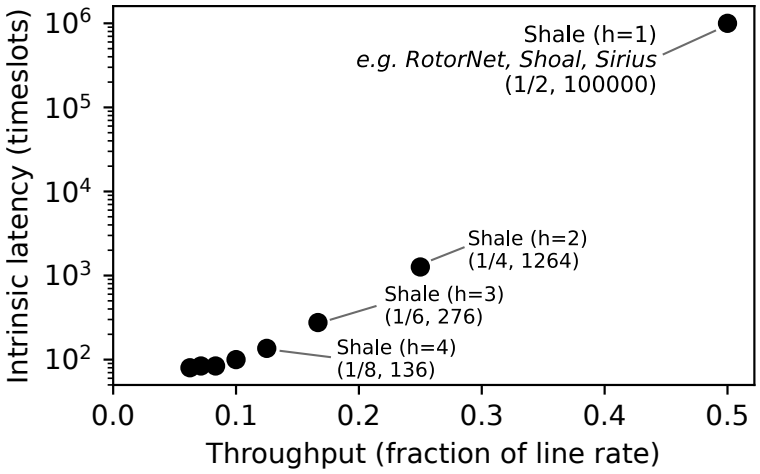


Figure 2.1: A comparison of the throughput and intrinsic latencies achievable by various tunings of Shale for a 100,000-node network. For our evaluation of Shale, timeslots begin every 5.632 ns.

existing systems such as Shoal, RotorNet, and Sirius (which all reduce to a specific tuning of Shale), as shown in Figure 2.1. Shale achieves this tunable tradeoff by carefully choosing a schedule from a collection of ORN schedules. As we show in Section 3.3.4, each of these schedules achieves a tradeoff between throughput and latency that is Pareto optimal among all ORN designs. Shale additionally proposes a novel interleaving technique (Section 2.2.2) that carefully combines multiple ORN schedules. This allows multiple traffic classes with different needs, such as latency-sensitive and throughput-sensitive traffic, to each be served on their ideal schedule.

Next, in order to achieve the Pareto optimal throughput-latency tradeoff, the traffic needs to be uniformly load balanced across all nodes in a Shale network (Section 2.2.1). This high degree of multi-pathing poses a challenge for existing congestion control algorithms, which fall short in maintaining low queuing in the network (Section 2.2.3). To overcome this, we design a novel congestion control algorithm which extensively modifies an existing ORN congestion control [68] to work in Shale's multi-hop context. Our algorithm achieves bounded queuing with minimal resource and processing overheads. Finally, we extend our congestion control mechanism to communicate node and link failures in real time, allowing

traffic to be rerouted with minimal impact on throughput and latency (Section 2.2.4).

Using both an FPGA-based hardware prototype and large-scale packet simulations, we show that Shale's mechanisms achieve close to theoretical throughput and latency guarantees. Both our hardware prototype and our custom packet simulator are available under an open-source license at `https://reconfigurable-networks.github.io/shale`.

2.2 Design

We begin by describing Shale's schedule and routing scheme, which generalize those used by the SRRD, followed by how Shale combines its schedules to support multiple traffic classes. Then, we describe Shale's congestion control. Finally, we explain how Shale reacts to failures.

2.2.1 Schedule and Routing Scheme

Our theoretical analysis of ORNs in Chapter 3 shows that within the ORN design space, many Pareto optimal tradeoffs are possible between worst-case throughput and *intrinsic latency* (latency resulting from the properties of the schedule and routing scheme, rather than queuing and propagation delay). Intuitively, this tradeoff exists because reducing intrinsic latency requires using paths with more hops, which have the flexibility to reach all nodes in the system using a shorter schedule. However, using more hops means each cell consumes more bandwidth over its entire path, reducing throughput. Shale adapts this idea to generalize the SRRD to use h spraying and h direct hops, rather than just one of each. This allows Shale to achieve maximum intrinsic latency of $O(h\sqrt[h]{N})$ and support

Timeslot	AA	BA	CA	⋯	BC	CC
1	BA	CA	AA	⋯	CC	AC
2	CA	AA	BA	⋯	AC	BC
3	AB	BB	CB	⋯	BA	CA
4	AC	BC	CC	⋯	BB	CB

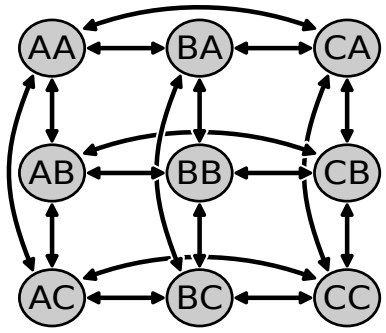


Figure 2.2: Shale's schedule with $h = 2$ for nine nodes. Here, each node is labeled with two letters, and participates in round robins with nodes differing by only one letter.

throughput of $\frac{1}{2h}$, achieving a configurable tradeoff between throughput and latency. As we show in Section 3.3.3, these tradeoffs are Pareto optimal for ORN designs, up to a constant factor in latency.

In the Shale schedule, rather than participating in a single round-robin among all nodes, nodes instead participate in h smaller round-robins, which we refer to as *phases*. Each node is assigned a unique set of h coordinates ranging from 1 to $\sqrt[h]{N}$. During a given phase, a node connects to each of the $\sqrt[h]{N} - 1$ nodes that match in all but a specific coordinate. One full schedule iteration, or *epoch*, consists of one phase for each of the h coordinates. An example of this schedule is shown in Figure 2.2.

Shale's routing scheme uses VLB, and is divided into two halves: a *spraying semi-path*, and a *direct semi-path*. The spraying semi-path sends a cell to a random intermediate node in the system, and consists of h *spraying hops*. As in the SRRD, the first spraying hop is taken to the first available neighboring node. Following this, during the subsequent $h - 1$ phases, a random hop is taken. After all h spraying hops, all h coordinates have been randomized and the cell is at a random intermediate node. The direct semi-path then forwards the cell to its final destination. During the each of the h phases immediately following the end of the spraying semi-path, a *direct hop* is taken to the node which matches the destination in the corresponding coordinate. After taking up to h such hops, all coordinates are correctly set and the cell arrives at its destination.

For example, in $h = 2$ Shale, a cell could be sent from node AA to node CC via the path AA→BA→BB→CB→CC. In this path, the portion from AA to BB is the spraying semi-path, and from BB to CC is the direct semi-path.

Overall, paths in Shale are up to $2h$ hops long, and take place over up to $2h$ adjacent phases. This corresponds to an intrinsic latency equivalent to 2 epochs, or $2h(\sqrt[h]{N} - 1)$ timeslots. Because of the use of VLB, flow is well-distributed throughout the network on average for all workloads, leading to a worst-case throughput guarantee of $\frac{1}{2h}$ times line rate.

2.2.2 Handling Multiple Traffic Classes

Interleaving

Shale's tuning parameter h provides multiple different schedules, each achieving a different tradeoff between throughput and latency for all traffic. This enables a new method of supporting multiple traffic classes, which we call *interleaving*, in which multiple sub-schedules are integrated into a single schedule which combines their benefits. Each sub-schedule is used as-is with the routing unmodified, and each cell is routed on only one sub-schedule, ensuring that the properties of each schedule are maintained. In its simplest form, interleaving can be achieved by simply alternating between two different sub-schedules every other timeslot; for example, every even timeslot could be mapped to an entry of the higher-throughput $h = 2$ schedule, while every odd timeslot could be mapped to an entry of the lower-latency $h = 4$ schedule. This allows short flows to be sent on a low latency schedule while sending long flows on a high throughput schedule, maximizing the benefits of both. Depending on the workload and desired performance, different ratios of timeslots can be allocated to each schedule. In the future, Shale could

even be interleaved with demand-aware sub-schedules, which may be beneficial for mixed or partially known demands.

Performance impacts. An interleaved ORN sub-schedule has increased latency and reduced throughput proportional to the fraction of timeslots allocated to the sub-schedule. For example, a sub-schedule allocated half of the timeslots will take twice as long to complete each schedule iteration, doubling both the intrinsic latency and the sensitivity to queuing. However, propagation delay remains unaffected, and the degree of queuing in a low-latency schedule may be reduced if long flows are delegated to a separate, high-throughput sub-schedule. Similarly, a sub-schedule allocated half of the timeslots will only be able to support half as much throughput. However, the total throughput supported across interleaved high-throughput and low-latency sub-schedules will be greater than that supported by the low-latency schedule run in isolation. We evaluate multiple interleaved configurations of Shale in Section 2.4.2.

Alternative approaches

Other attempts to add support for multiple traffic classes to ORNs are motivated by the slow reconfiguration times of microsecond-scale optical circuit-switched technology, which can penalize the completion times of short flows when used with ORNs. Opera [47] attempts to resolve this problem by running a single round-robin schedule in parallel using multiple transceivers at each node. Under Opera's schedule, at every point in time, nodes are connected in a random expander graph. Each configuration is held for several microseconds, allowing short flows to be routed using multiple hops in a single expander topology. Meanwhile, long flows are sent using the RotorLB transport protocol, a transport protocol which attempts to send traffic only when the source node is directly connected to the destination [47]. For unbalanced workloads, RotorLB sends indirectly using VLB.

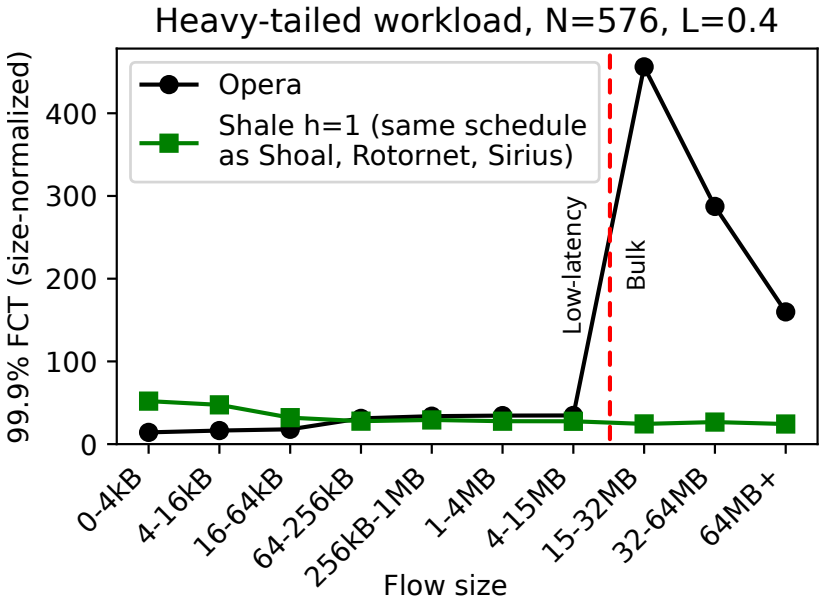


Figure 2.3: Under Opera, long flows are penalized due to long timeslots (over 8000 ns) and use of RotorLB with a large number of nodes. Shale's far shorter timeslots (~5ns) leverage rapidly-reconfiguring switches to enable low latencies for short flows without these compromises.

While this approach completely removes the effect of reconfiguration delay from short flow latencies (allowing it to outperform pure ORNs for the shortest flows), it also forces the use of long timeslots during which nothing is reconfigured. This is because each circuit configuration must be held at least as long as an end-to-end RTT, ideally longer to ensure most short-flow packets make it to their destination before a reconfiguration. Due to the speed of light, this duration can be orders of magnitude longer than the timeslot lengths that would otherwise be possible with new, rapidly-reconfiguring switches. When rapidly-reconfiguring switches are available, Opera has the effect of supporting low latencies at the expense of exacerbating certain scalability issues in the SRRD.

Figure 2.3 shows a comparison between Shale with $h = 1$ (using the same schedule as systems like Rotornet, Shoal, and Sirius), and Opera using similar 576-node configurations. We describe our Shale simulator setup, as well as the heavy-tailed workload used here, in Section 2.4. For Opera, we use the publicly available simulator [50] with minor modifica-

tions to enable a similar simulation setup[1]. Additionally, we use the same 15MB bulk flow cutoff as was used in its original evaluation when testing this workload [47]. Both Shale and Opera are configured with a total aggregate bandwidth of 400 Gbps at each node and a propagation delay of 500 ns. While Shale is able to use a timeslot length of only 5.632 ns, Opera must hold each of its configurations for 8167 ns.

Due to RotorLB, bulk flows are heavily penalized by Opera, experiencing tail flow completion times nearly 400× slower than if they were sent directly to the destination at line rate. This is because in this system size, a node with a bulk flow to send will only be connected to the destination $\frac{1}{575}$ of the time, a problem which will only worsen with system size. This slowing of long flows imposes a severe scalability limitation and makes RotorLB impractical for very large systems.

At the same time, one of the advantages of RotorLB is that bulk traffic can be primarily buffered in the end-host's networking stack until a direct path is available. Abandoning this approach requires buffering traffic at intermediate nodes. However, as timeslot lengths increase, the amount of traffic that must be buffered similarly increases. This greatly increases the memory requirements to run such a system, compounding the high resource utilization imposed by scaling up SRRD-based systems (demonstrated in Figure 2.8).

Note that in Figure 2.3, the longest flows have a decreasing size-normalized FCT. This behavior also appeared in the original evaluation of Opera [47]. Despite this, for flows 128 MB and above, even the flow with the lowest size-normalized FCT in Opera was slower than the highest size-normalized FCT in Shale.

[1]Opera's simulator assumes end-hosts are connected to top-of-rack switches which actually participate in Opera. To compare to Shale which directly connects end-hosts, we simulate only one end-host connected to each TOR using a single 400 Gbps link, and configure each TOR with 8 x 50 Gbps uplinks.

2.2.3 Congestion Control

Although Shale is based on an ORN design that achieves theoretically optimal intrinsic latency, in practice queuing can contribute a large fraction of total realized latencies. ORNs are particularly sensitive to queuing because their schedules empty at a rate of one cell per schedule iteration, rather than at line rate[2]. Congestion control is a core ingredient of Shale.

Shale differs from traditional packet-switched and circuit-switched networks in ways that make many conventional congestion control mechanisms a poor fit. First, Shale relies heavily on multi-pathing, which is necessary to achieve good throughput on arbitrary workloads in ORNs (as implied by the proof in Section 3.5). These paths overlap in intricate ways, resulting in complex fate-sharing relationships that frustrate attempts to address congestion on an end-to-end basis. At the same time, the fixed size of cells makes it expensive to send short control messages. Space can be reserved in each timeslot for exclusive use by control messages, but if they are sent end-to-end, care must be taken to ensure they do not themselves get congested. Finally, congestion is likely to occur during all hops in a path. This differs from traditional datacenter networks where congestion is most likely to occur on the final hop due to *incast* (a situation where multiple senders send to the same destination, exceeding its available bandwidth). These differences make congestion control algorithms developed for packet-switched networks [28, 21, 53] potentially a poor fit for Shale.

[2]In Shale, increasing h reduces the length of the schedule, allowing queues to drain faster, but also increases the number of hops, and therefore the number of queues each cell traverses. Our evaluation shows that with effective congestion control, increasing h does indeed reduce latency even when congestion is considered.

Causes of congestion in Shale

We identify two primary forms of congestion in Shale: *egress* and *path-collision* congestion.

Egress congestion arises when cells with the same destination accumulate in queues leading to their destination. As in other network contexts, this can occur due to incast. If this congestion is not addressed immediately, it quickly results in large amounts of queuing, as multiple senders can send cells more quickly than they can be delivered.

In Shale, egress congestion can also arise spontaneously due to the paths taken by cells, even when the destination's bandwidth is not exceeded. Randomized spraying ensures that traffic is evenly distributed between the final links to a given destination *on average*. However, during a given time period, some final links may receive more cells than others. Once a final link develops queuing, it remains until either fluctuations or congestion control result in less flow being routed along that link, or the total sending rate to the destination decreases. Egress congestion is thus likely to be sustained even in the absence of incast, particularly for heavy-tailed workloads.

Path-collision congestion arises when cells with unrelated destinations happen to be enqueued at the same node to send on the same link. Due to Shale's use of many indirect paths (required to achieve good performance), path collisions can occur between any two source-destination pairs. Unlike egress congestion, path-collision congestion occurs for all workloads with high utilization. However, because Shale uses many indirect paths for each flow, long-running flows do not create sustained load on queues distant from their destination. Path-collision congestion is both less workload dependent and more transient than egress congestion.

We address egress congestion with our **`hop-by-hop`** design. It extensively modifies a congestion control mechanism proposed for the SRRD [68] so that it can operate on

Shale's schedule. We address path-collisions with our **`spray-short`** design, which allows intermediate nodes to slightly deviate from Shale's routing algorithm based on their local queue lengths. We refer to the combination of the two as **`HBH+spray`**.

Hop-by-hop congestion control

ORN proposals based on the SRRD have made use of a hop-by-hop congestion control design [68]. When an intermediate node receives a cell to be forwarded, it counts how many cells are currently present in its send queue to that destination node. When it next connects to the original sender, it uses space reserved in the cell header to send the previously measured queue length. Because the queues are first in, first out (FIFO) and empty at a constant rate of one cell per epoch, the queue length allows the sending node to predict exactly when the previously sent cell will be forwarded. The sending node then waits that many epochs before sending a subsequent cell.

This design maintains the following invariant: At each intermediate node and at every point in time, there is no more than one cell enqueued that was received from the same neighboring node and is intended for the same destination. In the context of the SRRD, this invariant limits the number of forwarded cells in each queue to the incast degree of the destination, bounding egress congestion. Additionally, due to the short path lengths used by the SRRD, path-collision congestion can only occur on the first hop. Thus, total congestion on each queue is limited to the sum of the outcast degree of the sender, and the incast degree of the destination.

Extending to Shale. To adapt this into our **`hop-by-hop`** design intended for use in Shale, we make several significant changes. The first change is necessary to support the longer multi-hop paths used in Shale. To maintain a similar invariant in this context, intermediate

nodes may also need to wait to send particular cells on a given link. However, it is not always possible to predict in advance when a node will be able to forward a cell on its next hop. The first change is thus: rather than sending a queue length immediately when a cell is received, an intermediate node waits until it actually does forward the cell, and then sends a token back to the previous hop. This token gives the previous hop permission to send a new cell with the same destination via the given intermediate node; such cells are now *eligible* to be sent via the given link. Note that after a cell is sent on its final hop, there is no subsequent hop and thus no token needs to be generated; cells are always eligible to be sent on their final hop without regard to tokens.

To prevent head-of-line blocking due to awaiting tokens, the second change is to transition away from strictly FIFO queues to PIEO (Push In Extract Out) queues [64]. These queues allow the first eligible cell to be extracted, even if it is not the first cell in the queue. PIEO queues can be efficiently implemented in hardware, as we demonstrate in our hardware prototype. If there are no eligible cells in its send queue, a node can instead generate a new cell from a flow it is sending, assuming that flow is itself eligible.

The third change prevents the possibility of deadlocks. Due to the arrangement of paths used by Shale, it is possible for a cycle of nodes to want to send each other cells with the same destination. If cells are only differentiated based on their destination, this would form a credit loop, causing a deadlock. To avoid this, **`hop-by-hop`** assigns cells to *buckets*, with each bucket corresponding to a given destination and an *index* representing the number of remaining spraying hops. A cell's eligibility is determined based on the bucket to which it will be assigned at the next hop, and tokens indicate which buckets have become eligible, rather than which destinations. This results in a slightly different invariant compared to hop-by-hop in the SRRD: At each intermediate node and at every point in time, there is no more than one cell present in each bucket that was received from the same neighboring node.

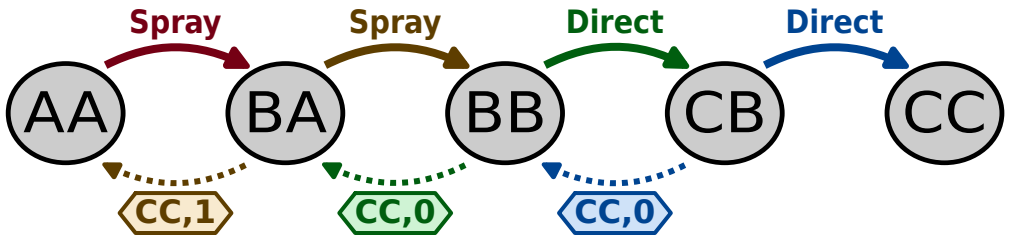


Figure 2.4: An example path which a cell might take in $h = 2$ Shale using **hop-by-hop**, showing both the hops taken by the cell and the returned tokens.

This design avoids deadlocks by eliminating cycles between buckets. In the spraying semi-path, each bucket leads to another bucket with a lower index, so cycles are not possible. In the direct semi-path, all buckets used have an index of 0. Cycles are prevented by the fact that each hop leads to a node that matches more coordinates of the destination.

For an example of how **hop-by-hop** works in practice, consider a cell sent from node AA to node CC via the path shown in Figure 2.4. When the cell arrives at node BA after its first hop, it is tagged with the previous hop AA, and because there is one remaining spraying hop, it is assigned to bucket $\langle$CC, 1$\rangle$. Once node BA forwards the cell on its next hop, it subsequently sends the token $\langle$CC, 1$\rangle$ to node AA at the first opportunity. Following this, node AA may send another cell to node BA which will be assigned to bucket $\langle$CC, 1$\rangle$.

One more change completes the design of **hop-by-hop**. Because a single node can generate multiple tokens intended for the same neighbor in a single epoch in our design, we reserve space in each cell header for two tokens, ensuring that any backlogs drain quickly.

Impacts. Because **hop-by-hop** separates cells with different destinations into different buckets, it primarily addresses egress congestion. The invariant maintained by **hop-by-hop** limits the number of cells enqueued at each node destined to each destination, bounding the queue lengths that can be created by egress congestion. In addition, **hop-by-hop** sends cells to uncongested destinations before to congested destinations,

improving performance during incast.

The first negative impact of **`hop-by-hop`** is an increase in the size of cell headers. This increased overhead slightly decreases the achievable throughput for all workloads.

More significant is the potential for **`hop-by-hop`** to limit throughput when the propagation delay is too large relative to the epoch length. Once a node sends a cell, even if the next hop immediately forwards it, it still takes at least twice the propagation delay to receive a token back. This limits the sending rate of cells in the same bucket (particularly when propagation delay is large relative to epoch length). This reduction in sending rate can be prevented by allowing nodes to send multiple cells before receiving back a token. We describe this strategy in detail below, although we do not evaluate it in this work.

We introduce a parameter T, the *token budget*. Rather than only sending one cell from the same bucket on a given link before receiving a token back, nodes may send up to T such cells. Increasing T increases the maximum sending rate of individual flows under high propagation delay, but decreases **`hop-by-hop`**'s ability to prevent congestion.

Increasing T is the most effective on the first hop. This is because the most constrained parts of the path between any two nodes are the first and final hop. While cells can always be sent on the final hop without waiting for tokens, this is not the case for the first hop. We therefore introduce an additional parameter T_F, the *first-hop token budget*. This parameter operates the same as T but only affects first hops, achieving most of the benefits of increasing T while reducing its negative effects.

For permutation traffic, Shale's throughput guarantee can be met as long as the propagation delay is no greater than hT_FE, where E is the epoch length. Increasing T_F beyond this point allows senders to take advantage of periods of reduced overall network activity to send above Shale's throughput guarantee. Shale's use of VLB ensures that there is high fan-out on subsequent hops in the spraying semi-path, and high fan-in along the direct

semi-path. The degree of fan-in / fan-out is $\sqrt[h]{N} - 1$, so Shale's throughput guarantee can be met as long as $\frac{1}{2h(\sqrt[h]{N}-1)}$ of the bandwidth of the penultimate link is available. This is true when the propagation delay is no greater than $hT(\sqrt[h]{N} - 1)E$. For systems with longer propagation delay, the T parameter can be increased to mitigate this effect.

Spray via short queues

Our second design, **`spray-short`**, targets path-collision congestion. This design modifies Shale's routing algorithm slightly: when a node enqueues a cell on a spraying hop, instead of choosing a random queue in the next phase, it instead chooses the one with the fewest enqueued cells. If multiple queues are tied, the tie is broken randomly.

Impacts. Enqueuing a cell on a spraying hop in the shortest possible send queue ensures that it collides with as few other cells as possible. This reduces the average number of path collisions in the network as a whole, in turn reducing path-collision congestion. **`spray-short`** may also ensure a more even load across the queues of a given node, lowering tail queue lengths and reducing total queuing at each node.

Because **`spray-short`** only uses locally available information at each node to decide which spraying hop to take, it adds no additional overhead to cell headers. However, **`spray-short`** modifies the routing algorithm to no longer be oblivious, departing from the theoretical model we use in Chapter 3 and potentially breaking the throughput guarantee. When using **`spray-short`**, flow is not guaranteed to be evenly distributed among spraying hops, even on average. Depending on the workload, some links might be largely avoided as spraying hops due to heavy load by traffic on direct hops. When combined with **`hop-by-hop`**, **`spray-short`** might cause uneven fan-in near destination nodes, increasing **`hop-by-hop`**'s sensitivity to propagation delay (described in Section 2.2.3).

Despite these theoretical possibilities, we did not observe any throughput reduction due to **`spray-short`** in our experiments. Realizing a sustained reduction in throughput due to **`spray-short`** would require many flows to interact with each other such that for each flow, a small set of suboptimal spraying hops reliably have the shortest queues when cells from that flow arrive at intermediate nodes. This highly specific interaction appears to be extremely unlikely in practice, especially over a sustained period. We leave further theoretical investigation to future work.

2.2.4 Failures

In order to support arbitrary workloads with good performance, ORNs must route traffic via a large number of indirect paths, as discussed in Section 2.2.1. As a result, a single failure in Shale impacts all flows, as cells must avoid paths that traverse the failed node or link. To achieve this, failures must be both detected and communicated throughout the system. We accomplish this through an extension of **`hop-by-hop`**.

To detect failures in Shale, note that every epoch, each node both sends and receives a cell from each of its neighbors. If a node i does not receive a cell from a neighboring node j, it assumes that either the node or the link is failed. Once node i establishes that a failure has occurred, it immediately stops sending cells to node j, ensuring symmetric detection of link failures in case both nodes are still otherwise active.

To communicate information about failed links (and by extension, failed nodes), we introduce *invalidation tokens* and *re-validation tokens*. These tokens have the same format as the regular tokens used by **`hop-by-hop`**, and can be sent using the same portion of the header by adding two bits to differentiate them. Nodes send invalidation tokens to neighbors to indicate that they have no valid route for cells belonging to a given bucket. A re-validation token reverses the effect of a previously-sent invalidation token.

For buckets with zero remaining spraying hops, which correspond to direct hops, a node can send a single invalidation token to communicate all destinations that cannot be reached through direct semi-paths due to a single failed link. This is possible because of the fact that direct semi-paths are deterministic and form a tree. An invalidation token of the form $j, 0$ invalidates all direct semi-paths to node j itself, as well as to all child nodes of j in this tree.

Sending and reacting to invalidation tokens. We first address direct hops. When node i determines that its link in phase p to neighboring node j has failed, it immediately drops all cells awaiting their last hop to node j. Cells being sent on their direct semi-path via node j to another destination are reset to their first spraying hop and re-enqueued, while cells on spraying hops via node j are simply re-enqueued to a different neighboring node in the same phase. Node i then sends the invalidation token $\{j, 0\}$ to all of its neighbors.

When a node k receives an invalidation token of the form $\{j, 0\}$ from neighbor k' in phase q, it learns that the final link in the direct path from k' to j has failed. Since direct paths are deterministic and form a tree, it can compute the final link (i, j) and the phase p that this final link occurs in by considering the direct path the invalidation token must have traveled on over the past few phases. Node k reacts by first dropping all cells enqueued to be sent on a direct semi-path to node j via node k'. It then finds all cells enqueued on direct semi-paths via the failed link to destinations other than j, and resets them to their first spraying hop. Finally, node k forwards the invalidation token $\{j, 0\}$ to all neighboring nodes it connects with in phases p through $q - 1$, inclusive. These are the neighbors that could themselves forward a cell to node k whose direct path would then traverse the failed link.

We now address spraying hops. Once a node k receives an invalidation token from a neighbor k' of the form $\{j, n\}$, it from then on avoids sending cells to destination j on

their $(h-n)$th spraying hop via k'. If it has any such cells currently enqueued, it instead attempts to spray them via a different spraying hop in the same phase which has not yet been invalidated. If k has already received identical invalidation tokens from all other neighbors in the same phase p, this means that node k cannot reach destination j on paths with n spraying hops remaining via any of its neighbors in phase p. Instead of re-spraying the cells, it drops them. Additionally, it sends an invalidation token of the form $\{j, n+1\}$ to all neighboring nodes in phase $p-1$.

Performance under failures. Routing around unusable paths can impact the throughput of remaining flows, especially those whose source or destination shares a phase group with a failed node. Fortunately, as we will show in Section 3.7, as long as failed nodes are well-distributed, with no more than h nodes missing from any given phase group, Shale's throughput guarantee is only reduced by a small factor. Performance can be maintained under arbitrary failures by occasionally exchanging the coordinates of failed nodes in order to ensure an even distribution. In Section 2.4.4, we show that even when up to 8% of nodes are failed, Shale maintains high throughput for the remaining nodes.

2.3 Implementation

We implement an FPGA-based prototype for Shale's end-host in Bluespec System Verilog [12], a high-level hardware description language that compiles to Verilog. Our prototype targets Terasic DE5-Net boards [73] comprising the Altera Stratix V FPGA [72], 234 K Adaptive Logic Modules (ALMs), 52 Mbits (6.5 MB) SRAM, and four 10 Gbps network ports.

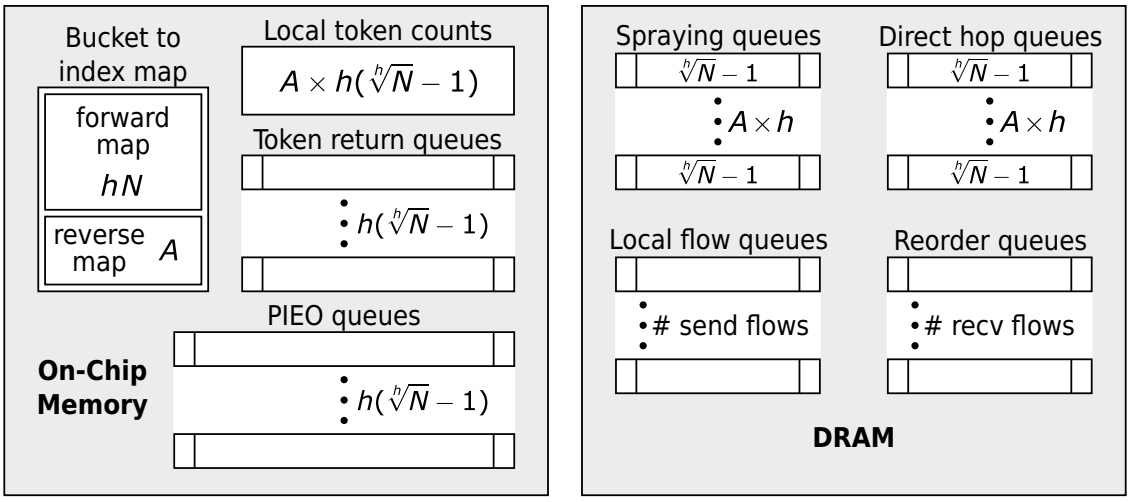


Figure 2.5: Memory layout of end-host implementation. Length of PIEO and token queues are tunable.

2.3.1 End-host Design

One of the main challenges of implementing Shale's congestion control mechanism is that under **`hop-by-hop`**, nodes must send the first *eligible* enqueued cell, even if this cell is not at the head of the queue. We accomplish this with per-neighbor PIEO queues [64, 65, 66], which efficiently implement this capability. We store the contents of cells to be forwarded in per-phase, per-bucket FIFO queues which can be stored in off-chip DRAM, allowing us to store just the bucket IDs in the PIEO queues. To ensure efficient eligibility testing, we store these PIEO queues along with the per-bucket available token counts in on-chip memory. Finally, we store a per-neighbor FIFO queue of tokens to be returned. We illustrate these data structures in Figure 2.5.

RX Path. When a node receives a cell, it first checks if it is the final destination of the cell. If it is, it places the cell into a reorder queue to be delivered to the application. Otherwise, the node determines whether the cell's next hop should be a spraying or a direct hop, and determines the next hop as described in Section 1.1. It decreases the remaining spraying hops in the header if needed, and enqueues the corresponding bucket ID in the PIEO

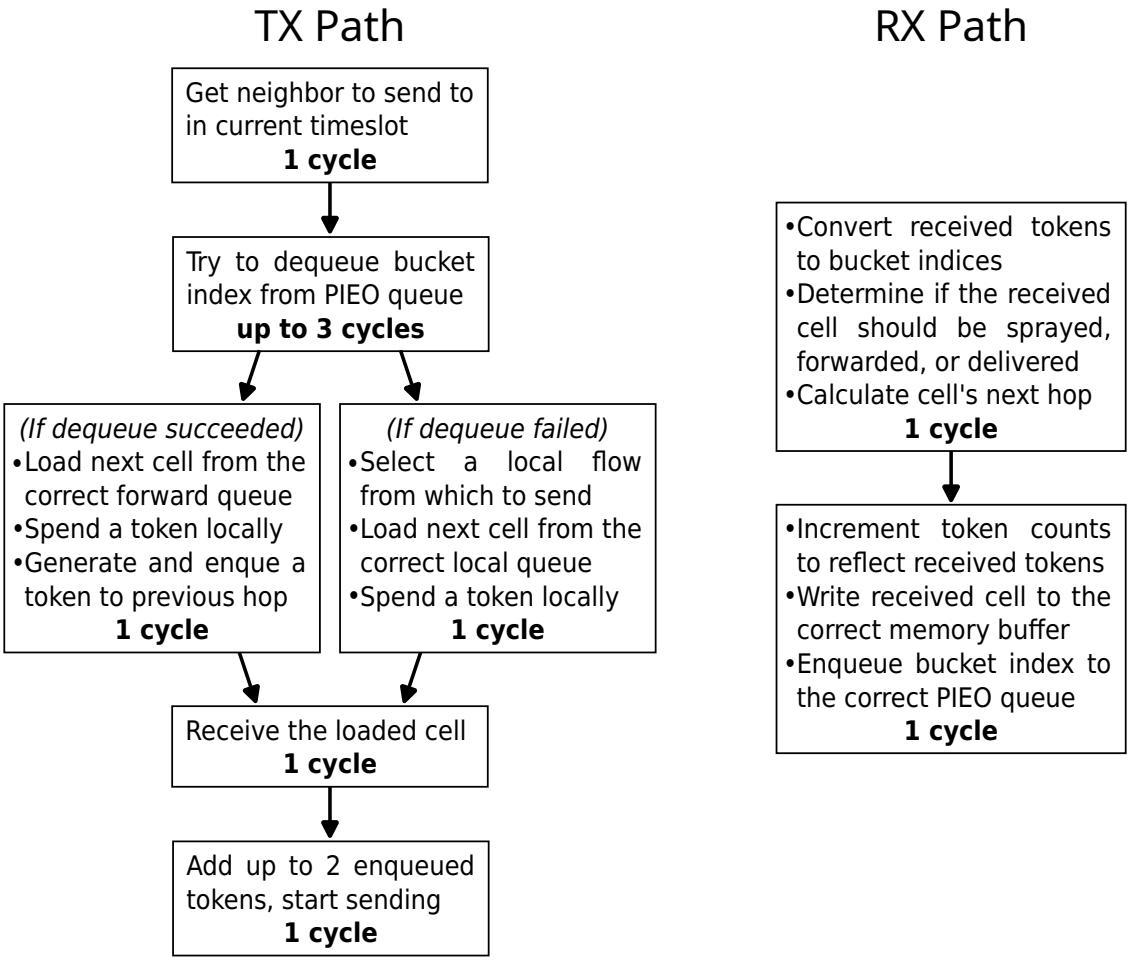


Figure 2.6: Cycle-level breakdown of the critical path for receiving and sending in Shale's end-host implementation.

queue for the next hop. It then writes the cell contents to the FIFO buffer associated with the cell's bucket and the phase of the next hop. In parallel, the node also updates its local token counts to reflect any tokens that were received in the cell header. These operations take 2 clock cycles in the critical path, as visualized in Figure 2.6.

TX path. At the start of each timeslot, each node updates its current phase and neighbor, and attempts to dequeue an eligible bucket ID from the corresponding PIEO queue. If an eligible bucket exists in the queue, the first such bucket is returned, and the node reads a cell from the corresponding FIFO. If there is no eligible bucket, but there is an eligible local flow, it sends a cell from the local flow's queue. Otherwise, the node defaults to sending a

source id	15 bits
remaining sprays	2 bits
token 1	17 bits
CRC checksum	8 bits

destination id	15 bits
sequence number	22 bits
token 2	17 bits

Figure 2.7: 12-byte header format valid for up to 32,768 nodes.

dummy cell.

Once the cell to be sent is retrieved, the node adds up to 2 tokens enqueued for the current neighbor to the header, and starts sending. In total, the operations in the TX path take up to 7 clock cycles in the critical path, as visualized in Figure 2.6.

Supported sending rates. For our evaluation in Section 2.4.1, timeslots are long enough that the entire TX and RX paths complete within a single timeslot. However, both our TX and RX paths can be pipelined to enable much faster timeslot periods. The main limit is PIEO operations, which occupy the PIEO hardware module for four cycles. Our design can easily support four-cycle timeslots by using a dedicated PIEO module for both the RX and TX paths. Our packet simulations in Sections 2.4.2 to 2.4.5 begin a new timeslot every 5.632 ns. Today's networking ASICs (Application-Specific Integrated Circuits) frequently have a clock rate of around 1 GHz [42], which is more than sufficient to support this timeslot period.

In order to minimize the overhead of reconfiguration, we set the timeslot period based on the length of the guard band (i.e., the reconfiguration delay) and the port count at each node. When these two parameters are kept constant, the same frequency ASIC is sufficient to support our routing and congestion control even at higher line rates. If needed, timeslots could potentially be started more frequently by either using a higher-frequency PIEO module, or using multiple PIEO modules in parallel.

Cell headers. In our evaluation of Shale, we assume 256-byte cells with a 12-byte header and 244-byte payload. Figure 2.7 shows a header structure that is valid for Shale with $h \leq 4$ and up to 32,768 nodes. This header supports **`hop-by-hop`**, which we combine with **`spray-short`** (which does not require support from the header).

2.3.2 Optimizations

A naive version of our design would store the cells to be forwarded in per-neighbor, per-bucket FIFO queues. For each bucket and phase, **`hop-by-hop`** ensures that we only receive one cell from each neighbor in the phase. For spraying hops, because all $\sqrt[h]{N} - 1$ of these cells could share the same bucket and next hop, each per-neighbor, per-bucket queue must individually be capable of holding all of them. This inflates the memory required by a factor of $\sqrt[h]{N} - 1$, meaning each node must reserve space for a total of $h^2N(\sqrt[h]{N} - 1)^2$ cells across all neighbors and buckets. We use two optimizations to dramatically reduce this memory requirement.

For our first optimization, we observe that for spraying hops, per-bucket queues can be shared across all neighbors in a given phase. This resolves the factor $\sqrt[h]{N} - 1$ over-allocation of memory described in the previous paragraph. Note that because cells on their spraying hops can be sent via any neighbor in the same phase, this optimization does not affect the correctness of the routing algorithm. While this optimization does not apply to direct hops, all direct hops to the same destination received from the same phase will use the same next hop anyway. This optimization reduces the total memory across all buckets and phases to $h^2N(\sqrt[h]{N} - 1)$.

Our second optimization uses the fact that empirically in our simulations, at any node and at any time, only a small fraction of the hN total possible buckets are active (with enqueued cells or outstanding tokens). Therefore, we only allocate cell buffers and token

storage for a limited number of *active* buckets, set by the parameter A. To allocate indices, we use a freelist bitmap and a priority encoder. We store the mapping from bucket IDs to active bucket indices using a size hN array, and the reverse mapping using a size A array. This reduces the memory required for cell buffers to $2Ah(\sqrt[h]{N} - 1)$ cells, and for local token counts to $Ah(\sqrt[h]{N} - 1)$.

2.3.3 Hardware Resource Scaling

An important concern for Shale's scalability is its memory usage, both on-chip and in DRAM. Assuming each node has at most A active buckets at a time, at most Q_P bucket IDs enqueued in each PIEO queue, and at most Q_T tokens enqueued to return to neighbors, the total on-chip memory required by Shale is $O(h(\sqrt[h]{N} - 1)(Q_P + Q_T + A) + hN)$. Additionally, as described in the previous section, Shale allocates space in DRAM to store $2Ah(\sqrt[h]{N} - 1)$ cells to be forwarded to the next hop.

Shale's most significant compute requirements stem from its use of PIEO queues, which use priority encoders for eligibility testing and rank comparison. Because we only dequeue from one PIEO queue at a time, multiplexers can be used to share a single set of priority encoders across all PIEO queues at the same node, ensuring scalability on this front.

Figure 2.8 shows how Shale's memory requirements scale compared to existing designs, using Shoal [68] as a representative example. Shale's memory requirements are based on the maximum number of active buckets and PIEO queue length observed in our scalability experiments (Section 2.4.5), which are both doubled to account for other potential workloads. The disparity in memory requirements is due to Shale's reduced epoch length, which reduces the number of neighbors. Additionally, our memory optimizations (which in practice do not transfer well to existing schedules) allow us to reduce the amount of on-chip memory required per neighbor.

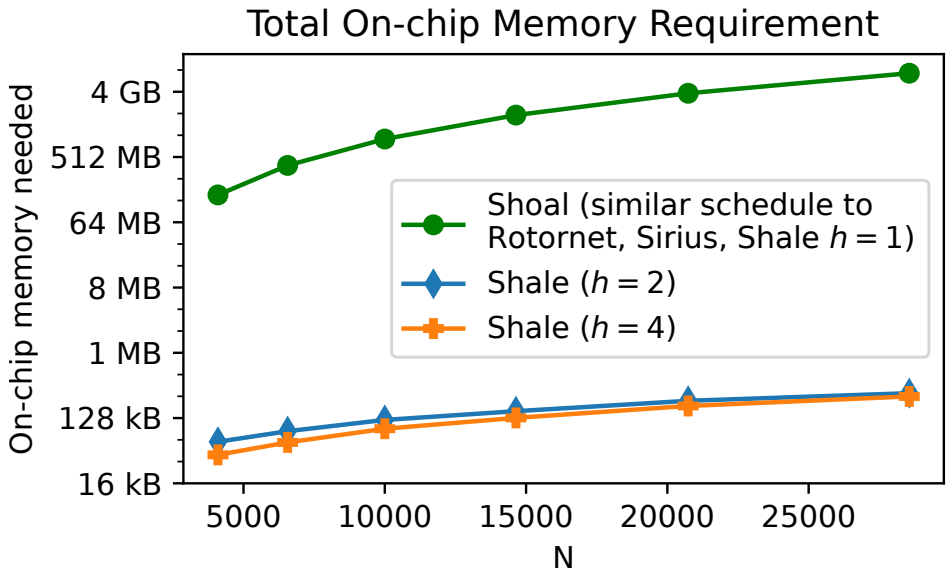


Figure 2.8: On-chip memory required by Shoal (representative of Rotornet and Sirius due to similar schedules and routing) and Shale for $h = 2$ and $h = 4$.

Takeaways. Shale with $h > 1$ uses orders of magnitude fewer hardware resources at datacenter scales than existing ORNs such as Rotornet, Sirius, and Shoal.

2.4 Evaluation

We evaluate Shale using a custom packet-level simulator. In our simulation setup, end-hosts are equipped with a 400 Gbps network interface composed of eight 50 Gbps lanes, a configuration used by 400 Gbps interfaces [55]. We assume a slot time of 45.056 ns, composed of 40.96 ns of usable slot time, followed by a 4.096 ns guard band. The usable slot time of 40.96 ns, combined with the per-lane bandwidth of 50 Gbps, results in a total cell size of 256 bytes. We reserve 12 bytes of each cell for the cell header, resulting in a payload of 244 bytes. Similar to [68], we take advantage of the eight lanes offered by each link to connect to neighbors in parallel, allowing a new timeslot to begin every 5.632 ns on average. The bandwidth of each lane, as well as size of the guard band, are achievable by an optical ORN based on tunable lasers, as demonstrated in [7]. To approximate

a datacenter-scale network, we use a propagation delay of 0.5 μs, or 89 timeslots, and simulate 10,000 nodes except where otherwise specified.

We use synthetic workloads with flow sizes modeled after published datacenter traces. Flows arrive according to a Poisson process, and sources and destinations are chosen with uniform probability across all nodes. The first workload [10], which we call the *short flow workload*, uses the flow size distribution reported in a measurement study of production datacenters. The largest flow size in this workload is 3 MB, and it produces primarily path-collision congestion. We run its simulations for 2 million timeslots. The second workload [22], which we call the *heavy-tailed workload*, mimics a data mining workload. This workload features flow sizes of up to 1 GB, and thus produces significant levels of egress congestion. We run its simulation for 50 million timeslots.

To demonstrate performance under high utilization, we run our workloads at load factors near the theoretical throughput guarantee. The load factor L is defined as the average sending rate at each node, divided by the total available bandwidth at each node. On $h = 2$, we use a load factor of $L = 0.24$, while on $h = 4$ we use $L = 0.12$, approaching the throughput guarantee of each.

In order to evaluate performance across a wide range of flow sizes, we normalize each flow's completion time based on its size. For a flow with a length of F cells in a system with a propagation delay of P timeslots, the amount of time it would take to transmit the flow at line rate across one hop is $F + P$. We divide the actual flow completion time t by this value to compute the **size-normalized FCT**, $\frac{t}{F+P}$. After normalizing each individual flow's FCT, we divide flows into buckets based on their flow size and compute statistics independently for each bucket.

2.4.1 Hardware Prototype and Simulator

We begin by validating both our hardware prototype and our packet-level simulator by simulating identical permutation workloads on both, shown in Figure 2.9. To ensure that the hardware simulation is tractable, we simulate only 16 end hosts, using both $h = 2$ and $h = 4$ schedules. We simulate our hardware prototype using the ModelSim FPGA simulator [52]. Our simulation uses a clock speed of 156.25 MHz (resulting in a clock cycle of 6.4 ns) and 10 Gbps links, as found on the Terasic DE5-Net board equipped with Altera Stratix V FPGA [72]. To connect the nodes, we also implement a switch which connects these nodes according to Shale's connection schedule. We simulate a propagation delay of 2.5 μs and use 512B cells. All hosts are synchronized to the same clock. Accounting for overheads, we begin a new timeslot every 68 cycles, resulting in an available bandwidth of 9.412 Gbps.

Takeaways. Both our hardware prototype and packet simulator achieve almost exactly the same throughput and maximum queue lengths in these simulations (the differences are due to different randomization in spraying). For both $h = 2$ and $h = 4$, the throughput is above the theoretical guarantee of 2.353 Gbps and 1.176 Gbps, respectively. This helps confirm that both are correctly implemented, and that the hardware optimizations described in Section 2.3.1 do not impact correctness.

2.4.2 Interleaving

Figure 2.10 shows the results of simulating the heavy-tailed workload on various interleaved schedules with 10,000 nodes. For each schedule, the s parameter indicates what portion of the timeslots are used by the higher-h sub-schedule, which is used by short flows.

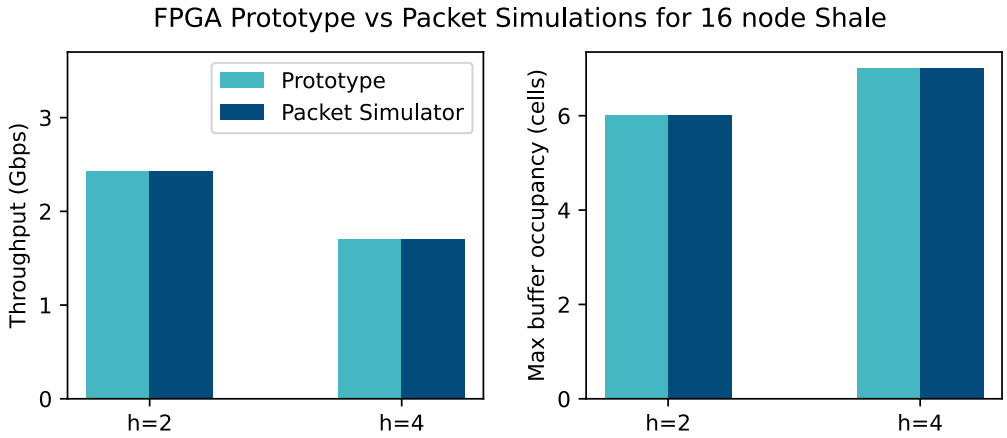


Figure 2.9: Throughput and queuing for the hardware prototype compared to packet simulations.

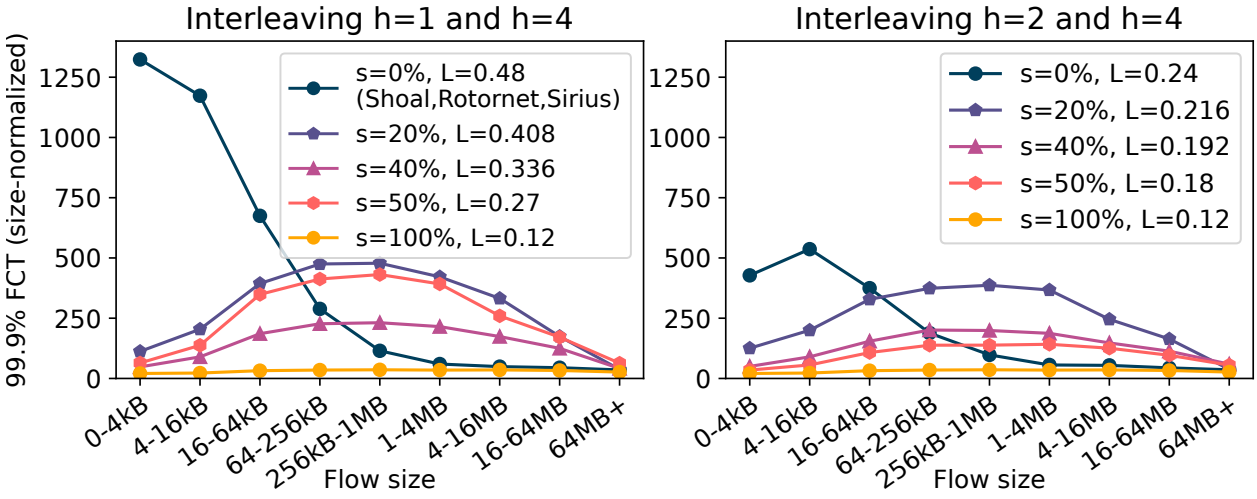


Figure 2.10: Size-normalized FCTs for the heavy-tailed workload under various interleaved schedules. s is the portion of timeslots allocated to the $h = 4$ subschedule, while L is the load factor used for each schedule.

Longer flows are sent on the lower-h sub-schedule which uses the remaining timeslots. The cutoff length is chosen to allow equivalent utilization of both for our given workload. Here, we run each schedule at almost the theoretical throughput limit, demonstrating increased throughput with interleaving. Note that when $s = 100\%$, the higher-h sub-schedule is used for all traffic, and when $s = 0\%$, only the lower-h sub-schedule is used. For $h = 1$, the schedule is identical to Rotornet, Shoal, and Sirius.

Interleaving allows the lower short-flow latencies of higher-h schedules to be achieved

while supporting much higher throughput. As fewer timeslots are allocated to the higher-h subschedule, the time taken for each subschedule iteration increases, usually resulting in longer latencies. At the same time, the reduced flow-size cutoff means that more flows are allocated to the low-h subschedule instead. This can occasionally improve latency in the higher-h subschedule, as with the $s = 40\%$ schedule outperforming the $s = 50\%$ schedule when interleaving $h = 1$ and $h = 4$ on this workload.

Takeaways. Interleaving allows Shale to simultaneously and scalably achieve high throughput and low latency. Depending on the specific requirements, network operators can choose an interleaving that balances latency and throughput.

2.4.3 Congestion Control Mechanism

We first evaluate our congestion control mechanisms, demonstrating the effectiveness of the combined **`HBH+spray`**.

Baselines. We use four alternative mechanisms as baselines:

1. **`none`** provides a baseline of Shale with no additional congestion control. Shale implicitly prioritizes forwarded cells over newly originated cells, providing a primitive admission control (which remains active in other designs).
2. **`priority`** provides a baseline of Shale with in-network scheduling. Prior works [2] have shown that sending packets from shortest flow first results in near-optimal mean flow completion times. In **`priority`**, when a cell arrives at a node, it is assigned a priority value $p = t + hE$ based on its arrival time t, the overall size h of

the flow to which it belongs (preventing starvation), and the epoch length E. Cells with lower priorities are sent first.

3. **`ISD`** (Idealized Sender-Driven) provides a very optimistic upper bound for the performance achievable by an end-to-end sender-driven congestion control mechanism which aims to achieve fair sharing between senders, such as the TCP suite of algorithms. In this design, nodes have clairvoyant, up-to-date knowledge of how many flows are currently being sent to each destination in the network. Whenever a node starts or finishes sending a flow, the global view of flows is immediately updated for all nodes. Nodes limit their sending rate to ensure that the total amount of traffic being sent to each receiver does not exceed a maximum bandwidth R. We use an R of $\frac{1.25}{2h}$, a low value that still allows the throughput bound to be reached.

4. **`RD`** and **`NDP`** are both based on the receiver-driven transport protocol used by NDP [28], but simplified to only use PULL messages. **`NDP`** also uses packet trimming, providing a close approximation to NDP within the context of Shale. As with **`hop-by-hop`**, this protocol reserves a portion of each timeslot to send small control messages. However, here control messages are sent end-to-end using the same routing as regular cells. To reduce the queuing encountered by these control messages, only one PULL message is sent for every 20 cells received from each sender.

   NDP has good performance when used with packet spraying on packet-switched networks. In this environment, queuing is primarily due to incast, occurring between the destination top-of-rack switch (TOR) and end-host [28]. This greatly differs from Shale, where queuing can occur throughout the path. For our **`NDP`**, we use a maximum queue length of 100 to prevent too many cells from being dropped and reducing throughput due to retransmissions. Despite this conservative cutoff, when running the heavy-tailed workload under $h = 4$, over 3% of packets are dropped.

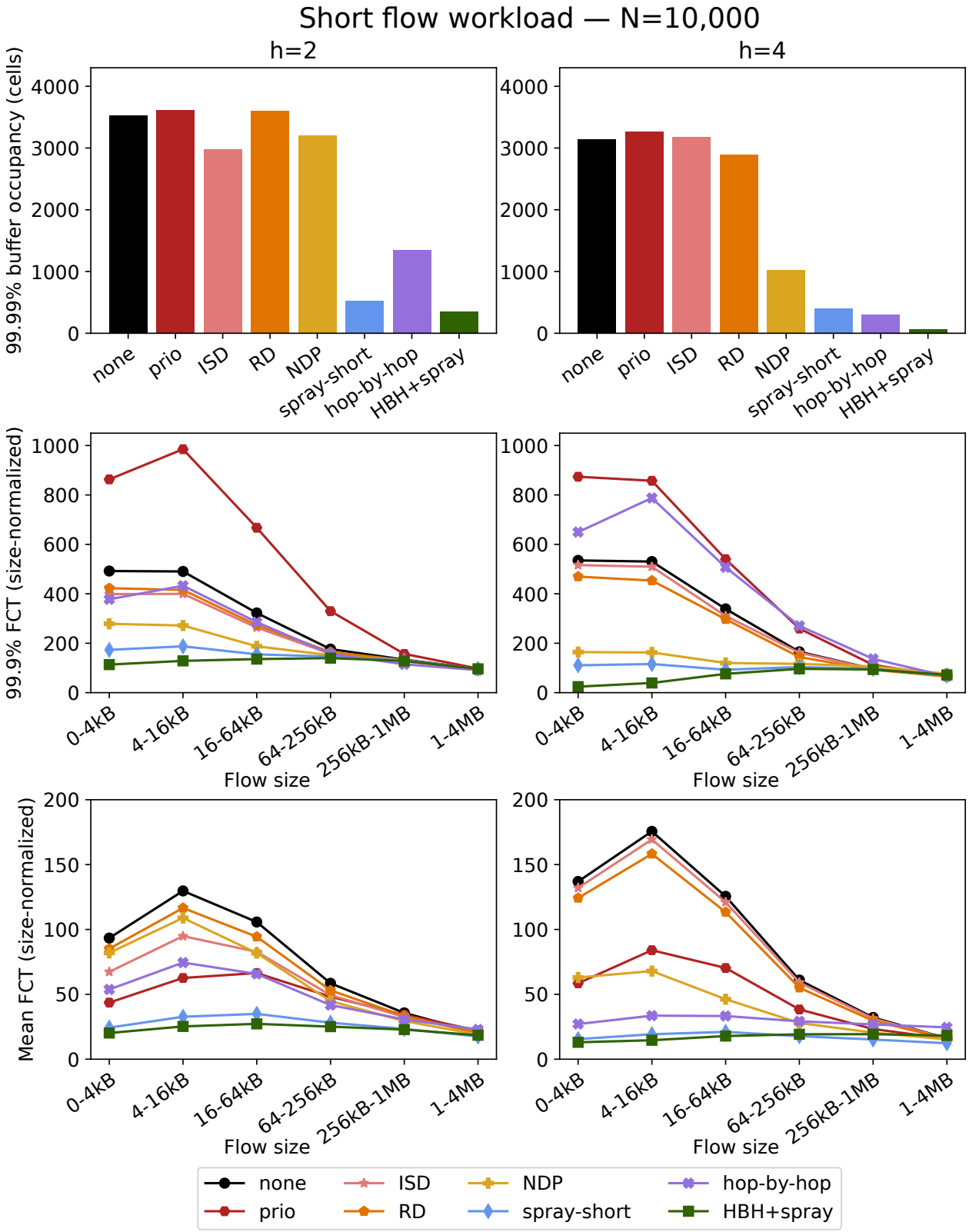


Figure 2.11: Buffer occupancies and normalized flow completion times for the short flow workload.

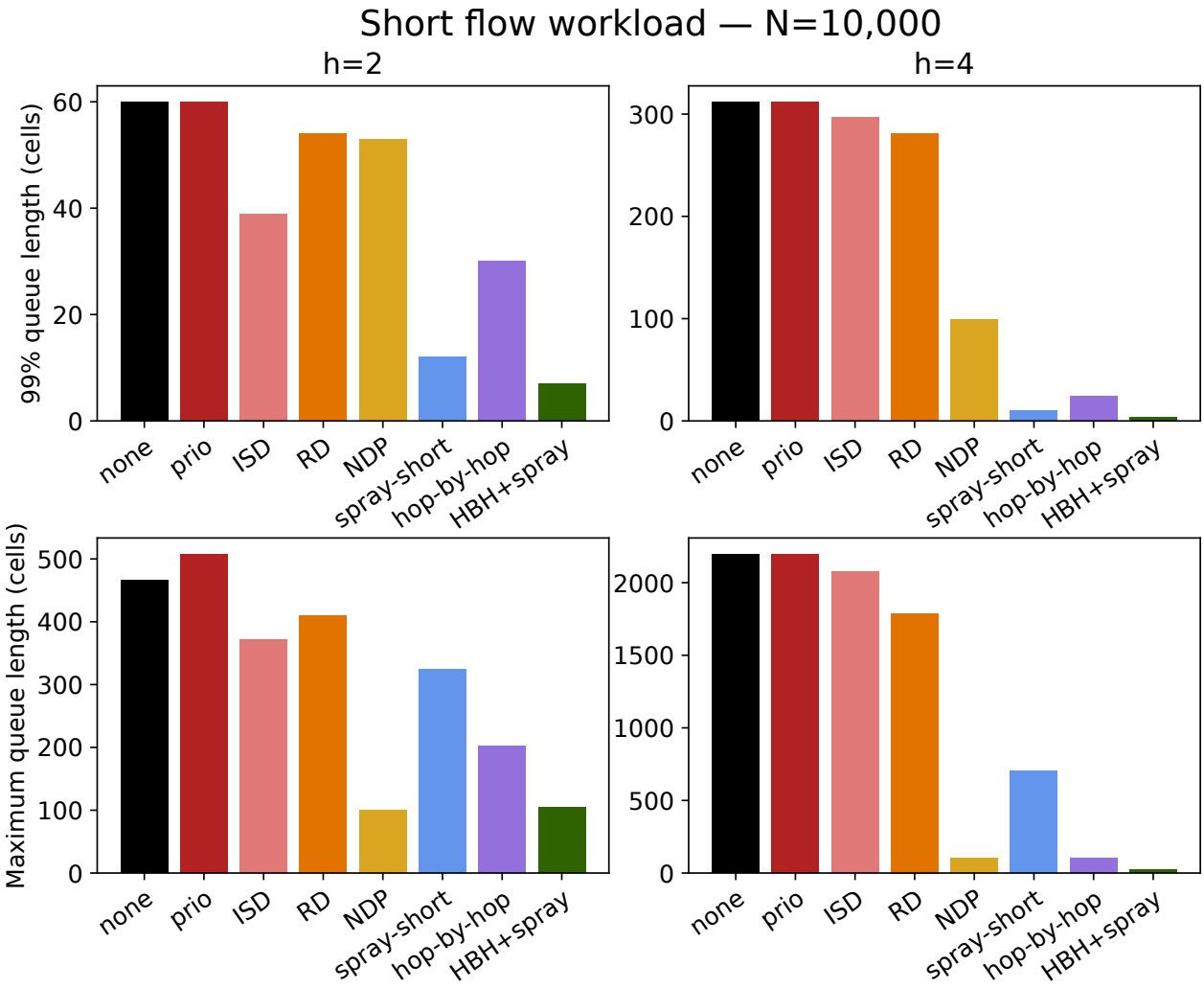


Figure 2.12: Queue lengths observed during the short flow workload

Short flow workload. We begin by examining the performance of our various congestion control mechanisms and baselines using the short flow workload, which primarily creates path collision congestion.

The lower graphs in Figure 2.11 show the FCTs achieved by various congestion control mechanisms for the short flow workload. As expected, **`spray-short`** is quite effective at improving tail flow completion times for all flow size categories. While **`hop-by-hop`** on its own is not well-suited for this workload, as it mainly addresses egress congestion, it still limits queuing, especially for $h = 4$. The best queuing and FCTs are achieved by the combined **`HBH+spray`**.

priority's scheduling policy optimizes mean flow completion time [2]; here, this comes at the expense of tail latency. **ISD** and **RD** are clearly mismatched with this workload, having only marginal differences compared to **none**. This demonstrates the need to address path-collision congestion using in-network interventions, rather than end-to-end rate limiting. While **NDP** performs decently in this test, it is not quite as effective as **spray-short** and **HBH+spray**.

Figure 2.12 shows the tail and maximum queue lengths observed during this workload. Even though **NDP** had a lower maximum queue length than **spray-short** for both $h = 2$ and $h = 4$. This made cells more likely to encounter long queuing delays.

The top two graphs in Figure 2.11 show the tail buffer occupancies, or total number of enqueued cells, at each node over the course of the simulation. As with flow completion times, **spray-short** is highly effective at reducing buffer occupancy for this workload. Combining with **hop-by-hop** results in even lower buffer occupancy, particularly for $h = 4$.

Finally, we measured throughput over the course of the workload. For this workload, following an initial ramp-up period, all mechanisms achieved average throughput within 2.5% of L, the target load. In particular, **spray-short** does not negatively affect the throughput of this workload.

Heavy-tailed workload. This workload produces considerable egress congestion, leading to FCTs and queuing differing by multiple orders of magnitude between different congestion control mechanisms.

The lower graphs in Figure 2.13 show tail FCTs. For this workload, **hop-by-hop** is effective at reducing tail latencies. In particular, short flows have their FCTs reduced by 2-3 orders of magnitude compared to **none**. **HBH+spray** brings an additional improvement.

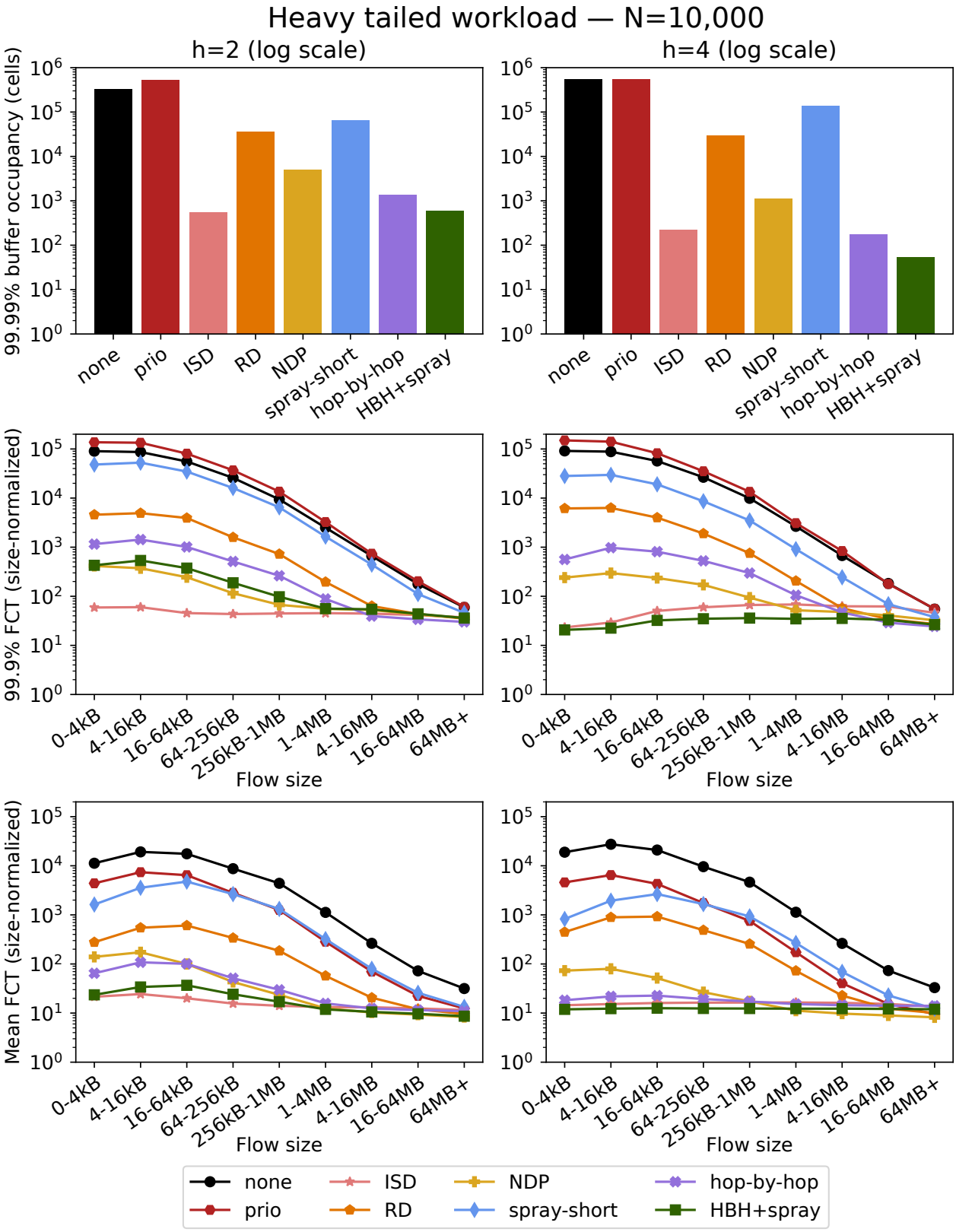


Figure 2.13: Buffer occupancies and normalized flow completion times for the heavy-tailed workload.

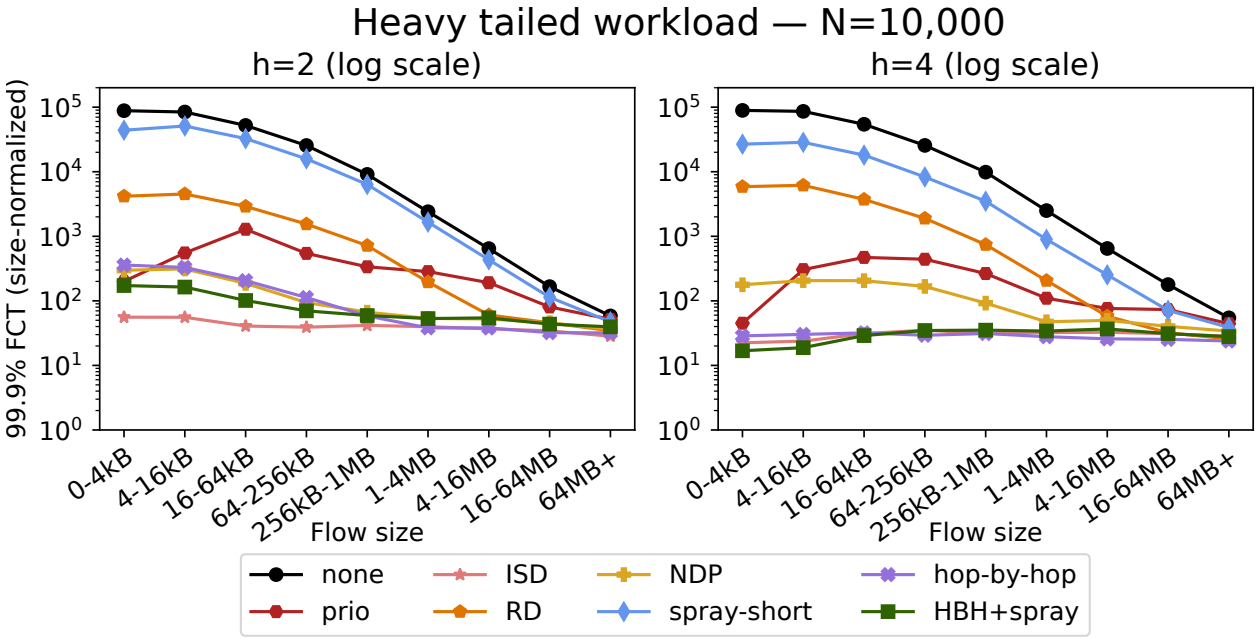


Figure 2.14: Tail latencies for the heavy-tailed workload, ignoring flows that experience incast with very long (>256 MB) flows.

In particular, for $h = 4$, **HBH+spray** achieves tail normalized FCTs of under 24 for short flows. Because of $h = 4$'s use of 8-hop paths, this is within 3x of the theoretical limit without queuing.

For $h = 2$, while **HBH+spray** limits queuing, tail latencies are slightly higher than for **NDP** and significantly higher than the idealized **ISD**. This is due to cases where short flows are incasted with long flows. **hop-by-hop** treats all cells to the same destination identically, so cells from short flows experience the same egress congestion as cells from incasted long flows. At our system size, Shale with $h = 2$ has an order of magnitude more neighbors at each node than $h = 4$, allowing longer per-bucket queues to build up at each node. Figure 2.14 shows the tail latencies only for flows that are not being incast to the same destinations as ongoing flows above 256 MB. When those flows are excluded, **HBH+spray** with $h = 2$ performs more similarly to **ISD**.

The top graphs in Figure 2.13 show the observed tail buffer occupancy. While **spray-short** is able to slightly reduce tail buffer occupancies, by maintaining its queuing

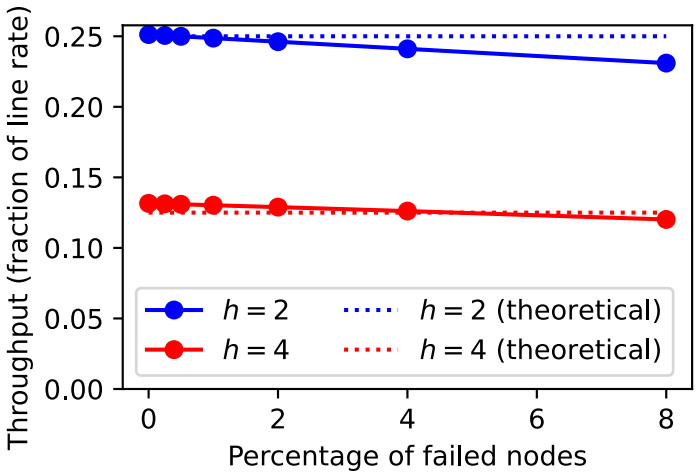


Figure 2.15: Throughput achieved with 10K nodes under failures, and the lower bound without failures.

invariant, **hop-by-hop** is able to reduce them by several orders of magnitude, outperforming both **RD** and **NDP**. **HBH+spray** achieves even better performance: for $h = 4$, nodes at the 99.99th percentile had fewer than 100 cells enqueued in total.

Takeaways. Shale's congestion control, **HBH+spray**, outperforms all of our baselines on both the short flow workload, and on the heavy-tailed workload when $h = 4$. In the latter case, **HBH+spray** achieves over an order of magnitude better short flow FCTs and buffer occupancy than **NDP**. For the heavy-tailed workload on $h = 2$, it is only significantly outperformed by the idealized **ISD** baseline.

2.4.4 Performance under Failures

We evaluate the effect of node failures on throughput for 10K nodes with Shale $h = 2, 4$. We use a synthetic workload consisting of 10 overlaid permutation traffic matrices to specifically benchmark the effect on throughput. Our permutations do not include the failed nodes, ensuring that we properly measure the throughput achievable by the remaining nodes. We run this simulation for 2 million timeslots and report the average

destination throughput over the course of the run. Figure 2.15 shows the results of our experiment.

Takeaways. Under node failures, throughput is reduced roughly in proportion to the number of failed nodes. As long as most nodes are active, good throughput is maintained.

2.4.5 Scalability

To demonstrate the scalability of Shale, we consider two metrics: (i) the hardware resource requirements to buffer packets and (ii) the tail flow completion times. We simulate the short flow workload with systems of various sizes between 4,096 nodes and 50,625 nodes, as shown in Figure 2.16. The top graphs show the maximum number of active buckets and maximum PIEO queue length observed, two important values for determining the hardware resources required (as discussed in Section 2.3.3). The bottom graphs show the 99.9% size-normalized FCTs.

Even when scaling system size by over an order of magnitude, $h = 2$ Shale only uses 2.5 times more active buckets, and PIEO queue lengths appear to plateau. $h = 4$ Shale is even more resource efficient, with the number of active buckets remaining nearly flat, and queue lengths increasing by only around 50%. For $h = 2$, there is at most a 2x increase in short-flow FCTs, while for $h = 4$, Shale maintains almost exactly the same short flow latencies.

Takeaways. Shale has exceptional scalability. Both latencies and hardware resource requirements are similar even when scaling the system size by over an order of magnitude, especially for $h = 4$.

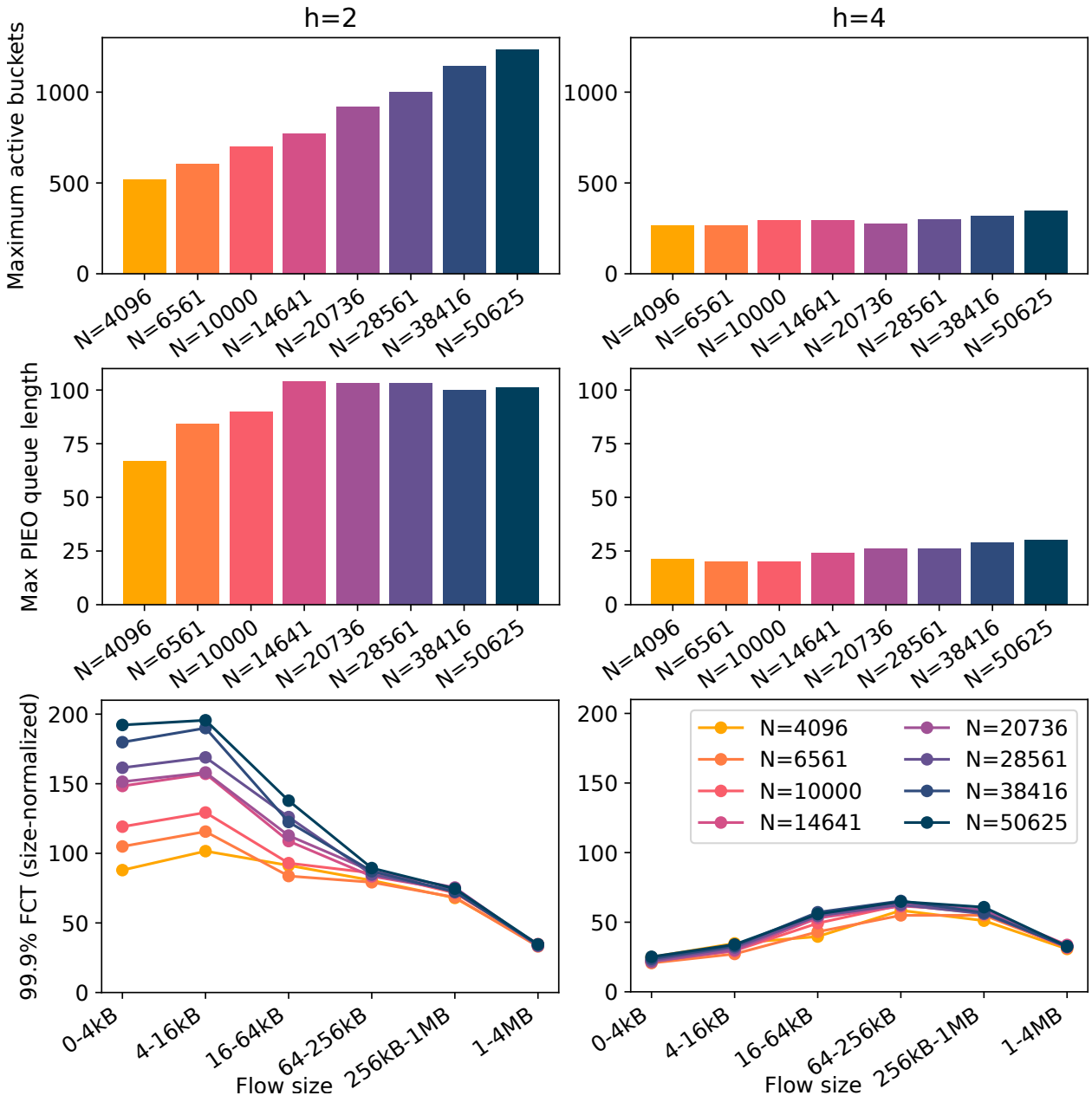


Figure 2.16: Active buckets, queue lengths, and FCTs for the short flow workload as system size scales.

2.5 Summary

This chapter examines how to design an ORN that is capable of operating at datacenter scale while supporting both low latency and high throughput. We develop Shale, an ORN design that aims to achieve this goal. Shale generalizes Single Round Robin Design (SRRD)-based ORN designs to achieve a tunable tradeoff between throughput and latency scaling that is Pareto optimal among all ORN designs (up to a constant factor). We show that by interleaving multiple tunings in parallel, both latency- and throughput-sensitive flow can be routed on the most advantageous schedule. To address queuing, we develop congestion control mechanisms tailored to the unique context of multi-hop ORNs. We also extend Shale's congestion control to communicate node and link failures, and showed that Shale continues to operate well even under failures. Finally, we implement an FPGA-based prototype end-host, showing that Shale can scale to datacenter-sized networks while using orders of magnitude fewer hardware resources than existing ORN designs.

To evaluate Shale, we create both an FPGA-based hardware prototype and a custom packet-level network simulator which can efficiently simulate tens of thousands of nodes. Both of these are available under an open-source license at `https://reconfigurable-networks.github.io/shale`.

CHAPTER 3

OPTIMAL OBLIVIOUS RECONFIGURABLE NETWORKS

This chapter presents our mathematical investigation of the latency-throughput tradeoffs possible with ORNs. We fully characterize which tradeoffs are possible, up to a constant factor. This work first appeared in *Optimal Oblivious Reconfigurable Networks*, published in the ACM Symposium on Theory of Computing (STOC) 2022 with co-authors Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal; as well as *Extending Optimal Oblivious Reconfigurable Networks to all N*, published in the SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS) 2023 with co-authors Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, and Robert Kleinberg.

3.1 Introduction

Oblivious Reconfigurable Networks (ORNs) represent an enticing design paradigm for datacenter network development. However, the space of possible ORN designs is poorly understood. Early designs have been highly similar to each other, limiting a broader understanding of what is possible with ORNs. Even as new designs, such as Shale, are created, it is difficult to evaluate how successful they are without a more rigorous theoretical understanding of the broader design space. While oblivious routing in general has a large body of theoretical research, thus far it has only been studied in the context of static networks, where the links in the network are fixed at the beginning and do not change over time. In this chapter, we make the first attempt to formally study the problem of oblivious routing in the context of reconfigurable networks.

There are two key objectives that ORNs must aim to optimize. First, since it is costly to overprovision networks (especially for modern high-bandwidth links), datacenter network operators aim for extremely high throughput, utilizing a large constant factor of the

available network capacity at all times if possible. Second, it is desirable to minimize latency, the worst-case delay between when a packet arrives to the network and when it reaches its destination. There is a vital need to understand oblivious network designs for reconfigurable networks that guarantee high throughput and low maximum latency.

The objectives of maximizing throughput and minimizing latency in reconfigurable networks are in conflict: due to degree constraints, most nodes cannot be connected by a direct link at all times. One must either use indirect paths, which comes at the expense of throughput, or settle for higher latency while waiting for reconfigurations to yield a more direct path. Since different deployments and applications may necessitate different tradeoffs between these two conflicting objectives, the main question that our work investigates is the following:

> *For every throughput rate r, what is the minimum latency achievable by an oblivious reconfigurable network design that guarantees throughput r?*

We fully resolve this question to within a constant factor[1] for *d-regular* reconfigurable networks[2]. That is, for every constant rate r, we identify a function $L^*(r, N)$ such that any N-node d-regular reconfigurable network guaranteeing throughput r must have maximum latency bounded below by $\frac{1}{d}L^*(r, N)$. Complementing this lower bound, we design oblivious networking schemes that guarantee throughput r and have maximum latency bounded by $O(\frac{1}{d}L^*(r, N))$, for every constant $r \in (0, \frac{1}{2}]$, $d \in \mathbb{N}$, and infinitely many N. These designs together provide an upper bound which is tight with our lower bound up to

[1] One could, of course, ask the transposed question: *for every latency bound L, what is the maximum guaranteed throughput rate achievable by an oblivious routing scheme with maximum latency L?* Our work also resolves this question, not only to within a constant factor, but up to an additive error that tends to zero as $N \to \infty$. As noted below in Section 3.1.2, optimizing throughput to within a factor of two, subject to a latency bound, is much easier than optimizing latency to within a constant factor subject to a throughput bound. The importance of the latter optimization problem, *i.e.* our main question, is justified by the high cost of overprovisioning networks: due to the cost of overprovisioning, datacenter network operators tend to be much less tolerant of suboptimal throughput than of suboptimal latency.

[2] except when d is very large — bounded below by a constant power of N

a constant factor. Note that some of these designs formalize the schedule and routing scheme of Shale, as described in Section 2.2.1, proving that Shale's ORN design is Pareto optimal up to a constant factor.

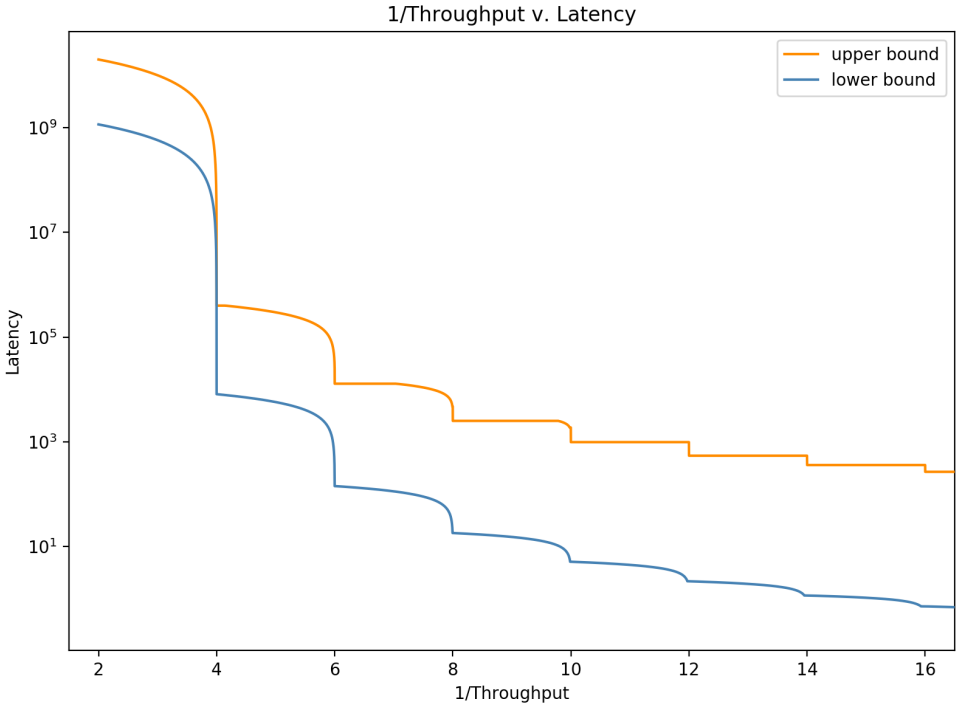


Figure 3.1: A plot of the upper and lower bounds for the latency of an ORN containing 10^9 nodes that can guarantee a given throughput.

The shape of the optimal tradeoff curve between throughput and latency is quite surprising. Figure 3.1 depicts the curve for $N = 10^9$ and $d = 1$; the x-axis measures the inverse throughput, $1/r$, while the y-axis (in log scale) measures maximum latency. The curve is scallop-shaped, with particularly favorable tradeoffs occurring when $1/r$ is an even integer. Between even-integer values of $1/r$, the maximum latency improves slowly at first, then precipitously as $1/r$ approaches the next even integer. The proof of our main result explains these key features of the tradeoff curve: its non-convexity, the special role played by even integer values of $1/r$, and the steep but continuous improvement in $L^*(r, N, d)$ as $1/r$ approaches the next even integer. In Section 3.1.2 below we sketch the intuitions that account for these features. Before doing so, we pause to explain more fully

our model and notation.

3.1.1 Our Model and Results

Our model of ORNs aims to formalize the description of ORNs given in Section 1.2. We assume that a network consists of N nodes, connected using circuit switches. The switches reconfigure periodically, with the time between reconfigurations referred to as a *timeslot*. An *ORN design* in our model is specified by two ingredients: a *connection schedule* and an *oblivious routing scheme*.

The connection schedule designates which pairs of nodes are connected in each timeslot. We visualize this in the form of a *virtual topology*: a directed graph in which each node represents a particular network node during a particular timeslot. The virtual topology allows us to encode paths that network traffic can take, including the precise timing in which each link is traversed. It is composed of *physical edges*, which represent sending traffic during a given timeslot, as well as *virtual edges*, which represent traffic remaining put on a given node during a given timeslot. For the majority of this chapter, we limit our model to 1-regular connection schedules. We generalize our results to d-regular schedules in Section 3.2.2 and Section 3.6.

The oblivious routing scheme determines how network traffic is routed over the virtual topology. For each source-destintaion pair (a, b) and timeslot t, the oblivious routing scheme designates a set of routing paths that begin at node a and timeslot t, and eventually reach destination b. Additionally, it includes a probability for each of these paths to be used, such that the total probability across all paths for a given source-destination pair and timeslot sums to 1.

We evaluate ORN designs according to two quantities: maximum latency (L) and

guaranteed throughput (r). Latency of a path measures the difference between the timeslots in which it begins and ends. An ORN design with maximum latency L uses no routing paths of latency greater than L.

The definition of guaranteed throughput is more subtle. First, we model traffic demand using a function that specifies, for each source-destination pair and each timeslot, the amount of flow with that source and destination originating at that time. We say that an ORN design *guarantees throughput r* if the routing scheme is guaranteed not to exceed the capacity of any link, whenever the traffic demand satisfies the following property: At every timeslot, the total amount of demand originating from any source, or bound to any destination, never exceeds r. Our main result can now be stated in the following form.

Theorem 1. *Consider any constant $r \in (0, \frac{1}{2}]$. Let (h, ε) to be the unique solution in $\mathbb{N} \times (0, 1]$ to the equation $\frac{1}{2r} = h + 1 - \varepsilon$, and let $L^*(r, N)$ be the function*

$$L^*(r, N) = h\left(N^{1/(h+1)} + (\varepsilon N)^{1/h}\right).$$

For every $N > 1$ and every ORN design on N nodes that guarantees throughput r, the maximum latency is at least $\Omega(L^*(r, N))$. Furthermore for infinitely many N there exists an ORN design on N nodes that guarantees throughput r and whose maximum latency is $O(L^*(r, N))$.

3.1.2 Techniques

To begin reasoning about the latency-throughput tradeoff in ORNs, we make the following counting argument to prove a weak lower bound. First, we note that for any node in the virtual topology, the number of distinct routing paths originating at that node which use exactly p physical edges and have maximum latency L is $\binom{L}{p}$. In order for a node to be able to reach a majority of other nodes within L timeslots using at most h physical edges, we must satisfy the inequality $\sum_{p=0}^{h} \binom{L}{p} \geq N/2$. A simple calculation verifies that

this inequality implies $L = \Omega\left(hN^{1/h}\right)$. At the same time, a routing scheme in which the routing path between a random source and a random destination contains h physical links, on average, cannot guarantee throughput greater than $1/h$. This suggests a latency-throughput relationship of the form $L = \Omega\left(\frac{1}{r}N^r\right)$. This lower bound can be made rigorous with a little bit of work, but it differs from the tight bound asserted in Theorem 1 in two significant ways.

1. Whereas $\frac{1}{r}N^r$ is a smooth convex function for $r > 0$, the function $L^*(r, N)$ is non-smooth and non-convex; when plotted as a function of $1/r$ it exhibits a scalloped shape with cusps at even integer values of $1/r$.
2. The exponent of N in the function $L^*(r, N)$ is approximately $2r$, rather than r. In other words, the naive bound $L \geq \frac{1}{r}N^r$ is tight up to a factor of 2 in terms of throughput, but off by a factor of about N^r in terms of latency. (As remarked in Footnote 1, sacrificing a factor of 2 in throughput is typically regarded by network operators as much more costly than sacrificing a constant factor in latency.)

The first of these differences is explained by a refinement of the counting argument at the start of this section. In order to guarantee throughput r, the average number of physical hops on the routing paths used (under any traffic demands with at most r units of flow based at any source or destination) must be at most $1/r$. However, the number of physical hops in any path must be an integer. Thus, if $1/r$ is not an integer, at least a constant fraction of routing paths must have $\lfloor 1/r \rfloor$ physical hops or fewer. Subject to any upper bound on latency, paths with a limited number of physical hops are much less numerous than those with a larger number of physical hops, so the requirement to use a large number of distinct paths with $\lfloor 1/r \rfloor$ or fewer physical hops places a significantly stricter lower bound on maximum latency, leading to the non-convex shape with regularly spaced cusps depicted in Figure 3.1.

To give intuition for the factor-two difference in throughput between the naïve lower bound and the true function $L^*(r, N)$, it is useful to recall *Valiant load balancing (VLB)*, an ingredient in many of the earliest and most practical oblivious routing schemes. VLB constructs a random path from source s to destination t by choosing a random intermediate node, r, and concatenating minimum-cost paths from s to r and from r to t. This inflates the number of physical hops used in routing paths by a factor of two, but is beneficial because it prevents congestion under worst-case demands. The fact that the exponent of N in $L^*(r, N)$ is approximately $2r$ rather than r can be interpreted as confirming that the factor-two inflation due to VLB is unavoidable, for oblivious routing schemes that guarantee throughput r. To prove this fact, we formulate optimal oblivious routing for a given virtual topology as a linear program and interpret the dual variables as endpoint-specific edge costs that can be summed to ascribe a cost to every path connecting a given pair of endpoints. We prove that, regardless of the virtual topology, one can always design a carefully-constructed dual solution that penalizes paths containing a large number of physical hops, and doubly penalizes physical hops that are too close to both endpoints. Paths that avoid the double penalty must use twice as many physical hops as minimum-cost paths, exactly as in VLB routing. The most delicate part of the proof is the verification that the dual solution is feasible, which requires carefully bounding the number of nodes reachable from any source within a given cost budget.

To prove that the lower bound $L^*(r, N)$ is tight, we need to construct an ORN design that matches the bound up to a constant factor. Our design is easiest to describe when $r = \frac{1}{2h}$ and $N = n^h$ for positive integer h and prime number n. In that case, we use a design that we call the *Elementary Basis Scheme* (EBS), which formalizes the schedule and routing scheme of Shoal described in Section 2.2.1. EBS identifies the set of N nodes with elements of the group[3] $(\mathbb{Z}/(n))^h$. Let $\mathbf{e}$ be the elementary basis consisting of the columns of the $h \times h$

[3]This should be thought of as the h-dimensional vector space over $\mathbb{Z}/(n)$. While we describe taking the elementary basis for simplicity here, the EBS scheme itself does not require $\mathbb{Z}/(n)$ to be a field, thus we use the word group here.

identity matrix. EBS uses a connection schedule whose timeslots cycle through the nonzero scalar multiples of elements of $\mathbf{e}$. In a timeslot devoted to $s \cdot \mathbf{e}_i$, the network is configured to allow each node x to send to $x + s \cdot \mathbf{e}_i$. Over the course of one complete cycle, any two nodes can be connected by a "direct path" consisting of h physical hops (or fewer) that modify the coordinates of the source node one by one until they match the coordinates of the destination. The EBS routing scheme constructs a random path connecting a given source and destination using VLB: it chooses a random intermediate node and concatenates two "semi-paths": the direct paths from the source to the intermediate node and from the intermediate node to the destination.

To generalize this design to all non-integer values of $\frac{1}{2r}$, we need to enhance EBS so that a constant fraction of semi-paths use h physical hops and a constant fraction use $h + 1$ physical hops. This necessitates a modified ORN design that we call the *Vandermonde Bases Scheme* (VBS). Assume $r = h + 1 - \varepsilon$ for $h \in \mathbb{N}, 0 < \varepsilon < 1$, and that $N = n^{h+1}$ for prime n, so that the nodes can be identified with the vector space $\mathbb{F}_n^{h+1}$. Instead of one basis corresponding to the identity matrix, we now use a sequence of distinct bases each corresponding to a different Vandermonde matrix. In addition to the single-basis semi-paths (which now constitute $h + 1$ physical hops), this enables the creation of "hop-efficient" semi-paths composed of h physical hops belonging to two or more of the Vandermonde matrices in the sequence. Hop-efficient semi-paths have higher latency than direct paths, but we opportunistically use only the ones with lowest latency to connect a subset of terminal pairs, joining the remaining pairs with direct semi-paths. A full routing path is then defined to be the concatenation of two random semi-paths, as before. Proving that the routing scheme guarantees throughput r boils down to quantifying, for each physical edge e, the net effect of shifting load from direct paths that use e to hop-efficient paths that avoid e and vice-versa. The relevant sets of paths in this calculation can be parameterized by unions of affine subspaces of $\mathbb{F}_n^{h+1}$, and the use of Vandermonde matrices in the connection schedule gives us control over the dimensions of intersections of these subspaces, and thus

over the size of their union.

3.2 Definitions

This section presents definitions that formalize the notion of an oblivious reconfigurable network (ORN). We assume a network of N nodes communicating in discrete, synchronous timeslots. The nodes are joined by a communication medium that allows an arbitrary pattern of unidirectional communication links to be established in each timeslot, subject to a degree constraint that each node participates as the sender in at most d connections, and as the receiver in at most d connections. Throughout most of this chapter we specialize to the case $d = 1$; see Section 3.2.2 below for a discussion of why the general case reduces to this special case.

In systems that instantiate reconfigurable networking, data is encapsulated in fixed-size units called *cells* or *packets*. In this work we instead treat data as a continuously-divisible commodity, and we allow sending fractional quantities of flow along multiple paths from the source to the destination. This abstraction is standard in theoretical works on oblivious routing, and it can be justified by interpreting a fractional flow as a probability distribution over routing paths, with each discrete cell being sent along one path sampled at random from the distribution. Under this interpretation, flow values represent the expected number of cells traversing a link.

Definition 1. A *connection schedule* $\boldsymbol{\pi}$ with size N and period length T is a sequence of permutations $\pi_0, \pi_1, \ldots, \pi_{T-1}$, each mapping $[N]$ to $[N]$. The interpretation of the relation $\pi_k(i) = j$ is that node i is allowed to send one cell to node j during any timeslot t such that $t \equiv k \pmod{T}$.

Definition 2. The *virtual topology* of a connection schedule $\boldsymbol{\pi}$ is a directed graph $G_{\boldsymbol{\pi}}$ with

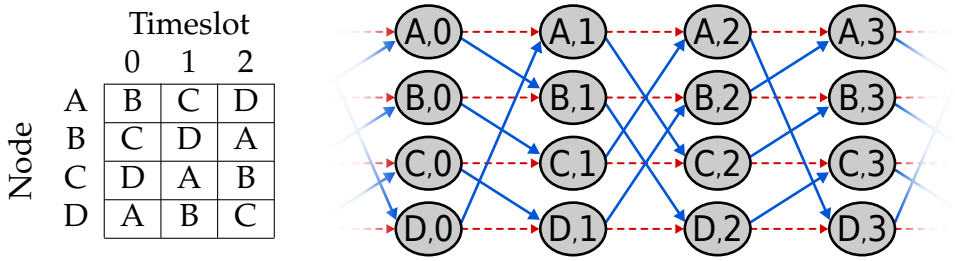


Figure 3.2: A connection schedule among four nodes, as well as part of its corresponding virtual topology. The full virtual topology represents a countably infinite number of timeslots.

vertex set $[N] \times \mathbb{Z}$. The edge set of G_π consists of the union of E_{virt} and E_{phys}. E_{virt} is the set of *virtual edges*, which are of the form $(i,t) \to (i,t+1)$ and represent the cell waiting at node i during the timeslot t. E_{phys} is the set of *physical edges*, which are of the form $(i,t) \to (\pi_t(i), t+1)$ and represent the cell being transmitted from i to $\pi_t(i)$ at timeslot t.

We interpret a path in G_π from (a,t) to (b,t') as a potential way to transmit a cell from node a to node b, beginning at timeslot t and ending at some timeslot t'. For a node $a \in [N]$ let $[\![a]\!]$ denote the set $\{a\} \times \mathbb{Z}$, consisting of all copies of a in G_π. Let $\mathcal{P}(a,b,t)$ denote the set of paths in G_π from the vertex (a,t) to $[\![b]\!]$. Finally, let $\mathcal{P} = \bigcup_{a,b,t} \mathcal{P}(a,b,t)$ denote the set of all paths in G_π.

Definition 3. A *flow* is a function $f : \mathcal{P} \to [0,\infty)$. For a given flow f, the amount of flow traversing an edge e is defined as:

$$F(f,e) = \sum_{P \in \mathcal{P}} f(P) \cdot \mathbf{1}_{e \in P}$$

We say that f is *feasible* if for every physical edge $e \in E_{\text{phys}}$, $F(f,e) \le 1$.

Definition 4. The *latency* $L(P)$ of a path P in G_π is equal to the number of edges it contains (both virtual and physical). Note that traversing any edge in the virtual topology (either virtual or physical) is equivalent to advancing in time by the duration of one timeslot, so the number of edges in a path is proportional to the elapsed time. For a nonzero flow f,

the *maximum latency* is the maximum over all paths in the flow

$$L_{max}(f) = \max_{P \in \mathcal{P}} \{L(P) : f(P) > 0\}$$

Definition 5. An *oblivious routing scheme* R is a function that associates to every $(a, b, t) \in [N] \times [N] \times \mathbb{Z}$ a flow $R_{a,b,t}$ such that:

1. $R_{a,b,t}$ is supported on paths from (a, t) to $[\![b]\!]$, meaning $\forall P \notin \mathcal{P}(a, b, t) \;\; R_{a,b,t}(P) = 0$.
2. $R_{a,b,t}$ defines one unit of flow. In other words, $\sum_P R_{a,b,t}(P) = 1$.
3. R has period T. In other words, $R_{a,b,t+T}$ is equivalent to $R_{a,b,t}$ (except with all paths transposed by T timeslots, as required to satisfy point 1).

Definition 6. A *demand matrix* is an $N \times N$ matrix which associates to each ordered pair (a, b) an amount of flow to be sent from a to b. A *demand function* D is a function that associates to every $t \in \mathbb{Z}$ a demand matrix $D(t)$ representing the amount of flow $D(t, a, b)$ to originate between each source-destination pair (a, b) at timeslot t. The *throughput requested by demand function* D is the maximum, over all t, of the maximum row or column sum of $D(t)$.

Definition 7. For a given oblivious routing scheme R and demand function D, the *induced flow* $f(R, D)$ is defined by:

$$f(R, D) = \sum_{(a,b,t) \in [N] \times [N] \times \mathbb{Z}} D(t, a, b) R_{a,b,t}.$$

Definition 8. An oblivious routing scheme is said to *guarantee throughput* r if the induced flow $f(R, D)$ is feasible whenever the demand function D requests throughput at most r.

Definition 8 can be interpreted as meaning that the network is able to simulate a "big switch" with N input and output ports having line rate r: as long as the amount of data originating at any node a or destined for any node b does not exceed rate r per timeslot, the network is able to route all data to its destination without violating capacity constraints.

3.2.1 Assumptions

The definitions in this section are based on the following implicit assumptions, which we employ to greatly simplify our analysis and make it tractable.

Fractional flow and no queueing We assume that flow is divisible into arbitrarily small fractional quantities which may be sent on multiple routes. One can interpret this fractional amount of flow as an expected number of cells traversing each edge.

Following this assumption, the ORN designs we construct in this chapter divide each unit of flow evenly across all possible routes. Additionally, they send small fractions of flow from multiple different paths across a single physical link, during a single timeslot. However, in a real network, each cell must traverse exactly one path, and only a single cell may traverse each link during a single timeslot. As a result, multiple cells may end up at the same intermediate node, all with paths that traverse the same outgoing link. In this case, cells must be sent one at a time, resulting in queuing delay. Accounting for this source of delay would require fundamentally reworking our model for ORNs and would make analysis extremely difficult, if not impossible. This queuing is best addressed using a congestion control mechanism, such as the one we develop in Section 2.2.3.

No propagation delay We assume no propagation delay. That is, we assume that all flow scheduled to traverse an edge in one timeslot t is received by the end of that timeslot, and could potentially traverse a new edge during the next timeslot $t + 1$.

In a real network, there is a delay between when a cell is transmitted on one hop, and when it is received and ready to be transmitted on its next hop. This delay is referred to as the propagation delay. To simplify our analysis, we ignore this source of delay in this chapter. However, our model could be enhanced to account for it by adjusting the virtual

topology. Rather than connecting physical edges from (i, s) to $(j, s + 1)$, they could instead connect to $(j, s + d_{ij})$, where d_{ij} is a whole number representing the propagation delay from i to j in timeslots. As in our basic model, nodes of the virtual topology in this enhanced model would be constrained to belong to at most one incoming and at most one outgoing physical edge, though if d_{ij} varies with i and j then the set of physical edges would no longer be described by a sequence of permutations.

3.2.2 Allowing degree $d > 1$ in a timeslot

While our formalization of ORNs describes networks in which nodes have a degree of 1 in every timeslot, it can be generalized to networks that support a d-regular connectivity pattern in each timeslot. When $d > 1$, we interpret a demand matrix D which requests throughput r as one in which the row and column sums of D are bounded above by dr.

The connectivity of $N \times \{t, t + 1\}$ is d-regular bipartite. By Kőnig's Theorem, this edge set can be decomposed into d edge-disjoint perfect matchings, which we use to "unroll" into d consecutive timeslots of a 1-regular ORN. Therefore, a d-regular ORN design which guarantees throughput r with maximum latency L unrolls into a 1-regular ORN design which guarantees throughput r with maximum latency dL.

Under this framework, a lower bound $L^*(r, N)$ for 1-regular ORN designs trivially implies the lower bound $\frac{1}{d}L^*(r, N)$ for d-regular designs. However, an upper bound for 1-regular designs does not necessarily imply a similar upper bound for d-regular designs, because the routing scheme could route paths containing two or more physical edges in timeslots belonging to the same "unrolled" segment of the 1-regular virtual topology. This would correspond to traversing two or more edges at once in the d-regular topology. We show in Section 3.6 that such a problem will never occur due to our construction. Specifically, we show that our construction can be modified to never allow flow to be

routed along two edges within any block of d consecutive timeslots, provided $d \leq N^{1/(h+1)}$. This modification will add a factor of at most 2 to the maximum latency. Then, by inverting the unrolling process, we will obtain a d-regular ORN design with maximum latency $L = O(\frac{1}{d}L^*(r, N))$. This confirms that the tight bound on maximum latency for d-regular ORN designs is $\Theta(\frac{1}{d}L^*(r, N))$ whenever $d \leq N^{1/(h+1)}$ and justifies our focus on the case $d = 1$ throughout the remainder this chapter.

3.3 Upper Bound: Elementary Basis Scheme

To prove an upper bound on the latency achievable while guaranteeing a given throughput, we define an infinite family of ORN designs which we refer to as the Elementary Basis Scheme (EBS). The upper bound given by EBS is within a constant factor of the previously described lower bound for most values of r. To tightly bound the remaining values of r, we describe a second infinite family of ORN designs which we refer to as the Vandermonde Bases Scheme (VBS). Combined, EBS and VBS give a tight upper bound on maximum latency for all constant r. We address the upper bound for d-regular networks with $d > 1$ by modifying EBS and VBS in Section 3.6.

3.3.1 Connection Schedule:

In EBS's connection schedule, each node participates in a series of sub-schedules called round robins. Consider a cyclic group $\mathbb{Z}/n$ acting freely on a set S of n nodes, where we denote the action of $t \in \mathbb{Z}/n$ on $i \in S$ by $i + t$. A round robin for S is a schedule of $n - 1$ timeslots in which each element of S has a chance to send directly to each other element exactly once; during timeslot $t \in [n-1]$ node i may send to $i+t$. The number of round-robins in which each EBS node participates is controlled by a tuning parameter h which we refer

to as the *order*. Similar to the previous section, h will be half of the the maximum number of physical hops in an EBS path.

Let $n = N^{1/h}$, so that the node set $[N]$ is in one-to-one correspondence with the elements of the group $(\mathbb{Z}/n)^h$. Each node $a \in [N]$ is assigned a unique set of h coordinates $(a_0, a_1, ..., a_{h-1}) \in (\mathbb{Z}/n)^h$ and participates in h round robins, each containing the n nodes that match in all but one of the h coordinates. We refer to these round robins as *phases* of the EBS schedule. One full iteration of the EBS schedule, or *epoch*, contains h phases. Because each phase is a round robin among n nodes, each phase takes $n - 1$ timeslots, resulting in an overall epoch length of $T = h(n - 1) = h(N^{1/h} - 1)$.

We now describe the EBS schedule formally. We express each node $\boldsymbol{i}$ as the h-tuple $(i_0, i_1, \ldots, i_{h-1}) \in (\mathbb{Z}/n)^h$. Similarly, we identify each permutation π_k of the connection schedule using a scale factor s, $1 \leq s < n$, and a phase number p, $0 \leq p < h$, such that $k = (n-1)p + s - 1$. Let $\mathbf{e}_p$ denote the standard basis vector whose p^{th} coordinate is 1 and all other coordinates are 0. The connection schedule is then $\pi_{(n-1)p+s-1}(\boldsymbol{i}) = \boldsymbol{i} + s\mathbf{e}_p = \boldsymbol{j}$. Since $\mathbf{e}$ is the standard basis, $j_x = i_x$ for $x \neq p$, and $j_p = i_p + s \pmod{n}$.

The EBS schedule can be seen as simulating a flattened butterfly graph between nodes [38]. This schedule generalizes existing ORN designs which have thus far all been based on the same schedule: a single round robin among all nodes, simulating an all-to-all graph. When $h = 1$, the EBS schedule reduces to this existing schedule. On the other hand, when $h = \log_2(N)$, the EBS schedule simulates a direct-connect hypercube topology. By varying h, in addition to achieving these two known points, the EBS family includes schedules which achieve intermediate throughput and latency tradeoff points.

Node \ Timeslot	0	1	2	3
A,A	B,A	C,A	A,B	A,C
B,A	C,A	A,A	B,B	B,C
C,A	A,A	B,A	C,B	C,C
.	.	.	.	.
.	.	.	.	.
B,C	C,C	A,C	B,A	B,B
C,C	A,C	B,C	C,A	C,B

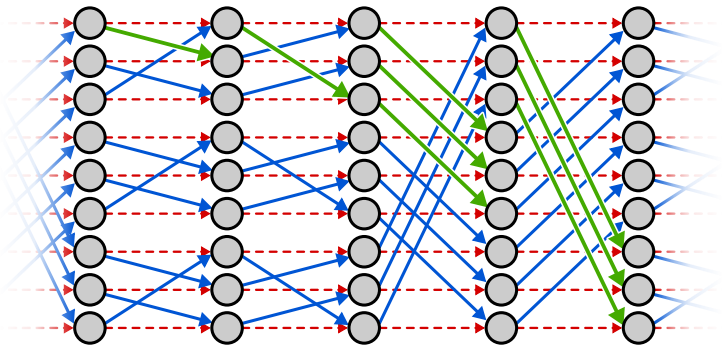


Figure 3.3: Connection schedule for 9 nodes in $h = 2$ EBS, as well as part of the corresponding virtual topology. Physical edges used on semi-paths from ((A,A),0) to other nodes are highlighted in green. This schedule can be seen as a generalization of the one presented in Figure 1.3.

3.3.2 Oblivious Routing Scheme

The EBS oblivious routing scheme is based around VLB, which operates in two stages: First, traffic is routed from the source to a random intermediate node in the network. Then, traffic is routed from the intermediate node to its final destination. This two-stage design ensures that traffic is uniformly distributed throughout the network regardless of demand. We refer to the path taken during an individual stage as a *semi-path*, and we use the same algorithm to generate semi-paths in either stage.

To create a semi-path between a node (a, t) and $[\![b]\!]$, the following greedy algorithm is used starting at (a, t): for the current node in the virtual topology, if the outgoing physical edge leads to a node with a decreased Hamming distance to b (i.e. it matches b in the modified coordinate), traverse the physical edge. Otherwise, traverse the virtual edge. This algorithm terminates when it reaches a node in $[\![b]\!]$. Note that because there are h coordinates, the largest Hamming distance possible is h, and the longest semi-paths use h physical links.

In order to construct a full path from (a, t) to $[\![b]\!]$, first select an intermediate node c in the system uniformly at random. Then, traverse the semi-path from (a, t) to $[\![c]\!]$. Let t' be

the timeslot at which we reach $[\![c]\!]$. If $t' < t + T$, traverse virtual edges until node $(c, t + T)$ is reached. Finally, traverse the semi-path from $(c, t + T)$ to $[\![b]\!]$.

The EBS oblivious routing scheme is formed as follows: for $R_{a,b,t}$, for all intermediate nodes c, construct the path from (a, t) to $[\![b]\!]$ via c as described above, and assign it the value $\frac{1}{N}$. Assign all other paths the value 0. Because there are N possible intermediate nodes, each of which is used to define one path from (a, t) to $[\![b]\!]$, this routing scheme defines one unit of flow.

3.3.3 Latency-Throughput Tradeoff of EBS

Proposition 1. *For each $r \leq \frac{1}{2}$ such that $h = \frac{1}{2r}$ is an integer, and each $N > 1$ such that $N^{1/h}$ is an integer, the EBS design of order h on N nodes guarantees throughput r and has maximum latency $\frac{1}{r}\left(N^{2r} - 1\right)$.*

The proof of Proposition 1 is contained in the following two subsections, which address the latency and throughput guarantees respectively.

Latency

Recall that $h = \frac{1}{2r}$ and that $n = N^{1/h} = N^{2r}$, so the latency bound in Proposition 1 can be written as $2h(n - 1)$. Since the epoch length is $T = h(n - 1)$, the latency bound asserts that every EBS routing path completes within a time interval no greater than the length of two epochs. An EBS path is composed of two semi-paths, so we only need to show that each semi-path completes within the length of a single epoch.

Let (a, t) denote the first node of the semi-path. If t occurs at the start of a phase, then after p phases have completed the Hamming distance to the semi-path's destination

address must be less than or equal to $t - p$; consequently the semi-path completes after at most h phases, as claimed. If t occurs in the middle of a phase using basis vector $\mathbf{e}_p$, let s denote the number of timeslots that have already elapsed in that phase. Either the semi-path is able to match the p^{th} destination coordinate before the phase ends, or the coordinate can be matched during the first s timeslots of the next phase that uses basis vector $\mathbf{e}_p$. In either case, the p^{th} destination coordinate will be matched no later than timeslot $t + T$, and all other destination coordinates will be matched during the intervening phases.

Throughput

Lemma 1. *Let R be the EBS routing scheme for a given N and h. For all demand functions D requesting throughput at most $\frac{1}{2h}$, the flow $f(R, D)$ is feasible.*

Proof. Consider an arbitrary demand function D requesting throughput $r = \frac{1}{2h}$, and consider an arbitrary physical edge $e \in E_{\text{phys}}$ from (i, t_e) to $(j, t_e + 1)$, where t_e is the timeslot during which the edge begins. Let $t_e \equiv (p_e, s_e)$ such that p_e is the phase in the schedule corresponding to t_e, and s_e is the scale factor used during t_e. We wish to show that $F(f(R, D), e) \leq 1$.

We first select a new demand function D' such that for all t, $D'(t)$ has row and column sums exactly equal to r, and $D'(t)$ bounds $D(t)$ above. Due to the latter condition, it follows that $f(R, D')$ bounds $f(R, D)$ above; thus $F(f(R, D'), e) \geq F(f(R, D), e)$. Henceforward, we focus on proving $F(f(R, D'), e) \leq 1$.

Valid paths in EBS include two components: the semi-path from the source node to an intermediate node, and the semi-path from the intermediate node to the destination node. We can therefore decompose the paths in $F(f(R, D'), e)$ into two components as follows:

first, we define R', a routing scheme defined such that $R'_{a,b,t}(P)$ equals 1 if P is the semi-path from (a, t) to $[\![b]\!]$, and 0 otherwise. Because EBS uses the same routing strategy for both source-intermediate semi-paths and intermediate-destination semi-paths, R' is used for both components. Then, we introduce two demand functions: $D'_{a\to b}$ represents demand on semi-paths from origin nodes to intermediate nodes, while $D'_{b\to c}$ represents demand on semi-paths from intermediate nodes to destination nodes. Note that for all physical edges e,

$$F(f(R, D'), e) = F(f(R', D'_{a\to b}), e) + F(f(R', D'_{b\to c}), e).$$

To characterize $D'_{a\to b}$, note that regardless of source and destination, R samples intermediate nodes uniformly. Therefore, for all $(t, a, b) \in \mathbb{Z} \times [N] \times [N]$,

$$D'_{a\to b}(t, a, b) = \frac{1}{N} \sum_{c\in[N]} D'(t, a, c) = \frac{r}{N}$$

Similarly, because semi-paths from an intermediate node to the destination always commence exactly T timeslots after the starting vertex, we can characterize $D'_{b\to c}(t, b, c)$ as follows:

$$D'_{b\to c}(t, b, c) = \frac{1}{N} \sum_{a\in[N]} D'(t - T, a, c) = \frac{r}{N}$$

Note that $D'_{a\to b} = D'_{b\to c} = D^{ALL}$, where D^{ALL} is the uniform all-to-all demand function $D^{ALL}(t, a, b) = \frac{r}{N}$ for all $(t, a, b) \in \mathbb{Z} \times [N] \times [N]$. Therefore, $F(f(R, D), e) \le 2F(f(R', D^{ALL}), e)$.

Claim 1. *For all $e \in E_{phys}$, there are exactly Tn^{h-1} triples (t, a, b) such that the semi-path from (a, t) to $[\![b]\!]$ traverses e.*

Proof of claim. Denote the endpoints of edge e by (i, t_e) and $(i + s \cdot \mathbf{e}_p, t_e + 1)$. The semi-path of a triple (t, a, b) traverses e if and only if the semi-path first routes from (a, t) to (i, t_e), and $(b - a)_p = s$.

Because semi-paths complete in T timeslots, only semi-paths beginning in timeslots in the range $[t_e - T + 1 \dots t_e]$ could possibly reach node (i, t_e) and traverse e. For every

$t \in [t_e - T + 1..t_e]$, where $t \equiv (p_t, s_t)$, we can construct n^{h-1} such triples as follows: First, we select $\boldsymbol{d}$, a vector representing the difference between a and b in the triple we will construct. To satisfy the second condition on (t, a, b), we must set $\boldsymbol{d}_p = s$. However, the remaining $h - 1$ indices of $\boldsymbol{d}$ can take on any of the n possible values. Thus, there are n^{h-1} possibilities for $\boldsymbol{d}$.

For any semi-path (t, a, b) such that $b - a = \boldsymbol{d}$, the timeslots in which a physical edge is traversed can be determined from $\boldsymbol{d}$. For any given timeslot $t' \equiv (p', s')$ such that $t \leq t' < t + T$, a physical edge is traversed if and only if $\boldsymbol{d}_{p'} = s'$. These are the edges that decrease the Hamming distance to b by correctly setting coordinate p. We thus construct a as follows: For every index p, if $(\boldsymbol{d}_p, p)$ is between k_t and $k_e - 1$ inclusive, we set $a_p = \boldsymbol{i}_p - \boldsymbol{d}_p$. Otherwise, we set $a_p = \boldsymbol{i}_p$. Once we have constructed a, b is simply $a + \boldsymbol{d}$. This choice of a and b ensures that by timeslot t_e, the semi-path from (a, t) to $[\![b]\!]$ reaches $[\![i]\!]$.

For each of the T timeslots for which semi-paths originating in the given timeslot may traverse e, there are n^{h-1} such semi-paths. This gives a total of Tn^{h-1} semi-paths that traverse e over all timeslots. Note that because each such semi-path has a unique $(t, \boldsymbol{d})$, none of the constructed semi-paths are double counted. In addition, because the $(t, \boldsymbol{d})$ pair determines the timeslots in which physical links are followed, and because there is only one physical link entering and leaving each node during each timeslot, there cannot be more than one choice of a for a given $(t, \boldsymbol{d})$ pair such that the semi-path includes (i, t_e). Because the Tn^{h-1} count includes all possible choices of $\boldsymbol{d}$ for every timeslot, all semi-paths that traverse e are accounted for. □

Now we continue with the proof of Lemma 1. Since exactly Tn^{h-1} triples (t, a, b) correspond to semi-paths that traverse e, and D^{ALL} assigns $\frac{r}{N}$ flow to each semi-path, $F(f(R', D^{ALL}), e) = \frac{r}{N} Tn^{h-1} = \frac{r}{N} h(n-1)n^{h-1}$. Thus:

$$F(f(R,D),e) \leq 2F(f(R',D^{ALL}),e) = 2\frac{r}{N}h(n-1)n^{h-1} < 2\frac{r}{N}hn^h = 2\frac{r}{N}h(N^{1/h})^h = 2rh$$

When $r \leq \frac{1}{2h}$, for all physical edges e, $F(f(R,D,e)) \leq 1$. Thus, $f(R,D)$ is feasible. □

3.3.4 Tightness of EBS Upper Bound

Lemma 2. *For $0 < r \leq \frac{1}{2}$ let $h = \left\lfloor \frac{1}{2r} \right\rfloor$ and $\varepsilon = h + 1 - \frac{1}{2r}$. The EBS design of order h attains maximum latency at most $CL^*(r,N)$, provided that*

$$\varepsilon \geq 2\sqrt{\frac{2h}{\pi}}\left(\frac{2e}{C}\right)^h.$$

Proof. Theorem 3 and Proposition 1 together show the following about the maximum latency of EBS compared to the maximum latency lower bound:

$$L_{EBS} \leq 2hN^{1/h}$$

$$L^*(r,N) \geq \frac{h}{e}(\varepsilon N)^{1/h}\left(\frac{\sqrt{\frac{\pi h}{2}}}{4h}\right)^{1/h}$$

Note that this interpretation of the maximum latency lower bound is taken from equation (3.3) in the proof of Theorem 3.

Suppose we wish to assert $L_{EBS}/L^*(r,N) \leq C$. Given C and h, we will derive the possible values of ε for which this assertion holds.

$$C \geq \frac{2hN^{1/h}}{\frac{h}{e}(\varepsilon N)^{1/h}\left(\frac{\sqrt{\frac{\pi h}{2}}}{4h}\right)^{1/h}} = \frac{2e}{\left(\frac{\varepsilon\sqrt{\pi h/2}}{4h}\right)^{1/h}}$$

$$\frac{\varepsilon\sqrt{\pi h/2}}{4h} \geq \left(\frac{2e}{C}\right)^h$$

$$\varepsilon \geq 2\sqrt{\frac{2h}{\pi}}\left(\frac{2e}{C}\right)^h.$$

□

When ε falls outside this range, the maximum latency of the EBS design is far from optimal. In the following sections we present and analyze an ORN design which gives a tighter upper bound when ε falls outside this range, in other words when $\varepsilon < 2\sqrt{\frac{2h}{\pi}}\left(\frac{2e}{C}\right)^h$.

3.4 Upper Bound: Vandermonde Bases Scheme

In order to provide a tight bound when ε is very small, we define a new family of ORN designs which we term the Vandermonde Bases Scheme (VBS). VBS is defined for values of N which are perfect powers of prime numbers. We begin by providing some intuition behind the design of VBS.

For $h = \left\lfloor \frac{1}{2r} \right\rfloor$ and $\varepsilon = h + 1 - \frac{1}{2r}$, a small value of ε indicates that r is slightly above $\frac{1}{2(h+1)}$. This indicates that the average number of physical hops in a path can be at most slightly below the even integer $2(h+1)$. EBS is only able to achieve an average number of physical hops equal to an even integer as N becomes sufficiently large. In small ε regions, the difference between the highest average number of physical hops theoretically capable of guaranteeing r throughput and the average number of physical hops used by EBS approaches 2. This suggests that EBS achieves a throughput-latency tradeoff that favors throughput more than is necessary in these regions, penalizing latency too much to form a tight bound. A more effective ORN design for these regions would use paths with $2(h+1)$ physical hops, but mix in sufficiently many paths with fewer physical hops to ensure that the average number of physical hops per path is at most $2(h+1-\varepsilon)$.

VBS achieves this by employing two routing strategies for semi-paths alongside each other. The first strategy, single-basis (SB) paths, resembles the semi-path routing used by

EBS for $h' = h + 1$. The second strategy, hop-efficient (HE) paths, will rely on the fact that VBS's schedule regularly modifies the basis used to determine which nodes are connected to one another. HE paths will consider edges beyond the current basis, enabling them to form semi-paths between nodes using only h hops, even when this is not possible within a single basis. The more future phases are considered, the more nodes can be connected by HE paths. This tuning provides a high granularity in the achieved tradeoff between throughput and latency, and enables a tight bound in regions where ε is small. It is interesting that the quantitative reasoning underlying this scheme is reminiscent of the proof of the Counting Lemma (Lemma 4), which similarly classifies paths into short paths and long paths and counts the number of destinations reachable by short paths.

We define VBS for $N = n^{h+1}$ such that n is a prime number. The connection schedule and routing algorithm of VBS depend on a parameter δ, which represents a target for the fraction of semi-paths that traverse HE paths. We later describe how to set Q, the number of future phases considered for HE path formation, such that the number of destinations reachable by HE paths is approximately δN.

3.4.1 Connection Schedule

Before describing the connection schedule of VBS, it is instructive to revisit the schedule of EBS. EBS's schedule consists of h' phases. Each of these phases is defined based on an elementary basis vector $\mathbf{e}_p$, connecting each node $\boldsymbol{i}$ to nodes $\boldsymbol{i} + s\mathbf{e}_p$ for all possible nonzero scale factors s. VBS is defined similarly, except instead of elementary basis vectors, Vandermonde vectors (to be defined in the next paragraph of this section) are used to form the phases. In addition, rather than using a single basis, the VBS connection schedule is formed from a longer sequence of phases, with any set of $h + 1$ adjacent phases corresponding to a basis.

For VBS, we assume the total number of nodes in the system is $N = n^{h+1}$ for some prime number n. As in EBS, each node a is assigned a unique set of $h+1$ coordinates $(a_0, a_1, ..., a_h)$, each ranging from 0 to $n-1$. This maps each node to a unique element of $\mathbb{F}_n^{h+1}$. We identify each permutation π_k of the connection schedule using a scale factor s, $1 \leq s < n$ and a phase number[4] p, $0 \leq p < n$, such that $k = (n-1)p + s - 1$. Each phase p is formed using the Vandermonde vector $\boldsymbol{v}(p) = (1, p, p^2, ..., p^h)$. This produces the connection schedule $\pi_{(n-1)p+s-1}(\boldsymbol{i}) = \boldsymbol{i} + s\boldsymbol{v}(p)$.

3.4.2 Routing Algorithm

As with EBS, VBS's oblivious routing scheme is based around VLB. First, traffic is routed along a semi-path from the source to a random intermediate node in the network, and then traffic is routed along a second semi-path from the intermediate node to its final destination. As in EBS, the same algorithm is used to generate semi-paths in both stages of VLB. However, unlike in EBS, semi-paths are only defined starting at phase boundaries. Thus, the first step of a VBS path is to traverse up to $n-2$ virtual edges until a phase boundary is reached. Paths are then defined for a given (q, a, b) triple, where $q = t/(n-1)$ for some timeslot t at the beginning of a phase (hence t is divisible by $n-1$). Following the initial virtual edges to reach a phase boundary, we concatenate the semi-path from the source to the intermediate node, followed by the semi-path from the intermediate node to the destination.

Depending on the current phase and the source-destination pair, we either route via a single-basis path or a hop-efficient path. The routing scheme always selects a hop-efficient semi-path when one is available, and otherwise it selects a single-basis path. We describe both path types below.

[4]The mnemonic is that p stands for "phase number", not "prime number". We beg the forgiveness of readers who find it confusing that the size of the prime field is denoted by n, not p.

Single-basis paths The single-basis path, or SB path, for a given (q, a, b) is formed as follows: First, we define the distance vector $\boldsymbol{d} = b - a$, as well as the basis $Y = (v(q), v(q + 1), ..., v(q + h))$. Note that the vectors in the basis Y are those used to form the $h + 1$ phases beginning with phase q. Then, we find $\boldsymbol{s} = Y^{-1}\boldsymbol{d}$. Over the next $h + 1$ phases, for every timeslot $t' \equiv (p', s')$, if $s' = \boldsymbol{s}_{p'}$, the physical edge is traversed. Otherwise, the virtual edge is traversed. This strategy corresponds to traversing $\boldsymbol{d}$ through its decomposition in basis Y, beginning at node a and ending at node b.

Although this algorithm for SB paths completes within $h + 1$ phases, following this virtual edges are traversed for a further Q phases. This ensures that both SB and HE paths take $h + 1 + Q$ phases to complete. Note that it is possible for an SB path to have fewer than $h + 1$ hops, although this becomes increasingly rare as N grows without bound.

Hop-efficient paths A hop-efficient path, or HE path, is formed as follows: First, for $h + 1$ phases, only virtual edges are traversed. This ensures that the physical hops of HE and SB paths beginning during the same phase q use disjoint sets of vectors (assuming $n > h + 1 + Q$), which simplifies later analysis. Following this initial buffer period, h phases are selected out of the next Q phases, and one physical hop is taken in each selected phase. During all other timeslots within the Q phases, virtual hops are taken.

For a given starting phase q and starting node a, there are $\binom{Q}{h}(n-1)^h$ possible HE paths. Because there are a total of N destinations reachable from a, we would like δN destinations to be reachable by HE paths. Ignoring for now the possibility of destinations reachable by multiple HE paths, we set Q to the lowest integer value such that:

$$\binom{Q}{h}(n-1)^h \geq \delta N \impliedby \binom{Q}{h} \geq \delta n$$

Note that for this value of Q, $\binom{Q-1}{h} < \delta n$. For some (q, a, b), more than one HE path may exist. In this case, an arbitrary selection can be made between these multiple paths; the specific path chosen does not affect our analysis of VBS.

3.4.3 Latency-Throughput Tradeoff of VBS

Latency

A VBS path begins with at most $n - 2$ virtual edges traversed until a phase boundary is reached. Following this, the first semi-path immediately begins, followed by the second semi-path. Because both SB and HE paths are defined to take $h + 1 + Q$ phases, the latency of a single semi-path is $(n - 1)(h + 1 + Q)$. This gives a total maximum latency of $(n - 2) + 2(n - 1)(h + 1 + Q) = (n - 1)(3 + 2h + 2Q) - 1$ for VBS paths.

Throughput

Lemma 3. *Let R be the VBS routing scheme for a given N, h, and δ, such that $\delta \leq \frac{1}{4(h+1)(1+\frac{1}{2h})^2}$. For all demand functions D requesting throughput at most $\frac{1}{2(h+1-\varepsilon)}$, where $\varepsilon = \frac{1}{4}\delta$, the flow $f(R, D)$ is feasible.*

Proof. Consider an arbitrary demand function D requesting throughput at most r, and consider an arbitrary physical edge $e \in W_{\text{phys}}$ from (i, t_e) to $(j, t_e + 1)$, where t_e is the timeslot during which the edge begins. Let $t_e \equiv (p_e, s_e)$ such that p_e is the phase in the schedule corresponding to t_e, and s_e is the scale factor used during t_e. We wish to show that $F(f(R, D), e) \leq 1$.

As in our proof of the throughput of EBS (Lemma 1), we begin by inflating D into D'.

Similarly, we define R', the routing scheme for semi-paths, and we decompose $f(R, D')$ into $f(R', D'_{a\to b})$ and $f(R', D'_{b\to c})$. Note that because semi-paths begin only on phase boundaries, R' in this case does not strictly follow our definition for an oblivious routing scheme. Instead, we define $R'_{a,b,q}$ using phases q, rather than timeslots t, for the domain. The path used for $R'_{a,b,q}$ begins during the first timeslot of phase q. This is reflective of the definitions for semi-paths in VBS.

To generate $D'_{a\to b}$, note that R first batches (a, b, t) triples over the $n-1$ timeslots preceding an epoch boundary, before sampling intermediate nodes uniformly. Therefore, for all (q, a, b)

$$D'_{a\to b}(q, a, b) = \frac{1}{N} \sum_{t\in[n-1]} \sum_{c\in[N]} D'(q(n-1) - t, a, c) = \frac{(n-1)r}{N}$$

Similarly, because semi-paths from an intermediate node to the destination always commence exactly $h + 1 + Q$ phases after the beginning of the first semi-path, we can define $D'_{b\to c}(t, b, c)$ as follows:

$$D'_{b\to c}(q, b, c) = \frac{1}{N} \sum_{t\in[n-1]} \sum_{c\in[N]} D'((q - h - 1 - Q)(n-1) - t, a, c) = \frac{(n-1)r}{N}$$

Note that $D'_{a\to b} = D'_{b\to c} = D^{ALL}$, where D^{ALL} is the uniform all-to-all demand function $D^{ALL}(q, a, b) = \frac{(n-1)r}{N}$ for all $(q, a, b) \in \mathbb{Z}\times[N]\times[N]$. Therefore, $F(f(R, D), e) \le 2F(f(R', D^{ALL}), e)$.

To calculate $F(f(R', D^{ALL}), e)$, we compute the number of (q, a, b) triples that traverse edge e. We calculate this number as follows: First, we calculate $\#_{SB}$, which represents the number of (q, a, b) triples that have an SB path that traverses edge e. Then, we calculate $\#_{missing}$, the number of such triples that have an HE path available (and thus do not traverse e). Finally, we determine $\#_{HE}$, the number of triples that traverse e using an HE path. The total flow traversing edge e is then $F(f(R', D^{ALL}), e) = \frac{(n-1)r}{N}(\#_{SB} - \#_{missing} + \#_{HE})$.

To find $\#_{SB}$, we use reasoning similar to that used in Lemma 1. In order for a given (q, a, b) to have an SB path that traverses edge e, the SB path for (q, a, b) must reach node (i, t), then traverse edge e. The only values of q for which this is possible are those in the range $q_e - h \leq q \leq q_e$. For each of these q, we can generate n^h distinct (q, a, b) triples that have SB paths that traverse edge e as follows. First, select an arbitrary $\boldsymbol{s}$ such that $s_{q_e - q} = s_e$. Then, set $a = \boldsymbol{i} - \Sigma_{q'=q}^{q_e-1} s_{q'-q} v(q')$, and $b = a + \Sigma_{q'=q}^{q+h} s_{q'-q} v(q')$. In this case, $\boldsymbol{s}$ corresponds to a distance vector between a and b, expressed in terms of the basis used for SB paths starting in phase q. Because of how a is set, it is clear that the SB path for (q, a, b) must traverse (i, t). In addition, because $s_{q_e - q} = s_e$, the SB path will traverse edge e instead of another edge during the same phase.

For a given q, there are n^h possible values for $\boldsymbol{s}$, because all but one of its $h + 1$ elements can be set to any value in $[n]$. There are $(h + 1)$ possible values for q, giving a total of $\#_{SB} = (h + 1)n^h$

To find $\#_{missing}$, we compare the distance vectors of (q, a, b) triples that have SB paths which traverse e with those of (q, a, b) triples that have valid HE paths. Each vector found in the overlap between these two sets corresponds to one (q, a, b) triple that contributes to $\#_{missing}$. To reason about the former set of vectors, we return to the construction of $\boldsymbol{s}$ used to find $\#_{SB}$. For a given starting phase q, each $\boldsymbol{s}$ such that $\boldsymbol{s}_{q_e - q} = s_e$ represents a distance vector that can traverse e, expressed in terms of the basis used for SB paths starting in phase q. We can construct this basis as $Y = (v(q), v(q + 1), ..., v(q + h)$. For each $\boldsymbol{s}$, $\boldsymbol{d} = Y\boldsymbol{s}$ is the same distance vector expressed using the elementary basis. The range of possible distance vectors $\boldsymbol{d}$ reachable while traversing e forms D_e, an h-dimensional affine subspace of $\mathbb{F}_n^{h+1}$ that is parallel to W_e, the linear subspace spanned by the set $Y \setminus \{v(q_e)\}$.

Next, we consider which triples have valid HE paths. For a given starting phase q, there are Q phases which are considered for forming HE paths. Let I be a set of h phase numbers chosen from these Q phases, and let $V(I)$ be the linear subspace spanned by the vectors

corresponding to the phase numbers in I. There are $\binom{Q}{h}$ ways of choosing such a set I. For each possible choice, $V(I)$ forms an h-dimensional linear subspace in F_n^{h+1}, corresponding to the distance vectors reachable via HE paths using the chosen phases. (Note that $V(I)$ must be h-dimensional because every h distinct Vandermonde vectors are linearly independent.) Because $V(I)$ and W_e are spanned by distinct sets of h Vandermonde vectors, these linear subspaces are not equivalent, implying that $V(I)$ and D_e are not parallel. Thus, $V(I) \cap D_e$ is an affine subspace with dimension $h-1$ and contains n^{h-1} distance vectors.

Some distance vectors lie in more than one such intersection. In order to avoid overcounting $\#_{missing}$, we must remove at least this many vectors from our count. Given two sets of h chosen phase numbers I and J, $V(I)$ and $V(J)$ form two different linear subspaces of $\mathbb{F}_n^{h+1}$. As linear subspaces, both I and J contain the zero vector, as does the $(h-1)$-dimensional $I \cap J$. D_e does not contain the zero vector, so $D_e \cap I \cap J$ can only be $(h-2)$-dimensional, containing n^{h-2} distance vectors. There are fewer than $\binom{Q}{h}^2$ ways of choosing two distinct sets I and J.

Thus, for a given starting q, there are fewer than $\binom{Q}{h} n^{h-1} - \binom{Q}{h}^2 n^{h-2}$ distance vectors in the overlap between D_e and the union of all possible $V(I)$. Because there are $h+1$ possibilities for the starting q, this gives the following lower bound for $\#_{missing}$:

$$\begin{aligned}
\#_{missing} &> (h+1)\left(\binom{Q}{h} n^{h-1} - \binom{Q}{h}^2 n^{h-2}\right) \\
&\geq (h+1)\left((\delta n) n^{h-1} - \left(\binom{Q-1}{h}\frac{Q}{Q-h}\right)^2 n^{h-2}\right) \\
&> (h+1)\left(\delta n^h - \left(\delta n \frac{Q}{Q-h}\right)^2 n^{h-2}\right) \\
&= (h+1)\left(\delta n^h - \delta^2 n^h \left(\frac{Q}{Q-h}\right)^2\right)
\end{aligned}$$

To find $\#_{HE}$, note that a given (q, a, b) can only traverse edge e if $q_e - h - Q \leq q < q_e - h$,

since q_e must be in the set of Q phases considered for HE paths for (q, a, b). For a given q, we can construct an HE path by selecting $h-1$ additional phases from the $Q-1$ remaining phases, and then selecting one of the $n-1$ edges within that phase to traverse. Some of these paths may lead to the same destination, causing an overcount, but it is fine to overcount $\#_{HE}$ slightly.

$$\begin{aligned}
\#_{HE} &\leq Q\binom{Q-1}{h-1}(n-1)^{h-1} \\
&= Q\binom{Q-1}{h}\frac{h}{Q-h}(n-1)^{h-1} \\
&< \delta nh\frac{Q}{Q-h}(n-1)^{h-1} \\
&< \delta hn^h\frac{Q}{Q-h}
\end{aligned}$$

Now that we have found $\#_{SB}$, $\#_{missing}$, and $\#_{HE}$, we can finally bound $F(f(R,D),e)$:

$$\begin{aligned}
F(f(R,D),e) &\leq 2F(f(R',D^{ALL}),e) \\
&= 2\frac{(n-1)r}{N}(\#_{SB} - \#_{missing} + \#_{HE}) \\
&< 2\frac{(n-1)r}{N}\left((h+1)n^h - (h+1)\left(\delta n^h - \delta^2 n^h\left(\frac{Q}{Q-h}\right)^2\right) + h\delta n^h\frac{Q}{Q-h}\right) \\
&= 2\frac{(n-1)r}{N}(h+1)n^h\left(1 - \left(\delta - \delta^2\left(\frac{Q}{Q-h}\right)^2\right) + \frac{h}{h+1}\delta\frac{Q}{Q-h}\right) \\
&< 2r(h+1)\left(1 - \delta\left(1 - \frac{h}{h+1}\frac{Q}{Q-h}\right) + \delta^2\left(\frac{Q}{Q-h}\right)^2\right)
\end{aligned}$$

For $Q \geq 2h^2 - h$, $\frac{Q}{Q-h} \leq \frac{h+\frac{1}{2}}{h}$. This gives:

$$\begin{aligned}
F(f(R,D),e) &< 2r(h+1)\left(1-\delta\left(1-\frac{h}{h+1}\frac{h+\frac{1}{2}}{h}\right)+\delta^2\left(\frac{h+\frac{1}{2}}{h}\right)^2\right) \\
&= 2r(h+1)\left(1-\delta\left(1-\frac{h+\frac{1}{2}}{h+1}\right)+\delta^2\left(1+\frac{1}{2h}\right)^2\right) \\
&= 2r(h+1)\left(1-\frac{1}{2}\frac{1}{h+1}\delta+\delta^2\left(1+\frac{1}{2h}\right)^2\right) \\
&= \frac{1}{2(h+1-\varepsilon)}2(h+1)\left(1-\frac{1}{2}\frac{1}{h+1}\delta+\delta^2\left(1+\frac{1}{2h}\right)^2\right) \\
&= \frac{1}{h+1-\varepsilon}\left(h+1-\frac{1}{2}\delta+(h+1)\delta^2\left(1+\frac{1}{2h}\right)^2\right) \\
&\leq \frac{1}{h+1-\varepsilon}(h+1-\varepsilon) \\
F(f(R,D),e) &< 1
\end{aligned}$$

Note that because of how we set ε and restrict δ, $\varepsilon \leq \frac{1}{2}\delta - (h+1)\delta^2(1+\frac{1}{2h})^2$. Because the amount of flow traversing any physical edge e is less than 1, the flow $f(R,D)$ is feasible.

□

3.4.4 Tightness of Overall Upper Bound

Theorem 2. *For all $r \in (0, 1/2]$, there is a VBS design or an EBS design which guarantees throughput r and uses maximum latency*

$$L_{max} \leq O(L^*(r,N)). \tag{3.1}$$

Proof. The VBS design of order h with parameter δ gives maximum latency $L \leq (h+1)(n-1) + Q(n-1)$ for $h = \left\lfloor \frac{1}{2r} \right\rfloor$, $\binom{Q}{h} \geq \delta n$, as long as $\delta \leq \frac{1}{4(h+1)(1+\frac{1}{2h})^2}$. Let $\varepsilon = h + 1 - \frac{1}{2r}$, and set $\delta = 4\varepsilon$.

We chose Q such that $\binom{Q-1}{h} < \delta n$ and $\binom{Q}{h} \geq \delta n$. Then $\binom{Q}{h} < \delta n \frac{Q}{Q-h} \leq \delta n \frac{h+\frac{1}{2}}{h}$, due to

$Q \geq 2h^2 - h$. Hence $Q \leq h\left(\delta n \frac{h}{h+(1/2)}\right)^{1/h}$. We upper bound the max latency of VBS in the following way.

$$
\begin{aligned}
L_{max} &\leq \max\{(h+1)(n-1) + Q(n-1), (h+1)(n-1) + (2h^2 - h)(n-1)\} \\
&\leq 2(h+1)(n-1) + 2h^2(n-1) + h\left(4\varepsilon n \frac{h+\frac{1}{2}}{h}\right)^{1/h}(n-1) \\
&\leq 2(h+1)n + 2h^2 n + hn(4\varepsilon n)^{1/h}\left(\frac{2h+1}{2}\right)^{1/h} \\
&\leq (h+1)[2N^{1/(h+1)} + hN^{1/(h+1)} + (4\varepsilon N)^{1/h}\left(\frac{2h+1}{2}\right)^{1/h}] \\
&\leq O(h[hN^{1/(h+1)} + (\varepsilon N)^{1/h}])
\end{aligned}
$$

For sufficiently large N (determined by ε and h, both functions of r), the second term will dominate. Thus, for large N:

$$
L_{max} \leq O\left(h\left[(\varepsilon N)^{1/h} + N^{1/(h+1)}\right]\right) = O\left(L^*(r, N)\right).
$$

By Lemma 3, VBS only gives a tight latency bound when $4\varepsilon = \delta \leq \frac{1}{4(h+1)(1+\frac{1}{2h})^2}$. When ε is greater than this value, we use EBS instead. By Lemma 2, EBS gives a factor C tight bound when $\varepsilon > 2\sqrt{\frac{2h}{\pi}}\left(\frac{2e}{C}\right)^h$. We check to make sure that there exists a constant C which works for all $\varepsilon > \frac{1}{4} \cdot \frac{1}{4(h+1)(1+\frac{1}{2h})^2}$

$$2\sqrt{\frac{2h}{\pi}}\left(\frac{2e}{C}\right)^h \leq \frac{1}{4}\cdot\frac{1}{4(h+1)\left(1+\frac{1}{2h}\right)^2}$$

$$\frac{2e}{C}\left(2\sqrt{\frac{2h}{\pi}}\right)^{1/h} \leq \left(\frac{1}{16(h+1)\left(1+\frac{1}{2h}\right)^2}\right)^{1/h}$$

$$C \geq 2e\left(2\sqrt{\frac{2h}{\pi}}\right)^{1/h}\left(16(h+1)\left(1+\frac{1}{2h}\right)^2\right)^{1/h}$$

$$C \geq O\left(\sqrt{h}^{1/h}\left((h+1)\left(\frac{2h+1}{2h}\right)^2\right)^{1/h}\right) = O(1)$$

Since there exists such a factor C, the following holds for EBS in the regions of interest.

$$L_{max} \leq O\left(h\left[(\varepsilon N)^{1/h} + N^{1/(h+1)}\right]\right) = O\left(L^*(r,N)\right)$$

□

3.5 Lower Bound

In this section we prove the lower-bound half of Theorem 1, which says that when $\frac{1}{2r} = h + 1 - \varepsilon$ with $h \in \mathbb{N}$ and $0 < \varepsilon \leq 1$, any d-regular, N-node ORN design that guarantees throughput r must have maximum latency $\Omega(\frac{h}{d}[N^{1/(h+1)} + (\varepsilon N)^{1/h}])$. As noted in Section 3.2.2, the general case of this lower bound reduces to the case $d = 1$, and we will assume $d = 1$ throughout the remainder of this section.

Because the full proof is somewhat long, we begin by sketching some of the main ideas in the proof, beginning with a much simpler argument leading to a lower bound of the form $\Omega(\frac{1}{r}N^r)$ when $1/r$ is an integer. This simple lower bound applies not only to oblivious routing schemes, but to *any* feasible flow f that solves the uniform multicommodity flow

problem given by the demand function $D(t, a, b) = \frac{r}{N-1}$ for all $t \in [T]$ and $b \neq a$. The lower bound follows by combining a few key observations.

1. Define the cost of a path to be the number of physical edges it contains. Since every source sends out r units of flow at all times, the flow f sends out rNT units of flow per T-step period, in a network whose physical edges have only NT units of capacity per T-step period. Consequently the average cost of flow paths in f must be at most $\frac{1}{r}$.
2. For any source node (a, t) in the virtual topology, the number of distinct destinations $[\![b]\!]$ that can be reached via a path with maximum latency L and cost p is bounded above by $\binom{L}{p}$.
3. If $L \leq \frac{1}{2er}N^r$, we have $\binom{L}{1/r} \leq N/4$ and $\sum_{p=1}^{1/r}\binom{L}{p} \leq N/2$, so the majority of source-destination pairs cannot be joined by a path with latency L and cost less than $\frac{1}{r} + 1$. In fact, even if we connect every source and destination with a minimum-cost path (subject to latency bound L), one can show that the average cost of paths will exceed $\frac{1}{r}$.
4. Since a feasible flow must have average path cost at most $\frac{1}{r}$, we can conclude that a feasible flow does not exist when $L \leq \frac{1}{2er}N^r$.

When $1/r$ is an integer, this lower bound of $L_{max} \geq \frac{1}{2er}N^r$ for feasible uniform multicommodity flows turns out to be tight up to a constant factor. However for oblivious routing schemes, Theorem 1 shows that maximum latency is bounded below by a function in which the exponent of N is roughly twice as large. Stated differently, for a given maximum latency bound, the optimal throughput guarantee for oblivious routing is only half as large as the throughput of an optimal uniform multicommodity flow.

The factor-two difference in throughput between oblivious routing and optimal uniformly multicommodity flow solutions aligns with the intuition that oblivious routing

schemes must use indirect paths (as in VLB) if they are to guarantee throughput r, whereas uniform multicommodity flow solutions (in a well-designed virtual topology) can afford to satisfy all demands using shortest-path routing. The proof of the lower bound for oblivious routing needs to substantiate this intuition.

To do so, we formulate oblivious routing as a linear program and interpret the dual variables as specifying a more refined way to measure the cost of paths. Rather than defining the cost of a path to be its number of physical edges, the duality-based proof amounts to an accounting system in which the cost of using an edge depends on the endpoints of the path in which the edge is being used. For a parameter θ which we will set to $h+1$ (unless ε is very small, in which case we'll set $\theta = h+2$), the dual accounting system assesses the cost of an edge to be 1 if its distance from the source is less than θ, plus 1 if its distance from the destination is less than θ. Thus, the cost of an edge is doubled when it is close to both the source and the destination. The doubling has the effect of equalizing the costs of direct and indirect paths: when the distance between a source and destination is at least θ, there is no difference in cost between a shortest path and one that combines two semi-paths each composed of θ physical edges.

Viewed in this way, it is intuitive that the proof manages to show that VLB routing schemes, which construct routing paths by concatenating random semi-paths with the appropriate number of physical edges, correspond to optimal solutions of the oblivious routing LP. The difficulty in the proof lies in showing that the constructed dual solution is feasible; for this, we make use of a version of the same counting argument sketched above, that bounds the number of distinct destinations reachable from a given source under constraints on the maximum latency and the maximum number of physical edges used.

3.5.1 Lower Bound Theorem Proof

Before presenting the proof of Theorem 3, we formalize the counting argument we reasoned about in our proof sketch.

Lemma 4. ***(Counting Lemma)*** *If in an ORN topology, some node a can reach k other nodes in at most L timeslots using at most h physical hops per path for some integer h, then $k \leq 2\binom{L}{h}$, assuming $h \leq \frac{1}{3}L$.*

Proof. If node a can reach k other nodes in $\leq L$ timeslots using exactly h physical hops per path, then $k \leq \binom{L}{h}$. Additionally, the function $\binom{L}{h}$ grows at least exponentially in base 2 — that is, $\binom{L}{h} \geq 2\binom{L}{h-1}$ — up until $h = \frac{1}{3}L$. Therefore, the number of such k is at most $\sum_{i=1}^{h}\binom{L}{i} \leq 2\binom{L}{h}$. □

Theorem 3. *Given an ORN design $\mathcal{R}$ which guarantees throughput r, the maximum latency suffered by any routing path P with $R_{a,b,t}(P) > 0$ over all a, b, t is bounded by the following inequality.*

$$L_{max} \geq \Omega\left(h\left\lceil (\varepsilon N)^{1/h} + N^{1/(h+1)}\right\rceil\right) \tag{3.2}$$

where $h = \left\lfloor \frac{1}{2r} \right\rfloor$ and $\varepsilon \in (0, 1]$ is set to equal $h + 1 - \frac{1}{2r}$. In other words, (h, ε) is the unique solution in $\mathbb{N} \times (0, 1]$ to the equation $\frac{1}{2r} = h + 1 - \varepsilon$.

Proof. Consider the linear program below which maximizes throughput given a maximum latency constraint, L, where we let $\mathcal{P}_L(a, b, t)$ be the set of paths from $(a, t) \to [\![b]\!]$ with latency at most L.

LP

maximize r

subject to $\sum_{P \in \mathcal{P}_L(a,b,t)} R_{a,b,t}(P) = r \qquad \forall a, b \in [N],\ t \in [T]$

$\sum_{a \in [N]} \sum_{t=0}^{T-1} \sum_{P \in \mathcal{P}_L(a,\sigma(a),t): e \in P} R_{a,\sigma(a),t}(P) \leq 1 \qquad \forall \sigma \in S_N, e \in E_{\text{phys}}$

$R_{a,b,t}(P) \geq 0 \qquad \forall a, b \in [N],\ t \in [T],\ P \in \mathcal{P}_L(a,b,t)$

The second set of constraints, in which the parameter σ ranges over the set S_N of all permutations of $[N]$, can be reformulated as the following set of nonlinear constraints in which the maximum is again taken over all permutations σ:

$$\max_{\sigma} \left\{ r \sum_{a \in [N]} \sum_{t=0}^{T-1} \sum_{P \in \mathcal{P}_L(a,\sigma(a),t): e \in P} R_{a,\sigma(a),t}(P) \right\} \leq 1 \quad \forall e \in E_{\text{phys}}$$

Note that given an edge e, this maximization over permutations σ corresponds to maximizing over perfect bipartite matchings with edge weights defined by $w_{a,b,e} = \sum_{t=0}^{T-1} \sum_{P \in \mathcal{P}_L(a,b,t): e \in P} R_{a,b,t}(P)$. This prompts the following matching LP and its dual.

Matching LP

maximize $\sum_{a,b} u_{a,b,e} w_{a,b,e}$

subject to $\sum_{b \in [N]} u_{a,b,e} \leq 1 \qquad \forall a \in [N]$

$\sum_{a \in [N]} u_{a,b,e} \leq 1 \qquad \forall b \in [N]$

$u_{a,b,e} \geq 0 \quad \forall a, b \in [N], e \in E_{\text{phys}}$

Matching Dual

minimize $\sum_{a \in [N]} \xi_{a,e} + \sum_{b \in [N]} \eta_{b,e}$

subject to $\xi_{a,e} + \eta_{b,e} \geq w_{a,b,e} \quad \forall a, b \in [N]$

$\xi_{a,e} \geq 0 \quad \forall a \in [N], e \in E_{\text{phys}}$

$\eta_{b,e} \geq 0 \quad \forall b \in [N], e \in E_{\text{phys}}$

We then substitute finding a feasible matching dual solution into the original LP, replace the expression $w_{a,b,e}$ with its definition $\sum_{t=0}^{T-1} \sum_{P \in \mathcal{P}_L(a,b,t): e \in P} R_{a,b,t}(P)$, and take the dual again.

LP

$$
\begin{aligned}
\text{maximize} \quad & r \\
\text{subject to} \quad & \textstyle\sum_{P \in \mathcal{P}_L(a,b,t)} R_{a,b,t}(P) = r && \forall a, b \in [N],\ t \in [T] \\
& \xi_{a,e} + \eta_{b,e} \geq \textstyle\sum_{t=0}^{T-1} \sum_{P \in \mathcal{P}_L(a,b,t): e \in P} R_{a,b,t}(P) && \forall a, b \in [N], e \in E_{\text{phys}} \\
& \textstyle\sum_{a \in [N]} \xi_{a,e} + \sum_{b \in [N]} \eta_{b,e} \leq 1 && \forall e \in E_{\text{phys}} \\
& \xi_{a,e} \geq 0 && \forall a \in [N], e \in E_{\text{phys}} \\
& \eta_{b,e} \geq 0 && \forall b \in [N], e \in E_{\text{phys}} \\
& R_{a,b,t}(P) \geq 0 && \forall a, b \in [N],\ t \in [T],\ P \in \mathcal{P}_L(a,b,t)
\end{aligned}
$$

Dual

$$
\begin{aligned}
\text{minimize} \quad & \textstyle\sum_e z_e \\
\text{subject to} \quad & \textstyle\sum_{a,b,t} x_{a,b,t} \geq 1 \\
& z_e \geq \textstyle\sum_b y_{a,b,e} && \forall a \in [N], e \in E_{\text{phys}} \\
& z_e \geq \textstyle\sum_a y_{a,b,e} && \forall b \in [N], e \in E_{\text{phys}} \\
& \textstyle\sum_{e \in P} y_{a,b,e} \geq x_{a,b,t} && \forall a, b \in [N],\ t \in [T],\ P \in \mathcal{P}_L(a,b,t) \\
& y_{a,b,e}, z_e \geq 0 && \forall a, b \in [N],\ e \in E_{\text{phys}}
\end{aligned}
$$

The variables $y_{a,b,e}$ can be interpreted as either edge costs we assign dependent on source-destination pairs (a, b), or demand functions designed to overload a particular edge e. We will use both interpretations, depending on if we are comparing $y_{a,b,e}$ variables to either $x_{a,b,t}$ or z_e variables respectively. According to the fourth dual constraint, the

variables $x_{a,b,t}$ can be interpreted as encoding the minimum cost of a path from (a,t) to $[\![b]\!]$ subject to latency bound L. According to the second and third dual constraints, the variables z_e can be interpreted as bounding the throughput requested by the demand function $D(t,a,b) = y_{a,b,e}$. We will next define the cost inflation scheme we use to set our dual variables.

Cost inflation scheme For a given node $a \in [N]$ and cutoff $\theta \in \mathbb{Z}_{>0}$, we will classify edges e according to whether they are reachable within θ physical hops of a, counting edge e as one of the hops. (In other words, one could start at node a and cross edge e using θ or fewer physical hops.) We define this value $m^+_\theta(e,a)$ as follows.

$$m^+_\theta(e,a) = \begin{cases} 1 & \text{if } e \text{ can be reached from } a \text{ using at most } \theta \text{ physical hops (including } e) \\ 0 & \text{if otherwise} \end{cases}$$

We define a similar value for edges which can reach node b.

$$m^-_\theta(e,b) = \begin{cases} 1 & \text{if } b \text{ can be reached from } e \text{ using at most } \theta \text{ physical hops (including } e) \\ 0 & \text{if otherwise} \end{cases}$$

To understand how these values are set, consider some path P from $(a,t) \to [\![b]\!]$. If we consider the m^+_θ, m^-_θ weights on the edges of P, then the first θ physical hop edges of P have weight $m^+_\theta(e,a) = 1$ and the last θ physical hop edges of P have weight $m^-_\theta(e,b) = 1$. It may be the case that some edges have both $m^+_\theta(e,a) = m^-_\theta(e,b) = 1$, if P uses fewer than 2θ physical hops. And if P uses θ or fewer physical hops, then every physical hop edge along P has weight $m^+_\theta(e,a) = m^-_\theta(e,b) = 1$. All other weights may be 0 or 1 depending on whether those edges are otherwise reachable from a or can otherwise reach b.

We start by setting $\hat{y}_{a,b,e} = m^+_\theta(e,a) + m^-_\theta(e,b)$. Also set $\hat{x}_{a,b,t} = \min_{P \in \mathcal{P}_L(a,b,t)}\{\sum_{e \in P} \hat{y}_{a,b,e}\}$. Note that by definition, $\hat{x}$ and $\hat{y}$ variables satisfy the last dual constraint. We will next find a

lower bound $w \le \sum_{a,b,t} \hat{x}_{a,b,t}$ and use that to normalize the $\hat{x}, \hat{y}$ variables to satisfy the first dual constraint.

Note that $\sum_{e \in P} \hat{y}_{a,b,e} \ge \min\{2\theta, 2|P \cap E_{\text{phys}}|\}$. Then we can bound the sum of $\hat{x}$ variables by

$$\sum_{a,b,t} \hat{x}_{a,b,t} \ge \sum_{a,t} \sum_{b \neq a} \min_{P \in \mathcal{P}_L(a,b,t)} \{2\theta, 2|P \cap E_{\text{phys}}|\}$$

Note that $\hat{x}_{a,b,t} < 2\theta$ only when there exists some path from (a,t) to $[\![b]\!]$ which uses less than θ physical edges. We can then use the Counting Lemma to produce an upper bound on the number of $b \neq a$ which have such paths: this is at most $2\binom{L}{\theta-1}$.

So, assuming that $2\binom{L}{\theta-1} \le N$ and that $\theta - 1 \le L/3$, we have

$$\sum_{a,t} \sum_{b \neq a} \hat{x}_{a,b,t} \ge NT \left(2\theta \left(N - 2\binom{L}{\theta-1}\right) + \binom{L}{\theta-1}\right)$$

Set

$$w = NT \left(2\theta \left(N - 2\binom{L}{\theta-1}\right) + \binom{L}{\theta-1}\right),$$

and then set $y_{a,b,e} = \frac{1}{w}\hat{y}_{a,b,e}$ and $x_{a,b,t} = \frac{1}{w}\hat{x}_{a,b,t}$.

Next, we set $z_e = \max_{a,b}\{\sum_a y_{a,b,e}, \sum_b y_{a,b,e}\}$. By construction, the values of $x_{a,b,t}, y_{a,b,e}, z_e$ that we have defined satisfy the dual constraints. Then to bound throughput from above, we upper bound the sums $\sum_a y_{a,b,e}$ and $\sum_b y_{a,b,e}$, thus upper bounding the sum of z_e's.

$$\sum_a y_{a,b,e} = \frac{1}{w} \sum_a (m_\theta^+(e,a) + m_\theta^-(e,b)) \le \frac{1}{w}\left(\sum_a m_\theta^+(e,a) + N - 1\right) \le \frac{1}{w}\left(2\binom{L}{\theta-1} + N - 1\right)$$

where the last step is an application of the Counting Lemma. Similarly,

$$\sum_b y_{a,b,e} = \frac{1}{w} \sum_b (m_\theta^+(e,a) + m_\theta^-(e,b)) \le \frac{1}{w}\left(N - 1 + \sum_b m_\theta^-(e,b)\right) \le \frac{1}{w}\left(N - 1 + 2\binom{L}{\theta-1}\right)$$

Recalling that $z_e = \max_{a,b}\{\sum_a y_{a,b,e}, \sum_b y_{a,b,e}\}$, we deduce that

$$z_e \leq \frac{1}{w}\left(N - 1 + 2\binom{L}{\theta - 1}\right).$$

Using this upper bound on z_e, we find that the optimal value of the dual objective — hence also the optimal value of the primal, i.e. the maximum throughput of oblivious routing schemes — is bounded by

$$\begin{aligned}
r \leq \sum_e z_e &\leq \frac{NT}{w}\left(N - 1 + 2\binom{L}{\theta - 1}\right) \\
&= \frac{N - 1 + 2\binom{L}{\theta-1}}{2\theta N - 4\theta\binom{L}{\theta-1} + 2\binom{L}{\theta-1}} \\
&\leq \frac{N - 1 + 2\binom{L}{\theta-1}}{2\theta N - 4\theta\binom{L}{\theta-1}} \\
&= \frac{N - 1 + \frac{2(L!)}{(\theta-1)!(L-\theta+1)!}}{2\theta N - 4\theta\frac{L!}{(\theta-1)!(L-\theta+1)!}} \\
&= \frac{(N-1)(\theta-1)!(L-\theta+1)! + 2(L!)}{2\theta(N(\theta-1)!(L-\theta+1)! - 2(L!))} \\
&= \frac{1}{2\theta} + \frac{4(L!)}{2\theta(L-\theta+1)!\left(N(\theta-1)! - 2\frac{L!}{(L-\theta+1)!}\right)} \\
&\leq \frac{1}{2\theta} + \frac{4L^{\theta-1}}{2\theta(N(\theta-1)! - 2L^{\theta-1})}
\end{aligned}$$

using the fact that $\frac{a!}{(a-b)!} \leq a^b$. At this point, we can rearrange the inequality to isolate L.

$$\begin{aligned}
r - \frac{1}{2\theta} &\leq \frac{4L^{\theta-1}}{2\theta(N(\theta-1)! - 2L^{\theta-1})} \\
\left(r - \frac{1}{2\theta}\right)(2\theta N(\theta-1)!) - \left(r - \frac{1}{2\theta}\right)4\theta L^{\theta-1} &\leq 4L^{\theta-1} \\
\left(r - \frac{1}{2\theta}\right)2\theta N(\theta-1)! &\leq L^{\theta-1}\left(4 + \left(r - \frac{1}{2\theta}\right)4\theta\right) \\
\frac{(r - \frac{1}{2\theta})2\theta N(\theta-1)!}{4 + (r - \frac{1}{2\theta})4\theta} &\leq L^{\theta-1} \\
\left(\frac{(r - \frac{1}{2\theta})2\theta N(\theta-1)!}{4 + (r - \frac{1}{2\theta})4\theta}\right)^{\frac{1}{\theta-1}} &\leq L
\end{aligned}$$

Now that we have a closed form, we simplify. We use Stirling's approximation, in the form $(k!)^{\frac{1}{k}} \geq \frac{k}{e}\sqrt{2\pi k}^{\frac{1}{k}}$.

$$
\begin{aligned}
L &\geq \left(\frac{(r - \frac{1}{2\theta}) 2\theta N (\theta - 1)!}{4 + (r - \frac{1}{2\theta}) 4\theta} \right)^{\frac{1}{\theta - 1}} \\
&= N^{\frac{1}{\theta-1}} (\theta - 1)!^{\frac{1}{\theta-1}} \left(\frac{(r - \frac{1}{2\theta}) 2\theta}{4 + (r - \frac{1}{2\theta}) 4\theta} \right)^{\frac{1}{\theta-1}} \\
&\geq \frac{\theta - 1}{e} N^{\frac{1}{\theta-1}} \left(\frac{(r - \frac{1}{2\theta}) 2\theta \sqrt{2\pi(\theta - 1)}}{4 + (r - \frac{1}{2\theta}) 4\theta} \right)^{\frac{1}{\theta-1}} \geq \frac{\theta - 1}{e} N^{\frac{1}{\theta-1}} \left(\frac{(r - \frac{1}{2\theta}) \theta \sqrt{\frac{\pi(\theta-1)}{2}}}{\theta r + \frac{1}{2}} \right)^{\frac{1}{\theta-1}}
\end{aligned}
$$

To set the parameter θ, first note that the above bound is positive when $r > \frac{1}{2\theta}$. Additionally, we would like to set θ as large as possible, and θ must be an integer value (otherwise the Counting Lemma doesn't make sense). Taking this into account, we set $\theta = \left\lfloor \frac{1}{2r} \right\rfloor + 1$, the nearest integer for which $(r - \frac{1}{2\theta})$ produces a positive value.

To simplify our lower bound further, let $h = \left\lfloor \frac{1}{2r} \right\rfloor$ and $\varepsilon = h + 1 - \frac{1}{2r}$. These can be interpreted in the following way: h represents the largest number of physical hops we take per path (approximately), and ε is directly related to how many pairs take paths using h physical hops instead of paths using fewer than h physical hops. Note that $\varepsilon \in (0, 1]$. This gives the restated bound below.

$$
\begin{aligned}
L &\geq \frac{h}{e} N^{1/h} \left(\frac{(r - \frac{1}{2(h+1)})(h+1)\sqrt{\frac{\pi h}{2}}}{(h+1)r + \frac{1}{2}} \right)^{1/h} \\
&= \frac{h}{e} N^{1/h} \left(\frac{\left(\frac{\varepsilon}{2(h+1)(h+1-\varepsilon)}\right)(h+1)\sqrt{\frac{\pi h}{2}}}{1 + \frac{\varepsilon}{2(h+1-\varepsilon)}} \right)^{1/h} \\
&= \frac{h}{e} N^{1/h} \left(\frac{\varepsilon\sqrt{\frac{\pi h}{2}}}{2(h+1-\varepsilon)+\varepsilon)} \right)^{1/h} \\
&\geq \frac{h}{e} (\varepsilon N)^{1/h} \left(\frac{\sqrt{\frac{\pi h}{2}}}{4h} \right)^{1/h} \\
&= \frac{h}{e} (\varepsilon N)^{1/h} \cdot \Omega(1) = \Omega\left(h(\varepsilon N)^{1/h}\right)
\end{aligned}
\tag{3.3}
$$

As $\varepsilon \to 0$, this bound goes toward 0, making it meaningless for extremely small values of ε. However, for such values of ε, we simply set $\theta = h + 2$ instead, which gives the following

$$
L_{max} \geq \Omega\left((h+1)N^{1/(h+1)}\right)
$$

To combine the two ways in which we set θ, we take the average of the two bounds. This gives the bound from our theorem statement,

$$
L_{max} \geq \Omega\left(h\left[(\varepsilon N)^{1/h} + N^{1/(h+1)}\right]\right) = \Omega\left(L^*(r,N)\right).
$$

□

3.6 EBS and VBS for Degree $d > 1$

Recall from Section 3.2.2 that an upper bound for 1-regular designs will only imply a similar upper bound for d-regular designs if we can ensure that the routing scheme does not route flow paths on multiple edges in the same "unrolled" segment of the 1-degree virtual topology. EBS and VBS always route flow on paths which use at most 1 edge from each phase, where a phase constitutes $(n-1)$ timeslots. Trivially, if d divides $(n-1)$, then these constructions already have the property we need. However, even if d does not divide $(n-1)$, as long as $d < n-1$, we can modify EBS and VBS as follows.

We change the connection schedule to iterate through each phase twice before moving on to the next. So for VBS, $\pi_{(n-1)p+s-1}(\boldsymbol{i}) = \boldsymbol{i} + s\boldsymbol{v}(\lfloor p/2 \rfloor)$. We also change the definition of single-basis and hop-efficient paths to use exclusively even-numbered phases or exclusively odd-numbered phases, depending on the parity of the next phase number after the request originates. With this modification, single-basis and hop-efficient paths always use physical edges that occur at least $(n-1)$ timeslots apart from each other. Therefore, in the "rolled up" virtual topology, our flow paths will always use at most one physical edge per timeslot. This at most doubles the maximum latency, and does not affect throughput.

3.7 EBS with Missing Nodes

EBS formalizes Shale's schedule and routing scheme, described in Section 2.2.1. Because of this, our proof that EBS achieves a Pareto optimal latency-throughput tradeoff, found in Section 3.3.3, can also be applied to Shale. However, there is an important caveat: EBS is only defined for very specific system sizes, namely perfect h-powers. This poses a problem for realistic deployments: datacenter operators would prefer to size their datacenters to match their anticipated workloads, rather than to fit an esoteric requirement of their

network design. Further, even if a datacenter operator were to exactly size their network to match EBS's requirements, datacenters in the real world regularly experience node failures. Failed nodes are unable to forward traffic. Due to EBS's use of VLB, this can affect the overall throughput of the network, potentially invalidating our proof of Pareto optimality in the case of failures.

Here, we design Dummy EBS, a scheme that modifies EBS to work with some nodes missing. We will show that by distributing the missing nodes evenly, Dummy EBS is still able to achieve a Pareto optimal latency-throughput tradeoff, up to a constant factor. This proof means that Shale can easily be adapted to work for arbitrary system sizes while preserving a Pareto optimal latency-throughput tradeoff. Additionally, it provides the basis for our claim in Section 2.2.4 that Shale can maintain good performance under failures, provided failed nodes are well-distributed.

3.7.1 Dummy EBS Design

Let h, ε be defined given r as before: $h = \left\lfloor \frac{1}{2r} \right\rfloor$ and $\varepsilon = h + 1 - \frac{1}{2r}$. Let N be the number of nodes we would like to run Dummy EBS on. That is, N is an integer that is not an integer h-power.

Let $M \geq N$ be the smallest such M for which there exists an integer m and $M = m^h$, and consider the h-level EBS design on M nodes. By Theorem 3.2, this design can guarantee throughput $\frac{1}{2h}$ within max latency $(2h + 1)(m - 1) - 1$ in a network of exactly M nodes.

We will designate a set of potential dummy nodes $\mathcal{D}$, and define the EBS dummy node design at level h on N nodes in the following way: choose a subset $D \subseteq \mathcal{D}$ such that $N + |D| = M$. All flow paths that do not travel through nodes in D remain untouched and continue to route flow as specified by EBS on M nodes. Flow paths that travel through D

no longer send flow.

Note that this design will be able to guarantee a slightly smaller throughput value than previously, and some node pairs and starting timeslots may have more flow attributed to them by the routing scheme than others. To fix this second point, one can normalize the amount of flow attributed to each pair by the minimum over all pairs.

To prove what throughput value this design guarantees, we will show that if we eliminate all flow that would have been routed through nodes in $\mathcal{D}$ using EBS, no pair loses more than a δ fraction of flow for sufficiently small δ.

Designate the following set $\mathcal{D}$ as the potential dummy node set,

$$\mathcal{D} = \left\{ \left(i_0, i_1, ..., i_{h-2}, h + \sum_{j=0}^{h-2} i_j \right) : i_0, ..., i_{h-2} \in [m] \text{ and } h \in \{0, ..., h-1\} \right\}$$

Before we check that eliminating all flow on a, b, t triples that would have routed through $\mathcal{D}$ still guarantees a high enough throughput rate, we should first check to ensure that $|\mathcal{D}|$ is large enough. That is, it must be at least $M - N$. Note that N cannot be smaller than $(m-1)^h$, as otherwise there would be a smaller integer h-power M' with $M > M' \geq N$. Therefore, it is enough to show that $|\mathcal{D}|$ is at least $m^h - (m-1)^h$.

Using the fundamental theorem of calculus, one can see that $\int_{m-1}^{m} hx^{h-1} = m^h - (m-1)^h$. The integrand hx^{h-1} will always be no more than hm^{h-1}, since x takes values in the interval $[m-1, m]$. Therefore, $\int_{m-1}^{m} hx^{h-1} \leq hm^{h-1} = |\mathcal{D}|$.

3.7.2 Latency-Throughput Tradeoff of Dummy EBS

Lemma 5. *The EBS dummy node design on N nodes can guarantee throughput* $\frac{1}{2h}\left(1 - \frac{2h^2}{m}\right)$ *and maximum latency* $(2h+1)(m-1) - 1$ *for* $h = \left\lfloor \frac{1}{2r} \right\rfloor$ *and M equal to the smallest integer h-power larger than N, for sufficiently large N.*

Proof. We can bound the guaranteed throughput rate by bounding the amount of flow between any worst case triplet that goes through the set $\mathcal{D}$. Since EBS uses VLB, we can bound the fraction of flow that gets eliminated by dummy nodes for any triple a, b, t by using the total fraction of semi-paths that get eliminated when routing semi-paths of a worst-case node a starting at timeslot t to all intermediate nodes v_{int}.

Consider node $a = (a_0, ..., a_{h-1})$ and dummy node $d = (d_0, ..., d_{h-1}) \in \mathcal{D}$, and suppose t occurs in phase p. We would like to count the number of intermediate nodes v_{int} for which the semi-path from a to v_{int} starting at timeslot t goes through d.

Suppose d and a match in all but one coordinate. Then in order for d to be on the semi-path from a to v_{int} starting at timeslot t, it must be the case that d and a are mismatched in coordinate $p + 1$, and that v_{int} matched d in the $(p + 1)$-th coordinate. There are m^{h-1} such v_{int}, leaving a $\frac{1}{m}$ fraction with the property. If we choose all our dummy nodes from $\mathcal{D}$, then there are at most h dummy nodes that match a in all but the $(p + 1)$-th coordinate.

Consider more generally if a d and a match in all but k consecutive coordinates, starting at phase $p + 1$. If we choose all our dummy nodes from $\mathcal{D}$, then there are at most hm^{k-1} such dummy nodes. And given such a dummy node, the fraction of v_{int} that get eliminated is at most $\frac{1}{m^k}$.

Thus, if we pick all our dummy nodes from $\mathcal{D}$, we can bound the fraction of flow δ_{EBS} that gets eliminated at each node. Note that we gain a factor of 2 due to VLB, by applying this argument for both semi-paths $a \to v_{int}$ and $v_{int} \to b$.

$$\begin{aligned}\delta_{EBS} &\leq 2 \sum_{k=1}^{h} \frac{1}{m^k} h m^{k-1} \\ &\leq 2h \sum_{k=1}^{h} \frac{1}{m} = \frac{2h^2}{m}\end{aligned}$$

So, we can guarantee throughput

$$\begin{aligned} r &\geq \frac{1}{2h}(1 - \delta_{EBS}) \\ &\geq \frac{1}{2h}\left(1 - \frac{2h^2}{m}\right) \end{aligned}$$

□

Since the above goes toward $\frac{1}{2h}$ as $M \longrightarrow \infty$, if we wanted to guarantee a throughput value r for which $\varepsilon \neq 1$, then the EBS dummy node design on any number of nodes N can guarantee throughput r for sufficiently large N.

Lemma 6. *Let $r \in (0, \frac{1}{2})$ be a throughput value and ε, h defined as usual; $h = \left\lfloor \frac{1}{2r} \right\rfloor$ and $\varepsilon = h + 1 - \frac{1}{2r}$. If $\varepsilon \in [\frac{1}{2} \cdot \frac{1}{16(h+1)(1+\frac{1}{2h})^2}, 1)$, then the EBS dummy node design on N nodes achieves maximum latency $O\left(L^*(r, N)\right)$ for sufficiently large N.*

Proof. Let $r \in (0, \frac{1}{2})$ be given as above and let M be the smallest integer h-power larger than or equal to N.

Since $\varepsilon < 1$, then $r < \frac{1}{2h}$, meaning that for large enough N (and therefore M), $r \geq \frac{1}{2h}\left(1 - \frac{2h^2}{m}\right)$. Therefore for large enough N, the EBS dummy node scheme can guarantee throughput r and achieve maximum latency $(2h + 1)(m - 1) - 1$.

To show that $(2h + 1)(m - 1) - 1 \leq O\left(L^*(r, N)\right)$, we use the fact that $N \geq (m - 1)^h$. Therefore $2hN^{1/h}$ is at least $2hm\left(1 - \frac{h}{m}\right)^{1/h}$ which is no more than a constant factor less than $(2h + 1)(m - 1) - 1$ for sufficiently large N.

Finally, when $\varepsilon \geq \frac{1}{2} \cdot \frac{1}{16(h+1)(1+\frac{1}{2h})^2}$, by Lemma 2 the maximum latency $2hN^{1/h} \leq O\left(L^*(r, N)\right)$, completing our proof. □

3.8 Summary

This chapter makes the first attempt to formally study ORNs from a theoretical perspective. We develop a formalized model of ORNs which we use to determine the fundamental limits of the throughput and latency achievable by ORNs which support arbitrary traffic demands. Specifically, we prove both an upper and lower bound on the Pareto front of optimal ORN designs. These bounds are tight with each other up to a constant factor in latency, fully characterizing the Pareto front.

To prove our upper bound, we mathematically constructed two families of ORN designs. One of these, EBS, uses the same schedule and routing scheme as Shale. This allows us to prove the Pareto optimality of Shale. Additionally, we show that our constructed designs can be extended in two ways while maintaining Pareto optimality: We extend both designs to degree $d > 1$, and we extend EBS to work with missing nodes. This shrinks the gap between our theoretical results and practical systems, which will often have failed nodes and may have multiple network ports at each node.

CHAPTER 4

RELATED WORK

ORN designs based on the Single Round Robin Design RotorNet [49] proposes a datacenter-wide ORN based on Rotor switches, a form of optical circuit switch that requires committing to a fixed schedule. Prototype Rotor switches were demonstrated with a reconfiguration time of 150 μs, and production switches have since been demonstrated with a reconfiguration time of around 7 μs [48]. Sirius [7] evolves on RotorNet by using tunable lasers and diffraction gratings to reconfigure the network far more rapidly, achieving a guard band of only 3.84 ns. Shoal [68] proposes an electronic ORN within a resource-disaggregated rack. By using circuit switches, Shoal reduces power consumption compared to packet switches, supporting 100s of nodes in a single rack.

While there has been clear progress in improving the hardware capabilities and reach of ORNs, these designs all use a schedule and routing scheme similar to the one described in Section 1.1. As a result, all of these systems have poor latency scalability. Shale represents a major step forward for ORNs by introducing a tunable tradeoff between throughput and latency. Because it is agnostic to the specific hardware implementation, Shale can extend all three of these designs, enabling them to scale to far larger environments.

Other ORN and ORN-related designs Opera [47] uses greatly expanded timeslots, configuring the network as a new expander graph during each timeslot, allowing latency-sensitive traffic to be routed via multiple hops on a single configuration. We discuss the consequences of this design and compare it to interleaving in Shale in Section 2.2.2. Cerberus [23] uses a derivative of the RotorNet ORN design as one component of an optical datacenter network, along with demand-aware reconfiguration and static graphs. This demonstrates the potential of using ORNs as a building block of demand-aware networks, as we propose to do with interleaving. Mars [1] analyzes ORNs through the properties of

the time-collapsed connection graph, and uses this to derive a tradeoff between throughput and buffering of forwarded cells. It proposes a schedule based on emulating De Bruijn graphs due to their low diameter and degree. However, as with [4] it does not consider the effects of queuing at intermediate nodes. Duo [83] expands on Mars by adding support for dynamic links based on observed traffic demands. This dissertation significantly contributes to this active, rich research area.

Non-oblivious reconfigurable networks. Jupiter Evolving [57] augments a traditional datacenter network by connecting machine aggregation blocks with optical switches. These switches reconfigure infrequently, and use traffic engineering to ensure efficient use of the resulting direct-connect topology. TopoOpt [79] uses optical switches to create an optimized direct-connect topology for deep neural network model training, and customizes the all-reduce algorithm to best match the resulting topology. The Lightwave Fabrics paper [45] combines both an in-chip-interconnection and a datacenter-wide optical network to create optimized fixed topologies for machine learning jobs. These works demonstrate the incredible potential of optical circuit switches, but require extensive knowledge or even engineering of workloads, reducing their flexibility compared to ORNs.

Ethernet flow control. Ethernet flow control attempts to prevent packet loss, creating a lossless link layer. PAUSE frames [70] allow an overloaded node to request that its neighbors pause sending. This pause affects all traffic, potentially creating new congestion at previous hops which can cascade throughout the network [61]. Priority Flow Control, or PFC [71], allows separate pausing of 8 traffic classes, but otherwise suffers from similar weaknesses. Thanks to Shale's highly regular schedule and routing, we show that it is practical to avoid head-of-line blocking in **`hop-by-hop`**, avoiding these issues.

Asynchronous Transfer Mode (ATM). ATM networks operate using virtual circuits, in which end hosts must inform intermediate switches of their traffic characteristics and negotiate a path before sending traffic. Under credit-based congestion control [31], ATM switches implemented flow control on a per-virtual-circuit basis, with nodes exchanging credits to indicate free buffer space, similar to tokens in **`hop-by-hop`**. By maintaining credit per-destination, **`hop-by-hop`** is able to achieve similar isolation capabilities to credit-based congestion control while being coordination free.

TCP in reconfigurable networks. TCP is the most commonly used network transport protocol both within datacenters and in the wider internet. It provides both reliable delivery and congestion control. **Time-Division TCP** [14], as well as the techniques developed in [56], adapt TCP to better utilize traditional reconfigurable datacenter networks. These techniques are suited for hybrid networks (a combination of packet and circuit switches), which use circuit switching technology with high reconfiguration delays, and thus switch less frequently and in response to the traffic.

Load-Balanced Switches. The load-balanced switch architecture proposed by Chang [13] uses static schedules and sends traffic obliviously via intermediate nodes. While there are significant similarities between this architecture and ORNs, it differs in its use of specialized intermediate nodes (rather than sending traffic via multiple end-hosts), as well as its focus on monolithic switches.

Oblivious routing in general networks. Räcke's seminal 2002 paper [58] proved the existence of polylog(n)-competitive oblivious routing schemes in general networks. Subsequent work improved the competitive ratio [29] and devised polynomial-time algorithms for computing an oblivious routing scheme that meets this bound [11, 29, 6]. Räcke's 2008

paper [59] yielded an $O(\log n)$-competitive oblivious routing scheme, computed by a fast, simple algorithm based on multiplicative weights and the randomized approximation of general metric spaces by tree metrics in [18]. The effectiveness of Räcke's 2008 routing scheme for wide-area traffic engineering in practice was demonstrated in [5, 40]. Additionally, Gupta, Hajiaghayi, and Räcke [25] show a polylog(n) competitive ratio for routing schemes oblivious to both traffic and the cost functions associated with each edge. While these works achieve excellent congestion minimization over general networks, they do not specifically consider throughput or latency, and do not attempt to co-design the network with their routing scheme.

With respect to throughput, Hajiaghayi, Kleinberg, Leighton, and Räcke [27] prove a lower bound of $\Omega(\frac{\log n}{\log \log n})$ on the competitive ratio achievable by oblivious routing in general networks. However, their definition of throughput differs from the one used in this dissertation; they simply mean the combined flow rate delivered to all sender-receiver pairs. With respect to latency, the competitive ratio of average latency of oblivious routing over general networks is analyzed by [30]. Their model of latency differs from the one used in this dissertation; they assign resistance values to each edge, and they only provide an oblivious routing scheme achieving the $O(\log(N))$-competitive ratio when routing to a single target.

Valiant load balancing in hypercubes and other architectures. Leslie Valiant introduced oblivious routing in [75]. Valiant and Brebner introduced the Valiant load balancing (VLB) scheme for randomized routing in the hypercube, and showed it to be optimal in this context [76, 75]. While these works evaluate latency under queuing, they do not evaluate throughput. Additionally, they use a direct-connect torus topology. The proofs in Chapter 3 can be interpreted as proving that VLB is the optimal oblivious routing scheme to use in conjunction with an optimally-designed reconfigurable network topology. This provides

further theoretical justification for the widespread usage of VLB in practice when oblivious routing is applied on handcrafted network topologies.

For the case where $N = 2^h$, EBS can be seen as simulating VLB over a hypercube. In this case, the connection schedule simulates an h-dimensional hypercube by cycling through h different connection graphs, each with degree 1. While the latency bound achieved by EBS in this case is $O(log(N))$, the same bound found in [75], the differing contexts make these bounds incomparable. In particular, while Valiant considered queuing delay, we consider latency intrinsic to the scheduled nature of ORNs.

A lower bound for deterministic oblivious routing in d-regular networks with N nodes was proven in [34]; the same paper shows this bound is tight for hypercube networks, in which $d = \log(N)$.

CHAPTER 5

FUTURE WORK

Throughout this dissertation, we have focused on studying optimal, scalable Oblivious Reconfigurable Networks (ORNs) which can support arbitrary traffic demands. Shale provides a blueprint for achieving many different Pareto optimal latency-throughput tradeoffs when this flexibility is needed. However, many applications have known demand characteristics. There is significant room for future advances by thoughtfully designing and extending ORNs to match the needs of particular applications.

ORNs for Machine Learning Model Training Machine learning is a massive and growing area of interest and investment. Model training, which underpins machine learning, is a major driver of growing datacenter traffic demands [9]. Fortunately, model training exhibits extremely predictable traffic demands, providing a natural opportunity for optimization. When training the same model with the same parallelization strategy, network accesses follow a consistent, periodic pattern [79]. Conventional datacenter networks based on a multi-tiered arrangement of packet switches are ill-suited to the particular traffic demands of model training, which has resulted in the network becoming a performance bottleneck for model training.

Together, these observations have motivated the use of OCSes to create static, optimized network topologies for model training [79, 45]. In these systems, machine learning models are trained on purpose-built clusters which use circuit switches to implement their internal network. Before a machine learning model is trained, a model-specific optimized network topology is generated for it and the circuit switches are reconfigured appropriately.

ORNs could be adapted for this context in a similar way. A purpose-made schedule could be created for each model, simulating an optimized topology for its particular traffic demand. This would no longer strictly speaking be an *oblivious* reconfigurable network,

potentially allowing it to significantly outperform the latency-throughput tradeoff shown in Chapter 3.

Semi-Oblivious Reconfigurable Networks for Datacenters Compared to model training, general datacenter traffic is far more difficult to precisely predict. Most datacenter applications do not exhibit the same tightly synchronized, periodic behavior as machine learning model training. However, while it is impossible to predict individual flows in advance, large-scale patterns in datacenter traffic demands are often quite predictable and stable. This suggests that datacenters may be well-served by *semi*-oblivious reconfigurable networks (SORNs), which operate as ORNs on short timescales, but periodically adapt their schedules and routing schemes as large-scale patterns change over time.

The predictability of datacenter workloads comes from the predictability of the network use of individual applications. Different applications, such as user-facing web services, machine learning, and caches, each exhibit characteristic traffic and communication patterns. This predictability is made easier to exploit by the fact that applications often have predictable spatial distributions within the datacenter, due to distinct hardware requirements. Servers and racks in a datacenter are arranged in a spatial hierarchy of pods, clusters, or blocks to facilitate management [24, 62, 69]. Scheduling and job placement algorithms use this hierarchy to map requests to resources, and similar applications are often co-located due to coinciding hardware requirements. For their own workloads, datacenter operators sometimes designate specific clusters of servers for distinct roles [62]. However, even in public clouds, network requirements often fit into a spatial hierarchy. For example, resource requests can specify preferences for co-location within the datacenter, and operators report that the distribution of these requests follows predictable patterns [26].

Application workloads vary over time, but on the order of minutes or hours rather than microseconds. By periodically optimizing its schedule to match current structural

patterns, a fast circuit-switched network should be able to consistently remain in step with large-scale demand patterns. This can be done on far longer timescales than would be required to react to individual flows, making it practical even at datacenter scales. While it may be difficult to exploit the specific patterns of each individual application, some patterns consistently appear in different variations across different workloads.

Addressing In-Network Computing Tasks in ORNs With the advent of in-network computing, datacenter networks have become active participants in data processing. Programmable packet switches have been used to accelerate applications including storage [43, 33], consensus [32, 16], telemetry [85, 37, 46], network functions [82, 81], and load balancing [51, 35], with recent work bringing support for multitenancy [78, 39, 36]. However, OCSes are specifically designed not to process the data that traverses them. It is not yet understood how to best integrate computing within networks based on rapid OCSes.

Some in-network computing systems primarily use a single top-of-rack switch to load-balance or cache requests to nodes within the rack [43, 33]. These can be supported by existing fast circuit-switched network designs, assuming that the nodes participating in the circuit-switched network are themselves top-of-rack switches as proposed in several ORN designs [49, 47, 7]. However, other designs require multiple switches to participate, potentially in multiple locations in a single path. For example, P4xos uses spine, aggregate, and top-of-rack switches to serve different roles in Paxos [16]. Fast circuit-switched networks use a completely different structure than existing networks which changes the nature and organization of in-network computing.

In an ORN, while the switches are unable to participate, indirect routing creates an opportunity for in-network computing as data traverses intermediate nodes. Indeed, Shale makes use of this to implement its hop-by-hop congestion control. However, in

existing, uniform network designs, paths can traverse arbitrary nodes on the way to their destination, potentially requiring applications to be deployed to every single node in the system. Future ORN designs may focus on routing between localized cliques, which would allow applications to be deployed within a single clique only.

One potential alternative may be to relocate workloads which were once accelerated by switches into new, purpose-built accelerators. Such purpose-built accelerators are already in use in some production datacenters to reduce the need for programmable FPGA NICs at end-hosts [8]. In this case, a significant portion of traffic may need traverse these accelerators in order to be processed on the way to its destination. This would significantly change the traffic demands placed on the network, offering an opportunity to optimize the network topology and organization to ideally serve this predictable demand pattern.

CHAPTER 6

CONCLUSION

As packet switched networks reach scaling limits, optical circuit switches stand poised to take over as the new dominant switch technology in datacenter networks. In particular, rapid nanosecond-scale circuit switches promise to fully replace packet switches for all traffic, but require novel network designs to do so. This dissertation identifies the Oblivious Reconfigurable Network (ORN) design paradigm as one which can most effectively use the capabilities of rapid circuit switches. It significantly advances the state of the art in this research area, bringing the vision of a fully circuit-switched datacenter network closer to reality.

First, this dissertation shows that ORNs can be made practical at datacenter scale. We develop Shale, an ORN which achieves a tunable, Pareto optimal tradeoff between throughput and latency scalability. At datacenter scale, Shale achieves orders of magnitude better latency and memory requirements than earlier ORNs, which were universally based on similar single round-robin designs. We additionally develop congestion control mechanisms suitable for multi-hop ORNs, and show how ORNs can be composed through interleaving to combine the benefits of multiple schedules. Finally, we implement a prototype end-host for Shale, demonstrating several optimizations that can be used to efficiently implement both Shale and our multi-hop compatible congestion control.

Second, this dissertation undertakes the first attempt to formally study ORNs from a theoretical perspective. We create a mathematical model of ORNs, and use it to prove both an upper and a lower bound on the Pareto optimal latency-throughput tradeoffs achievable by ORNs which support arbitrary traffic demands. Our analysis provides the proof that Shale's tradeoffs are Pareto optimal. It also provides a rigorously proven reference point for evaluating future ORN designs.

Overall, this dissertation makes multiple major advances to help ORNs mature into a viable network technology. As datacenter operators seek to end their reliance on packet switches, the results presented here will be crucial ingredients in their next-generation networks.

BIBLIOGRAPHY

[1] Vamsi Addanki, Chen Avin, and Stefan Schmid. Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 7(1):1–43, 2023.

[2] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Pfabric: Minimal near-optimal datacenter transport. In *SIGCOMM*, 2013.

[3] Ethernet Alliance. 2024 ethernet roadmap. `https://ethernetalliance.org/technology/ethernet-roadmap/`, 2024.

[4] 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, pages 1339–1352, New York, NY, USA, 2022. Association for Computing Machinery.

[5] David L. Applegate and Edith Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. 2003.

[6] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Räcke. Optimal oblivious routing in polynomial time. 2003.

[7] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, et al. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 782–797, 2020.

[8] Deepak Bansal, Gerald DeGrace, Rishabh Tewari, Michal Zygmunt, James Grantham, Silvano Gai, Mario Baldi, Krishna Doddapaneni, Arun Selvarajan, Arunkumar Arumugam, Balakrishnan Raman, Avijit Gupta, Sachin Jain, Deven Jagasia, Evan Langlais, Pranjal Srivastava, Rishiraj Hazarika, Neeraj Motwani, Soumya Tiwari, Stewart Grant, Ranveer Chandra, and Srikanth Kandula. Disaggregating stateful network functions. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1469–1487, Boston, MA, April 2023. USENIX Association.

[9] S. J. Ben Yoo. Prospects and challenges of photonic switching in data centers and computing systems. *Journal of Lightwave Technology*, 40(8):2214–2243, 2022.

[10] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. Association for Computing Machinery.

[11] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. 2003.

[12] Bluespec SystemVerilog. `http://wiki.bluespec.com/bluespec-systemverilog-and-compiler`.

[13] Cheng-Shang Chang, Duan-Shin Lee, and Yi-Shean Jou. Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering. *Computer Communications*, 25(6):611–622, 2002.

[14] 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, pages 19–35, New York, NY, USA, 2022. Association for Computing Machinery.

[15] Q. Cheng, A. Wonfor, J. L. Wei, R. V. Penty, and I. H. White. Demonstration of the feasibility of large-port-count optical switching using a hybrid Mach-Zehnder interferometer-semiconductor optical amplifier switch module in a recirculating loop. *Opt. Lett.*, 39(18):5244–5247, Sep 2014.

[16] Huynh Tu Dang, Pietro Bressana, Han Wang, Ki Suh Lee, Noa Zilberman, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé. P4xos: Consensus as a network service. *IEEE/ACM Transactions on Networking*, 28(4):1726–1738, 2020.

[17] M. Ding, A. Wonfor, Q. Cheng, R. V. Penty, and I. H. White. Scalable, low-power-penalty nanosecond reconfigurable hybrid optical switches for data centre networks. In *2017 Conference on Lasers and Electro-Optics (CLEO)*, pages 1–2, May 2017.

[18] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3), 2004.

[19] 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. 2010.

[20] Daniel Firestone. Hardware-accelerated networks at scale in the cloud. Keynote at ACM SIGCOMM 2017 Workshop on Kernel-Bypass Networks, 2017.

[21] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. pHost: Distributed near-optimal datacenter transport over commodity network fabric. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, pages 1–12, 2015.

[22] 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 *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[23] 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. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–33, 2021.

[24] 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 on Data Communication*, SIGCOMM '09, New York, NY, USA, 2009. Association for Computing Machinery.

[25] Anupam Gupta, Mohammad Taghi Hajiaghayi, and Harald Räcke. Oblivious network design. 2006.

[26] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, et al. Protean:{VM} allocation service at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 845–861, 2020.

[27] Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, Frank Thomson Leighton, and Harald Räcke. New lower bounds for oblivious routing in undirected graphs. 2006.

[28] 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 *SIGCOMM*, 2017.

[29] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. 2003.

[30] Prahladh Harsha, Thomas P. Hayes, Hariharan Narayanan, Harald Räcke, and Jaikumar Radhakrishnan. Minimizing average latency in oblivious routing. 2008.

[31] Raj Jain. Congestion control and traffic management in atm networks: Recent advances and a survey. *Computer Networks and ISDN systems*, 28(13):1723–1738, 1996.

[32] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. NetChain: Scale-Free Sub-RTT coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, Renton, WA, April 2018. USENIX Association.

[33] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 121–136, New York, NY, USA, 2017. Association for Computing Machinery.

[34] Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. Tight bounds for oblivious routing in the hypercube. *Math. Syst. Theory*, 24(4):223–232, 1991.

[35] 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*, pages 1–12, 2016.

[36] Sajy Khashab, Alon Rashelbach, and Mark Silberstein. Multitenant {In-Network} acceleration with {SwitchVM}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 691–708, 2024.

[37] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, Lawrence J Wobker, et al. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*, volume 15, pages 1–2, 2015.

[38] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Reetuparna Das, Yuan Xie, Vijaykrishnan Narayanan, Mazin S. Yousif, and Chita R. Das. A novel dimensionally-decomposed router for on-chip communication in 3d architectures. 2007.

[39] Johannes Krude, Jaco Hofmann, Matthias Eichholz, Klaus Wehrle, Andreas Koch, and Mira Mezini. Online reprogrammable multi tenant switches. In *Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms*, ENCP '19, page 1–8, New York, NY, USA, 2019. Association for Computing Machinery.

[40] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiunlin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. 2018.

[41] Ki Suh Lee, Han Wang, Vishal Shrivastav, and Hakim Weatherspoon. Globally

synchronized time via datacenter networks. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 454–467, New York, NY, USA, 2016. Association for Computing Machinery.

[42] Jason Lei and Vishal Shrivastav. Seer: Enabling Future-Aware online caching in networked systems. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 635–649, Santa Clara, CA, April 2024. USENIX Association.

[43] Jialin Li, Jacob Nelson, Ellis Michael, Xin Jin, and Dan R. K. Ports. Pegasus: Tolerating skewed workloads in distributed storage with In-Network coherence directories. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 387–406. USENIX Association, November 2020.

[44] 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.

[45] Hong Liu, Ryohei Urata, Kevin Yasumura, Xiang Zhou, Roy Bannon, Jill Berger, Pedram Dashti, Norm Jouppi, Cedric Lam, Sheng Li, et al. Lightwave fabrics: At-scale optical circuit switching for datacenter and machine learning systems. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 499–515, 2023.

[46] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[47] 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.

[48] William M. Mellette, Alex Forencich, Rkushani Athapathu, Alex C. Snoeren, George Papen, and George Porter. Realizing RotorNet: Toward practical microsecond-scale optical networking. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '24, New York, NY, USA, 2024. Association for Computing Machinery.

[49] 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. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 267–280, New York, NY, USA, 2017. Association for Computing Machinery.

[50] William M. Mellette and George Porter. opera-sim. `https://github.com/TritonNetworking/opera-sim`, 2020.

[51] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[52] ModelSim-Intel® FPGAs Standard Edition Software. https://www.intel.com/content/www/us/en/software-kit/750637/modelsim-intel-fpgas-standard-edition-software-version-20-1.html.

[53] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *SIGCOMM*, 2018.

[54] Timothy Prickett Morgan. Like a drumbeat, Broadcom doubles Ethernet bandwidth with "Tomahawk 5". `https://www.nextplatform.com/2022/08/16/like-a-drumbeat-broadcom-doubles-ethernet-bandwidth-with-tomahawk-5/`, August 2022.

[55] QSFP-DD MSA. QSFP-DD hardware specification for QSFP double density 8x pluggable transceiver. `http://www.qsfp-dd.com/wp-content/uploads/2020/08/QSFP-DD-Hardware-rev5.1.pdf`, 2020.

[56] 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.

[57] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. Jupiter evolving: transforming Google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 66–85, 2022.

[58] H. Räcke. Minimizing congestion in general networks. 1975.

[59] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. 2008.

[60] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath TCP. volume 41, pages 266–277. ACM New York, NY, USA, 2011.

[61] S-A Reinemo, Tor Skeie, Thomas Sodring, Olav Lysne, and O Trudbakken. An overview of QoS capabilities in InfiniBand, advanced switching interconnect, and ethernet. *IEEE Communications Magazine*, 44(7):32–38, 2006.

[62] 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.

[63] Ken-ichi Sato. Optical switching will innovate intra data center networks [invited tutorial]. *Journal of Optical Communications and Networking*, 16(1):A1–A23, 2023.

[64] Vishal Shrivastav. Fast, scalable, and programmable packet scheduler in hardware. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, pages 367–379, New York, NY, USA, 2019. Association for Computing Machinery.

[65] Vishal Shrivastav. Programmable multi-dimensional table filters for line rate network functions. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '22, pages 649–662, New York, NY, USA, 2022. Association for Computing Machinery.

[66] Vishal Shrivastav. Stateful multi-pipelined programmable switches. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '22, pages 663–676, New York, NY, USA, 2022. Association for Computing Machinery.

[67] Vishal Shrivastav, Ki Suh Lee, Han Wang, and Hakim Weatherspoon. Globally synchronized time via datacenter networks. *IEEE/ACM Transactions on Networking*, 27(4):1401–1416, 2019.

[68] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A network architecture for disaggregated racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, Boston, MA, 2019. USENIX Association.

[69] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, and et al. Jupiter

rising: A decade of Clos topologies and centralized control in google's datacenter network. *SIGCOMM Comput. Commun. Rev.*, 45(4):183–197, August 2015.

[70] IEEE Computer Society. IEEE standards for local and metropolitan area networks: Specification for 802.3 full duplex operation. *IEEE Std 802.3x-1997 and IEEE Std 802.3y-1997 (Supplement to ISO/IEC 8802-3: 1996/ANSI/IEEE Std 802.3, 1996 Edition)*, 1997.

[71] IEEE Computer Society. IEEE standard for local and metropolitan area networks–media access control (MAC) bridges and virtual bridged local area networks–amendment 17: Priority-based flow control. *IEEE Std 802.1Qbb-2011 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011 and IEEE Std 802.1Qbc-2011)*, pages 1–40, 2011.

[72] Intel® Stratix® Series FPGAs and SoCs. `https://www.intel.com/content/www/us/en/products/details/fpga/stratix.html`.

[73] DE5-Net FPGA development kit. `http://de5-net.terasic.com.tw`.

[74] International Telecommunications Union. Measuring digital development: Facts and figures 2023. `https://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx`, 2023.

[75] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11(2):350–361, 1982.

[76] 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.

[77] 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. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 327–338, New York, NY, USA, 2010. Association for Computing Machinery.

[78] Tao Wang, Hang Zhu, Fabian Ruffy, Xin Jin, Anirudh Sivaraman, Dan R. K. Ports, and Aurojit Panda. Multitenancy for fast and programmable networks in the cloud. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association, July 2020.

[79] 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, 2023.

[80] Wikipedia. Ethernet physical layer — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Ethernet_physical_layer&oldid=1232800774`, 2024. [Online; accessed 10-July-2024].

[81] Lior Zeno, Ang Chen, and Mark Silberstein. In-network address caching for virtual networks. 2024.

[82] Lior Zeno, Dan RK Ports, Jacob Nelson, Daehyeok Kim, Shir Landau-Feibish, Idit Keidar, Arik Rinberg, Alon Rashelbach, Igor De-Paula, and Mark Silberstein. {SwiSh}: Distributed shared state abstractions for programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 171–191, 2022.

[83] Johannes Zerwas, Csaba Györgyi, Andreas Blenk, Stefan Schmid, and Chen Avin. Duo: A high-throughput reconfigurable datacenter network using local routing and control. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(1), mar 2023.

[84] Johannes Zerwas, Wolfgang Kellerer, and Andreas Blenk. What you need to know about optical circuit reconfigurations in datacenter networks. In *2021 33th International Teletraffic Congress (ITC-33)*, pages 1–9. IEEE, 2021.

[85] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. Flow event telemetry on programmable data plane. 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 76–89, 2020.

GLOSSARY

Bandwidth. The capacity at which a network can transmit data. Depending on the context, bandwidth may refer to a specific link (in which case it is equivalent to line rate), to the total capacity between two portions of the network (as in bisection bandwidth), or to the total capacity across the entire network. *See also: throughput* & *line rate*

Cell. A packet with a fixed length. Standardizing on a fixed length enables ORNs to use the entirety of each fixed-length timeslot. 7, *See also: packet*

Circuit. A dedicated, preestablished communication link between two ports in a circuit switch, or more broadly two nodes in a network. 4, *See also: circuit switch*

Circuit reconfiguration. The process of setting or changing the circuits implemented by a circuit switch. 5, *See also: circuit* & *circuit switch*

Circuit switch. A network switch that forwards data according to preestablished circuits. 4, *See also: circuit*, *network switch*, *packet switch* & *optical circuit switch*

Connection schedule. A schedule which dictates which circuits should be deployed on a circuit switch at which time. Oblivious reconfigurable networks use a periodic traffic-oblivious schedule to determine which circuits to deploy. 6, *See also: circuit*, *circuit switch*, *oblivious reconfigurable network*, *timeslot* & *guard band*

Constraint. The conditions that a solution to a linear program must satisfy. For example, consider the linear program on page 90, which aims to generate a routing scheme for a given connection schedule that maximizes the guaranteed throughput r while respecting a given maximum latency. This linear program has three constraints, encoding the following three properties the routing scheme must satisfy:

- The routing scheme routes r units of flow from each source to each destination.
- The routing scheme does not overload any physical edges, regardless of the traffic demand.

- The routing scheme does not route negative flow on any path.

13, *See also: linear program* & *objective value*

Direct hop. A single hop within the direct semi-path. 8, 20, *See also: direct semi-path*

Direct semi-path. In this dissertation, the portion of the path from the intermediate node to the final destination (under Valiant load balancing). 20, *See also: Valiant load balancing* & *direct hop*

***d*-regular.** A graph is d-regular if every vertex has degree d. In the context of our theoretical analysis of ORNs, this can be interpreted as every node having d ports, allowing it to connect to d neighbors simultaneously. 56

Dual linear program. A linear program (LP) derived from another LP, referred to as the primal LP, using the following systematic method:

- Each constraint in the primal LP becomes a variable in the dual LP.
- Each variable in the primal LP becomes a constraint in the primal LP.[1]
- The objective direction (maximization or minimization) is reversed.

According to the weak duality theorem, any feasible solution of the dual LP bounds the feasible objective values of the primal LP, and vice versa. 13, *See also: primal linear program*, *linear program*, *constraint* & *objective value*

End-host. A device that originates or receives data within a network, such as a server or computer. 5

Epoch. One single iteration of the full connection schedule of an oblivious reconfigurable network. 8, 20, *See also: oblivious reconfigurable network*, *connection schedule* & *phase*

[1]More specifically, for an LP expressed in standard form, the constraint matrix (which contains the coefficient of each variable in each constraint) is transposed, and the roles of the coefficient vector and right-hand-side vector are reversed.

Flow. A sequence of packets sent from a particular source to a particular destination within a network. 5

Guard band. In an oblivious reconfigurable network, the portion of the connection schedule in between timeslots. The guard band is used to reconfigure switches and absorb minor clock desynchronization. 7, *See also: oblivious reconfigurable network*, *connection schedule* & *timeslot*

Header. The portion of a packet that carries control information. This includes the destination, as well as any other information needed to route the packet. 3, *See also: packet*, *packet switch*, *payload* & *routing*

Intrinsic latency. Latency in an ORN resulting solely from the properties of the connection schedule and the oblivious routing scheme. Intrinsic latency ignores other sources of latency, such as queuing delay and propagation delay. 19, *See also: latency*, *connection schedule* & *oblivious routing scheme*

Latency. The amount of time taken for an individual piece of data to arrive at at its destination after being sent. 8, *See also: throughput* & *intrinsic latency*

Line rate. The speed at which data is physically sent over a given network interface. 3, *See also: bandwidth* & *network interface*

Linear program. A particular problem to be optimized through linear programming. A linear program comprises a set of constraints and an objective value to be minimized or maximized. 13, *See also: linear programming*, *objective value* & *constraint*

Linear programming. A method to optimize the solution of a mathematical model when both the objective value (the value to be optimized) and the constraints (requirements imposed on the solution) can be represented by linear relationships. 13, *See also: linear program*, *dual linear program*, *objective value* & *constraint*

Moore's law. An empirical observation that the number of transistors on an integrated circuit doubles roughly every two years. While this observation held for several decades, scaling has slowed since 2010 and is expected to slow further. 3

Network interface. A hardware device that enables a network node to send and receive data over a given physical medium. *See also: port*

Network switch. A device which is used to connect and direct data between devices in a network. Switches forward data they receive via a port leading towards the destination. *See also: packet switch* & *circuit switch*

Node. A device attached to a network which is capable of sending and receiving data. Some ORN designs assume that the nodes participating in the ORN are all end-hosts. Other designs assume that the nodes participating in the ORN are top-of-rack packet switches which the end-hosts are directly connected to. 7, *See also: end-host* & *packet switch*

Objective value. The value a linear program aims to optimize. For example, consider the linear program on page 90, which aims to generate a routing scheme for a given connection schedule that maximizes the guaranteed throughput while respecting a given maximum latency. Here, the objective value is r, the maximum guaranteed throughput. 13, *See also: linear program* & *constraint*

Oblivious reconfigurable network. A network design paradigm in which circuit switches are reconfigured according to a traffic-oblivious connection schedule, and data is routed following an oblivious routing scheme. This dissertation identifies this paradigm and proposes its use in networks which use rapid circuit switches. 6, *See also: circuit switch*, *circuit reconfiguration*, *connection schedule*, *oblivious routing scheme* & *rapid circuit switch*

Oblivious routing. A method for routing in which the paths used to route data depend only on the source and destination. Routing paths are not adjusted in light of network

conditions such as traffic demand or congestion. *See also: routing*, *oblivious routing scheme* & *traffic demand*

Oblivious routing scheme. In a network using oblivious routing, the set of predetermined paths used to route data between each source-destination pair. Oblivious routing schemes may also use weights to set the probability for each path to be used. 6, 7, *See also: oblivious routing* & *routing*

Optical circuit switch. A circuit switch that operates on optical signals. Optical circuit switches can maintain data in light form as it traverses the switch, unlike packet switches. 4, *See also: circuit*, *circuit switch*, *network switch* & *packet switch*

Optical transceiver. A device that converts electrical signals to optical signals, and vice versa. 4

Packet. A formatted unit of data carried by a packet-switched network. A packet consists of control information (the header) and application data (the payload). 2, *See also: header*, *payload* & *packet switch*

Packet switch. A network switch that forwards data in the form of packets. Packet switches read the packet header of each packet they receive and process it to determine where it should be sent next. 2, *See also: packet*, *header*, *network switch* & *circuit switch*

Pareto front. The set of Pareto optimal solutions to a given problem. Chapter 3 of this dissertation investigates the possible tradeoffs ORNs can achieve between throughput and latency. It fully characterizes the Pareto front up to a constant factor. 13, *See also: Pareto front*

Pareto optimal. A state in which different resources are optimized such that it is impossible to improve one resource without worsening another. Chapter 3 of this dissertation explores Pareto optimal tradeoffs between throughput and latency in ORNs. 12, *See also: Pareto front*

Payload. The portion of a packet that carries application data. *See also: packet* & *header*

Phase. In Shale's schedule, a single round-robin. Shale's schedule is composed of h phases, one for each of the h coordinates. The nodes that connect to each other in a given phase only differ in the coordinate corresponding to that phase. 20

Port. A physical opening which a network cable can be plugged into. 4, *See also: network interface*

Primal linear program. The original linear program used to generate a dual linear program. 13, *See also: dual linear program* & *linear program*

Rapid circuit switch. In the context of this dissertation, a circuit switch that can reconfigure its circuits within a few nanoseconds. 6, *See also: circuit switch* & *circuit reconfiguration*

Routing. The process of selecting a path for traffic to travel through a network. 3, *See also: oblivious routing*

Spraying hop. A single hop within the spraying semi-path. 8, 20, *See also: spraying semi-path*

Spraying semi-path. In this dissertation, the portion of the path from the original sender to the intermediate node (under Valiant load balancing). 20, *See also: Valiant load balancing* & *spraying hop*

Throughput. The rate at which data is successfully delivered to its destination. In this dissertation, throughput is frequently expressed as a fraction of the total available bandwidth. 9, *See also: bandwidth*

Timeslot. In an oblivious reconfigurable network, the portion of the connection schedule in which circuits are fixed and data can be sent. Oblivious reconfigurable networks use fixed-length timeslots which are long enough to transmit a single cell. 7, *See also: oblivious reconfigurable network*, *connection schedule*, *guard band* & *cell*

Traffic demand. The set of flows that the nodes in a network desire to send. 5, *See also: flow*

Valiant load balancing. A strategy often used in oblivious routing in which data is first sent to a randomly chosen intermediate node, and then forwarded directly to the destination node. Valiant load balancing helps to convert unbalanced traffic demands into more balanced, all-to-all demands, making it an effective strategy for oblivious routing in many networks. *See also: oblivious routing*, *traffic demand*, *spraying semi-path* & *direct semi-path*