UNIVERSITY OF CALIFORNIA SAN DIEGO

Improving the Ecological Effect of Datacenter Networking

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

James Robert McGuinness III

Committee in charge:

Professor George Porter, Chair
Professor George Papen
Professor Alex C. Snoeren
Professor Deian Stefan
Professor Geoffrey M. Voelker

2021

The dissertation of James Robert McGuinness III is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2021

DEDICATION

For those who told me yes, when I feared the answer was no.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Thanks to Professor George Porter, my committee chair, for being the best advisor I could ever ask for. His support, direction, and advice were beyond compare.

Thanks to everyone at the Newman Center, for showing me that I was less than I wanted, but more than I was.

Thanks to Edwin and Sunjay, for being incredible friends and housemates.

Chapter 2 contains material from three different sources co-authored by the dissertation author. Each source is used in a different part of the chapter.

Sections 2.3 and 2.4 have material from a paper that was rejected from publication. William M. Mellette; Rob McGuinness; Alex Forencich; Joseph Ford; Alex C. Snoeren; George Papen; George Porter, 2016. The dissertation author was an investigator and co-author for this material.

Section 2.5 contains material from a publication that appeared in the Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17). William M. Mellette; Rob McGuinness; Arjun Roy; Alex Forencich; George Papen; Alex C. Snoeren; George Porter, Association for Computing Machinery, 2017. The dissertation author was an investigator and co-author for this material.

Section 2.6 incorporates material from a publication that appeared in the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). William M. Mellette; Rajdeep Das; Yibo Guo; Rob McGuinness; Alex C. Snoeren; George Porter, USENIX Association, 2020. The dissertation author was an investigator and co-author for this material.

Chapter 3, in full, is a reprint of the material as it appeared in the Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems (ANCS '18). Rob McGuinness; George Porter, Association for Computing Machinery, 2018. The dissertation author was the primary investigator and author for this paper.

Chapter 4, in full, is a reprint of the work as it appeared in the Workshop on Computing

within Limits (LIMITS 21). Rob McGuinness; George Porter, 2021. The dissertation author was the primary investigator and author for this paper.

VITA

2013-2014	Software Engineering Intern, Yelp, Inc.
2014	B. Eng. in Computer Science *cum laude*, Cornell University
2014	Undergraduate Teaching Assistant, Cornell University
2015	M. Eng. in Computer Science, Cornell University
2015	Graduate Teaching Assistant, Cornell University
2014-2015	Teaching Assistant, Cornell University
2016	Software Engineering Intern, Ph. D., Google, Inc.
2015-2021	Graduate Student Researcher, University of California San Diego
2016-2021	Graduate Teaching Assistant, University of California San Diego
2021	Ph. D. in Computer Science, University of California San Diego

PUBLICATIONS

William M. Mellette, Rob McGuinness, Alex Forencich, Joseph Ford, Alex C. Snoeren, George Papen, and George Porter. "The virtues of being selective: Fully connecting datacenters with incomplete switches". 2016.

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". 2017. *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 267280.

Rob McGuinness. "Evaluating flow control in packet and circuit switched networks". 2017.

Rob McGuinness and George Porter. "Evaluating the performance of software NICs for 100-gb/s datacenter traffic control". 2018. *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems (ANCS '18)*. Association for Computing Machinery, New York, NY, USA, 7488.

Shelby Thomas, Rob McGuinness, Geoffrey M. Voelker, and George Porter. "Dark packets and the end of network scaling". 2018. *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems (ANCS '18)*. Association for Computing Machinery, New York, NY, USA, 1-14.

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”. 2020. *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, USA, 1-18.

Rob McGuinness and George Porter. “Stipulated smartphones for students: The requirements of modern technology for academia”. 2021. *Workshop on Computing within Limits (LIMITS 21)*.

Jennifer Switzer, Rob McGuinness, Pat Pannuto, George Porter, Aaron Schulman, and Barath Raghavan. 2021. “TerraWatt: Sustaining sustainable computing of containers in containers”.

ABSTRACT OF THE DISSERTATION

Improving the Ecological Effect of Datacenter Networking

by

James Robert McGuinness III

Doctor of Philosophy in Computer Science

University of California San Diego, 2021

Professor George Porter, Chair

Datacenter networks are a critical component of computing infrastructure. With recent calls for reducing the carbon emissions of the globe in order to prevent global warming, it is necessary to examine datacenters and how they contribute to the amount of carbon in the atmosphere. I propose how datacenter networks can reduce both the total operational and embodied energy information and communications technology outputs in order to lower impact on the environment.

Future datacenter networks based on optical circuit switches use less operational energy than traditional counterparts. In order to make these a reality, servers must interact with the network in fundamentally different ways. I examine the systemic needs of servers in three

different optical circuit switched networks and develop several methods of enforcing precision transmission flow control in software. Using an in-depth study of the accuracy of networking software for microsecond precision flow control, I observe that networking software alone is only effective at rates up to 40 Gbps. This implies that hardware support is required to accurately transmit at the 400 Gbps rates needed in modern datacenter networking.

Datacenters are poised to aid lowering the embodied energy of smartphones as well. The embodied energy of smartphones are a highly significant contributor to total carbon emissions. Smartphones are increasingly discarded after shorter periods of time, which raises their impact on the environment further. To extend the lifetimes of smartphones and reduce their embodied energy, datacenters can run parts of a phone's browser application. To see how effective this would be in practice, I study how university students use smartphones on academic platforms. In particular, I analyze data from a university's learning management system and find that smartphones roughly four years past their manufacturing date become obsolete in academic settings. I discuss how using datacenter networks for a split-browser solution can reduce the embodied energy of smartphones in academic settings by providing increased device longevity.

Chapter 1

Introduction

Digital computing has become an essential component of everyday life in most areas around the globe. The market for computing servers has continued to grow by approximately 20% year over year [Cor20]. It is inarguable how critical computing infrastructure is to almost every part of society, from regulating power delivery substations to tracking medical record information in hospitals. Significant scientific research is conducted worldwide to further the field of computing and how it can impact and aid humanity.

The trend of increased research, development, and manufacturing of computing devices has caused an upward trend in their total global energy usage and general environmental impact. In 2013, Information and Communications Technology (ICT) accounted for 10% of total global electricity usage, and was projected to grow further [Mil13]. The contribution of smartphone devices to total greenhouse gas emissions (a critical metric for global warming) was 1.4% in 2007, but is projected to be over 14% of total emissions in 2040 [CAS21].

Until recent decades, this growth of energy usage and environmental impact did not garner significant attention. A negative environmental trend of rising global average temperatures, referred to as "global warming", has resulted in a significant interest in curbing these trends.

Energy production is cited as one of the major factors for global warming, contributing to two-thirds of total greenhouse gas emissions [Kha19]. The overall environmental impact of computing devices also goes beyond electricity, with negative ecological effects resulting from the material processing, manufacturing, and disposal of computing devices such as smartphones.

A large component of modern computing is the construction of aggregated networks containing some number of computer servers, known as "datacenter networks". Datacenters may contain anywhere from tens to tens of thousands of servers, and are commonly used for a wide range of tasks, such as running physics simulations [GZKS19] or providing internet search results [SOA+15]. The number of datacenters and the amount of data they process continues to grow, with the amount of internet traffic they process roughly doubling every year [SOA+15]. With the massive growth of datacenter networking, the amount of electricity they use has grown as well; datacenters are projected to consume 13% of total global electricity by 2030 [Sch20].

With datacenter networks being a critical infrastructural component of many modern technologies, it is imperative to both reduce the carbon footprint of datacenter deployments while also leveraging datacenters to lower the environmental impact of computing as a whole. Datacenter technologies are already commonly used to run flexible, offloaded tasks via cloud computing [MAV17]. Datacenters are poised to holistically reduce the carbon footprint of ICT as a whole.

In this dissertation, I answer the question: **How can new, high performance datacenter networks can reduce the environmental impact of computing technologies?** To do this, I focus on the two topics mentioned above: looking at how datacenters can be used to reduce the **operational energy** and **embodied energy** of computing. Operational and embodied represent the two forms of environmental impact computing devices have on the environment, and different chapters of this dissertation will approach each topic in turn.

1.1 A Brief History of Datacenter Networks

Datacenter networking has gone through many iterations over the past century. Original warehouse-scale computers from the 1940s may be thought of as a kind of "datacenter", but I will focus on the history of more modern datacenter designs in this dissertation. Traditional datacenter networking design focuses on the core problem of connecting large numbers of computer servers together in order to provide a greater amount of computation power in comparison to what a single server could provide. These networks provide high speed communications pathways between servers so that they may exchange and further process inputted data and ultimately provide some desired complex output. Modern datacenters achieve this typically via single computers connected together via some network architecture.

In this section, I review the terminology of datacenters as well as describe how datacenter topologies are constructed using various hardware and software components. I also describe how modern datacenter architectures relate to the scaling of operational energy.

1.1.1 The Terminology of Datacenter Networks

In this subsection, I will briefly define terminology relevant to datacenter networks that I use throughout this dissertation. I describe what each term refers to and how it is relevant to a datacenter network.

Server: A server is an industrial-grade computer that is the base of a datacenter network. The goal of a datacenter is to connect some number of servers together in order to create a system that can process and execute software more complex than a single server could alone. Servers may be composed of a traditional set of hardware components, including a CPU, disk, and memory, but recent designs may have more non-traditional servers containing different hardware configurations. A server is also called a **node**, **host/endhost**, or **endpoint**.

Gigabits per second (Gbps): This term represents a scale of the rate of information passed through a network connection. Datacenter networks operate by moving information from one server to another; the faster this operation takes place, the faster the overall computation becomes. This is also referred to as the **network bitrate** or **network speed**.

Network switches: A "switch" is a hardware component that connects some number of computers together. Switches are not exclusive to datacenters, but switches designed for datacenter environments typically require higher power and operate at a faster rate. Datacenter networks are created by connecting switches to a combination of servers and switches. A network switch contains a number of hardware **ports**, which are how inputs and output are connected to the switch.

Network links: A "link" is a connection between two points in the network, typically either between a switch and some other entity. Links usually are physical cables running some physical layer protocol to transfer data between the two connected points. Links may run at a variety of speeds depending on the cable and connected hardware configuration.

Packets: A "packet" is the unit of data that is transferred over network links. Each packet contains data and some amount of header information to direct information about the packet's source, destination, information type, and more. Packets are limited in their maximum size depending on the network configuration. A sequence of packets between a fixed source and destination is referred to as a **flow**, and the collection of packets sent over the network is referred to as **traffic**.

1.1.2 Modern Datacenter Topologies

Modern datacenter networks have gone through several design iterations [SOA+15, AWE19]. While many forms of datacenter architectures exist, most of them are not widely

deployed as they have specific benefits and drawbacks that make them less desirable for general use [ZZZ+12, CXW+16, CWM+15]. Many of these aim to solve problems that are for targeted datacenter workloads, such as supercomputer architectures targeted for physics simulations [GZKS19].

One of the vastly popular designs is the FatTree [AFLV08], which implements a type of datacenter consisting of multiple switching tiers in a pattern fitting to its namesake. Many large datacenter providers use a tree like architecture similar to a FatTree, often comparing to them for performance metrics [SOA+15, GMP+16, MDG+20]. These are also sometimes referred to as "Clos topologies". A multi-tiered tree datacenter topology is of importance when examining energy savings- because the tree must scale to support larger numbers of hosts in the datacenter, these architectures have a superlinear growth of energy required to operate. I will show this effect in Chapter 2.

1.2 Operational Energy: Better Datacenter Networks

"Operational energy" is energy that is consumed during operation, that is, electricity used to power and run a device. Operational energy is the more commonly known component of ICT's impact on global warming, as it composes the significant energy usage that ICT devices use every day. Reducing the operational energy of computing devices is critical to eliminate greenhouse gas emissions that result from high amounts of energy production around the world.

In Chapters 2 and 3, I discuss how to reduce datacenter operational energy via the design of new, more energy efficient datacenters in the form of optically circuit switched datacenter networks. I present my findings on the challenges of realizing such designs in practice, particularly the difficulties of software enforcement of the precise transmission control requirements for these networks. I conclude with a discussion of the limitations software encounters, and how hardware is needed to bridge the gap between the shortcomings that software currently encounters.

1.2.1 Reducing the Operational Energy of Datacenters

Reduction of the operational energy used by a datacenter can come via creating better hardware or software. For example, hardware that runs at a higher bitrate for the same energy cost will reduce the operational energy of a datacenter. Additionally, software that only requires half the network bitrate to execute the same amount of work will also reduce operational energy. Both of these point to how operational energy is a function of the amount of energy required to perform some amount of work. "Reducing" operational energy then, is achieved by **doing the same amount of work for less energy**.

Reducing the operational energy of a datacenter becomes more difficult when considering that new datacenter network designs must not only serve future demands, but simultaneously reduce the amount of total energy used. Datacenters have become increasingly complex, and many research efforts have focused on primarily increasing the *scale* at which datacenters can functionally operate [FGH+21, ASA+21] in order to serve the increasing demands that are placed on datacenter networks. With the work required by datacenters roughly doubling every year [SOA+15], this comes at no surprise, but it makes the task of reducing operational energy seem difficult.

One way to support new, more energy efficient datacenter designs is to increase the overall efficiency of the network. Running workloads on datacenters more efficiently means that the same work is being completed in a shorter period of time, which reduces the overall energy consumed by a datacenter over time. This is a primary goal on road maps aimed at reducing datacenter operational energy [All20], and it fits well with other goals to ensure that a new datacenter network can "meet in the middle" with the scaling requirements for the next-generation. My work in this dissertation will describe how a new form of datacenter networking can improve network efficiency.

Other methods of reducing datacenter operational energy come via more efficient indi-

vidual components, better cooling mechanisms, and power grid profiling. These are beyond the scope of my dissertation.

1.3 Embodied Energy: Longer Lasting Smartphones

Smartphones are now an everyday device that, while having an operational energy cost via charging the battery, come with a significant "embodied energy" cost as well. Embodied energy consists of all of the energy that goes into the construction and disposal of an object, from the mining and processing of raw materials, to recycling its parts for future use. Embodied energy is of critical significance because in many cases the full impact of technology on the environment is not seen by the operational energy alone. In fact, the embodied energy of a computing device frequently outweighs operational energy [RM11].

Reducing embodied energy means reducing the total amount of impact on the aforementioned construction, manufacturing, and disposal of a device. While efforts to create more environmentally friendly methods of manufacturing would be beneficial to reducing embodied energy, an equally direct and impactful method is to simply reduce the amount of work that has to be done.

This method comes in the form of **product lifetime extension**. Put simply, using a piece of equipment for longer means that less new equipment has to be made, reducing the total amount of global embodied energy used for that class of equipment. This does not directly change the embodied energy that went into creating that equipment to begin with, but rather reduces the overall energy and environmental demand by reducing the number of times that the embodied energy cost is paid.

Recent surveys put the life cycles of smartphones at roughly 20 months [Pan], far below what would be required to offset the energy cost of manufacturing and assembly of these devices [RM11]. Reducing the embodied energy of smartphones has proven to be a difficult task,

given their rapid growth as a market [CAS21]. Extending the lifetimes of smartphones to a period much beyond the current trend would reduce their environmental impact greatly.

Extending the lifetime of a product, particularly computing technology, is a complex task. Because purchasing habits of consumers are not beholden to a single factor, understanding how to approach the issue of smartphones in a particular focused environment can be especially helpful in order to create a more targeted and effective solution.

In Chapter 4, I perform a case study on the short lifespans of smartphone devices in academic settings, the negative environmental impact this creates, and discuss how datacenter networks can extend the lifetimes of smartphones to reduce the embodied energy created by the rapid lifecycles I observe among mobile devices.

1.3.1 "Outdated" Smartphones

Smartphones become outdated for a variety of reasons. Physical hardware failures may occur, such as batteries no longer holding charge. However, many phones become outdated due to lack of software support, as new features and software requirements make older devices unable to run the programs required of them. This is a critical component of why many smartphones become "obsolete", as if they cannot fulfill the user's software requirements, the user is incentivized to purchase a new smartphone that can.

However, software support is something that can be extended if the proper tools and platforms are provided. The conclusion of Chapter 4 describes how there are a variety of methods to support and extend older phones. Unfortunately, it often becomes a function of the phone manufacturer to provide support in some fashion [Ama21, App21].

1.3.2 Datacenter Networks and Longer-Lived Smartphones for Students

Datacenter networks can aid smartphones by providing a platform to run the software that the smartphone cannot. There has been research on how to use datacenter networks to support

mobile devices [MAV17], but these solutions rarely offer a holistic method of providing every user's needs. Because it is difficult to target every user simultaneously, focusing on a specific group becomes necessary.

Smartphones are common in academic settings, where there is an increasingly large portion of students who use them for everyday academic tasks [Bla18, Ins21]. Providing students with a method to leverage their smartphones for longer periods of time is not only beneficial for the environment by reducing embodied energy, but also helps many students who may not have the budget for a new phone [MSWH16].

1.4 Dissertation Overview

In this dissertation, I study how datacenter networks can reduce their future operational energy costs at scale while also analyzing their benefit for reducing the embodied energy of smartphones via product lifetime extension. Each of these topics I discuss in turn in two parts, with my work around operational energy first and the work on embodied energy second.

To tackle the problem of datacenter operational energy, in Chapter 2 I present a an alternative, more energy efficient datacenter. The goal of this datacenter is to reduce its total operational energy usage by making use of optical circuit switches. This design comes with a number of limitations compared to more typical methods of datacenter networking. The bulk of my work focuses on how to provide endhost servers with the software traffic control systems necessary to efficiently interact with this new type of network. I find that despite different methods of implementing traffic control in software, there are fundamental limitations that prevent endhosts from being performant at high network speeds.

This motivates my work in Chapter 3, where I analyze the limits of what software may achieve when sending over an optical circuit switch. I perform an extensive investigation of how and why traffic control software cannot run over an optical circuit switch at the bitrates modern

datacenter networks require, and present the limited cases in which software may still be effective.

In Chapter 4 I move towards discussing the embodied energy of smartphones. To provide datacenters an effective target for extending device lifetimes, my work seeks to understand the lifecycles of phones in a university setting where they are increasingly required to complete coursework. By studying a market of users that have little purchasing power but have a great necessity to purchase a phone, I seek to provide an answer for the maximum *software* lifetime of a phone. Because software is required for students to access coursework, the inability to run said software forces new phone purchases. I find that there is a typical maximum of 4-5 years that smartphones are operable in academia. This motivates a solution via offloading portions of online academic applications onto datacenters in order to extend the lifetime of smartphones.

Chapter 2

Optical Datacenter Networks: More data, less power

The operational energy of datacenters consumes roughly 1% of the world's global energy every day. This has remained constant despite rapid growth in internet traffic over recent years, which doubled between 2017 and 2019. Despite this growth, the usage of energy within datacenters has remained at a constant 1% [Kam20].

Modern datacenter designs focus on energy efficiency as a critical requirement, ensuring that the power draw of datacenters remains low despite datacenter traffic roughly doubling every year [SOA+15]. In order to continue this trend, future designs of datacenters must continue to evolve and leverage new technologies that can provide higher networking data rates for less power.

In this chapter, I discuss my work on optical datacenter networks, which are an evolving field of active research examining how to leverage an alternative method of constructing switches, the critical component of datacenter networks. I first discuss a background on optical datacenter networks and present the design of the two optical circuit switches (OCSes) I use in my research.

I follow with a comparison between the energy usage of possible datacenter architectures that use these OCSes, and the energy usage of equivalently performant traditional datacenter networks. I then discuss the systems challenges that come with using OCSes in practice, and the results of a variety of programming methods to solve these challenges.

In particular, I focus on time division multiple access (TDMA) networking, a method of implementing transmission control on endhosts for circuit switched networks. Running an accurate TDMA schedule accurately is essential to using the aforementioned optical circuit switched networks effectively and maximize their operational energy savings. Because endhosts do not have a readily available mechanisms for implementing TDMA, my work focuses on three different methods of doing so: kernel-based, kernel-bypass, and remote direct memory access (RDMA). Overall, I find that while there are circumstances where each mechanism functions adequately, there are notable drawbacks that prevent them from categorically functioning in a general manner.

2.1 An Introduction to Optical Datacenters

Optically-switched networks consist of one or more OCSes as the primary backbone method of transport. These networks are traditionally referred to as "circuit-switched" due to the nature of how information is exchanged and routed between links in the network itself. This is in contrast to with packet-switched networks, which is the vastly more common method of computer networking.

Circuit-switched networks are an increasingly popular topic of research. While both wide-area and local communications have traditionally used or leveraged parts of circuit-switched networks, modern datacenter networks are generally solely based on silicon-based electronic packet switches. Large datacenter operators such as Google and Facebook follow this pattern [SOA+15, RZB+15a], and have generally focused on improving their packet-switched

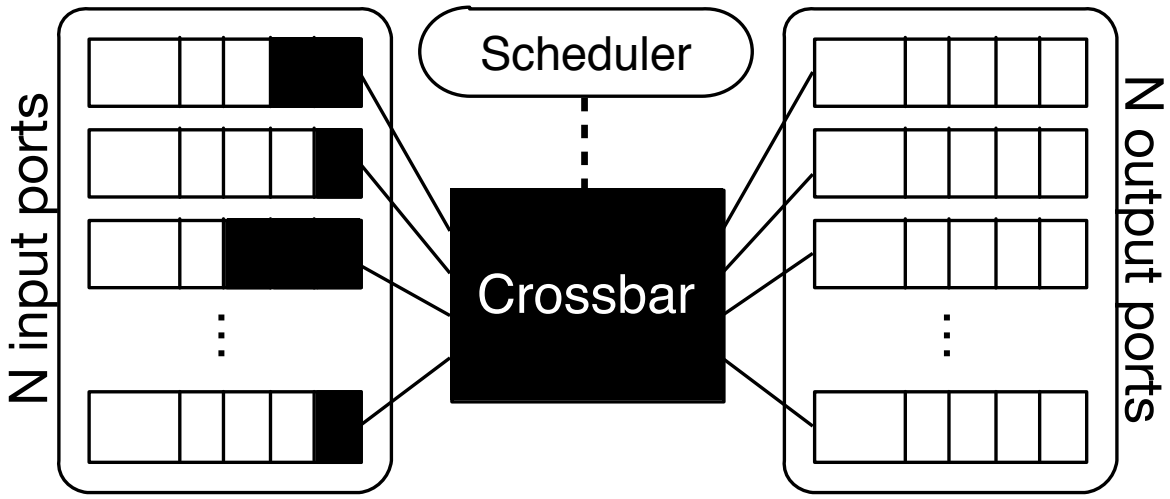


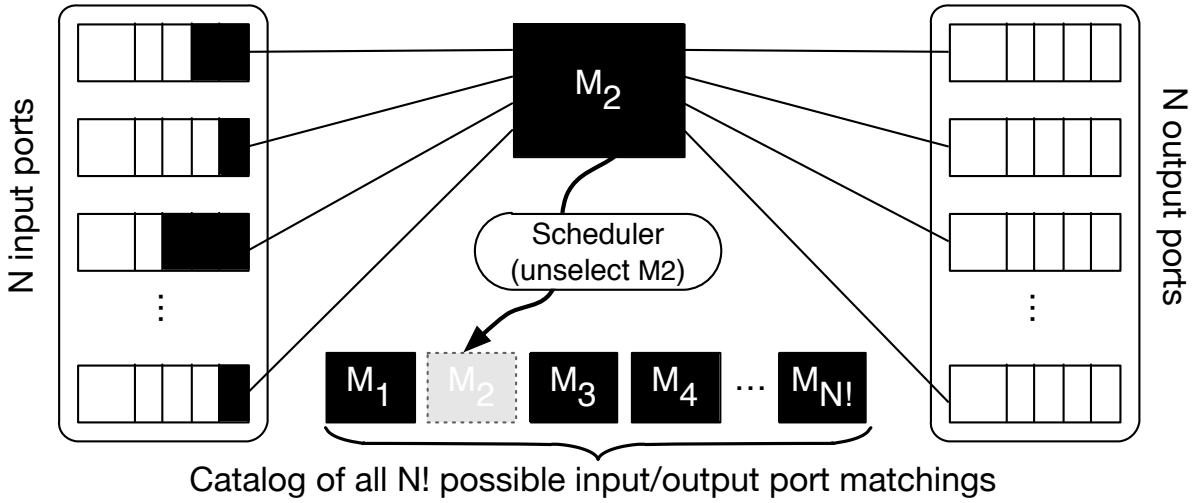
Figure 2.1: A standard crossbar-based packet switch uses an internal scheduling algorithm to dynamically reconfigure the crossbar to service demand.

architecture due to cost-efficiency [AWE19].

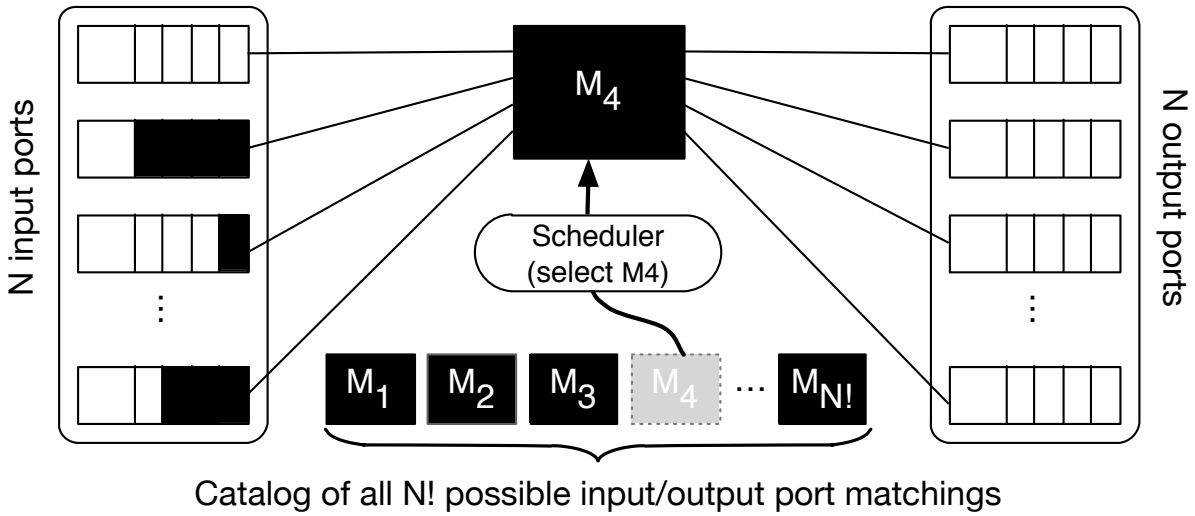
2.1.1 What is circuit switching?

The primary difference between a circuit-switched network and a packet-switched network is the nature of the switch itself. The latter is far more common and follows the traditional and common concept of a switch, where every packet entering the switch from any port can be routed out of any of the other ports at any point in time; the switch operates per-packet. The former instead operates per-circuit; the available connections between two ports change over time, and all possible pairwise connections (a full crossbar) may not be available on a per-packet basis. Further, circuit-switches often do not have a method of storing or buffering data that is transmitted to them, meaning they cannot "hold on" to information to be sent at a later time when the correct connection pattern is available. A common example of a circuit switch is a MEMS-based optical switch [FPR+10], which rotates an array of micromirrors to create a bipartite graph over the connected ports.

Datacenter networks frequently critically rely on the full-crossbar connectivity of silicon packet switches. Additionally, silicon packet switches can "buffer" packets, holding on to



(a) Deselecting the M_2 matching and returning it to the internal catalog.



(b) Selecting the M_4 matching from the internal catalog and installing it between the input and output ports.

Figure 2.2: A strawman circuit switch design. When the scheduler detects a change in demand, its scheduling algorithm changes the mapping of input ports to output ports by replacing the M_2 matching with M_4.

them to transmit at a later time if the switch is overloaded with traffic at the time the packet arrives. Datacenters depend on switches to make smart buffering and routing decisions to use the complex set of available paths efficiently [KHK+16, AFRR+10, AKE+12]. Datacenters may also rely on several other key features of these switches, such as multicast and quality of service enforcement [GSG+15].

None of these features are present in a circuit switch. Alternative methods must be used to realize the objectives that the above features achieve for datacenter traffic control. This is the primary problem in developing circuit switched networks, and research frequently attempts to solve this problem in a variety of ways [GMP+16, LLF+14]

An example of how a standard full-crossbar packet-switch connects to its input and output ports can be seen in Figure 2.1. Circuit switches typically employ a **matching** scheme. A matching over N ports consists of a single, fixed pattern of N input connections to output connections that is active over some continuous time. A packet switch with N input/output ports can be thought of as having all possible $N!$ matchings available at any given time. A simple strawman circuit switch with four available matching patterns can be seen in Figure 2.2.

Including every possible one of the $N!$ matchings in a circuit switch is possible, but results in a large **reconfiguration delay** [FPR+10]. A reconfiguration delay represents a period during which no data may be sent through the switch, as it is changing from one matching to another. Higher reconfiguration delays decrease net datacenter energy efficiency. Thus circuit switches incur a tradeoff: more connection patterns can enable a better connected network, but this comes at a cost of higher delays when changing that network.

Previous proposals for circuit-switched datacenter networks make several sacrifices in order to achieve their goal of a high-throughput, cost-effective, and long-term solution for high-bandwidth datacenters. Some of the common drawbacks include complex control planes [CSS+12], limited routing flexibility [WAK+10], and increased packet latency [MMR+17]. Several proposals suggest a hybrid approach, where a traditional packet-switched network sits

alongside a circuit-switched network, and network traffic traverses the network that best suits its needs [LLF+14, FPR+10, KPB09]. It is partly for this reason that I examine the need for a more efficient circuit switched network.

Circuit switching has significant benefits in the form of bandwidth efficiency for lower energy costs. Because circuit switches do not use silicon based hardware to route inputs to outputs, the amount of data they can process relative to their energy usage is much greater than in packet switches.

2.1.2 The Benefits and Limitations of Packet Switching

Despite research interest in circuit switching, packet switching has continued to dominate the market. Large datacenter operators have all relied upon scaling out packet-switched network fabrics to meet their ever-increasing bandwidth requirements [AFLV08, Fac14, GHJ+09a, SOA+15]. Google's 2015 data-center network design delivers 1.3 Pbps of cross-network bandwidth to hundreds of thousands of servers [SOA+15]. Since their deployment in the mid-2000s, packet-switched networks have leveraged the steadily increasing performance and decreasing cost of merchant switching silicon to keep pace with demand.

However, future high-speed electrical switches are projected to become prohibitively costly [LLF+14, FPR+10]. Merchant silicon chips used in packet switches face the same scaling limitations that currently hamper CPU manufacturers [Tay12]. Recent studies suggest [VSG+10] that switch chips have at most two more Moore's Law generations, culminating in the present decade in devices that can support link rates approaching 100 Gbps.

To continue supporting higher link speeds, future packet switch designs will require ganging multiple ports together. Switches like the Broadcom Tomahawk already implement 100 Gbps Ethernet by ganging together four 25 Gbps ports [Bro]: a 128-port 25 Gbps per port switch becomes a 32-port 100 Gbps switch. Such an approach dramatically impacts the feasibility of large network fabrics whose energy cost scales according to the square of the switch

radix [FWJ+13].

The future of packet switches thus will inevitably become inefficient in both energy usage and overall cost, which strongly motivates the need to find another solution. Newer architectures have made clever decisions to allow for these limitations in the short term [AWE19], but ultimately circuit switched datacenter networks are an extremely attractive solution to solve these challenges for reasons I will show in this chapter.

2.1.3 The Energy Savings of Circuit Switching

To show that circuit switched networks use energy more efficiently than packet switched networks, I compare the raw operational energy used by the networking hardware for both types of networking. In order to make this comparison, I first must establish the target packet switched datacenter network for comparison. A Fat Tree network [AFLV08] is a reasonable target, and is commonly used at large datacenter operators such as Google [SOA+15]. I compare against both a 1:1 oversubscription ratio (the optimal, but expensive) and a 3:1 oversubscription ratio (less efficient, but is used in practice [SOA+15]).

Second, I need circuit switched networks to compare against. Here, I use the types of optical networks I discuss in this chapter. These three networks represent the evolution of my work on optical datacenter networking, and how I and others in my research group approach different challenges that occur when creating networks that used OCSes. Each of these will be discussed in more detail in the following sections of this chapter.

For each network, I vary the number of nodes (servers) as the only independent variable. I assume that every node is connected with a 100 Gbps link. This allows me to compute the required number of top-of-rack (ToR) switches, and then the number of intermediate and core switches required to connect the ToR switches together, and thus the total power draw of the network. I do not include the energy cost of the nodes/servers in my calculation. The reason for this is that all network types will use the same amount of power on each node, so it is moot to

Table 2.1: Energy usage of two OCSes and two packet switches.

Switch Model	Watts Used
Optical Selector Switch v1	2.7 W
Optical Selector Switch v2	27.6 W
Barefoot Wedge 100BF-65X	276.9 W
Mellanox SN2700	90.5 W

include this in my result.

To aid in this, I conduct an independent measurement of the power draw of each switch used in this comparison. I use a Chroma 66204 Multi-Channel Digital Power Meter and directly connected the power supplies of each switch to examine how many Watts it uses in operation. No difference in power draw is observed between an idle switch and a switch under traffic load.

The two 64-port optical switches are used in the three optical networks mentioned, and will be described more in future sections. The power draw of the first version of the optical switch used in SelectorNet (Section 2.4) and RotorNet (Section 2.5) is roughly 2.7 Watts. The power draw of the second version of the optical switch used in Opera (Section 2.6) is approximately 27.6 Watts. An optimal version of the second optical switch would use approximately 10 Watts, and I include a projection of this case as well.

For the packet switched Fat Tree networks, I use measurements from two different 100 Gbps packet switches: a 64-port Barefoot Wedge 100BF-65X packet switch, and a 32-port Mellanox SN2700 packet switch. These are fairly modern packet switches, with the former containing features becoming increasingly used in datacenter networks [BDG+14]. I find that the 64-port Barefoot switch draws about 276.9 Watts, and the 32-port Mellanox switch draws about 90.5 Watts. These values are outlined in Table 2.1.

The projected energy usage of the aforementioned datacenter architectures is shown in in Figure 2.3. I only plot a 3:1 FatTree up to 24576 nodes, as scaling the network past this results in hundreds of thousands of nodes, which is not a viable comparison for the other networks given the switches I measure against.

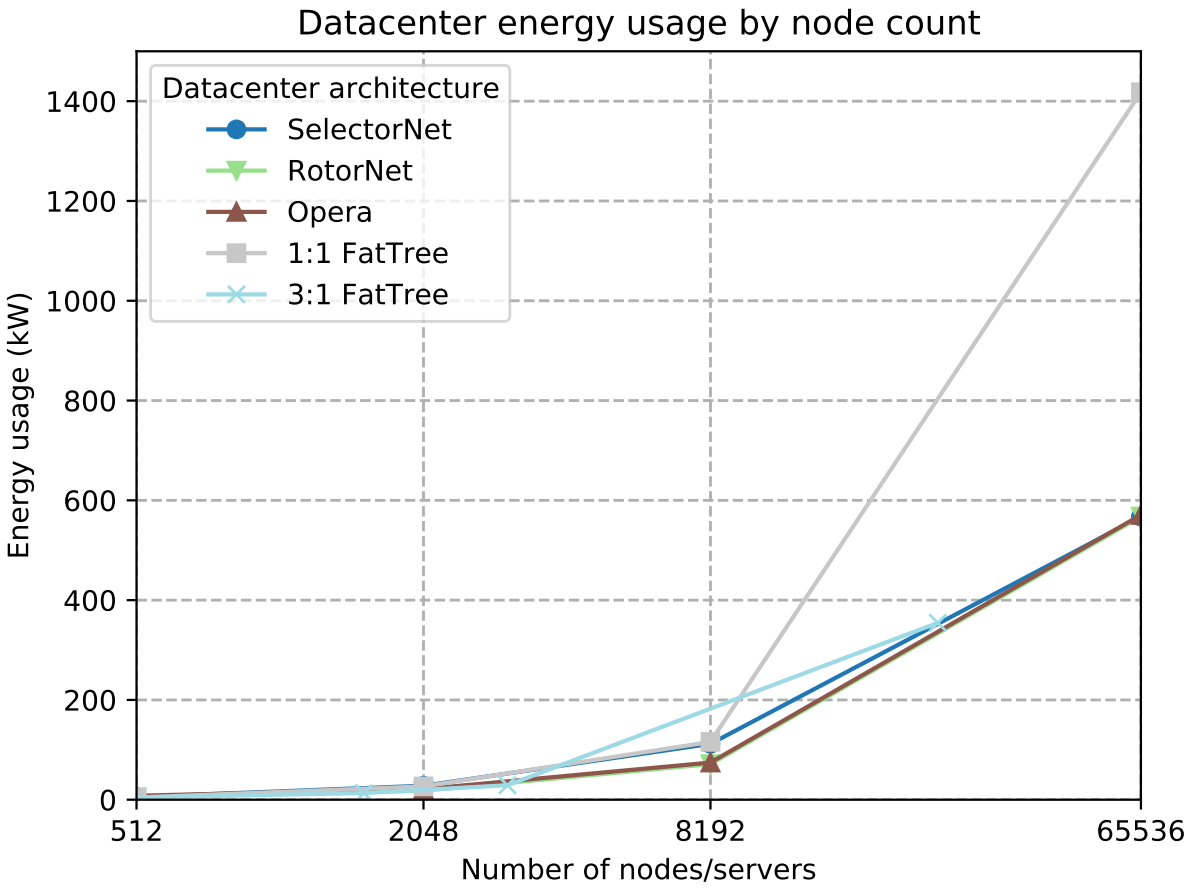


Figure 2.3: Datacenter energy usage based on observed power usage of electrical and optical switches. Energy for servers/nodes is not included.

The immediate observation is that FatTrees are roughly as efficient as optical networks when using thousands of nodes. This is due to the significant operational energy of the ToR switches, which dominate the network's power utilization along most of the graph. However, once tens of thousands of nodes are required, FatTrees draw a vastly greater amount of power due to the higher number of intermediate and core layer switches used to support them. Optical networks only scale up with the number of ToRs, with the very minimal energy impact of the optical switches not causing any additional significant power requirements.

While using a 3:1 FatTree may look like a good solution, this limits the bandwidth of the network to only one-third of the bandwidth found in a 1:1 FatTree, delaying computation on many

jobs and reducing overall efficiency (and thus increasing energy usage, as described below in Section 2.1.3). Indeed, Google's 2015 datacenter architecture is a Clos datacenter design, similar to a FatTree, that uses a 1:1 oversubscription ratio [SOA+15]. Additionally, said network uses *more* energy than a standard 1:1 FatTree, with the 65536 node version presented in their work using about 1559.67 kW if it used the 64-port Barefoot switch mentioned above.

Operational energy and workload efficiency

In the previous section, I only compare the operational energy of the networking hardware at a single point in time. Despite this, network efficiency is still an important metric. If a circuit switched network is not as efficient at delivering data as its packet switched counterpart, the energy savings may go to waste as more time is needed to complete software jobs, consuming more energy.

This is the core reason that the optical networks above have evolved over time. As I describe each network in this chapter, I note that the design decisions made are done so to increase the operational efficiency of the optical datacenter network, rather than the raw energy consumption; the latest iteration, Opera, uses slightly more energy to gain significant benefits in workload efficiency.

2.2 The Challenges of Circuit Switched Networking

Circuit switched networking comes with significant challenges due to its fundamental characteristics. The core reality of not having all paths available between two endpoints and not being able to buffer packets creates design challenges not traditionally seen in standard, packet switched datacenter networks. Endhosts typically rely on buffering and path availability in switches so that they may send packets at any time. The critical question is **how to balance the drawbacks of circuit-switching while leveraging its benefits to achieve maximum network**

performance.

There are two primary challenges in solving this question that drive the work I will present in this chapter: **when** to send the data, and **where** to send the data. These two questions have immediate, obvious answers, but implementing them in practice creates issues that I will discuss at length.

2.2.1 When to send the data

Data cannot be sent at all times inside of a circuit switched network; inputs cannot go to every output, and circuit switches cannot store and buffer traffic like packet switches can. Circuit switches typically have an **uptime** and a **downtime**, with the former representing when data may be sent through the switch to a destination, and the latter being the period when a switch cannot accept data while it changes the connection pattern.

Sending traffic to a circuit switch during its downtime results in the data being lost. Senders must know the difference between downtimes and uptimes and send data at precise intervals through the switch to use the network properly. Enforcing this is a significant challenge for traditional datacenter switches and servers. Electrical, packet switched networking assumes an always available framework, so hardware is not designed with precise transmission control in mind.

This means that when traffic is sent by an endhost, the actual packet often leaves the endhost far later than was originally intended. There are two components to this delay. The first is a fixed, minimum value that cannot be avoided as data moves through the system and onto the wire when finally sent. This value can be measured and compensated for; simply sending all traffic early to compensate for this will solve the issue. I refer to this as a **preallocation delay**.

The second part of the delay is far more impactful and difficult to solve. Software and hardware has not been designed with real time, fast optical transmission in mind, which creates a random amount of additional delay that occurs when sending data over a network. This means

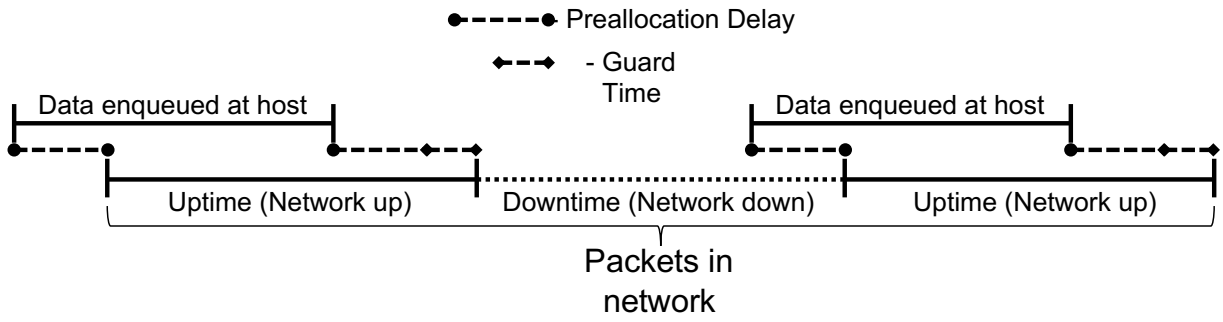


Figure 2.4: How transmission delays affect the time between sending from a host and when they appear on the wire in a TDMA setting. Transmission delay and guard time must be accounted for to avoid downtime transmissions.

that when data is sent close to the beginning of a switch's downtime, it can randomly instead be sent *during* the downtime instead, resulting in data loss and inefficiency. To protect against this, hosts have to stop sending traffic early so that there is zero probability data is sent during a downtime. I refer to this as the **guard time** of an optical network.

Figure 2.4 shows how an optical switch moves between periods of uptime and downtime, and how preallocation delay and guard time are accounted for in the endhost when transmitting data over the network. In my work, I always observe guard time to be smaller than the reconfiguration delay.

Guard times are the most impactful aspect of an optical network as they "remove" time during which data can be sent. Reducing the guard time required at an endhost means the endhost can send more data during the uptime of the switch, increasing network efficiency. Note that the guard time and preallocation delay values are independent of any aspect of the optical switch itself; they are purely a function of the hardware components in the endhost.

2.2.2 Where to send the data

When active, circuit switches only have one fixed connection pattern available. Unfortunately, even when endhost traffic patterns have some predictability, there is still plenty of data that

needs to be sent across the network at unpredictable times [RZB+15a]. Any datacenter network must ensure there are opportunities available for endhosts to send arbitrary traffic to its final destination.

One option to solve this in a circuit-switched network with limited connectivity is **cut-through indirection**. This has an endhost send data to a connected intermediate endhost, and the intermediate then forwards it to its final destination. Indirection allows for achieving full connectivity when direct paths are not available, making it a strong solution. However, it is not foolproof- redirecting traffic through an intermediate host uses extra bandwidth, called a **bandwidth tax**. By using multiple connections for a single packet, it is "taxing" the network by transmitting the packet multiple times.

Thus, it is even more effective to ensure that a direct path between a source and destination in the network is available within a reasonable period of time. To do this, the switch can be reconfigured more often: if the switch moves between states more frequently, the maximum amount of time an endhost will have to wait until a direct path is available will be lower.

However, the uptime of a circuit switch cannot be arbitrarily low; otherwise the switch spends most of its time reconfiguring. The ratio of the time the switch is up versus down is called the **duty cycle** of the optical network. In Figure 2.4, I have shown a duty cycle of 50%: the network is up half of the time, and down the other half of the time. The duty cycle of the network is directly tied to bandwidth- a 50% duty cycle means that the network can only send at 50% of its link rate over time, as half of the time the network cannot send traffic at all. Increasing the duty cycle means that there is more time that the network can send traffic, but direct paths will change less frequently.

2.2.3 When and Where: The Benefit of Low Cycle Times

Combining the two problems of when and where create the challenge my work investigates: **How can the network have a low duty cycle while also achieving maximum network**

effectiveness? The biggest impact I quickly observed was that when reducing the duty cycle of the network to increase path availability (and reducing the bandwidth tax), the guard time would significantly reduce the efficiency of the network further.

The ratio of the guard time over the configured uptime of the switch represents the overall maximum amount of time the network can operate. I will refer to this as **uptime efficiency** in the future; the higher the uptime efficiency, the more effectively the network is being used. For example, if the guard time of an endhost is $100\mu s$ and the uptime is configured to be $1ms$, then there is a $(1000\mu s - 100\mu s)/1000\mu s = 90\%$ uptime efficiency in the network. Getting this value as close to 100% as possible will ensure that a circuit switched network can be used to its maximum effectiveness.

The core issue that I will describe in this chapter is how to reduce the guard time at an endhost. This is an incredibly complex issue that is a function of how datacenter software and hardware has been designed for decades. There are a variety of technologies that can enable endhost software to interact more closely with network hardware, and these technologies are the focus of my work in this chapter.

2.3 Selector Switches: An Optical, Circuit-Switched Network

As discussed above, fundamental technological limits on merchant silicon will constrain future designs of silicon-based, packet-switched datacenters. Servers are already utilizing link rates of up to 400 Gbps [AWE19], and I show in Section 2.1.3 that scaling packet-switched datacenters with even 100 Gbps switch chips is energy inefficient. This scaling barrier will derail the decade-long cost-effectiveness trend of existing packet-switched topologies, forcing operators to consider alternative designs. While hybrid optical/electrical topologies have been proposed to bypass the limitations of packet based switching, existing approaches cannot scale to large host

counts and next-generation link rates due to the complexity of circuit scheduling and the need for costly optical amplification.

In this section I present a new form of optical switching in the form of **selector switches**. Selector switches use a limited number of matchings in order to achieve more rapid switching speeds and reduce downtime, thus increasing uptime efficiency in the network. They also scale with a large number of endhosts without additional energy costs, unlike packet switches. However, they come with drawbacks, such as not being able to store or buffer packet information, a critical feature in packet switched networks.

These selector switches are leveraged to create new alternative circuit switched topologies throughout this chapter. Each topology is an iteration on the last, though each uses a selector switch with the same properties. The first two topologies, SelectorNet and RotorNet, employ a real physical prototype switch created by Max Mellette. Opera, the final topology, uses a virtual selector switch created by Rajdeep Das as multiple real prototypes were not available for use. I begin by describing SelectorNet here, and describe changes in the topology construction in future sections as necessary.

2.3.1 SelectorNet's Design

SelectorNet is designed from the ground up to limit scheduling complexity and simplify the optical components. SelectorNet fully connects datacenters with a new type of optical device that abandons the crossbar abstraction: instead, it relies on indirection to deliver packets between hosts that are not directly connected by novel selector switches. The result is a network fabric that is not only cost-competitive with packet-switched designs, but continues to scale as link rates surpass 400 Gbps.

The SelectorNet architecture scales to hundreds of thousands of nodes. The optical switching devices can be scheduled independently and implemented entirely passively, making them both link-rate agnostic and compatible with commodity optical interconnects. The key

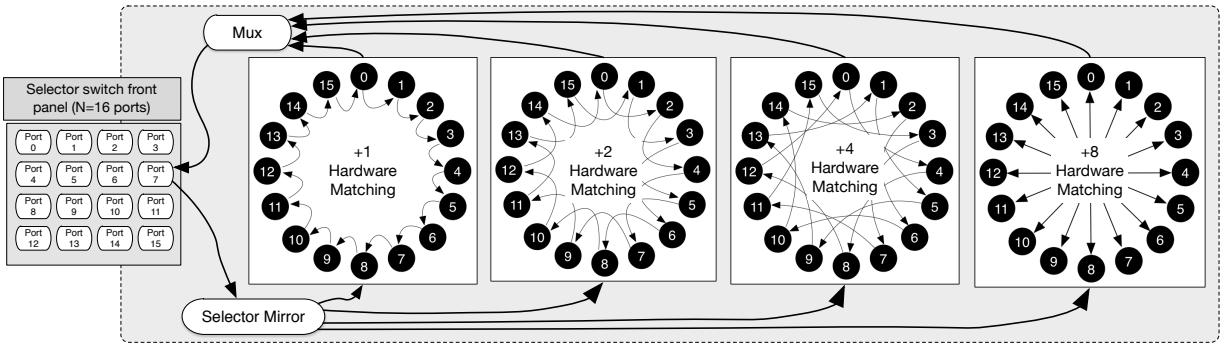


Figure 2.5: A selector switch forwards traffic from each of its input ports to one of k internal matchings, mapping traffic to destination ports at different powers-of-two distances away.

insight in this first iteration is that production datacenter networks are commonly over-subscribed: they do not deliver full bisection bandwidth. Unlike previous architectures, SelectorNet leverages this fact to construct a different type of network that may require packets to traverse the fabric multiple times via cut-through indirection, yet still fully connects all the servers in the datacenter.

2.3.2 Selector Switches and the Bandwidth Tax

The design of the selector switch is inspired by the Chord [SMK+01] overlay network. It employs the same idea by having $\log_2 N$ matchings in every selector switch. In Chord, paths are constructed by indirecting traffic over multi-hop routes in the Chord ring. Every node has a one-hop path to its successor node, which is its neighbor in the ring. To avoid long $O(N)$ paths, each node further maintains a logarithmic number of paths to nodes spaced at factors of two away from it in the ring. In this way, data can be forwarded to any node in at most $\log_2 N$ hops.

The physical design of the selector switch used in this work is based on a microelectromechanical systems (MEMS) tilt mirror switching to select between four matchings. The selector mirror is controlled by a Xilinx Spartan 6 FPGA on a Digilent Atlys board. The FPGA drives the mirror with precomputed drive waveforms through a DAC and high-voltage amplifier board. The only power draw of the design is from the motor used to move the MEMS mirror and the FPGA, which is why this first version of an optical switch uses so little power as presented in

Section 2.1.3. The selector switch I use was designed and constructed by Mellette and Ford, et. al [MSP+17].

Figure 2.5 shows the design of the 16-port selector switch I use in this work, containing $\log_2 16 = 4$ internal matchings. Output port i of each of the matchings is connected to a multiplexer which is in turn connected to the i-th output port on the front panel of the selector switch. The four matchings each realize a mapping of input ports to output ports spaced at logarithmic offsets, similar to the Chord ring. I denote these matchings as the +1, +2, +4, and +8 matchings. As the prototype device is optical, the i-th input port on the front panel connects to the MEMS mirror, which forwards the signal to only one of the internal matchings.

Restoring full connectivity

Unless a given input/output port mapping has an exact match within one of the $\log_2 N$ matchings, that mapping requires multiple passes through the selector switch. At each pass, data is buffered in the (external) device connected to the output port, which then reinjects it into the switch after a new matching is installed so it can continue to its next hop. Consider two example port mappings:

$0 \rightarrow 4$: The host wishes to send data from input port 0 to output port 4. Since the output port is an even power-of-two distance away ($2^2 = 4$), the selector switch is configured to connect input port 0 to the +4 matching, resulting in the one-hop path $\{0 \rightarrow 4\}$.

$0 \rightarrow 6$: None of the four matchings implement a one-hop path from input port 0 to output port 6, and so the host must rely on indirection. There are several equivalent options that can all affect the same end-to-end mapping. One is to first configure the selector switch to connect input port 0 to the +4 matching, mapping it to output port 4. Subsequently, the selector switch would connect port 4 to the +2 matching, mapping data to output port 6. The result is the two-hop path $\{0 \rightarrow 4 \rightarrow 6\}$.

The bandwidth tax

Depending on a packet's destination, it may need to transit the selector switch multiple times. The number of hops through the selector switch depends on the particular source-destination pair, and is no greater than $\log_2 N$. A multi-hop flow that transits the selector switch l times requires $l\times$ more bandwidth than a one-hop flow. Thus, the bandwidth tax of the packet on the network is l. The overall average path length across all flows as the selector switch's total bandwidth tax, which depends on the particular workload being served. With a workload evenly distributed among all node pairs, the average path length (and thus bandwidth tax) is $\frac{1}{2}\log_2 N$.

A SelectorNet datacenter network

Figure 2.6 shows an example of how a SelectorNet datacenter architecture could be constructed. The design shown supports a total of 32,768 servers, and can scale to work with 100 Gbps or 400 Gbps links depending on the ToR switches used (in Section 2.3, the 64 port ToRs are able to split the 100 Gbps links eight ways).

The network consists of "super racks" of 128 servers each, which allows each rack to have eight total ToR switches that, in turn, each connect to eight separate selector switches. This increases path availability as there are now a large number of possible selector switches for each server to choose from, which better supports both large and small flows of traffic simultaneously to increase network efficiency. Each ToR uses half of its ports to connect to servers, and the other half to connect to optical switches.

The total number of racks supported is limited by the maximum port count of a selector switch. While selector switches can scale to extremely large port counts [MSP+17], SelectorNet uses 256 port switches in order to allow for a reasonable number of matchings in each switch while achieving full network connectivity and supporting a high level of fault tolerance.

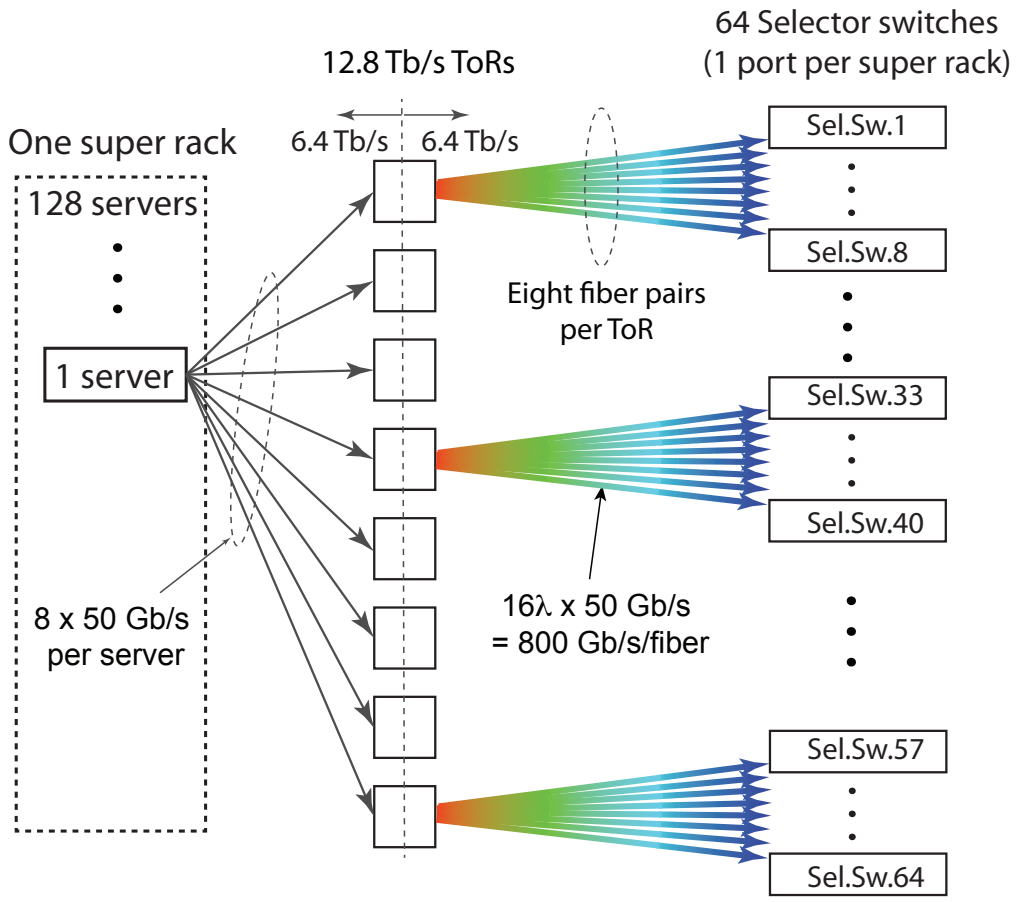


Figure 2.6: A SelectorNet datacenter architecture, using 128-server super racks. Each server's Ethernet links are divided into eight 50 Gbps lanes, divided between 8 ToRs within the super rack. Each ToR has eight fibers that connect to 8 distinct selector switches. These fibers are repeated for each of the 8 ToRs for a total of 64 fiber pairs leaving each super rack. Each selector switch has 256 ports (one per super rack).

2.4 Kernel-Based Programming: SelectorNet

Now that I have covered the construction of a SelectorNet topology, I will describe my work in how to promote high efficiency and maximize the benefits of employing the SelectorNet architecture. My work focuses on the endhosts in the network, and how they must interact with a circuit switched topology like SelectorNet

Endhosts in packet switched architectures are able to "fire and forget" packet data into the

network, as packet switches are able to buffer and store packets if they cannot be immediately sent to a destination. However, since selector switches cannot store packet information, endhosts now have to be discretely aware of when and where to send data through the network. Programming endhosts in a scalable, accurate fashion becomes paramount to use the SelectorNet architecture. This is done by providing scheduling information to endhosts, which I describe in this section.

After creating a schedule, the endhosts still need to enforce it and reduce the guard time incurred in the network. The second step I present in this section is the implementation of a microsecond scale precision transmission protocol via a module in the server's operating system kernel, which gives me more direct access to the networking hardware without incurring additional variable delays.

Finally, I create a testbed setup using a real selector switch and see how effective a kernel-based networking approach is for providing accurate and fast transmissions speeds over a circuit switched network. I find that there is a modicum of guard delay required at a moderately low networking speed of 10 Gbps.

2.4.1 Circuit switching through TDMA

To support SelectorNet, endhosts need to answer the questions outlined in Section 2.2: **when** will the switch be active, and **where** will data sent through the switch go? To store this information, it is useful to create an abstraction that can accurately convey this information to the endhost's networking stack so it can use the network appropriately.

I have found that using a style of network programming called **"time division multiple access" (TDMA)** networking [VPVS12] is suitable for this. TDMA networking is traditionally used in other types of physical computer networks, particularly for wireless and radio connections, to divide up when and where the network may be used, and also by whom; however, it has been used in packet-switched datacenter networks as well [POB+14]. In this case, the extra question of "whom" provides an additional benefit to abstract how I view indirected traffic. TDMA slots

can say which traffic source will be using a specific network link in the case of multi-hop paths.

TDMA networks encode this information by using a series of **time slots** that contain the start and stop times for the slot and the source and destination for the traffic. The method of how time slots are communicated to an endhost may vary. For SelectorNet, I develop using fixed, predefined cycles of time slots and a control host that gathers traffic information to dynamically send time slot information.

2.4.2 Implementing TDMA in the Endhost Kernel

Now that TDMA networking can be implemented by encoding timeslot information at the endhost via a predefined configuration, it is now necessary to enforce the correct transmissions to occur during each timeslot. The most critical component is ensuring that packets are transmitted with maximum accuracy relative to the timeslot and that *only* the designated packets are sent. Because endhosts must implement cut-through indirection, they cannot simply send any data they want during the slot.

For SelectorNet, I first implement TDMA scheduling at the hosts through a custom Linux queuing discipline (qdisc) that runs in the operating system kernel. The TDMA qdisc enqueues all traffic to a network interface into queues according to the destination of the packet. It then sends packets in a given queue to the NIC according to a provided schedule of timeslots. Each timeslot consists of the starting and ending times when the switch will employ a given matching. The qdisc emits only as many packets as can be sent during a timeslot without extending into the switch downtime period (where they would be lost). During a single timeslot, the qdisc may emit packets destined for multiple destinations, expecting that the connected destination will in turn indirect any packets that need to traverse multi-hop paths. To simplify scheduling, the qdisc always gives priority to traffic that did not originate at that host over locally generated traffic to the same destination.

The qdisc sends traffic using the lowest level kernel interface possible, directly calling a

function just above the network interface card (NIC) driver. I additionally modify the NIC driver to not reset the networking stack when it detects that the link is no longer available (a frequent occurrence, as the selector switch moves between up and down states constantly). Packets are enqueued individually in order to ensure that any delays can be detected and accounted for.

2.4.3 Kernel-level Guard Time and Preallocation Delay

The first step to configuring the kernel-level qdisc is to input the correct amount of guard and preallocation delay, so that the basic transmission schedule becomes accurate relative to the physical network link. Determining the appropriate guard time and preallocation delay for an endhost can be done in a variety of ways.

I present two separate methods used to determine the appropriate values for these parameters in SelectorNet. Both methods ultimately allow for characterizing how endhosts interact with the optical network, but one is more accurate than the other and provides additional insight into the random nature of endhost delays.

These methods are tested against the kernel-level qdisc described above. This method of transmission is the only method used in the SelectorNet project. Other methods of transmission will be presented in future subsections in this chapter for different circuit switched networks.

Guess and Check: Binary Search: It is possible to simply determine the correct value of preallocation delay and guard time by using a binary search. The preallocation delay can be initially determined by inspecting whether or not an endhost receives packets that were scheduled at the beginning of a timeslot; if not, then the preallocation delay is too small. If the preallocation delay is sufficient, then all packets at the beginning of the slot will arrive. Using this method, I am able to determine the correct preallocation delay within $\pm 25\mu s$.

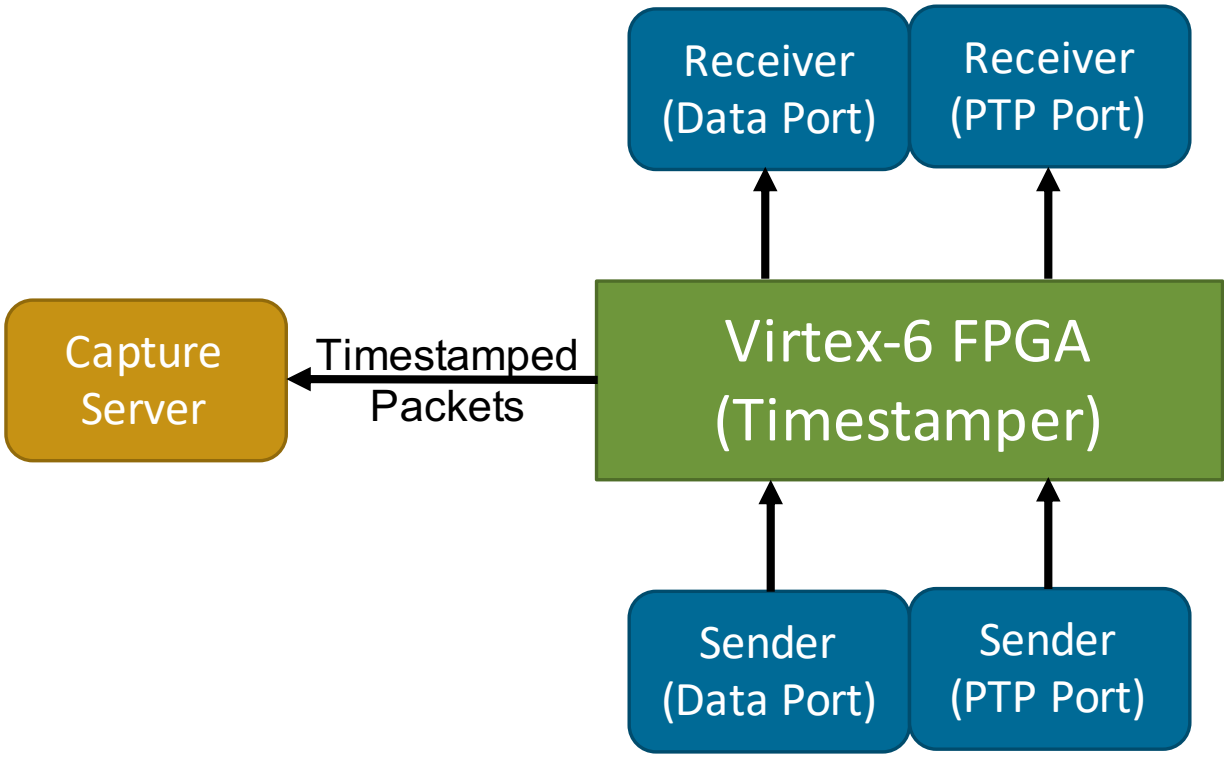


Figure 2.7: A diagram showing how an FPGA timestamping method is used to determine the time packets are actually transmitted on the wire from a server. The FPGA has a separate internal clock from the servers, which is why PTP packets are required to compute the delta between the FPGA clock and the servers' clocks.

Timestamped Packet Capture: The more accurate method of measuring packets is to use an intermediate capture board on the wire with timestamping capabilities. For this, I use a Xilinx Virtex-6 FPGA board that timestamps a copy of every packet sent through it by encapsulating them with a generic routing encapsulation (GRE) header. These copies are then sent to a separate server that records them for later analysis. This FPGA code was developed by Alex Forencich. Figure 2.7 shows a simplified overview of how endhosts connect to the FPGA in this system.

However, the FPGA clock is not in sync with the endhost clocks. To solve this, I send PTP packets through the FPGA and receive copies of those as well. All packets are copied in the order the FPGA sees them, regardless of the original input port. From this, I can determine the time that the endhost sent a packet relative to its own internal clock. Thus, I am able to determine the time between a packet being enqueued at the host and when it was actually sent, which allows me to easily compute preallocation delay and guard time.

To determine the time a single data packet p began transmitting on the wire, I also need PTP packets s_a and s_b. Let s_a to be the last PTP packet seen *before* p in the packet capture, and s_b to be the first packet seen *after* p in the packet capture. All times and timestamps are in nanoseconds. I define the following variables:

- p_b: The data size of p, in bytes.
- p_s: The time the packet p began transmitting, based on the sender's clock.
- p_f: The timestamp that the FPGA puts on the packet.
- s_{bh}: The *host's* timestamp of the PTP packet last seen *before* p.
- s_{ah}: The *host's* timestamp of the PTP packet first seen *after* p.
- s_{bf}: The *FPGA's* timestamp of the PTP packet last seen before p.
- s_{af}: The *FPGA's* timestamp of the PTP packet first seen after p.

From this, p_s can be determined using the following formula:

$$p_s = s_{bh} + (s_{ah} - s_{bh}) * \frac{p_f - s_{bf}}{s_{af} - s_{bf}} - k * (p_b + 24)$$

The constant of 24 is to account for additional physical Ethernet headers added on by the NIC when transmitting the packet over the wire. The constant k is the nanoseconds per byte rate of the link. For example, for a 10 Gbps link k is computed as:

$$k = \frac{10^9 ns}{s} * \frac{1 Gb}{2^{30} b} * \frac{1 s}{10 Gb} * \frac{8 b}{1 B} \qquad = 0.7451 \; ns/b$$

Where B represents bytes, b represents bits, ns represents nanoseconds, and s represents seconds. From these methods, I determined the appropriate guard time for the SelectorNet testbed is $400ns$, and the preallocation delay is $5.6\mu s$. These values are rather low relative to the downtime reconfiguration delay of $150\mu s$ the prototype switch uses.

2.4.4 Live Test of Kernel-Based TDMA

Next, I wish to ensure the accuracy that endhosts transmit at the correct time and with the correct route inside of an actual selector switch network when using my implementation. With a real selector switch provided by Max Mellette, I am able to test against a live, minified version of the SelectorNet architecture on real Linux endhosts.

I first describe the testbed that I use for conducing these experiments. Then, using the kernel-based TDMA-enforcement qdisc I develop, I run multiple different bulk traffic workloads through a real selector switch, and record how efficient the network operates relative to a projected maximum.

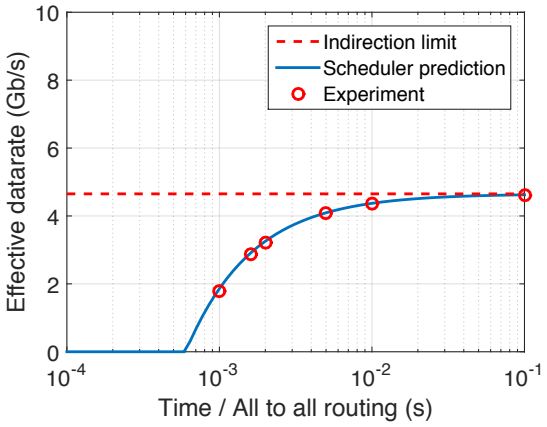
A SelectorNet testbed

The minified SelectorNet testbed emulates only the portion of a SelectorNet that a single Selector switch connects: a 16-port selector switch can support 16 super racks. Because I am evaluating a single selector switch, I need only consider one ToR per super rack and one fiber connection out of that ToR. Because the selector switch itself is link-speed agnostic, for purposes of evaluation I only use one lane per fiber pair (as opposed to the 16 lanes depicted in Figure 2.6), and drive it at 10 Gbps (rather than 50).

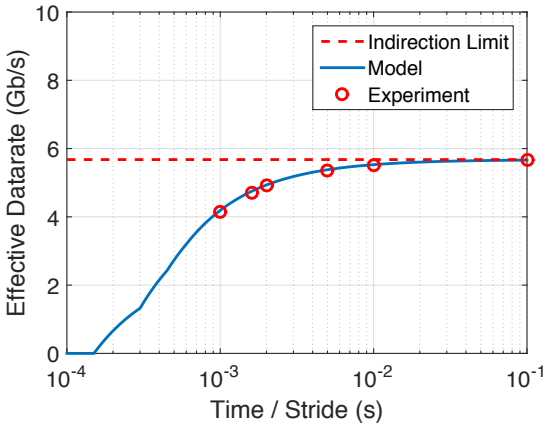
While the theoretical reconfiguration delay of a full version of the SelectorNet switch can be under 50μs, the delay used in the SelectorNet testbed is 150μs. Therefore, in order to achieve at least a 50% duty cycle, the network must have uptimes of at least 150μs as well. This creates a baseline for the uptime values described below in Section 2.4.3.

I emulate 16 super racks using only 8 machines. Each pair of super racks is emulated by an HP ProLiant DL360p Gen8 server with two hex-core Intel Xeon E5-2630 CPUs running at 2.30 GHz and 8×2 GB of 1600-MHz DDR3 RAM. Each server runs Ubuntu 14.04 and is equipped with a dual-port Myricom Myri-10G Dual-Protocol NIC. I modify the `iproute2` tables in Linux to prevent packets from being delivered locally to isolate the two super rack interfaces from each other.

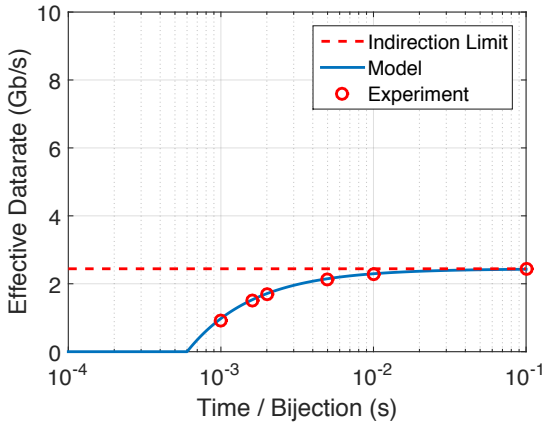
I use two user-level processes to send and receive UDP traffic at each (virtual) super rack (meaning each host runs four processes during an experiment). The sender processes open 15 different sockets (one for each other super rack) and the receivers listen on their respective interface and records traffic received from each of the senders. Because each host has two physical CPUs, I am able to pin each virtual super rack to a distinct CPU. A virtual super rack's kernel threads, UDP sender, UDP receiver, and NIC interrupts are further pinned to separate cores on the physical CPU.



(a) An all-to-all workload.



(b) A "rolling stride" workload.



(c) A random bijection workload.

Figure 2.8: A comparison of the scheduler-predicted vs. experimentally observed throughput for three workloads.

SelectorNet macrobenchmarks

I next consider three canonical traffic workloads: an all-to-all demand, a rolling stride pattern, and a time-varying random bijection workload. The goal in running these experiments is to see how closely the prototype's throughput compares to the expected throughput computed by a scheduling algorithm created by Max Mellette, accounting for the indirection penalty and selector switch reconfiguration overhead.

In each experiment I start a UDP sender application to fill the TDMA qdiscs with packets destined for each destination. I then send each virtual super rack schedules from a control server

that match the traffic pattern that I wish to test. A UDP receiver process on each virtual super rack records and logs the number of bytes it receives from each source, and these records are aggregated together at the end of the test.

Figure 2.8 shows the results. The selector switch has a downtime reconfiguration period of 150μs, and I vary the uptime allowed to serve each demand matrix. This determines the length of the timeslots for a single schedule. The y-axis shows the scheduler's predicted average throughput for that uptime. Circles signify experimental observations.

All-to-all: In this experiment I generate a uniform demand matrix representing every super rack sending to every other super rack, shown in Figure 2.8a. I run the experiment for a duration needed to serve 1000 demand matrices, and since they are uniform, I randomize the routing for each one. As expected, the observed throughput closely matches the expected indirection penalty. The overall throughput is lower for smaller uptimes, since the network must always pay a constant reconfiguration cost to switch between different matchings to serve traffic.

Stride: In a stride-x workload, each super rack sends data to a single destination shifted x mod N positions away. In a rolling stride workload, the value of x increases for all super racks in lock-step every T units of time. The results are shown in Figure 2.8b. I run rolling stride schedules for six different uptimes and found they corresponded to the projected effective data rate. Since four of the stride-x patterns match perfectly with the four matchings (specifically stride-1, stride-2, stride-4, and stride-8), the average indirection penalty is lower than $\frac{1}{2}\log_2 N$, and so the expected (and observed) throughput is higher.

Random bijection: The random bijection workload measures how well SelectorNet can serve a large number of randomly generated demand matrices, with the only restriction being that sources in the resulting demand matrices only send to one destination, and destinations only receive data from one source. The scheduler generates 1000 random demand matrices, with a switch uptime

ranging from 1ms up to 100ms, as indicated in Figure 2.8c. Compared to the Stride pattern, the overall throughput is lower since the amount of indirection, and thus the bandwidth tax, is greater on average.

In all cases, the results of the experiment show that a kernel-based TDMA approach is able to meet the projected maximum throughput that can be achieved for that uptime. The actual values on the graph are not 100% of what the projection predicts. There is some amount of loss that occurs invariably on every experiment due to how the Linux kernel performs thread maintenance. The Linux kernel occasionally stops threads from running in order to verify that the CPU core is not deadlocked, which causes packets to be sent at the incorrect times and creates some small amount (¡0.1%) of network inefficiency.

2.4.5 Kernel-Based TDMA Conclusions

SelectorNet is a very strong first entry into using a new, lower energy optical implementation of a circuit switched datacenter network. The kernel-based TDMA qdisc module achieves high throughput for large traffic workloads, and in a 10 Gbps setting does not encounter significant issues with guard times or traffic routing.

With that said, this implementation of kernel-based TDMA networking is very limited in what it may achieve. There are several parts of this method, and SelectorNet overall, that require investigation for future work in order to fully realize what is required in a datacenter:

Unpredictable Traffic: SelectorNet's architecture requires use of a scheduler that has a predefined knowledge of traffic matrices, i.e. it has to know everything about what, when, and where data is sent. This is obviously not realistic in practice, as datacenters are designed to handle all forms of traffic patterns, most of which are not known ahead of time. Many circuit switched networks use a central scheduler to act as an oracle [POB+14]. However, this comes with a huge sacrifice in scalability, the primary case where circuit switches networks create energy savings.

Bijective Flow Control: I only implemented networking tests using UDP traffic. This was far from an arbitrary decision; TCP and other bijective flow control protocols that require rapid communication between source and destination do not work effectively. This is due to a combination of the two issues listed below.

Small Packet Sizes: Tests on my qdisc were only performed with maximum (or close to maximum) size packets. This is far from the typical case in datacenters; small packets from functions such as remote procedure calls (RPCs) or TCP acknowledgments will often require small packets to be sent through the network. SelectorNet cannot handle small packets well due to how the TDMA module functions: several upkeep functions are run after every packet transmission to ensure accuracy. The less time the module has before the next packet should be enqueued (i.e. before the NIC has finished transmitting it), the less time the module has for these functions. Relaxing these functions, however, leads to transmission errors, I will discuss further in Chapter 3.

Latency: SelectorNet has a drawback in how selector switching affects packet latency. The scheduler I use has optimized for throughput by generating timeslots that are long relative to the selector switch's reconfiguration delay δ. If the scheduler uses short timeslots, it could trade lower latency for reduced network throughput. To understand the potential savings, I have been provided the times at which the first packet from a timeslot in each workload will be delivered to its final destination, as a function of δ. I have been given a weighted average latency seen by each workload: all-to-all has 2.27δ, rolling stride has 1.13δ, and random bijection has 2.26δ of latency. This idea of trading latency for throughput is of great significance, as many datacenter workload care greatly about low latency traffic [DB13].

Combined, kernel-based networking and SelectorNet are not yet fully-realized enough to fulfill the needs of future low-energy datacenters. I only test kernel-based networking at 10 Gbps, a very low speed that is already obsolete [AWE19], and I discuss in the next chapter how higher

link rates make kernel-based TDMA networking inaccurate. SelectorNet has central design issues that stop it from being as scalable as future datacenter network demands would require. These issues will become a driving force for the next two designs that I will present in this chapter. Next, I move into an upgraded version of the SelectorNet architecture, known as RotorNet.

2.5 Kernel-Bypass Programming: RotorNet

I now move to discussing a new method of implementing TDMA, kernel-bypass programming. This differs from the kernel-based approach in SelectorNet by skipping the kernel completely, allowing a program to access the networking drivers directly rather than still using standard kernel interfaces to send and receive traffic. This comes with the drawback that the program must now implement necessary networking software components that were previously provided by the operating system kernel, which may be rather complex in practice [AGM+10]. The need for a kernel-bypass approach is based on the new requirements for a revision of the SelectorNet architecture, called RotorNet. RotorNet focuses on solving a major problem with the SelectorNet architecture: the control plane.

In this section I again focus on how my work achieves maximum transmission efficiency through scheduling and transmitting packets over the wire within the RotorNet circuit switched topology. I describe the limitations of a centralized control plane in RotorNet, and a new method of flow scheduling, RotorLB, that circumvents this issue. I then move into describing how kernel-bypass networking is necessary to accurately implement RotorLB, and finalize by testing its accuracy on the same live testbed as in SelectorNet. RotorNet is primarily an iterative work on SelectorNet, so I will not be describing the RotorNet architecture as it is effectively the same as from SelectorNet (with some minor differences described below in Section 2.5.2).

2.5.1 RotorLB: Decentralized Indirection

RotorNet makes use of a new, decentralized flow control protocol called RotorLB. The need for this comes from one of the core problems of SelectorNet- the centralized control server. The SelectorNet control server issues the fixed timeslot information to all of the servers in the testbed. The tests in SelectorNet used a fixed, predefined set of timeslots, but a full deployment would want to examine a more dynamic solution to work with a variety of datacenter workloads. Unfortunately, timeslots in the network must be decided consistently and dynamically based on the entire network state, meaning a control server cannot be distributed, but centralized.

A large body of previous work on circuit switched networks also use a centralized control server to collect network-wide traffic demand [LLF+14, PSF+13, WAK+10, POB+14], and other bodies of work have focused on using this demand to create an optimal schedule of timeslots [BVAV16, LML+15]. Enforcing this also requires rate-limiting endhost transmissions [BVAV16, LLF+14, LML+15, PSF+13], and synchronizing every component of the network together [LLF+14].

While this is certainly a *common* practice, it is an untenable one. Scaling a control server to tens of thousands of nodes means the scheduling algorithm for timeslots will take longer than the total switching uptime RotorNet wishes to use, making the timeslots useless [BVAV16, LML+15]. Recall that tens of thousands of nodes are where circuit switched networks begin to have power savings over packet-switched networks, so it is a must to create a network that does not have this scaling restriction.

RotorLB Overview

RotorLB is the replacement for the control server in RotorNet. RotorLB, or RotorNet Load Balancing, is a lossless, fully distributed protocol based on the principle of Valiant load balancing [Val82]. The primary principle is that instead of having a host communicate to a central

control server, it instead directly asks a destination host how much traffic it may indirect through it. To do this, every host has full knowledge of the connection patterns of the network, and if a connected destination may be able to forward traffic promptly.

When indirecting traffic, RotorLB injects traffic into the network fabric exactly two times: traffic is first sent to an intermediate rack, where it is temporarily stored, and then forwarded to its final destination. RotorLB stitches together two-hop paths over time as required by the traffic demand of the network.

Unlike traditional VLB, which always sends traffic over random two-hop paths, RotorLB prioritizes sending traffic to the destination directly (over one-hop paths) when possible, and only injects new indirect traffic when that traffic will not subsequently interfere with the intermediate rack's ability to send traffic directly. These two policies improve network throughput by up to $2\times$ (for uniform traffic) compared to traditional VLB.

In RotorNet, each ToR switch is responsible for keeping an up-to-date picture of the demand of each end host within the rack and for exchanging in-band control information with other ToRs. There are two types of traffic the ToR must track: *local* traffic generated by hosts within the rack, and *non-local* traffic that is being indirected through the rack. Because I use virtualized ToRs in the test network, each server has readily available demand information.

RotorLB Algorithm and Example

RotorLB runs between a source and destination when two hosts become newly connected after a selector switch is done reconfiguring. RotorLB's purpose is to determine how much direct and indirect traffic to send during the switch's uptime. First, RotorLB prioritizes any previously indirected (i.e. non-local) traffic as it has previously made a guarantee to deliver that traffic. Second, RotorLB allocates any direct local traffic (i.e. does not need indirection), as this maximizes network efficiency and reduces the overall bandwidth tax on the network.

Lastly, RotorLB calculates any remaining link budget that will be available during that

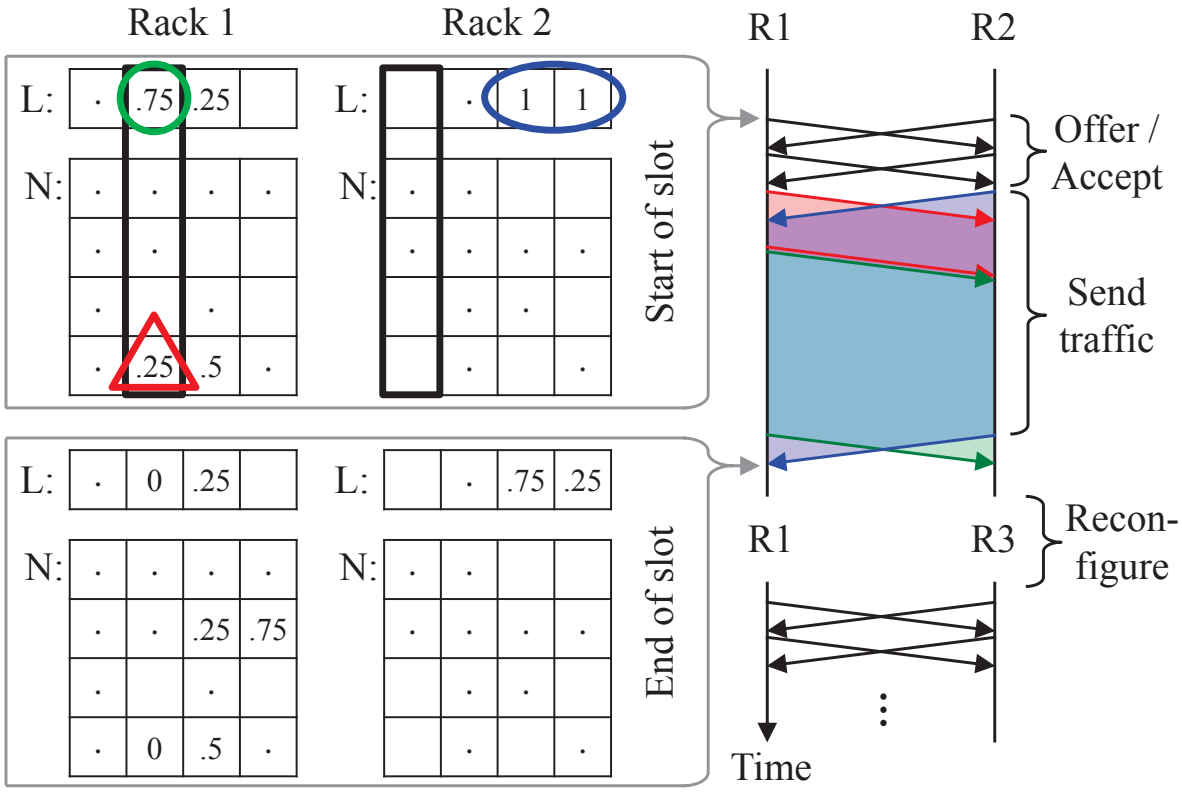


Figure 2.9: RotorLB example. Matrix rows represent sources and columns represent destinations; L and N represent local and non-local traffic queues, respectively; matrix elements show normalized traffic demand. In the current matching between racks 1 and 2, traffic which can be sent directly is bounded by black rectangles, stored indirect traffic is marked by a red triangle, one-hop direct traffic is marked by a green circle, and new indirect traffic is indicated by a blue oval. Adapted from [MMR+17].

Algorithm 1 RotorLB Algorithm. Adapted from [MMR+17].

```
function PHASE 1(Enqueued data, slot length)
    alloc ← maximum possible direct data
    capacity ← slot length minus alloc
    offer ← remaining local data
    send offer, capacity to connected nodes                    ▷ offer
    send allocated direct data
    remain ← size of unallocated direct data
    return remain
function PHASE 2(remain, LB length)
    recv offer and capacity from connected nodes
    indir ← no allocated data
    avail ← LB length minus remain
    offer_i ← offer_i if avail_i ≠ 0
    offerscl ← fairshare of capacity over offer
    while offerscl has nonzero columns do
        for all nonzero columns i in offerscl do
            tmpfs ← fairshare of avail_i over offerscl_i
            avail_i ← avail_i − sum(tmpfs)
            indir ← tmpfs
        offerscl ← offer − indir
        tmplc ← capacity − sum(indir)
        offerscl ← fairshare of tmplc over offerscl
    send indir to connected nodes                              ▷ accept
function PHASE 3(Enqueued local data)
    recv indir from connected nodes
    local_i ← enqueued local data for host i
    indir_i ← min(indir_i, local_i)
    send indir_i indirect local data for host i
```

switch uptime and "offers" to indirect traffic through the destination host. RotorLB only indirects traffic that can be ultimately delivered before the active source/destination pair will be matched together again. The destination then decides if it can deliver the indirected traffic in the future, by examining how much local traffic it has to send in the future as well. Algorithm 1 shows the pseudocode of the RotorLB algorithm.

To effectively balance load, I allow traffic from the same flow to be sent over RotorNet's single one-hop path and also to be indirected over multiple two-hop paths. This multipathing can

lead to out-of-order delivery at the receiver. Ordered delivery is ensured using a reorder buffer at the receiver.

Figure 2.9, shows an example of RotorLB in practice. Consider the ToRs of two racks, R1 and R2, which have current demand information for the hosts within each rack stored in non-local (N) and local (L) queues, as shown in Figure 2.9. In this example, demands are normalized so that one unit of demand can be sent over the ToR uplink in one matching slot. Note that, as described above, there is no central collection of demand; each host simply shares its demand with its ToR switch, and ToR switches share aggregated demand information in a pairwise fashion.

Phase 1: Send stored non-local and local traffic directly: RotorNet follows a fixed connection pattern, and each ToR switch anticipates the start of the upcoming matching slot as well as to which rack it will be connected. After taking a snapshot of the N and L queues, the ToR computes the amount of traffic destined for the upcoming rack. Delivery of stored non-local traffic on its second (and final) hop is prioritized to ensure data is not queued at the intermediate rack for long periods of time. Delivery of local traffic has the next priority level. In Figure 2.9, R1 has 0.25 units of stored non-local traffic (red triangle) and 0.75 units of local traffic (green circle) destined for R2, so it allocates the entire ToR uplink capacity for the matching slot duration to send this traffic. R2 has no stored non-local or local traffic for R1, so no allocation is made.

The ToR then forms a RotorLB protocol *offer* packet which contains the amount of local traffic and the ToR uplink capacity which will remain after the allocated data is sent directly. The smaller of the two quantities constitutes the amount of indirect traffic the ToR can offer to other racks. Once the matching slot starts, the ToR sends the offer packet to the connected rack. As an optimization, rather than waiting for the entire offer/accept process to complete, the ToR can also begin sending the stored non-local and local traffic which was been allocated for direct delivery to the destination.

Phase 2: Allocate buffer space for new non-local traffic: Shortly after the start of the slot,

the ToR switch receives the protocol packet containing the remote rack's offer of indirect traffic. At this point, it computes how much non-local traffic it can accept from the remote rack. To do this, the ToR examines how much local and non-local traffic remain from Phase 1. The amount of non-local traffic it can accept per destination is equal to the difference between amount of traffic that can be sent during one matching slot and the total queued local and non-local traffic. Because the amount of accepted indirect traffic is limited to the amount that can be delivered in the next matching slot (accounting for any previously-enqueued traffic), the maximum delivery time of indirect traffic is bounded to $N_m + 1$ matching slots, or approximately one matching cycle. The algorithm handles multiple simultaneous connections by fair-sharing capacity across them.

In Figure 2.9, R1 sees via the offer packet that R2 would like to forward 1 unit of traffic destined for each R3 and R4 (blue oval), and that R2 has a full-capacity link to forward that data. R1 already has 0.25 units of local traffic for R3 and 0.5 units of stored non-local traffic for R3. Therefore, it allocates space to receive $1 - 0.75 = 0.25$ units destined for R3 and 0.75 units for R4 from R2, which fully utilizes the remaining link capacity from R2 and ensures that all queued traffic at R1 will be admissible.

Once the allocation is made to receive non-local traffic, the ToR switch responds with a protocol *accept* packet informing the remote rack how much traffic it can forward on a per-destination basis.

Phase 3: Forward local traffic indirectly: Finally, the ToR switch receives the protocol accept packet from the remote rack. After it finishes sending direct traffic determined in Phase 1, it forwards new non-local traffic to the remote rack per the allocation specified by the accept packet.

In Figure 2.9, R2 receives an accept packet informing it that 0.25 units of traffic destined for R3 and 0.75 units destined for R4 may be sent. It forwards this traffic, which is stored as non-local traffic at R1. Finally, the Rotor switch reconfigures and establishes a new connection, and the RotorLB algorithm runs again.

2.5.2 RotorLB and Switch Matchings

RotorLB requires that it can communicate in a bidirectional fashion with the destination host. This is not how SelectorNet's switch matchings are architected; only one of the original matchings had hosts connected bidirectionally. RotorNet uses new switch matchings that create bidirectional matchings between endhosts, which changes how the network is connected. Additionally, RotorNet also uses $N_R - 1$ for N_R racks, distributed among the entire set of rotor switches. This also means that in order to create a datacenter architecture, ToRs must be connected to a number of selector switches equal to a power of two.

For the RotorNet experimental testbed, I run experiments with eight endpoints instead of the original sixteen, and the selector switch is then configured with the necessary seven matchings. Having seven matchings in a single switch is more than acceptable, as a large-scale RotorNet network would have greater than seven matchings in each selector switch.

2.5.3 Kernel-Bypass: Implementing RotorLB

RotorLB is a departure from the TDMA-style implementation in SelectorNet. For RotorLB, hosts need to send an exchange with the connected destination, which is composed of multiple smaller packet sizes. Recall from Section 2.4.5 that a primary issue with the TDMA module I use in SelectorNet is that it does not function well with smaller packet sizes.

Investigation of this restriction revealed that this is a function of how network interface hardware and firmware on servers is constructed. In order to ensure high link rates, network cards grab packets in batches in order to ensure that CPU processing of packet queues remains low. I measure the full effect of this in Chapter 3. To solve this restriction for RotorNet, I fill the link with "null-space" packets containing no data to ensure that the NIC is always transmitting data.

Unfortunately, the Linux kernel does not accommodate sending malformed packets without any data. To solve this restriction, RotorNet uses a "kernel-bypass" framework to transmit

data. Kernel-bypass networking is now a common practice to achieve high-speed networking inside of datacenter networks [ZLA+19]. It allows elimination of the operating system kernel from the traditional networking stack by allowing a regular application to interact directly with networking hardware. By doing so, applications achieve higher throughput and lower latency on the network.

The Cost of Kernel-Bypass

The benefits of kernel-bypass networking are not free. Traditional server networking uses an interrupt based method, in which the NIC alerts the operating system when data has been received. This allows the host's CPU to do other work while waiting for data, and in the past, this was the preferred method of interacting with the network interface in an operating system.

Kernel-bypass networks typically are not interrupt based; rather, they use a "polling" based method, where the host CPU constantly examines the NIC queues to determine if new data has been received. This uses a lot more CPU power on the endhost, but provides higher bandwidth and lower latency than an interrupt based method. With the advent of 100 Gbps networks and beyond, using a polling based method with a kernel-bypass interface has become common [ZLA+19].

For RotorNet, this means that I dedicate additional one additional CPU core per endhost to receiving incoming data, ensuring that RotorLB's initial negotiation completes as quickly as possible. For a 10 Gbps network, this additional CPU cost is high, but the servers I use in the RotorNet and SelectorNet work are fairly old and do not have nearly as much CPU power as modern servers (later in Section 2.6, I move to using hosts with 24 physical CPU cores).

Additionally, there is a simultaneous drawback and benefit with kernel-bypass networking, in that there is no longer support from the operating system kernel for forming and transmitting packets using common networking protocols. This means that this must be implemented manually for RotorNet. For my purposes this is suitable to ensure RotorLB is implemented correctly, but

would be problematic if RotorNet was required to run an arbitrary networking protocol in a full deployment.

Other reasons for moving to kernel-bypass

In addition to requiring a kernel-bypass API to implement the null-space packets required to make RotorLB work efficiently, SelectorNet also encounters an issue where after sending roughly 1 in every 2000 packets (0.2%), the Linux kernel context switches away from the sending thread to a NMI Watchdog process to verifies that the CPU core is still functional.

This process tends to take a long time, often hundreds of microseconds. In rare cases this causes packets to be sent out during a nighttime, causing loss. However, more commonly this means that some amount of throughput during the daytime is totally lost. For RotorNet, the design of RotorLB means that I must reduce rate of failure as much as possible.

Kernel-bypass APIs provide a solution as calling a "send" function inside of software is as close to the NIC as possible, so there are less instructions between the TDMA software assuming a packet was sent and it actually being sent to the NIC. This runs far less risk of the "send" call being interrupted by a context switch in the CPU core.

Kernel-Bypass APIs

For RotorNet, I experiment with two different kernel-bypass APIs to implement RotorLB. The first API is one unique to the Myricom Myri-10G Dual-Protocol NICs that are installed on the servers used for the experiments. This API allows direct interaction with NIC queues in a fairly generic fashion, not providing much in the way of additional features or support.

The second API is DPDK [Fou], a widely supported software framework that implements kernel-bypass support for a large number of server NICs. DPDK also provides a large number of memory management and packet processing tools, that permit construction of fairly complex custom software networking frameworks.

Both of these APIs are suitable for testing RotorLB, but I use the simpler Myricom API in order to ensure that I am able to use the same NICs as in SelectorNet when testing in order to draw an accurate comparison. I describe the results of a DPDK-based TDMA implementation in Chapter 3.

2.5.4 Kernel-Bypass Accuracy: RotorNet Results

I now move to using the testbed described in Section 2.4.4 to see how accurate and performant a kernel-bypass solution is for implementing RotorLB and RotorNet. I use the same endhosts as before running at 10 Gbps and the same real circuit switch, which still rotates between a fixed pattern of matchings that is preprogrammed in the endhosts. The goal is to determine how effective this implementation is in comparison to a projected maximum, as in Section 2.4.3, and understand what the limitations of kernel-bypass networking are.

I implement RotorLB on the endhosts as a user-level process, using the Myricom Sniffer API to directly inject and retrieve packets from the NIC. The only requirement to run RotorLB in practice is that endpoints be made aware of the Rotor switches' states. In a real implementation using ToR switches, each ToR could monitor the status of its optical links to determine when one matching ends (i.e., the link goes down), and the next matching begins (i.e., the link comes back up again). The Myricom NICs I use do not have a built-in low-latency method to detect link up/down events, so I use an out-of-band channel to notify end hosts of the switch reconfiguration events.

I emulate a RotorLB ToR switch on each server using distinct user-level threads to generate, send, and receive UDP traffic, with an additional thread to process state changes of the Rotor switches. To analyze performance under a variety of traffic conditions, I am provided traffic patterns with different numbers of "heavy" connections. Each heavy connection attempts to send data at line rate. I define the **traffic density** as the fraction of heavy connections out of all possible connections (56 in the 8-endpoint prototype). For each traffic density, I repeat the

experiments with 32 randomly-generated traffic matrices representing the inter-rack demand.

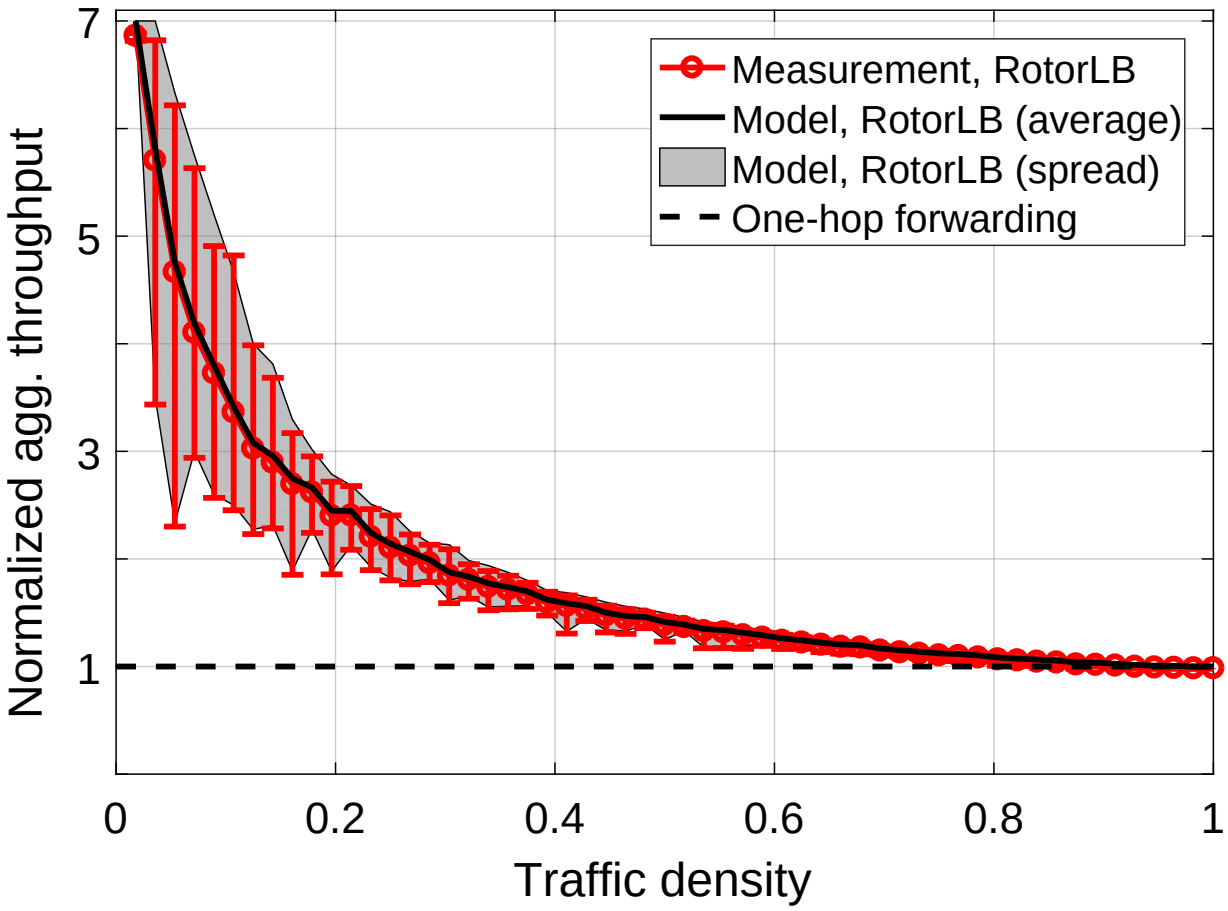


Figure 2.10: Measured and modeled throughput under RotorLB relative to that using one-hop forwarding. Circles represent the average throughput over 32 random traffic patterns; error bars show the maximum and minimum.

As a baseline, I first send data through the network using only one-hop forwarding. Next, I repeat the experiments with RotorLB running on the endpoints. Figure 2.10 shows the relative network throughput under RotorLB normalized to that of one-hop forwarding. I see that RotorLB significantly improves throughput for sparse traffic patterns. For a single active heavy connection, RotorLB improves throughput by the expected factor of $N_R - 1$ (7 in this case, with $N_R = 8$ virtual racks), as traffic can now take advantage of all paths through the network. Further, I observe that RotorLB adaptively converges to the throughput of one-hop forwarding for uniform traffic, as intended.

Figure 2.10 also overlays the modeled relative throughput of RotorLB to one-hop forwarding for the same set of traffic conditions I use in the measurements. The close agreement between model and measurement demonstrates that RotorLB operates as designed, and also validates the model's ability to accurately predict RotorLB performance in practice.

RotorNet TDMA Parameters

For RotorNet, I use a more aggressive set of TDMA parameters to achieve the above results. I find that a minimum guard time of roughly $400\mu s$ is necessary to achieve a packet loss rate of about 0.05%. To achieve loss rates more typically found in datacenter networks (a 10^{-7}) loss ratio), it is necessary to extend the switch uptime to roughly $10ms$ and use a $1ms$ guard time. This amount is large enough that the aforementioned context switching issue is now outweighed by the guard time, meaning a host almost never will send during a switch's downtime.

However, having such a long switch uptime and guard time creates a number of inefficiencies in the network, as I described earlier in Section 2.3.2. While this did allow me to achieve ideal results in Figure 2.10, it is not ideal for a real datacenter network. The unfortunate reality is that software can encounter a number of delays when trying to emit packets at precise intervals, causing unexpected and unacceptable loss in the network.

2.5.5 Kernel-Bypass and RotorNet Conclusions

RotorNet's design is primarily focused on removing the control plane as a necessity to create a circuit-switched datacenter network. It succeeds in this respect, but this design still does not solve all of the issues outlined in Section 2.4.5. This necessitates using a kernel-bypass framework to more tightly control when small packets are emitted from the NIC, with the trade-off of requiring more CPU power on the endhost.

Through proper configuration of my kernel-bypass implementation, I am able to send sufficient traffic over the network at 10 Gbps with 99.9% accuracy. However, the large guard time

of 1*ms* required to achieve this means that a long switching period of 10*ms* is used, reducing the efficiency the network would have when routing low latency traffic (one of the primary issues I note in Section 2.4.5). Kernel-bypass is useful for implementing a reasonable TDMA schedule combined with a rapid exchange protocol in very specific conditions.

Kernel-bypass networking and RotorNet's design begin to reveal that there is a persistent problem with precision networking in software. Ensuring that packets are not lost at an unacceptable rate is a critical issue, and one that is more prevalent in the next project in this chapter, Opera.

2.6 RDMA Networking: Opera

The final method of networking I will present in this chapter is remote direct memory access (RDMA) based networking. This is developed alongside the Opera project, which is the final project in the trio of optical circuit switched datacenter networking designs I contribute to developing. RDMA networks are traditionally used in high performance computing (HPC) datacenter architectures where the entire workload is controlled and predefined, but grew into working over more traditional datacenter network as well [SLLP09].

I use RDMA networking in Opera to achieve even tighter packet emission control than in the kernel-module and kernel-bypass frameworks. These previous frameworks do well with predicable, long periods of large packets, which fit very well into the SelectorNet and RotorNet project goals. However, Opera introduces some critical new factors that unfortunately render the previous methods unsuitable. Additionally, the testbed in Opera is extended to a 100 Gbps network, exacerbating these problems.

2.6.1 Opera: Anytime Low-Latency Traffic

Opera extends upon RotorNet by creating a new architecture based on a new optical switch [MFK+20]. This switch uses slightly more power but provides several advantages over the previous MEMS-based switch, including having a lower reconfiguration delay and supporting a higher port count. The Opera datacenter architecture is a variant of the RotorNet architecture in construction [MDG+20].

Opera's key contribution is how it handles low latency traffic. No tests in SelectorNet or RotorNet use low latency traffic, as it performs poorly on these networks due to the longer uptime periods required to support large, bulk traffic flows efficiently. Opera solves this problem by providing persistent, always available multi-hop paths for traffic. To do this, optical switches in Opera change at at non-uniform, staggered periods, ensuring that there is no "absolute" downtime of the network as in SelectorNet and RotorNet. Opera then allows designated low-latency traffic to be sent anytime through the network, using a path that is currently available. In the case that a path becomes unavailable during transmission, ToR switches will dynamically reroute the low-latency traffic through a different (possibly new) available path.

The combination of available paths at any time forms an expander graph. The end result is a single fabric that supports bulk and low-latency traffic as opposed to two separate networks used in hybrid approaches. Opera does not require any runtime selection of circuits or collection of traffic demands as in RotorNet, vastly simplifying its control plane relative to approaches that require active circuit scheduling, such as ProjecToR [GMP+16] and Mordia [PSF+13].

This changes how packets sent to unavailable paths are handled in Opera. Instead of simply dropping the packet, they are now routed on an updated available path at the time of the packet's arrival. However, if during the packet's journey, the circuit topology changes multiple times, it is possible the packet could be caught in a loop or redirected along a sub-optimal path. Dropping the packet immediately (and expecting the sender to resend it) as in RotorNet would

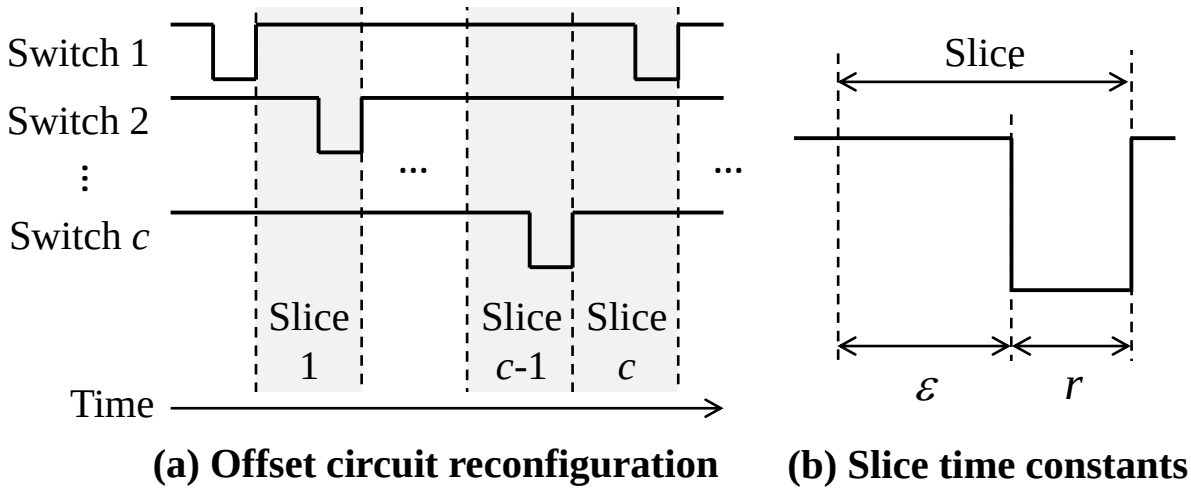


Figure 2.11: (a) A set of c circuit switches with offset reconfigurations forms a series of topology slices. (b) The time constants associated with a single slice: ε is the worst-case end-to-end delay for a low-latency packet to traverse the network and r is the circuit switch reconfiguration delay.

significantly lower network performance. To account for this, Opera changes how guard delay functions.

Figure 2.11 shows how the offset, staggered slot scheduling is performed in Opera. Opera requires that subsequent circuit reconfigurations be spaced by at least the sum of the end-to-end delay under worst-case queuing, ε, and the reconfiguration delay, r. I refer to this time period ε+r as a "topology slice". Any packets sent during a slice are not routed through the circuit with an impending reconfiguration during that slice. This way, packets always have at least ε time to make it through the network before a switch reconfigures.

The parameter ε depends on the worst-case path length (in hops), the switching queue depth, the link rate, and propagation delay. Path length is a function of the expander, while the data rate and propagation delay are fixed; the key driver of ε is the queue depth. Opera chooses a shallow queue depth of 24 KB (8 1500-byte full packets + 187 64-byte headers), and sets ε to 90 μs.

Because Opera does not use circuits that have an impending change at all during a topology slice, it does not need to necessarily guard against a delay for those paths. That is, if a packet is

sent late during a timeslot for a slice containing a path that is still available in the next slice, it is okay to be more aggressive in sending data during that timeslot. This is particularly important when considering Opera's switch uptime. Opera's example network contains 108 racks with 6 circuit switches, leading to a switch uptime of 10.8 *ms* [MDG+20]. Having a long uptime period means that hosts can send very large bursts of traffic over "safe" paths, provided low-latency traffic is accounted for.

Scheduling Around Low-Latency Traffic

Opera still wishes to support bulk traffic as in SelectorNet and RotorNet using RotorLB. Endhost scheduling, however, becomes more complex due to the competition of the link for low latency traffic. Low latency traffic must be the priority in order to satisfy its namesake, so bulk traffic must now dynamically react to unpredictable low latency traffic. To do this, endhosts must be aware at all times not only how much data to send, but **how much data has been sent already.** Unfortunately, the previous solutions of kernel-based and kernel-bypass networking do not provide a straightforward technical solution to this problem. While traffic control protocols such as TCP do provide round trip feedback on whether or not data has been received, it is expected that additional data be sent while an acknowledgment to the sender is in flight to ensure that the network link is saturated.

100 Gbps Networking and Mixed Packet Sizes

Recall in Section 2.4.5 that I describe how smaller packet sizes do not function well with kernel-based networking. For kernel-bypass networking, hosts are able to send a single small packet for RotorLB reliably, but rapid amounts of small packets still tend to struggle compared to larger packet sizes (I describe this in detail in Chapter 3). Low-latency flows will frequently contain packets of small sizes, meaning that these methods will further be hindered and more inaccurate due to the demands of Opera. Opera commands each endhost to send bulk traffic in

tandem with low-latency flows, and still be synchronized to the state of the network.

This problem is exacerbated by higher network rates (again, described further in Chapter 3). As I will describe below in Section 2.6.3, the Opera testbed is run on a 100 Gbps network. This means that the time required to send each packet over the network is now 10% of what it would be on RotorNet's 10 Gbps network, and thus any endhost delay that is guarded against (which is unchanged by using a faster network) now has 10 times the impact. As kernel-bypass is not suitable for Opera's needs, I must find a new solution to meet these demands.

2.6.2 The Solution of RDMA

RDMA networking provides a solution to these problems. RDMA provides "transaction-based" network primitives that give an explicit feedback signal to the sender when a transfer has completed. However, unlike TCP, these transfers can be of arbitrary size (up to a limit far beyond Opera's needs), allowing hosts to select an optimal "chunk" of bulk traffic to be safely sent during a timeslot in the network, assuming a maximum bound on the amount of low-latency traffic sent in the interim. Providing such a bound is simple when examining traffic patterns in datacenters; data published by Microsoft [AGM+10, GHJ+09b] and Facebook [RZB+15b] provide essential insights into the predictability of low-latency workloads, allowing Opera to define a fixed threshold of what constitutes low-latency traffic. In Opera, low-latency traffic is defined as any flow that is under 15 MB in size. With RDMA over Converged Ethernet (RoCE) [SLLP09], endhosts in Opera are able to use RDMA despite having a more traditional datacenter architecture.

RDMA is not flawless; it is a non-traditional choice for networking that requires data be given to the networking stack in bulk, as endhosts now need to make larger requests in order to solve the listed problems above. Traditional Ethernet networking is packet based, with packets being a maximum of 9000 bytes, which is .72 μs on a 100 Gbps network. To fill a timeslot with even a modicum of data, endhosts in Opera must operate on bulk traffic that is given in much

larger chunks as well. This assumption is suitable for Opera's needs, but may cause issues with "bulk" workloads that provide data incrementally [RZB+15b].

There are two methods of RDMA networking that I use in the Opera testbed: Message Passing Interface (MPI) and RDMA_WAIT.

MPI: Open MPI is a messaging protocol that can be used over RDMA networks, and is commonly used in high performance computing [GFB+04]. It functions by providing a way to pass messages between a source and destination in a tightly synchronized fashion, which is very suitable for a circuit-switched network. MPI automatically handles all RDMA transaction handling. Using MPI allows Opera to designate large block data transfers asynchronously to many different hosts, and then poll over each block to enqueue another block of an appropriate size after completion. This permits a more dynamic method of transmission that does not require additional complexity at the endhost.

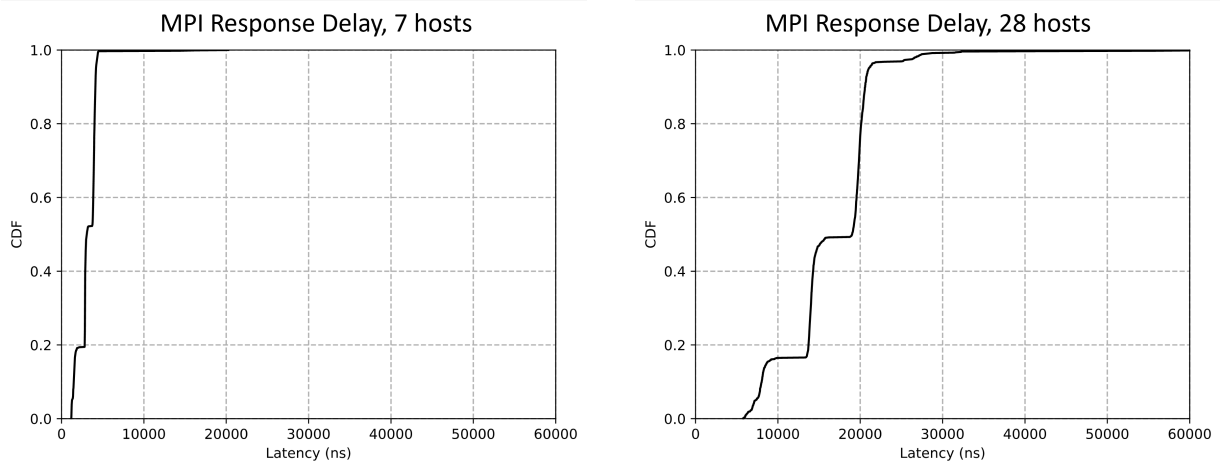
RDMA_WAIT: RDMA provides several different primitives that can interact with the network. A newer form of primitive is RDMA_WAIT, which signifies that a queue of traffic should be paused until it receives a signal. As Opera uses signals to specify when switch states are available (described further below), this is suitable for sending traffic appropriately between endhosts in a more controlled fashion. This requires additional complexity as it operates purely at the RDMA level, unlike MPI, but provides a roughly 2.5x latency improvement and additional consistently over MPI.

Each of these solutions perform well in a limited capacity, but when scaling up to larger numbers of hosts, begin to encounter an issue where the response latency to a switch uptime event grows larger. To understand how this affects network performance, I move to describing the Opera testbed.

2.6.3 The Opera Testbed

The Opera prototype uses a virtualized datacenter network implemented via a single physical 6.5 Tbps Barefoot Tofino switch. The virtualized network consists of eight ToR switches, each with four uplinks connected to one of four emulated circuit switches. The switch is programmed with a P4 program written by Rajdeep Das to emulate the circuit switches, which forward bulk packets arriving at an ingress port based on a state register, regardless of the destination address of the packet. The virtual ToR switches are connected to the four virtual circuit switches using eight physical 100 Gbps cables in loopback mode (logically partitioned into 32 10 Gbps links). Each virtual ToR switch is connected via a cable to one attached end host, which hosts a Mellanox ConnectX-5 NIC. There are eight such end hosts (one per ToR switch), and each end host runs four sets of parallel workloads that are rate limited to 10 Gbps each. The endhost connections are configured to run at 40 Gbps rather than 100 Gbps, for reasons I will explain below.

The parallel workloads mentioned above consist of two separate processes (for a total of eight per endhost): one is a shuffle-based bulk workload executed via MPI, and the other is a simple low latency "ping-pong" application that sends rapid RDMA messages to a random destination. Using synchronization functions in MPI (particularly the barrier primitive), I create a control server that is attached to the Tofino switch that sends signals through the switch to the virtualized network and connected endhosts. This updates a state register in the Tofino that triggers the appropriate network matching state according to the virtualized circuit switch configuration I wish the switch to emulate at that time. The control server then signals the endhosts to run the next step of their shuffle-based workload via MPI. The low latency application runs persistently through all network states, as would occur in a real Opera deployment.



(a) MPI response latency with 8 endhosts.

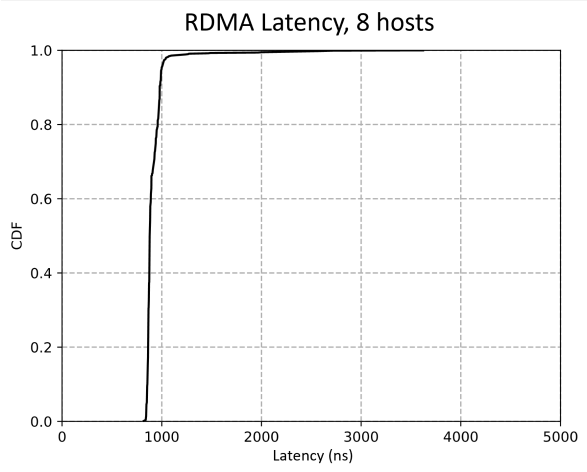
(b) MPI response latency with 32 endhosts.

Figure 2.12: CDF of latencies observed when signaling endhosts to begin transmitting data using MPI.

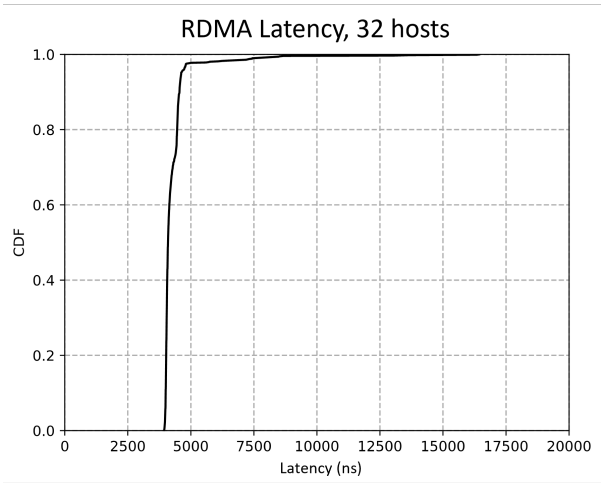
Scaling guard delay with host count

As mentioned above, guard delay scales as a function of the number of virtual servers contained in the Opera testbed. The reasoning for this is due to how the control server functions; a synchronized MPI primitive is very useful for providing a barrier for distributed systems that need logical consistency, but unfortunately the needs of a circuit-switched network do not require a logical consistency, but consistent timing instead. Opera uses a new method of providing timeslot information; rather than provide the host the knowledge of what states will be incoming, they instead only know to send what data the control server tells them to. This allows for a more simplified method of endhost programming, but the unfortunate side effect is that the control server adds additional delay scaling with the number of endhosts. The solution in SelectorNet and RotorNet is to include full future timeslot timing information, removing this delay.

This scaling artifact is exhibited via a simple experiment where I use Opera's control server to signal the endhosts to send data. To reduce delay, each endhost sends a single 1500 byte packet back to the control server. I use both MPI and RDMA_WAIT with both 8 virtual endpoints (1 process per server) and 32 virtual endpoints (4 processes per server). Processes are pinned to



(a) RDMA_WAIT response latency when signaling 8 endhosts.



(b) RDMA_WAIT response latency when signaling 32 endhosts.

Figure 2.13: CDF of latencies observed when signaling endhosts to begin transmitting data using RDMA_WAIT.

distinct CPUs to isolate them, and run over the distinct virtual 10 Gbps links described above.

The results for MPI are shown in Figure 2.12, and the results for RDMA_WAIT are shown in Figure 2.13. While RDMA_WAIT encounters roughly one-third the latency that MPI does in the 32-host case, the control server encounters a higher tail latency in both cases; roughly 20 μs for RDMA_WAIT, and 60 μs for MPI. This is expected for how a control server would operate in practice: running on a greater number of hosts means the control server must spend more time operating on processing and sending signals to endhosts. It should be noted that in Opera, these control signals are only given from the ToR to nodes connected to the ToR in the same rack. This scaling problem will not encounter the same critical issues as in SelectorNet.

However, the core issue is that the long tail of latency is still on the order of tens of microseconds for a low number of endhosts. A full datacenter rack will contain even more servers, and having a guard time of 100 μs or greater to account for these signaling delays reduces network efficiency for bulk traffic. Opera does not aim to optimize bulk traffic, but low-latency flows, so the efficiency of bulk traffic is not measured. This design is intentional due to how inefficient the endhosts are at operating on a TDMA schedule at 100 Gbps, which I describe further in

Chapter 3.

2.6.4 RDMA and Opera Conclusions

Opera is an architecture that aims to optimize a path for low latency traffic while still providing paths for bulk traffic to transit the network. It achieves the low latency traffic goals well, but there is still a problem of scaling bulk traffic to high data rates without encountering significant amounts of guard delay and network inefficiency. RDMA based transmission methods serve well at fulfilling the needs of signaling traffic from a ToR to the endhosts in a more controlled fashion due to the problems kernel-bypass frameworks face with controlling packet transmissions dynamically around unpredictable low latency traffic.

MPI and RDMA_WAIT are both valid ways of using RDMA to transmit traffic, with different tradeoffs. MPI provides a method of synchronizing endhosts using signaling in a more robust and supported fashion, while RDMA_WAIT provides a performance benefit in the signaling of endhosts. However, in both cases they are not able to remove the high amounts of guard delay that occurs in the long tail of cases when working with TDMA scheduling. While in the common case they function well, unfortunately guard delay must be scheduled around the 100th percentile of signal delay, which goes into the tens of microseconds for even a low number of endhosts.

2.7 Optical Networking Conclusions

Optical networking provides a future for highly energy efficient datacenter networks that can scale to high data rates. However, there is a significant challenge in having endhosts support the needs of TDMA networking protocols to interact with a circuit-switched architecture. Endhosts do not traditionally care exactly when a packet transits the wire, which leads to loss in a TDMA network when packets transit at the wrong time. A solution to create precise transmission is necessary to ensure packets are not lost and the network operates as efficiently as a traditional

packet-switched network.

Using kernel-based, kernel-bypass, and RDMA frameworks to send packets at precise times proves to be difficult. Scaling issues with increasing the number of endhosts and the data rate of the network causes packets to begin to become less precise over the network. Additionally, adding in a mixed workload of both large and small packet sizes creates additional unpredictability. Endhost software has to ultimately give packets to networking hardware, which may operate on packets at a different rate than the software expects or wishes.

While there are many issues with endhost networking in a circuit switched environment, there are cases where it does work effectively. It is necessary to understand the scaling limitations of endhost networking hardware in its current state in order to look towards what future work must be done in order to accurately and efficiently support the needs of future optical circuit-switched datacenters.

Chapter 2 contains material from three different sources co-authored by the dissertation author. Each source is used in a different part of the chapter.

Sections 2.3 and 2.4 have material from a paper that was rejected from publication. William M. Mellette; Rob McGuinness; Alex Forencich; Joseph Ford; Alex C. Snoeren; George Papen; George Porter, 2016. The dissertation author was an investigator and co-author for this material.

Section 2.5 contains material from a publication that appeared in the Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17). William M. Mellette; Rob McGuinness; Arjun Roy; Alex Forencich; George Papen; Alex C. Snoeren; George Porter, Association for Computing Machinery, 2017. The dissertation author was an investigator and co-author for this material.

Section 2.6 incorporates material from a publication that appeared in the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). William M. Mellette; Rajdeep Das; Yibo Guo; Rob McGuinness; Alex C. Snoeren; George Porter, USENIX Association,

2020. The dissertation author was an investigator and co-author for this material.

Chapter 3

Evaluating the Performance of Kernel-Bypass Software NICs for 100 Gbps Datacenter Traffic Control

In the previous chapter, I discuss how reducing the operational energy of datacenter networks can be achieved through creating new, high speed, optically circuit switched datacenters. However, the goal of ensuring that endhosts are able to function within the restrictions of the circuit switched networks was not completely achieved. My results show that endhosts are limited in how accurate they can be when transmitting data over restrictive synchronous periods. The question remains, just how inaccurate are endhosts in these scenarios, to show just how much hardware assistance is required?

In this chapter, I investigate the functional limitations of endhost networking software when using high speed networking hardware running on circuit switched networks. I find that there are a large number of limitations that are critical to understanding a future path forward for creating and deploying circuit switched networks that have lower energy and environmental costs.

3.1 An Introduction to Software NICs

The network interface card (NIC) is the interface between a host and the network. Traditionally the role and responsibilities of the NIC was clear, well-defined, and simple. The NIC transmitted packets generated by the operating system (OS) to the wire, and demultiplexed incoming packets from the wire to deliver them to the relevant input buffer. TCP's flow control and congestion control algorithms placed no particular requirements on when the NIC and OS actually transmitted packets on the wire, so long as no more than a certain number of packets were in flight at a time, as dictated by the receive and congestion windows. Recently, the adoption of datacenter networks, large-scale clusters, and rack-scale computers has fundamentally changed the interface between the server and the network. As a result, the NIC has become the "ground zero" of this reinvention, with commensurate changes in the requirements placed on it by application developers and network operators.

Several trends have driven the substantial change seen in endhost network stacks and network interfaces. End systems increasingly rely on virtualization to improve efficiency, either through virtual machines or via lightweight containers. The orchestration of traffic to and from these virtualized endpoints and the network requires network address translation, the implementation of access control lists (ACLs), and often custom forwarding rules, which are typically implemented in a virtual switch (vSwitch) abstraction. For performance reasons, the NIC has increasingly implemented this vSwitch functionality. A second trend is scaling across multi-core systems, which requires "steering" packets from the network directly to the core or hyperthread responsible for processing that flow.

Another prevalent trend is the introduction of radically new transport protocols. This includes protocols such as pFabric [AYK+12], NDP [HRA+17], Fastpass [POB+14], and Ethernet TDMA [VPVS12]. Unlike TCP, these new transports commonly impose stringent requirements on exactly when packets need to be transmitted on the wire, and further often require fine-grained,

per-flow rate limiting [JAM+12, SKG+11]. Finally, datacenter operators and rack-scale computer designers have begun to explore new, advanced topologies. As discussed in the previous chapter, one such topology, Opera, is based on expander graphs [KVS+17, VSDS16]. All of the circuit switched topologies in Chapter 2 rely on non-shortest path forwarding and multi-hop indirection. Other circuit-switched topologies topologies rely on RF [CXW+16, ZZZ+12] or optical [FPR+10, LLF+14] devices to physically reconfigure their structure, which necessitates sending data at precisely the correct time [LML+15, BVAV16] (and rate) to match the physical configuration, potentially relying on multi-hop indirection as well [BVAV16, MMR+17].

Rapidly increasing link rates make these trends even more challenging to address. Networks running at 10- and 40 Gbps have been deployed for years [SOA+15, RZB+15a], and 100 Gbps NICs and switches are now commodity. At the 2018 OpenCompute Summit, a number of vendors announced 400 Gbps networking gear to serve modern datacenter workloads, and Facebook now uses 400 Gbps networking equipment in their datacenter [AWE19]. While advancements in OS design have improved endhost performance, they are still hard to scale. This has led to the development of modular user-level, kernel-bypass frameworks called "software NICs" [HJP+15, PHJ+16, KPAK15]. Software NICs are highly programmable software interfaces between virtualized endpoints (e.g., VMs or containers) and the network. While software NICs are primarily targeted for environments where more complex operations are required, such as network function virtualization (NFV), they are beginning to gain traction as a standard medium for endhost traffic control [SDV+17].

The pairing of endhost-backed flow control with the fine-grained capabilities of software NICs invites the natural extension that software NICs should implement these endhost-based flow control proposals, as the extensibility and ease of use of software NICs makes them extremely attractive for both rapid development and deployment in datacenter and rack-scale environments. In order for software NICs to be suitable to today's environments, their performance must be satisfactory for both current and future workloads.

In this chapter, I investigate the feasibility of deploying complex endhost-based flow control and TDMA mechanisms on a representative modern software NIC, BESS [HJP+15]. As software NICs are deployed on faster links, first 40 Gbps and ultimately 100 Gbps, I seek to understand how they perform across a range of operational conditions. While I am evaluating BESS, its fundamental performance characteristics are largely due to DPDK [Fou], the underlying kernel-bypass framework that it is built upon, which I use in Section 2.5.

I create a number of experimental networking applications on the BESS framework and measure results up to and at 100 Gbps speeds. I focus primarily on a limited number of targeted applications to determine performance baselines across core flow control concepts. I discuss our experience developing for the BESS architecture during development and some additional features that would be beneficial to recent flow control proposals.

I find that while BESS and DPDK provide mechanisms for designing and implementing wide range of networking applications purely in software, there are still inherent limitations I observe that require NIC hardware support to solve. BESS provides great TDMA performance at speeds at around 40 Gbps, but fails to operate sufficiently at 100 Gbps in the majority of the scenarios I test.

3.2 Background of Software NICs

In this section I discuss software network interface cards (sNICs) and their architectures, taking note of the BESS author's original intentions of the problems they are intended to solve. Next I hypothesize how their software architecture can potentially provide a solution to current and future forms of high-speed network traffic control, particularly features that are often implemented via hardware rather than software. Finally, I review the core concepts of TDMA flow control, and how an sNIC may be leveraged for the requirements of TDMA schedules.

3.2.1 History of Software NICs (sNICs)

Software network interface cards are designed to supplement or reimplement functionality traditionally performed by NIC hardware, such as packet pacing, tenant isolation, and protocol offloading. Since these functions are implemented in software, they can be upgraded quickly, reducing bugs, enabling new features, and ensuring flexibility for future architectures and systems.

To provide high performance, sNICs layer directly on the hardware NIC by mapping portions of the NIC memory into the sNIC, rather than relying on an OS-provided device driver. Both BESS [HJP+15] and FlexNIC [KPAK15] directly claim to be software NICs, but I argue that NetBricks [PHJ+16], a framework targeted specifically for network function virtualization (NFV) falls in this purview as well. Both BESS and NetBricks leverage DPDK [Fou], which provides drivers for mapping hardware NIC memory into userspace and a generalized library for interfacing with these drivers.

Of course, software may perform any amount of packet processing it wishes; NIC hardware does not limit the actions that endhosts can perform inside the network. But while the Linux TC subsystem provides a workable interface for many forms of traffic control, such implementations are typically too slow for current demands [KNHM17], leading to the trend of hardware offloading described above.

As modern datacenters serve increasing amounts of demand each year [SOA+15], it is essential for sNICs to provide high performance alongside an easy-to-use framework for development. I argue that the long-term feasibility of sNICs is dependent on whether they can operate efficiently at high speeds or if iterations of specialized hardware will be necessary for future flow control needs.

For my evaluation, I use BESS as a representative case study for an sNIC. It supports recent versions of DPDK that interface with the 100 Gbps NIC used in my tests. My aim is to establish how well BESS achieves its goal for several different forms of traffic and flow control

(which I refer to more generally as "flow control" or "traffic control") in a 100 Gbps environment using several benchmarks. From this I posit the strengths and weaknesses of an sNIC and its potential to support new forms of traffic control.

3.2.2 Endhost Flow Control

The advertised features of sNICs make them a suitable target for quickly changing implementations of network traffic control. At its core, traffic control aims to maximize network performance. It is important to quantify multiple distinct vectors of measurement in order to properly evaluate sNICs as a networking utility. I focus on a few of these general concepts that I believe are critical for future datacenter architectures.

Rate limiting

Rate limiting is perhaps the most fundamental concept of flow control, with roots based in one of networking's earliest and most ubiquitous flow control proposal, TCP. The idea of rate limiting is simple: if the network cannot support the bandwidth that the host wishes to supply, simply make the host supply less bandwidth. This avoids packet loss, which can cause unnecessary retransmissions, increased latency, and lower overall network performance.

Although hardware-based rate limiters have become common in recent high-speed NICs, they are unlikely to support per-flow limiting for thousands of flows due to limitations in on-NIC memory and processing power [RGJ+14]. Recent work [SDV+17] has proposed a method to handle rate limiting in software with the help of an sNIC framework.

Many flow control proposals require rate limiting in order to avoid packet loss or restrict the bandwidth of low-priority flows. I list several such proposals in Table 3.1. It's clear that rate limiting is an essential component of flow control, and one that sNICs must support very well at scale in order to be a suitable alternative to hardware.

Packet pacing

Because bandwidth is measured as data per second, the delta time that bandwidth is computed for can be of any duration. However, packets are always sent at the configured speed of the link. Even if the rate a flow is sending at is properly limited, the packets from that flow may be arriving in bursts large enough to cause packet loss. This can be mitigated with larger packet queues, which has the side effect of increasing overall latency.

In order to allow for small queues (and thus low latency) without overloading the network, some proposals require that rate limiters also pace packets. Perfect packet pacing is achieved by ensuring each packet of a flow is sent on the wire with interpacket gaps equal to the amount of time the packet would have theoretically required to send on a link with the same aggregate speed of the rate limited flow.

This is generally a feature relegated to hardware for two reasons. First, it requires software to precisely time when individual packets are given to a NIC, which requires a dedicated CPU core. Additionally, supplying a single packet in each PCI transaction to the NIC drastically reduces the potential throughput of the system compared to sending a group of packets as a single unit.

Some flow control proposals leverage packet pacing mechanisms for additional performance. NDP [HRA+17] proposes a secondary queue for flow control administration that requires packet pacing in order to provide theoretical guarantees. I investigate what granularity of packet pacing an sNIC can perform to understand if it can potentially replace hardware solutions.

Flow scheduling and TDMA

Flow scheduling can refer to a number of different specific approaches within the flow control space. As mentioned in Chapter 2, TDMA flow scheduling is of interest due to its ability to be used in optical and RF-based reconfigurable datacenter networks [VPVS12, LLF+14, CSS+12, WAK+10]. In Table 3.1 I list some flow control proposals that leverage TDMA in

their implementation. These proposals aim to provide a future-proof way to scale the speed of a datacenter at the cost of path availability, requiring a TDMA-style form of control in order to function efficiently.

TDMA flow scheduling operates on the core idea that flows cannot be transmitted at all times. TDMA schedules require sending bursts of packets to destinations during precise time windows. The exact reason for this restriction is dependent on the datacenter architecture and flow control system. For example, optical networks implement a circuit-switched abstraction, and so endpoints can only send data to a particular destination or set of destinations when the circuit is established to those points. Fastpass [POB+14] implements this circuit-switched abstraction on a packet-switched network, rather than on a physical circuit-switch.

I review two central concepts to TDMA scheduling: the *period* of the schedule, and its *duty cycle*. The period of the schedule is the duration over which a single set of flows may be sent, and includes any downtime delay from reconfiguration or other sources. I divide a period into an uptime during which packets are sent and a downtime when nothing can transit the circuit

Table 3.1: Proposals that use rate limiting, TDMA, and/or indirection for flow control.

Proposal	**Rate Limit**	**TDMA**	**Indirect**
DCTCP [AGM+10]	✓		
EyeQ [JAM+12]	✓		
NDP [HRA+17]	✓		
TIMELY [MLD+15]	✓		
HUG [CLGS16]	✓		
Fastpass [POB+14]	✓	✓	
Diamond [CXW+16]	✓	✓	✓
WaveCube [CWM+15]	✓	✓	✓
RotorNet [MMR+17]	✓	✓	✓
Eclipse [BVAV16]	✓	✓	✓
OSA [CSS+12]	✓	✓	✓
TDMA [VPVS12]		✓	
Helios [FPR+10]		✓	
ReacToR [LLF+14]		✓	
c-Through [WAK+10]		✓	
Solstice [LML+15]		✓	

switch. The duty cycle of the schedule is simply the ratio of the uptime over the total period of the schedule, and represents the percentage of time that can be used to send data. The period and duty cycle of a dynamic TDMA schedule may change rapidly over time, such as in Fastpass.

TDMA flow scheduling can be difficult to efficiently implement on endhosts. Packets sent outside of the uptime can potentially cause increased queueing or even packet loss. This requires flows to start and stop transmitting as closely to the edges of the uptime as possible in order to maximize performance. Short periods or duty cycles make this a difficult task, but are theoretically beneficial by decreasing the scheduling latency of flows.

Little to no hardware support exists for TDMA scheduling, and enforcing precise forms of packet transmission is intractable with the standard Linux networking stack [VPVS12], as I discuss in Chapter 2. Software NICs may provide a potential solution to implementing this form of flow control at high speeds and realizing some TDMA-based flow control proposals which rely on simulations in order to compute their results [BVAV16, LML+15].

Multi-hop indirection

A unique form of flow control that has begun to appear in some systems is multi-hop indirection, a form of routing that has endpoints in the network forward a flow across multiple paths to its ultimate destination. Multi-hop indirection requires that components forward information back into the network along a transmission path, which reduces the potential outgoing throughput of the endhost or top-of-rack switch forwarding the flow.

Multi-hop indirection is rooted in the core ideas of Valiant load balancing (VLB), and has become a useful idea for flow control implementations in datacenter architecture proposals that do not always have a direct path between all endhost pairs. For example, the previously mentioned optical networking proposals often cannot implement a full crossbar using an optical switch. While an endhost could simply store the data until a direct connection is available, some proposals [MMR+17, CSS+12] instead leverage underutilized outgoing links

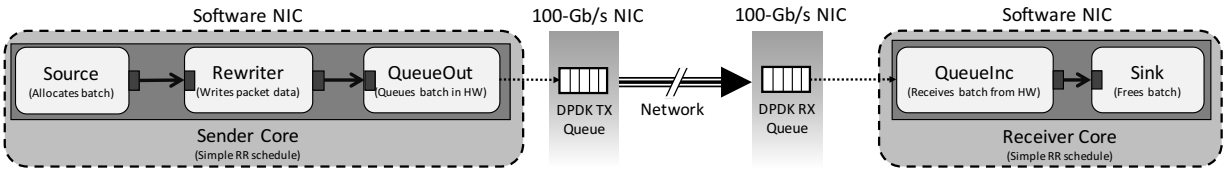


Figure 3.1: Simple sNIC sender and receiver. More cores may run concurrently with their own TX/RX queue.

at currently connected hosts and employ multi-hop indirection to reduce latency. Yet other proposals [VSDS16, CWM+15] may never have a direct connection between an arbitrary pair of hosts, and thus require indirection. I mark some proposals that use multi-hop indirection in Table 3.1.

Multi-hop indirection may be implemented via per-switch forwarding tables based on destination IP or other information, similar to how most tree-based datacenter networks typically operate. I examine a different method that provides forwarding information in each packet as an encapsulated GRE header, an idea taken from previous work in source-based datacenter routing [JFR16]. Packet encapsulation is typically done in hardware, and most enterprise-level switches support using GRE headers to determine the output port to forward an incoming packet.

Software NICs provide a low-overhead method of prepending headers to packets, and are able to implement multi-hop indirection. Future datacenter architectures may significantly benefit from this if hardware solutions for multi-hop direction at 100 Gbps speeds are unavailable or insufficient.

3.3 sNIC Testing Design

Software NICs are intended to be easily usable for a large range of networking applications, and must provide a consistent and extendable framework in order to fulfill this goal. They must accomplish this while also effectively leveraging an often unintuitive kernel-bypass framework in order to provide high performance for applications. BESS, the software NIC I analyze in this

work, achieves this by having only a few core interfaces that users can combine and interact with to implement complex network functions. These interfaces leverage or even partially reimplement DPDK libraries to gain the performance benefits DPDK provides. BESS also exposes programmable RPC endpoints to interact with these interfaces in order to enable more dynamic systems.

I briefly describe the interfaces that BESS provides and how they can be leveraged by users to perform a variety of actions. I also discuss the potential limitations of this architecture and how it may be improved to provide even further flexibility. I then focus on the framework I created to run and gather results, and the modifications I made to the general purpose tools provided by BESS.

3.3.1 Core functionality

The main component of the BESS sNIC consists of a C++ daemon that utilizes DPDK libraries. The daemon runs a gRPC (https://grpc.io) server that is used by a controller for setting up and configuring applications. A control client written in Python is provided to users in order to easily communicate with and execute tasks on a daemon. BESS applications consist of some combination of workers, schedulers, modules, and packets.

Components

A BESS worker is a logical thread pinned to a CPU core that executes a root scheduler object. Scheduler objects implement configurable scheduling policies, such as round robin or weighted fair queueing, and have one or more children attached to them, creating a tree. Each child may be either another scheduler object (a node), or a module that defines a task (a leaf). This is akin to how the Linux TC subsystem operates, and as such schedulers in BESS are also sometimes referred to as “traffic classes”.

BESS modules may or may not have a task that is executed by a scheduler, but every

module has some set of input and output gates that represent how packets flow through an application. An output gate may only connect to one input gate, though an input gate may have multiple output gates connected to it. A module that implements a task represents a starting point for some chain of modules that will handle batches of packets. When a module is run or receives some packets at an input gate, it will execute a function that may perform any operation it wishes over the batch, and then either stop all further processing of the batch or send the batch through an output gate to another module.

A range of basic modules are provided by the BESS codebase that implement several forms of common network functions, such as header encapsulation, field matching, and packet timestamping. Every module must supply a minimal set of RPC functions that allow the module to be created by the control client, and may supply further functions to allow users to inspect module state or change its configuration at runtime. Some modules provide wrappers around DPDK libraries, such as queue modules that may send or receive packets from a device queue on a DPDK-compatible NIC.

Simple applications

All traffic in my results is generated by software NICs. I define a flow generated by an sNIC to mean a continuous stream of a single unique duplicated UDP packet. Every byte of each packet is written by the sNIC, and additional headers are written into the packet based on the experiment being run. I program the packet generation modules to run at their maximum speed, meaning there is always infinite demand for every flow. This ensures that I primarily evaluate the performance of the scheduling and packet processing components; I defer evaluation of timely traffic generation and application feedback for TDMA traffic patterns to future work. One or more flows may be sent via a single TX/RX queue, and each TX/RX queue pair is handled by a dedicated CPU core.

The simple sending and receiving applications I run for my microbenchmarks presented in

Section 3.4.1 are shown in Figure 3.1. Source modules can be configured with different batch/burst sizes, which determines the number of packets that will be allocated and sent simultaneously to the pipeline, and ultimately over the network. The size of the data buffer given to the Rewriter module will determine the size of each packet in the batch. The statistics reported to the scheduler are used to determine sending and receiving rate.

3.3.2 Rate Limiting and Packet Pacing

BESS provides a rate limit scheduler that can attach to a single child task. I create one rate limit scheduler for each sending flow. Each worker uses a round robin scheduler as their root with rate limit schedulers as children. The rate limit scheduler uses an internal token bucket filter that is refilled at a user-programmed rate. After a batch of packets is sent by a flow, tokens are generated based on execution time. If a flow uses more tokens than are available, it is blocked for a period equivalent to the time it would have taken to produce those tokens.

This means that the rate limit scheduler attempts to perfectly pace the child when it is using bitrate as the variable for producing and consuming tokens. While the rate limit scheduler has a maximum burst parameter, this does not have any additional function when bitrate limiting is used. The true size of a burst for a flow is determined by the burst size that is input to the packet generation (Source) module, as this allocates a group of packets in a single execution loop of the scheduler. Because the rate limiter cannot block the flow in the midst of a single batch, large batch sizes at the Source module of a sending flow will result in a roughly equivalent burst of packets on the wire (n.b., the actual batch size sent is ultimately dependent on the NIC firmware).

3.3.3 TDMA Scheduling

TDMA support is not included in BESS. I implemented a custom TDMA module that runs on a dedicated CPU core with no other tasks. This ensures that the scheduler will not accidentally send packets at inappropriate times. While it would be expected to implement TDMA as a

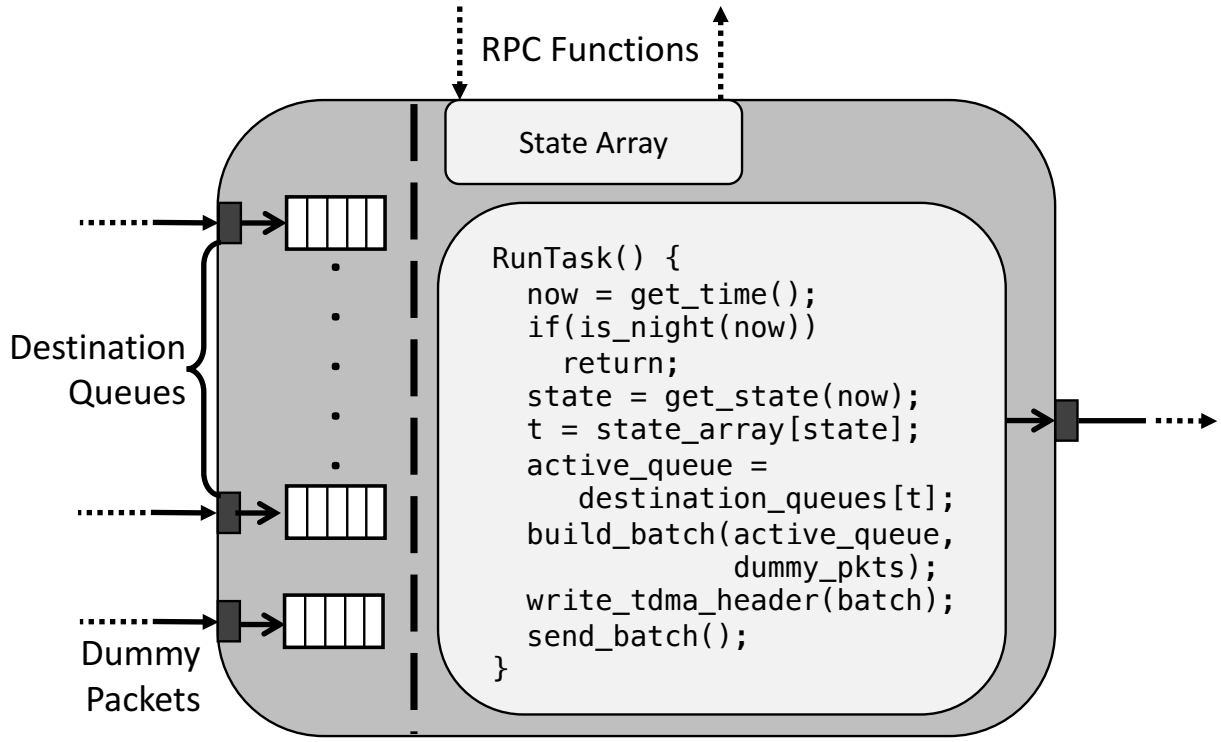


Figure 3.2: sNIC TDMA module. Destination queues may contain data from one or more flows. Dummy packets are used when little data is available to ensure the NIC transmits data packets promptly.

Figure 3.3: A TDMA schedule. Up-time preallocation and guard time correct for observed delays and variance in packet transmission from NIC hardware.

scheduling object, I have ultimately decided to implement TDMA as a monolithic module instead. The primary reasoning for this decision is twofold.

First, the module interface of BESS is clearly defined and allows for rapid development of new classes, and provides a direct way to create RPCs for these modules that can be used to inspect and modify the module's state. The scheduling interface does not permit heavy amounts of inspection or modification to overall state at runtime without significantly more programming effort, which made it difficult to use during development.

Further, the scheduling interface does not provide a way to the parent class to identify a specific node that has become blocked or unblocked, only that there was a child for which that event occurred. If a TDMA scheduler wishes to use rate limited flows or any flows that do not have infinite amounts of traffic, this then requires the scheduling object to manually iterate through all of its children to identify which child became blocked, resulting in an inconsistent and uncontrollable delay that scales with the number of flows the TDMA scheduler controls.

In general, I found the difficulty of developing new schedulers in BESS to be much higher than developing modules. Doing so requires a more in-depth knowledge of both the core daemon and control client, which may be disappointing for users that may be examining BESS as a general platform for future flow control systems. However, I have not examined the difficulty of creating a similar mechanism in the Linux TC subsystem, so I cannot comment on the difficulty of this task relative to BESS.

A custom TDMA module

The structure of my TDMA module is shown in Figure 3.2. The module is configured with the number of hosts in the network, and creates a destination queue for each. Flows provide packets to the appropriate destination queue via an equal number of input gates. The vertical line in the figure denotes a separation between the input gates which terminate the current task chain, and the output gate which executes a separate task on a dedicated core in order to ensure

scheduling accuracy.

RPC functions are used to program the state array of the module, which contains the up/down pattern that the TDMA module task should use, and the appropriate destination queue to use for each up-time. Once the last state is executed, the module will loop back to the beginning of the state array. The state array also contains a base index time that is used to synchronize the up/down periods of TDMA modules on separate machines. I use PTP on a separate interface not controlled by software NICs to synchronize the host clocks of each machine.

There is an additional input gate for dummy packets, which are generated by a separate Source module and contain a basic Ethernet header with zeroed MAC addresses. This is necessary in order to ensure that the physical NIC transmits packets quickly even if there is only a small number of packets waiting in a destination's queue when up-time begins. If a batch cannot be completely filled with real packets, dummy packets are used instead. Because the MAC addresses are zeroed, a network switch will drop these packets immediately upon receiving them.

Even with this precaution, there are still times that the physical NIC hangs onto packets too long, causing packets to be transmitted during down-times. Additionally, the delay between the NIC receiving and transmitting its first batch can be high. There is also a software delay from the time an up-time begins and the first batch of packets being handed off to the NIC. To solve these issues, I provide a *guard time* and an *up-time preallocation* value to a TDMA module.

The guard time value is used to "cut off" an up-time early to ensure that no packets get transmitted during the down-time. I evaluate the measured loss of various guard times in Section 3.4.3. The up-time preallocation value starts an up-time early in order to ensure packets fill as much of the up-time as possible. The combined effect of these two values is visualized in Figure 3.3. I show two up-times along with the down-time between them.

By using multiple cores each with a dedicated TDMA module, I can create multiple "virtual hosts" that are all controlled by a single sNIC. Each virtual host can then act in isolation, generating its own flows and sending to their own distinct destinations. For my results, I create

pairs of virtual hosts that only communicate with one another on fixed up/down intervals in order to provide a baseline evaluation of the capabilities of sNICs for TDMA scheduling.

Because BESS will run each module at full speed unless otherwise specified, I use a rate limiting scheduler with the BESS module as its child to ensure virtual hosts do not compete with one another for link bandwidth.

3.3.4 Multi-Hop Indirection

I measure the latency and accuracy of multi-hop indirection in sNICs to understand if they can be utilized for future datacenter architectures. I use packet encapsulation, which provides the most flexibility but slightly larger overhead due to the additional data and processing required for each packet in a flow. Another method of indirection could be to communicate a forwarding pattern as a scheduling state to the TDMA module, but this has extremely limited flexibility within my specific implementation.

BESS supplies generic header encapsulation and decapsulation modules to the user, and it can be used to implement both standard GRE headers as well as any arbitrary header the user wishes to define. DPDK packets provide a buffer on either side of the packet when they are allocated in order to rapidly prepend and append information without requiring the entire data buffer to be copied.

I create a simple forwarding application using basic modules that encapsulates each packet of a single flow with a 4-byte standard GRE header and timestamps packets before sending them to a receiving host. The receiver then removes the GRE header, swaps the MAC addresses in the Ethernet header, and then forwards the packet back to the sending host. The sending host records round-trip packet latency.

3.3.5 Statistics Modules

I create a few modules to gather statistics for my experiments. While basic rate information is provided by BESS, I wish to inspect interpacket gaps for packet pacing and record arrival times to analyze the performance of my TDMA module and multi-hop indirection application. I leverage the histogram object provided by BESS for TDMA scheduling statistics and round-trip latency.

Recording interpacket gaps cannot be done precisely in software, and I do not have a

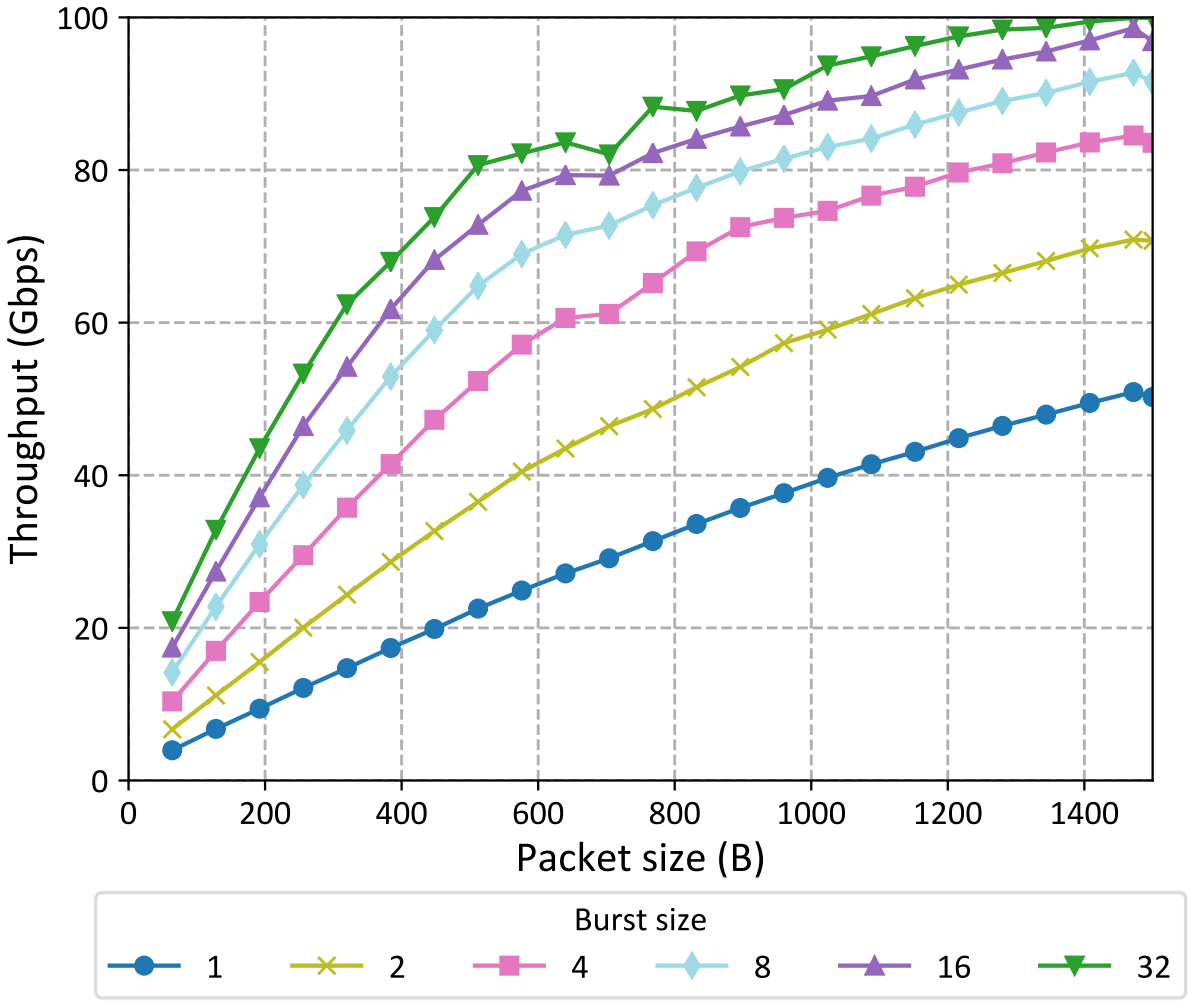


Figure 3.4: sNIC data throughput with one sending core.

hardware solution available for accurately timestamping packets at 100 Gbps. Instead, I create a module that records the arrival time and size of each batch of packets for a flow in a memory array for a fixed number of batches. This is then dumped to a file at the end of an experiment for later analysis.

While loss information can be provided by NIC counters, I create a sequencing module that detects packet loss in a flow. The sequencing module has a paired loss detection module that simply inspects the sequence number of each packet and records any gaps in a histogram. I slightly modify BESS to allow the sequencing module to retransmit sequence numbers if they were dropped at the sender by a later module.

3.4 sNIC Results

I now present results when using sNICs for a variety of flow control components along with microbenchmarks to provide an understanding of software NIC baseline performance. I execute my tests on a pair of Dell PowerEdge R630 servers, each with a 100 Gbps ConnectX-5 NIC. The NICs are connected together using a 100 Gbps Mellanox Spectrum Ethernet switch. Both servers are configured with two 12-core Intel Xeon E5-2650v4 CPUs, though only one is used to ensure I do not require any tasks to use QPI to communicate with the NUMA domain of the NIC. My servers run Ubuntu 16.04.3 LTS with Linux kernel version 4.4. I use a current open-source version of BESS that leverages DPDK version 17.11. The ConnectX-5 network card driver and firmware is provided by MLNX_OFED version 4.2-1.2.

3.4.1 Microbenchmarks

I begin by executing a few microbenchmarks that allow a basic understanding of the baseline performance of software NICs. This is a requirement when properly evaluating more complex applications, as I must understand if failures and limitations observed there are a result

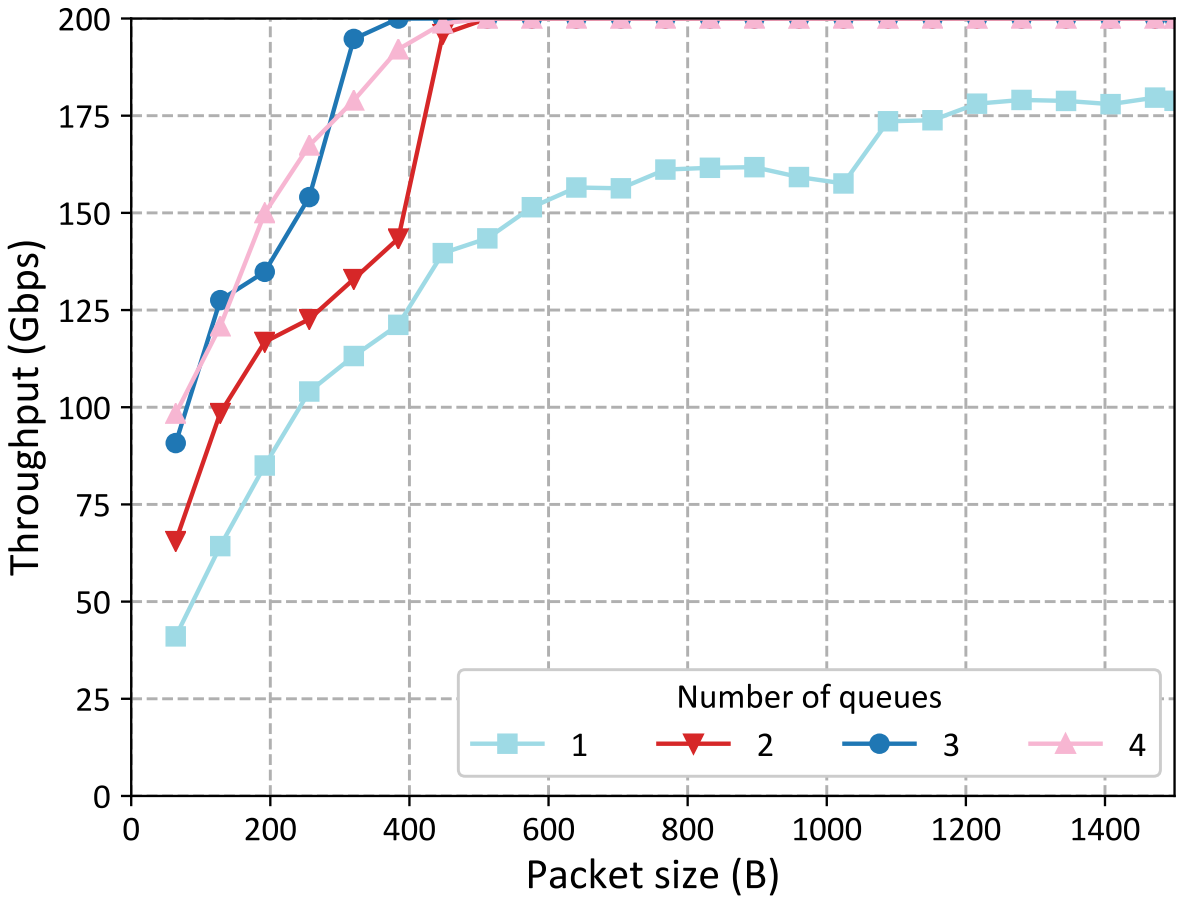


Figure 3.5: Combined sending/receiving throughput using bidirectional flows with a burst size of 32 packets.

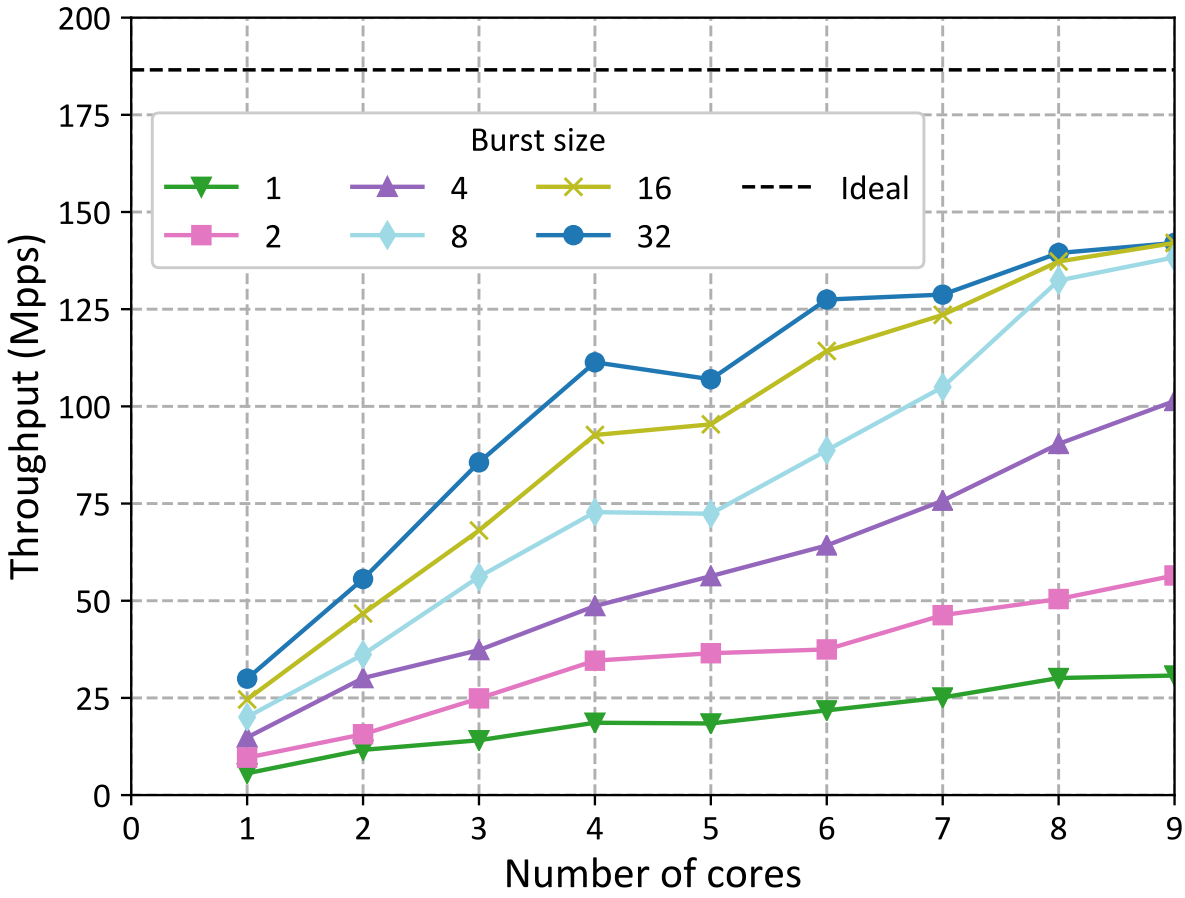


Figure 3.6: sNIC throughput with 64-byte packets.

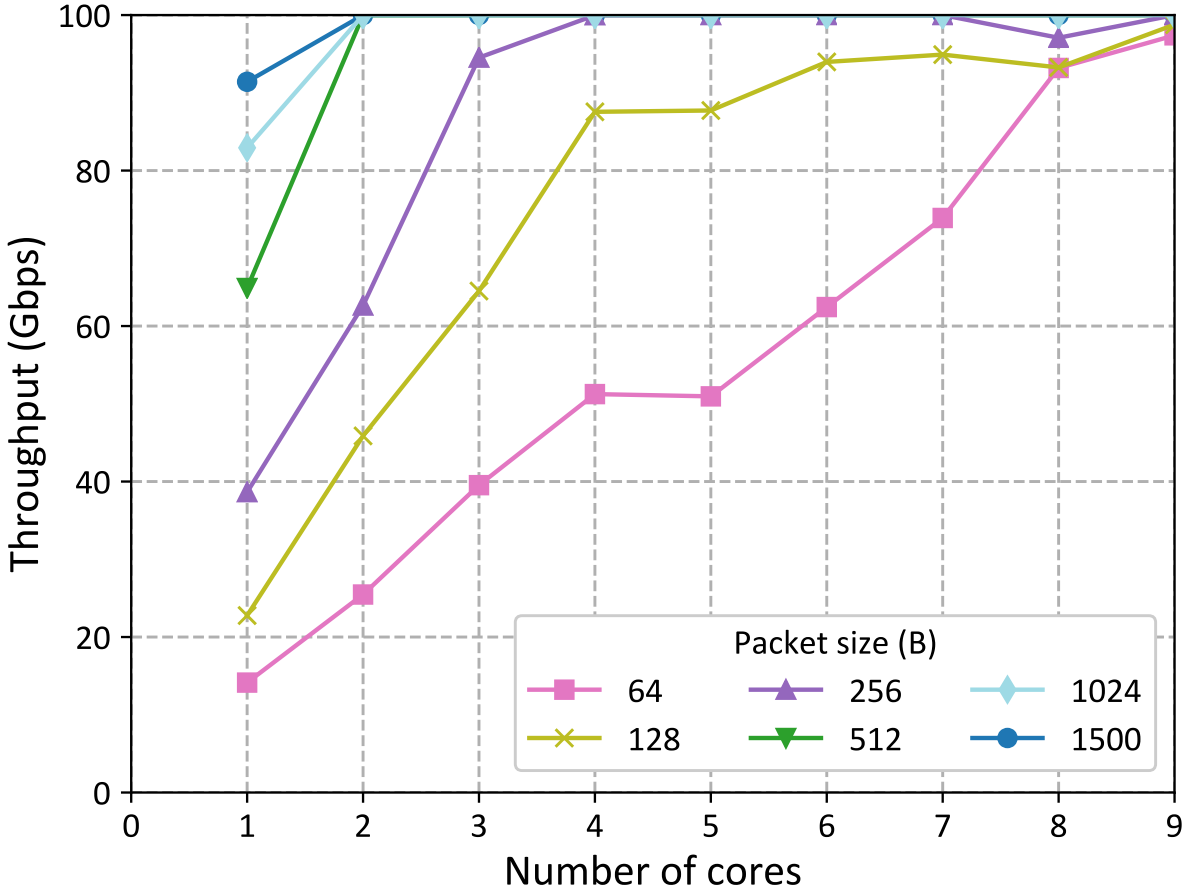
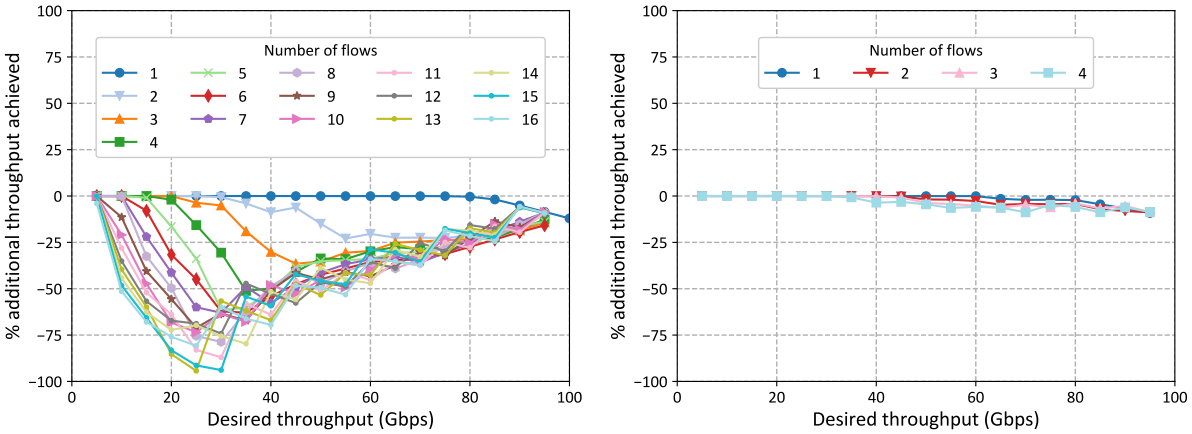


Figure 3.7: sNIC throughput with an 8 packet burst size.



(a) Flows from a single core.

(b) Flows from four cores.

Figure 3.8: Rate limiting accuracy of a primary flow. Remaining bandwidth is evenly split across all other flows (not shown). All flows use 1500 byte packets and a burst size of 8 packets.

of one or more basic restrictions. In all but one microbenchmark, one host acts as a dedicated sender, and the other as a dedicated receiver. I report the statistics at the sender, as the receiver was not the limiting factor in my microbenchmarks.

I first determine the throughput that a single core can provide in gigabits per second (Gbps) as a function of packet size. I also vary the burst size, which determines how many packets are created and sent in a single scheduling loop. A lower burst size means more time will be spent between allocating and sending each packet, but allows for finer grained rate limiting and packet pacing.

The results in Figure 3.4 suggests that a single core is able to saturate the entire link if maximum sized packets and the largest possible burst size (32 packets) is used. Even still, saturating the link with 40Gbps of bandwidth is very feasible with a large range of packet and burst sizes. As a baseline, a single core is quite capable of saturating link bandwidth. At 100Gbps, saturating with a single core is typically infeasible, but given 100Gbps is a large amount of data for a single core to handle in many scenarios, this is neither unreasonable nor unexpected.

Some of my later experiments use bidirectional flows, where each host acts as both a sender and receiver. I modify my previous microbenchmark and have each machine have a number of sending and receiving queues simultaneously handled by different CPU Cores. I measure the combined sending and receiving throughput at a single host.

I see from the results in Figure 3.5 that when a single queue is simultaneously sending and receiving data, it is unable to achieve 100 Gbps bidirectionally. I are unable to determine whether this is a limitation of BESS or DPDK, but this does mean that users should be aware that multiple queues may be required to achieve 100 Gbps bidirectionally on an endhost.

My next microbenchmark examines sending throughput and how this scales with multiple sending queues, where each queue is dedicated to a single physical CPU core. I measure the sending rate at the receiver in millions of packets per second (MPPS). I use 64 byte packets in order to ensure that the link can be saturated with the maximum possible number of packets

instead of data. I again test with different burst sizes at the sender.

The results are presented in Figure 3.6. I note that the advertised maximum MPPS for the NIC I use is roughly 140 MPPS [MT], meaning the theoretical ideal maximum shown in the figure is likely not achievable with my hardware. With the maximum possible burst size (32 packets), I can achieve just over 142 MPPS.

For my TDMA experiments in Section 3.4.3, I select a fixed burst size of 8 packets. The final microbenchmark I perform is the throughput scaling of multiple cores with this same fixed burst size and different packet sizes. I again have each queue on unique physical CPU cores.

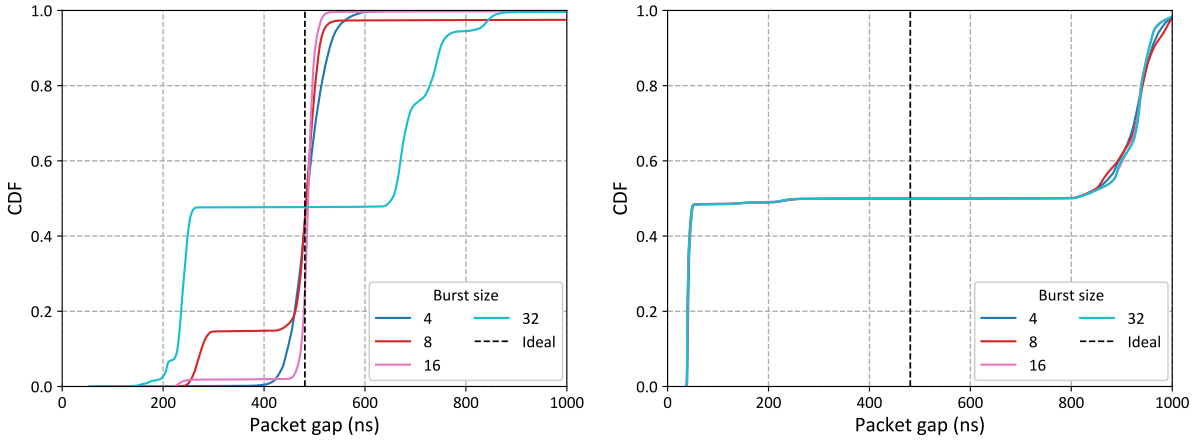
The data in Figure 3.7 asserts that demonstrate that adding a small number of additional cores can linearly scale throughput for all packet sizes, but as more cores are added, throughput grows at a slightly slower rate. Interestingly, throughput always increases when using 64 byte packets, but otherwise there is a slight loss of performance when using 8 cores. My hypothesis is again that there is some form of CPU cache contention at this point that has a more negative effect than the increased throughput provided by an additional core.

I find it interesting that performance decreased when increasing the number of sending queues or packet size in a few places, and are not able to provide a definitive explanation as to why. My hypothesis is that at these points the CPU cache became limited due to the increased memory required, offsetting the potential benefit.

I see from these microbenchmarks that although I am unable to completely saturate a 100 Gbps link with non-MTU size packets, it can send at a significant fraction of that with only a few cores and a moderate burst size. However, in order to ensure link saturation for future experiments with a lower number of queues, I use packet sizes of 1500 bytes.

3.4.2 Rate Limiting

I examine the accuracy of sNICs for rate limiting over a number of different active queues and flows per sending queue, using one dedicated core per queue. I again have dedicated sender



(a) A single flow.

(b) Four queues, each with one flow.

Figure 3.9: CDF of estimated inter-packet gaps with 1500 byte packets for a flow. All flows are limited to 25Gbps.

and receiver hosts. I use rate increments of 5Gbps and record the amount of bandwidth observed at the receiver. I create one receiver queue per sending flow to ensure the receiver accurately gathers sending statistics.

I sweep the desired rate for the first (primary) flow on the first queue I create. The remaining rate on the link is distributed evenly across every other flow in the experiment, including flows sent from other queues. I record both pacing and loss data along with the average throughput of each flow (see below).

I first increase the number of flows that are sent from a single core (and thus queue) and compute how close the achieved rate is to the desired rate for that flow. I use a burst size of 8 and a packet size of 1500 bytes in order to provide a balance between good throughput of single and multiple flows.

The clear takeaway from the data in Figure 3.8a experiments is that multiple flow tasks with different rate limits do not perform well on a single queue. With just two flow tasks sharing the link bandwidth the primary flow underperforms, even though with 1500 byte packets it is possible for a single flow to almost completely saturate the 100 Gbps link. However, I see that the primary flow never takes more bandwidth than it is allotted, which is a desirable invariant.

In order to further understand this issue, I perform the same test with four active queues and between one to four flows per queue. This means there are again at most 16 flows sending at any given time. Bandwidth not assigned to the primary flow is again split evenly across every flow across all queues.

The performance seen in Figure 3.8b is much more promising. I immediately see that while I am still unable to completely fulfill requested rate limits for high amounts of bandwidth, there is not a rapid collapse between the achieved versus requested rate. This argues that the sNIC is currently unable to handle multiple rate limits on a single core when that core is responsible for sending extremely large amounts of traffic, and does far better when some of the link bandwidth is offloaded onto other cores.

Packet pacing

During my rate limiting evaluation, I record the arrival time and size of each batch of packets for each flow at the receiver. I then assume that each packet in a batch was evenly spaced between the arrival time of that batch and the previous batch. While this is an optimistic evaluation of packet arrival times for a flow, it does permit me to coarsely inspect the distribution of inter-packet gaps for each flow without requiring an advanced hardware solution. I ensure that each receiver core only handles one flow.

I again present results for a ”primary” flow that is allocated a fraction of the overall link bandwidth on one core, with the remaining bandwidth distributed across all other flows both on the same and other cores. I plot the arrival times as a cumulative distribution function, with the ideal interpacket gap for the given bandwidth displayed as a vertical dotted line.

My results when sending with only a one flow and one queue at 25Gbps are plotted in Figure 3.9a. This exhibits that up to moderate burst size of 16 packets, the sNIC is capable of pacing packets to a reasonable degree of error, some of which may be attributed to my measurement method.

We've established previously that I need to send with multiple cores and hardware queues in order to permit multiple flows to fulfill requested rate limits. However, when increasing to sending with four separate queues with each sending at 25Gbps, I am now reliant on the NIC hardware to properly stripe sending across each hardware queue, rather than batching groups of packets from each queue together. As shown in Figure 3.9b, my NIC clearly does the latter and the burst size I configure no longer affects the distribution of interpacket gaps. This behavior may play a role as to why multiple queues are better able to fulfill requested rate limits.

3.4.3 Basic TDMA Schedules

My investigation into the performance of TDMA schedules on sNICs is done by sweeping a few different basic parameters in order to gain insight into the core limitations of software NICs. The results here are targeted towards the TDMA specific requirements described in Section 3.3. Because more complex TDMA schedules will have strict requirements in order to work effectively, I work here to identify scenarios where the performance of a software NIC is insufficient to identify future work for both networking software and hardware to satisfy future systems.

I run each test with a number of virtual TDMA hosts on the same two physical machines, with each machine hosting half of the virtual hosts for the experiment. Each virtual host requires two physical CPU cores; one for sending scheduled traffic, and the other for receiving traffic and gathering statistics. I use a third physical core for each virtual host to generate traffic to ensure infinite demand. Each virtual TDMA host only sends a single flow to one other virtual host in a repeating static up/down pattern, as described in Section 3.3.3. I use a fixed up-time preallocation value of 3 microseconds, which I determine based on preliminary results. The burst size of each virtual host is fixed at 8 packets to balance throughput and emission accuracy. I use dummy packets sized at 128 bytes, which are able to mitigate NIC transmission delays while not creating any observable overhead on performance.

Guard time

As discussed in Section 3.3.3, the appropriate guard time value varies due to both software and hardware. Because evaluating the effect of hardware in my case would require multiple 100Gbps NICs to cross reference their minimum appropriate guard time value, I instead simply present a sweep over my experimental setup to evaluate the effect of different guard times on the quantity of packets seen during down-time. I use 8 virtual hosts (4 per physical machine) rate limited to 10Gbps each.

My results are shown on Figure 3.10. While it may initially seem unintuitive that larger packet sizes are lost more often, this is caused by batches of packets transmitting just before the up-time ends, meaning some fraction of the batch will be lost. If the batch takes more time to transmit, more of the overall batch will end up being transmitted during down-time. There is observable variance in the results caused by the hardware and scheduling randomness. I see that a guard time of 4 microseconds is sufficient for preventing packets of any size from arriving during down-times. For the following results I select a value of 5 microseconds in order to provide a reasonable buffer and eliminate any possible variance in the transmission delay of packets that may cause packets to be transmitted during down-times.

Duty cycle

The purpose of a TDMA schedule's duty cycle is important to determine how to appropriately balance the schedule of flows, so it is important that a software NIC be able to support a range of up/down-time durations without sacrificing accuracy or performance. This becomes especially important if dynamic schedules are used, as the duration of the up-time or down-time will vary between shorter and longer values over time.

I fix the duration of the TDMA scheduling period and use a number of different up/down-time length ratios to determine the effect that the duty cycle may have on both achieved rate and packet loss. I compute the packet loss rate across all virtual hosts via both dropped packets via

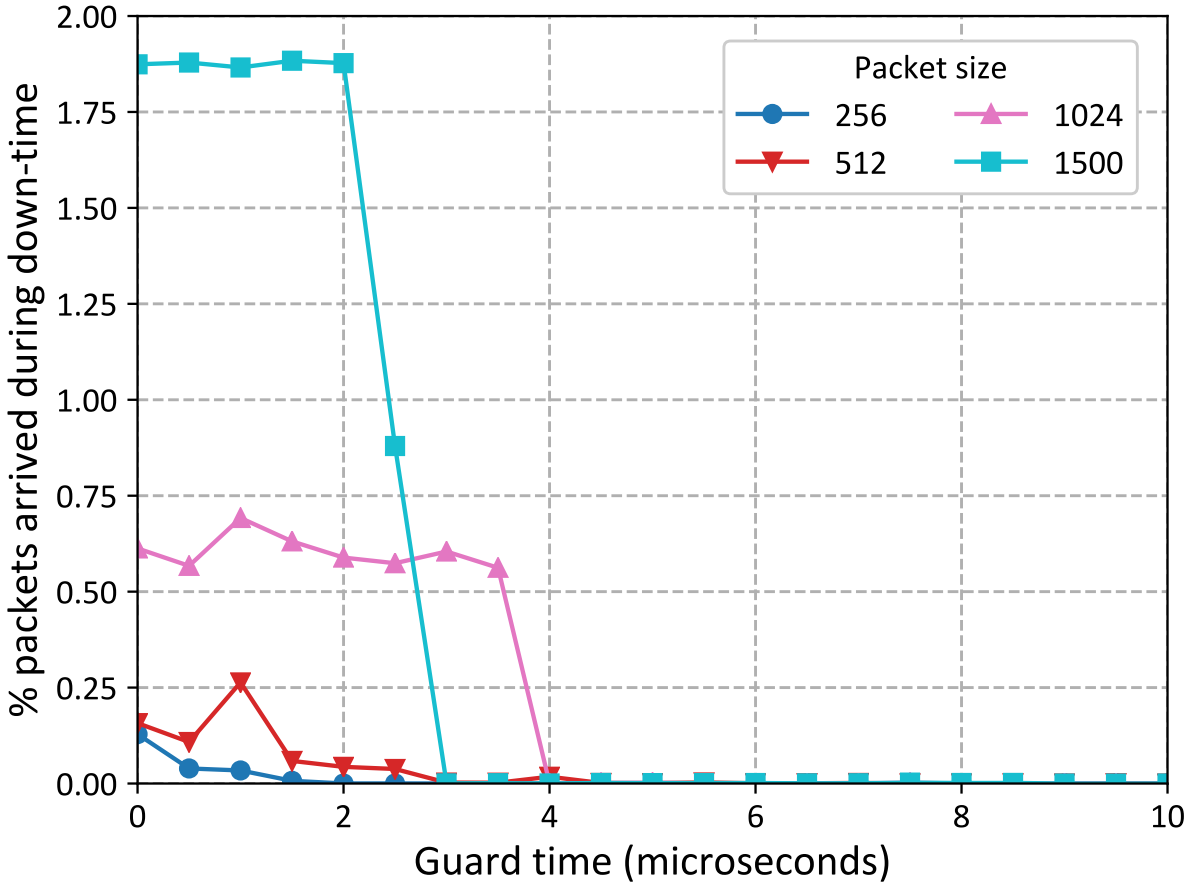


Figure 3.10: Effect of guard time on the amount of packets observed during down-times with 8 virtual hosts each sending at 10Gbps.

sequence numbers and observed down-time packets. I use a period of 500μs and limit each virtual host to 25 Gbps.

I see from Figure 3.11 that with both small up-times and down-times packet loss begins to increase (thus throughput suffers), and shorter down-times have a greater effect than shorter up-times. When the number of virtual hosts combined leverages all the available link bandwidth, a large amount of packet loss occurs and throughput suffers. With shorter down-times, I note that a reasonable fraction of the loss is caused by packets arriving during down-times. Interestingly, having a small number of virtual hosts that leverage a smaller fraction of the link bandwidth avoids most of the performance issues observed with higher speeds and more hosts, although

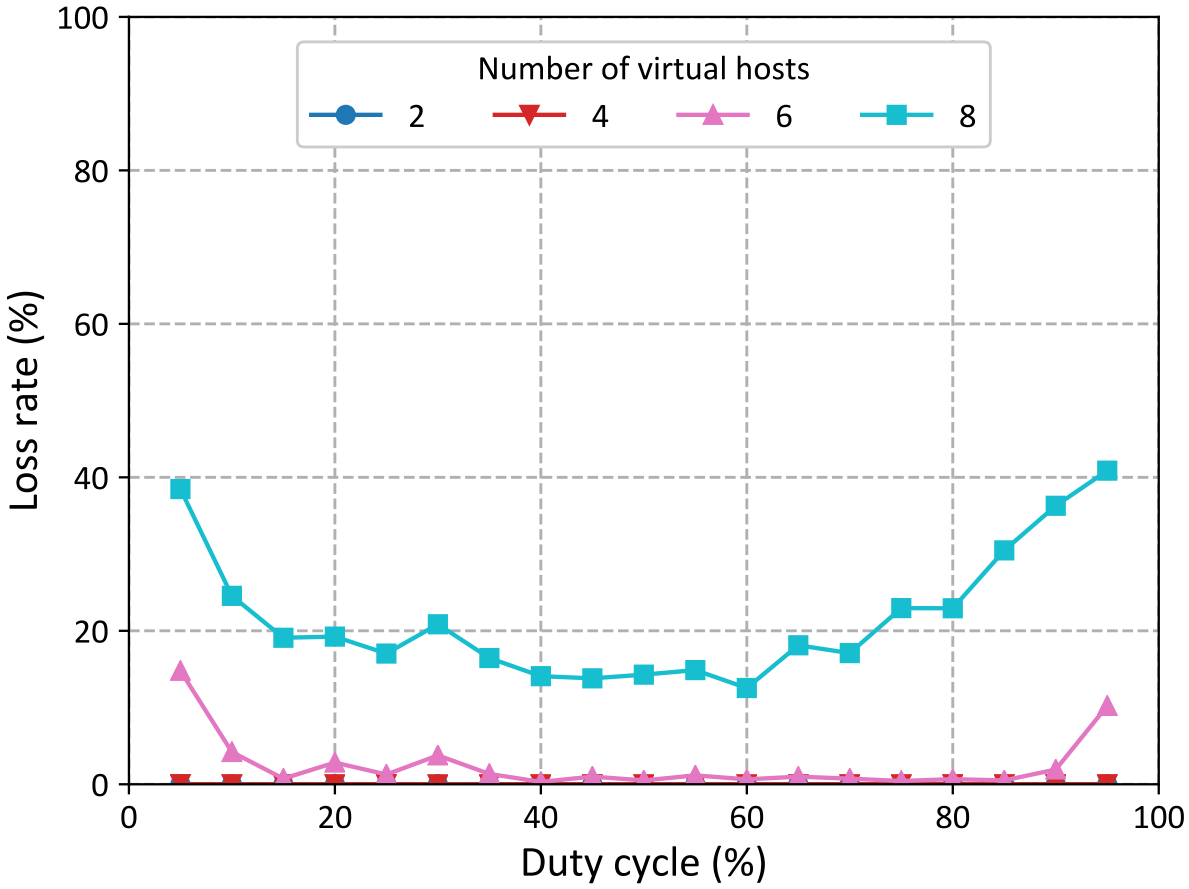


Figure 3.11: TDMA packet loss rate by duty cycle with a period of 500 μs and 25 Gbps per virtual host.

shorter up-times do seem to induce a small penalty.

Scheduling period

I next examine the performance of TDMA when the duty cycle is fixed, but the scheduling period is varied. I briefly mention in Section 3.3.3 that smaller scheduling periods allow new flows to be scheduled sooner, as they have to wait less time for the previous period to complete before they can begin sending traffic. Additionally, longer periods can be desirable in order to avoid down-time weighted duty cycles caused by high reconfiguration delays of physical networking hardware, such as 3D-MEMS optical switches.

I again fix the sending rate of each virtual host to 25Gbps, and use a fixed 50% duty cycle with a varying scheduling period. I compute statistics as in the duty cycle experiments.

In Figure 3.12, I observe similar trends with the number of active virtual hosts from the duty cycle experiment with no significant deviations. I notice that small scheduling periods still seem to cause significant packet loss at the receiver. The variance in performance becomes larger as the period increases at higher bitrates due to the increased duration that the sNIC must saturate the link, which I discover can be difficult in Section 3.4.2.

3.4.4 Multi-Hop Indirection

To test multi-hop indirection, I create a single sending queue with one flow sending 1500 byte packets at various rate limits to understand how forwarding latency is affected by network speed.

While rate limiting performance for a single queue and flow performed reasonably well in my rate limiting experiments, it is expected that any performance degradation caused by rate limiting will affect forwarding latency in any system. Because multi-hop indirection requires sacrificing the forwarding host's outgoing throughput on indirected traffic, it is typically necessary to rate limit forwarded data so that it does not completely consume the outgoing link's available

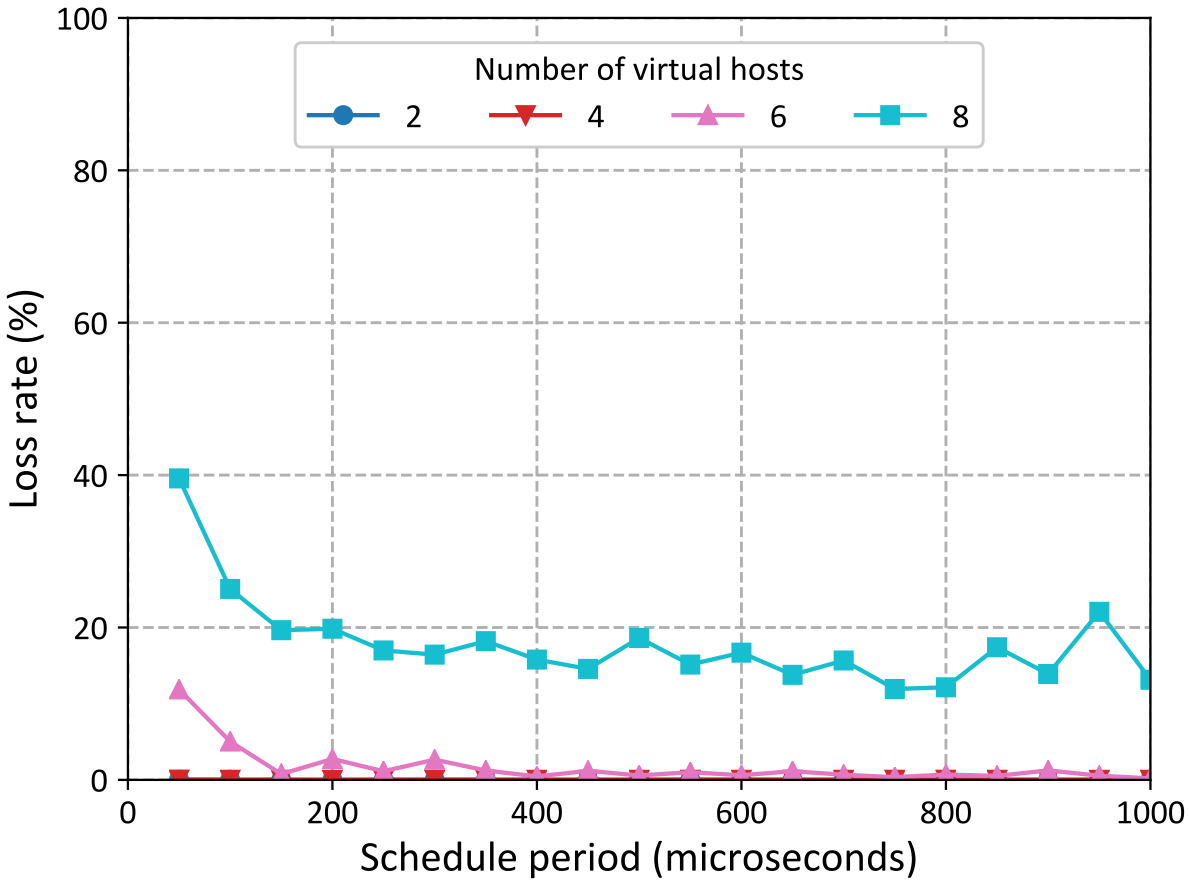


Figure 3.12: TDMA packet loss rate by scheduling period with a duty cycle of 50% and 25 Gbps per virtual host.

bandwidth.

The results of my experiment are in Figure 3.13. The boxes represent the standard quartiles for the results, and the whiskers represent the 0.1 and 99.9 percentiles of observed latency to account for scheduling variance during beginning the test and while collecting results. I record loss information (not shown), and measure roughly 18% packet loss only at speeds of 80Gbps and above, explaining the massive spike in latency at those points.

I see that the forwarding latency does slightly increase for higher speeds before loss occurs, implying that there is some software-associated overhead with forwarding encapsulated packets at high network speeds. I also observe that the forwarding latency at low speeds is actually rather large. I suspect this is due to the hardware NIC delaying packet transmissions until a sufficiently large batch has been enqueued, which explains the large gap between the median and the minimum values.

3.4.5 sNIC Performance Observations

Although these tests are not completely exhaustive, I have gathered enough results that I have gained a basic understanding of the limitations of sNICs for usage in a few different flow control contexts. I do not expect these results to be a statement on whether the idea of software NICs is well formed or invalid, but just an observation of the current state of affairs based on the results I have presented.

The amount of bandwidth that an sNIC can provide to the network is very good for 40 Gbps links: only a few cores are necessary to send at 40 Gbps with minimum size packets. However, scaling to 100 Gbps speeds becomes difficult for smaller packet sizes without using a large number of cores, which is prohibitive for non-experimental usage. I suspect that the microbenchmarks perform roughly equivalent to base DPDK due to the simplicity of the sNIC application used to run them, implying that the core technologies enabling software NICs at all may need further development in order to satisfy future networking demands.

Per-flow rate limiting performance in the sNIC was rather disappointing when using a single core. I note that given the performance of the microbenchmarks that this is primarily due to the large amount of bandwidth sent on a single CPU, especially since splitting the bandwidth across multiple CPUs drastically improved performance.

While packet pacing performance was disappointing, it was not unexpected. It is impossible for current software to control the exact method that the NIC firmware will use to select and send packets from various queues. I am impressed that the sNIC and DPDK managed to pace a single flow fairly well in isolation, given that the hardware NIC could have simply held onto packets until a minimum size threshold. Future hardware could provide a method through DPDK to configure some of the firmware's packet transmission parameters to make software-programmed packet pacing a viable alternative. At the moment, hardware is still a necessary solution to enforce pacing of flows.

The TDMA performance of the sNIC is satisfactory when it is bound within the limits found in the previous results and the parameters of the schedule are within some moderate restrictions. The only serious setback for TDMA is that scheduling periods at or less than 50μs begin to incur serious performance setbacks at higher speeds, which makes multi-hop indirection more essential when using sNICs for TDMA flow control.

Multi-hop indirection proved to have higher latency than I expect in even the best case, though I am unable to determine how much of this is due to software versus hardware. The current performance is sufficient to reduce flow latencies for TDMA based on the required scheduling periods I observe. However, many datacenter networks are beginning to use low flow latency as a primary metric, and the multi-hop forwarding latency of the sNIC is an order of magnitude higher than current expectations for flow latencies in datacenters [ZDM+12].

Additionally, the limited flexibility of the scheduling subsystem discussed in Section 3.3.3 prevents me from seeing sNICs as a perfect solution for future-proof TDMA flow control development. Nonetheless, the module subsystem is easily extensible and provided sufficient

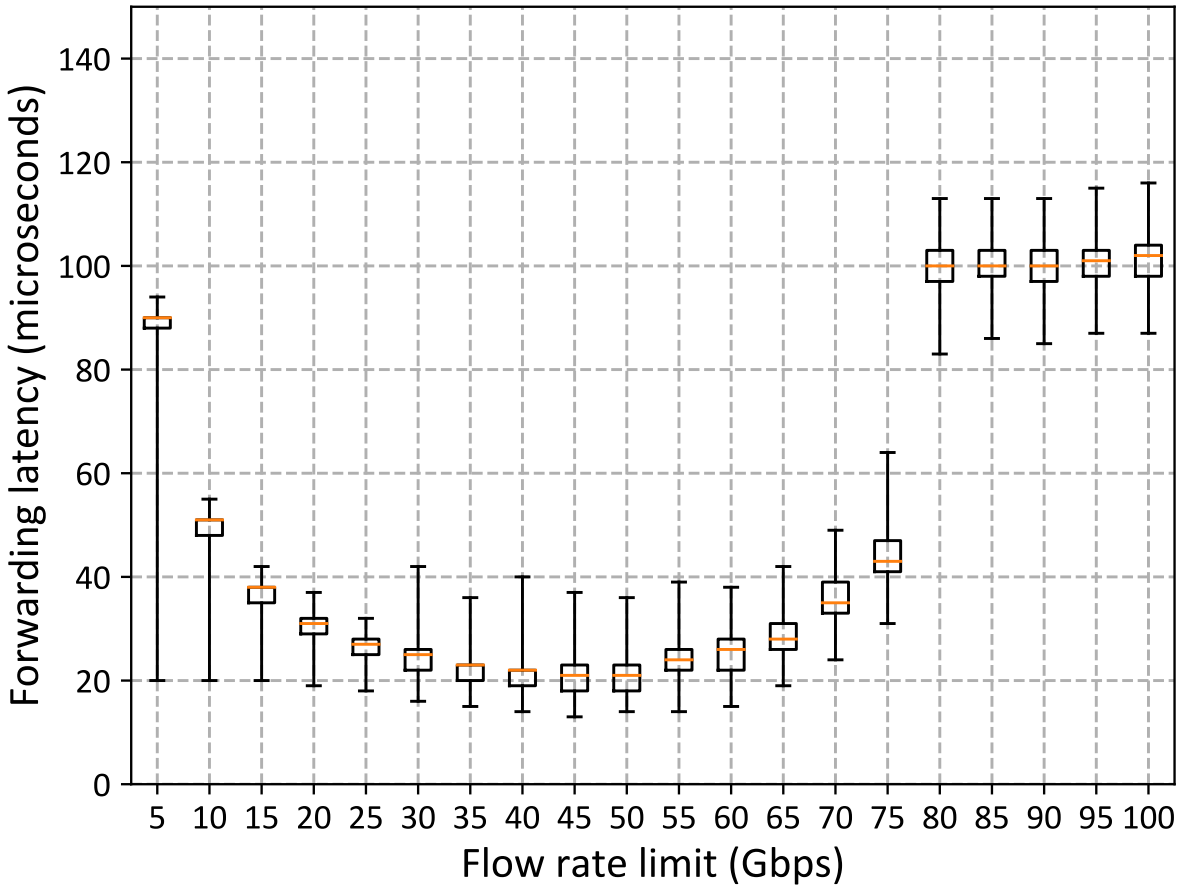


Figure 3.13: Packet forwarding latency using GRE encapsulation with 1500 byte packets and a single flow.

performance with a limited number of cores at 40 Gbps that I can highly recommend that users leverage sNICs in current datacenter networks in endhosts for a range of flow control systems.

3.5 sNIC Conclusions

Software NICs provide a flexible framework to aid to future flow control development, and their features aim to supplant some of the trends towards offloading networking functions onto NIC hardware. Rate limiting, packet pacing, and TDMA flow scheduling, and multi-hop indirection are traditionally difficult flow control problems for software, despite how prevalent

they are in many of today's flow control systems.

BESS, a software NIC developed specifically for endhosts and built on the DPDK software library, provides a modular system that I use to implement and measure the performance of these forms of flow control at 40 and 100 Gbps speeds. While sNICs allow for rapidly evolving applications with satisfactory performance for 40 Gbps networks, there is still work to be done in order for sufficient performance in 100 Gbps environments.

3.5.1 The Future of Circuit-Switched Networks

The support that software NICs and other TDMA implementations need will need to come from hardware rather than software. Throughout this work and the work in Chapter 2, the common factor I encounter is that software is not able to efficiently provide precise, nanosecond level timing information in the variety of situations that are required for flow and traffic control inside a datacenter network. I use a wide variety of possible tools to attempt to support the networking protocols required in circuit switched datacenter networks at high bitrates, but am unable to provide sufficient performance to justify the move away from a packet switched environment to a circuit switched one.

There is still hope for circuit switched networks. Hardware support has become increasingly available through complex programmable endhost NIC hardware [FSPP20, GYBS20], and new NIC hardware aims to more directly support precisely timed packet transmissions. However, it is clear from my work that software support alone cannot create circuit switched networks that achieve a higher rate of energy efficiency at scale over traditional packet-switched datacenters.

Next, I will move into discussing how datacenter networks can aid in extending the lifetimes of smartphones by understanding the current lifetime limitations in an academic environment, and what applications may specifically be offloaded into a datacenter to prevent device obsolescence via software incompatibility.

Chapter 3, in full, is a reprint of the material as it appeared in the Proceedings of the

2018 Symposium on Architectures for Networking and Communications Systems (ANCS '18). Rob McGuinness; George Porter, Association for Computing Machinery, 2018. The dissertation author was the primary investigator and author for this paper.

Chapter 4

Stipulated Smartphones for Students: Using Datacenters to Extend Device Lifetimes

I now move from discussing how the impact of computing on the environment can be reduced via lowering the operational energy of datacenters, to discussing how datacenters can be leveraged to lessen the embodied energy smartphones have on the environment. In particular, I look at the context of smartphones within an academic setting, as students are typically consumers with less purchasing power. This means that students are poised to want to keep their smartphone for longer.

Examining how long students can and do keep using their smartphones aids me in understanding the current limitations of smartphone lifetimes, and how datacenters can best be leveraged to extend them. I look at a set of student user data obtained by my campus and understand how this data reflects what future steps need to be taken. I then discuss solutions on how datacenters can make phones function for longer on the academic tools that are nowadays

often required for college coursework.

4.1 Introduction to Smartphones in Academia

Computing has become an invariable and essential part of everyday life. Individuals are now required to use digital devices in order to carry out daily tasks in both personal and professional settings. Supporting this change is a rapid and constant evolution in hardware and software capabilities. This is evident in the purchase rate of new smartphones, with users replacing them roughly every 20 months on average [Pan]. Users may replace their device for a large number of reasons, such as wanting higher resolution screens, better cameras, or better software performance.

This inevitable "march of progress" can result in newer, widely used software that that is cumbersome or impossible to use on older devices. Users that require this software for their work, or students requiring this software for their education may find that their older devices no longer work correctly and no longer receive necessary software updates. They may be able to compromise by relying on a degraded interface to these software services (e.g. via a browser instead of a dedicated app). However, as I will show later in this chapter, even browser interfaces often become unusable, forcing users to upgrade their devices to continue accessing the modern web. Those upgraded devices comes with numerous costs, both to users in the form of monetary cost and the world at large in the form of pollution, eWaste, and increased carbon output.

In this chapter, I examine these costs with respect to undergraduate college students, a sector of the population that now is required to use technology to receive an education in the wake of the COVID-19 pandemic. I then discuss how datacenter networks can reduce these costs using future novel systems ideas. Even before to the pandemic, learning management systems (LMS) such as Blackboard and Canvas have garnered widespread use [Bla18, Ins21], forcing students to use internet browsers or mobile apps to access and interact with coursework.

I look at three different types of costs that occur in relation to these educational users:

Monetary Costs: With smartphone costs doubling between 2014 to 2018 [Suc19], and continuing to rise, I note that students are disproportionately affected by the cost of frequently purchasing a new device. A majority of students now receive some form of financial aid to afford a college education [MSWH16], and some percentage of students additionally require aid for basic needs like food and shelter. Students that are required to purchase new devices to continue to access basic education materials may have to make compromises against their own personal well-being in order to fulfill class requirements. I argue that a student should never have to make this compromise in order to receive an education, especially since at a component level, their "older" devices are likely released only a few years ago.

eWaste Costs: The other significant cost is to the world at large in the form of the eWaste generated from discarding old devices. In 2016, the world produced over 44 million metric tons of eWaste, with computers (laptops, desktops, smartphones, and tablets) accounting for about a quarter of that total [BFG+17]. The vast majority of eWaste is not properly discarded or recycled [BS+17], which results in long-term damaging effects to the environment.

Manufacturing Costs: Additionally, the vast majority of environmental damage from computers and smartphones comes in their manufacturing process [BS+17]. Manufacturing incurs enormous environmental damage, involving mining for minerals, generating greenhouse gasses during transportation and assembly, and processing elements like cobalt, lithium, and mercury. At the same time, once a computer goes into service, it is quite efficient, due to lower power components, flash storage instead of spinning hard drives, and more efficient battery technologies, among other reasons.

Taken together, it is clear that upgrading devices unnecessarily has huge personal and environmental effects on students. And yet because the web-hosted services that students rely on

to complete their studies become increasingly complex over time, simply keeping older devices in use for longer is often no longer a feasible option. In this chapter I take a critical look at this issue.

My research in this chapter focuses on the question: **In the worst case, how long can a user realistically continue to use their device to access online educational resources?** I examine multiple sources of data to answer this question. I both independently test websites with a range of browser software and device hardware, and also perform analysis of real user data obtained from the authors' university, UC San Diego.

As a motivating use case, I look at websites used by an undergraduate computer science student. Using an online platform for browser testing, I survey a sample set of educational websites they might access across a range of legacy operating systems and browsers, dating back to 2012. I find that after about four years, the educational websites become increasingly inaccessible to older devices and software. I carry this information forward into my investigation of user data.

My user data investigation is done via several years of access logs to a web-based learning management system, Blackboard, used by classes at the authors' college campus. I study the average age of browsers and devices used by students across each year, and study the upgrade pattern of devices to understand how old the software and hardware is "in the wild". I find a reinforcement of the "four year" lifetime implied by my website survey, seeing that there are extremely few cases where a user operates with software or hardware outside of this timespan.

Because a traditional undergraduate degree at a university takes four years to complete (and often more than four years), this implies that a student may be forced to purchase a new device within their career as a student at a university in order to complete their degree. However, I argue that a student should not be required to make this purchase, as their old hardware should be more than suitable.

I acknowledge that there are a number of both technical and non-technical reasons why

people dispose of their computers to get new ones. However, performance and functionality are common concerns [BS+17]. In reality, "obsolete" devices are just as capable today as when they were brand new. There is a growing market of "refurbished devices" showing that users are willing to purchase older hardware, provided it still functions well [PCPS18]. I posit that this obsolescence is a function of the evolution of modern apps, websites, and web-based services that have grown increasingly more sophisticated and resource-hungry over time. Since most applications that are used on a day-to-day basis require some Internet-enabled functionality, the evolution of web-based services render devices prematurely obsolete. In an era of long-distance learning, it is of particular importance to increase device longevity for students who now require computers to complete even baseline academic tasks like attending class.

4.2 Background: The Use and Cost of Smartphones

4.2.1 The Ubiquity of Computing in Teaching and Learning

Academic platforms are commonplace as a support framework for presenting coursework and managing student submissions for assignments. An academic-focused learning management system (LMS), Blackboard, reported more than 100 million users in 2017 [Bla18]. A recently adopted LMS, Canvas, reported more than 30 million users as of 2019 [Ins21]. Students are expected to use these systems to access and complete coursework, and require computing devices in order to do so. Commonly, these are accessed using internet browsers or dedicated mobile application software.

Additionally, the COVID-19 pandemic has had an extreme effect on the role of technology in teaching by requiring that classes begin to be taught remotely over video conferencing platforms. Instructors need to use online teaching methods for courses [MGS20], which further reinforces the obligation for a student to own a device capable of accessing online learning platforms.

This trend is not particularly surprising when reflecting on the growth of technology

across the world as a whole. The proliferation of mobile networks has meant that over 80% of the world's population is now covered by a mobile broadband signal [BFG+17]. The number of users has grown dramatically as well, with over 4.2 billion mobile broadband subscriptions active as of 2017 [BFG+17], over 50% of the world global population.

4.2.2 The cost and life cycle of a computer

In order to access the aforementioned online learning platforms, a suitable computing device must be available for each student. I must examine the costs of a computing device to provide context the effects they have on students and eWaste generation.

There are variety of costs observed when examining the lifetime of a computing device. These include monetary costs to the users and the environmental costs of manufacturing and discarding the device. Each of these costs can have disproportionate effects in different ways. Extending the life cycles of devices will offset these costs.

Monetary costs

Purchasing a computer or smartphone is a non-trivial cost to a user. While users have varying upgrade rates depending on a large range of factors, users with less financial assets are more likely to be unable to upgrade their devices as frequently. With the cost of smartphones rising quickly in the past decade [Suc19], it is important to consider how students who operate on less funding can access technologies required to complete their coursework.

It is an unfortunate reality that many students require financial assistance. At my university campus, UC San Diego, a report [MSWH16] states that over 60% of students require some form of financial aid. Additionally, the on-campus food pantry provided food to thousands of students in a single quarter, with that number expected to climb in the following years. A separate report on community colleges [GRRH17] reported that two in three students were food insecure, and 13 to 14 percent of students were homeless.

With this in mind, I believe that the monetary costs for students to provide their own devices to access educational opportunities must be reasonable. Unfortunately, no programs to widely provide remote access technologies existed at the time of the shift to remote learning due to the COVID-19 pandemic at my campus, which is a relatively large institution (over 40,000 enrolled students in the 2020 academic year [Uni21]). Smaller and less funded institutions struggled disproportionately during the pandemic [MGS20].

Environmental costs

The environmental cost of a device comes in the form of eWaste. eWaste is defined as any device with a plug, electrical cord, or battery that is no longer used and thus has reached the end of its useful life [BS$^+$17, UIU19]. eWaste is divided into six categories: temperature exchange equipment, screens and monitors, lamps, large equipment, small equipment, and small IT and telecommunications equipment. In this chapter, I will use the term *devices* or *computers* to refer to the two categories of "screens and monitors" as well as "small IT and telecommunications equipment", which includes laptops, desktops, smartphones, and tablets. In 2016, the world produced 44.7 million metric tons of eWaste [UIU19], with screens and computers accounting for about a quarter of that total volume. And while eWaste only accounts for about 2% of the total waste volume in landfills, it represents about 70% of the volume of hazardous waste that makes its way back into the ecosystem [UIU19]. This waste is rich in precious, heavy, and rare-earth materials, with computers often consisting of up to 60 different elements from the periodic table [UIU19].

To understand the environmental impact of eWaste, it is important to understand the entire lifecycle of modern computers, which can be broken down into four phases [BS$^+$17]: Manufacturing, use, eWaste generation, and eWaste disposal. Each of these phases has a different environmental impact:

Phase 1: Manufacturing. During manufacturing, materials are brought together to create integrated circuits, flash memory, screens, and other components. This process is very resource intensive, relying on a number of materials including gold, silver, copper, platinum, and aluminum, and heavy metals such as mercury, cobalt, iridium, cadmium, lead, and lithium. Mining these materials results in significant environmental damage, and transporting components from their origins to be integrated and delivered to their ultimate destinations incurs a significant carbon footprint as well.

Phase 2: Use. In this phase, the device is put into use, either by the primary owner, or subsequent owners in secondary markets. Here the environmental impact of the device is primarily due to its energy demands (e.g. to recharge its internal batteries).

Phase 3: eWaste generation. At the end of a device's usable life, it is no longer used and becomes eWaste. Unfortunately, as I will highlight below, the replacement cycle for devices has become more rapid over time. Users report replacing devices to keep current with the most advanced models, obtaining new manufacturer warranties, and supporting increasingly complex and resource-demanding applications and apps [BFG+17]. Manufacturers have encouraged short replacement lifecycles through planned obsolescence, subsidized replacement programs, and making it difficult or impossible to maintain and repair devices past their planned "end of life" and warranty period [BFG+17].

Phase 4: eWaste disposal. There are two ways to dispose of eWaste. The first approach includes official eWaste recycling programs which safely recycle and reclaim materials before disposing of the devices. This method is preferred and has a negative environmental impact. The second method includes sending devices to landfills, incinerators, unregulated reuse and reclamation channels, and other untracked disposal methods. This latter case greatly impacts the environment as the aforementioned materials used in manufacturing cause damage when

eWaste is removed in this manner. Unfortunately, only 20% of eWaste is properly recycled, and the remaining 80% ends up in other channels [BS+17].

Is recycling the answer?

Recycling, by itself, is not able to fully address the scale and scope of these challenges. As mentioned above, only about 20% of eWaste is currently recycled. But even increasing that ratio to 100% would not solve the problem for two primary reasons. First, the internal components in modern computers are increasingly monolithic. For example, modern CPUs typically include not only compute cores, but external graphics support and GPUs, and even networking and wireless LAN support (e.g. the Atom x3). It simply isn't possible to recover the underlying elements from these chips and devices, in the same way that you can't unbake a cake to recover its underlying flour and sugar.

Even if components could be recovered, recycling by itself is not sufficient to address these problems. Research has shown that most of the impact of creating computing devices resides in the Phase 1 manufacturing step. Bakker et al. performed a lifecycle assessment (LCA) that evaluated laptops both in 1990 and in 2010, assuming they are used for only one year. They found that the total environmental impact of the manufacturing step rose from 68% in 1990 to 78% in 2010. However, the impact of the per-year use of the device fell from 31% in 1990 to 21% in 2010. Lastly, transportation accounted for a negligible percentage [BS+17]. Despite improvements to laptops in other respects, Kasulatis et al. found that between 1999 to 2008, the impact of the manufacturing phase of laptops did not decrease at all [KBK+15].

Further, a metastudy by Suckling and Lee found that, similar to laptops, the manufacturing phase of smartphones represents the majority of the device's environmental impact [SL15]. In fact, they found that for a set of smartphones manufactured after 2010, the manufacturing phase accounted for an average of almost 75% of the total impact, with the use phase making up the majority of the remaining 25%.

How long should devices last?

In recent years, use periods of devices have decreased, raising the impact of device manufacturing and disposal on the environment. Bakker et al. found that the Phase 2 (Use) period of laptops decreased from 4.3 years in 2000 to 4.1 years in 2005 [BS+17]. A study in 2016 [Pan] found that the lifespan of smartphones varied from under 18 months to just under 2 years.

Given the environmental impacts described above, Bakker et al. suggest that the optimal replacement point of a laptop is at least after seven years after manufacturing, based on the increased operational efficiency of laptops during their use [BS+17]. Further, Suckling and Lee suggest that smartphones need to be usable for approximately 5 years before impacts are amortized [SL15].

This information shows that devices are used, on average, for approximately half as long as they would need to be in order to offset the costs of manufacturing and eWaste disposal have on the environment. Unfortunately, as I discuss below, there are significant challenges that prevent users from simply using their devices for longer periods of time.

Extending a computer's life

Given the oversized role that cost and manufacturing plays in a computer's impact, a key to reducing that impact is extending the usable life of computing devices. This idea is referred to as a *Circular Economy* and aims to keep devices in circulation to avoid them become eWaste and to reduce the ownership burden of the device across its lifespan. This concept is also called *Product Lifetime Extension (PLE)*. Lifetime extension is sustainable, economical, good for the environment, and helps to address the "technology gap". As shown above, aiming to double the usable life of computing equipment would contribute greatly to solving economic and environmental challenges. Surveys of consumers find that they do wish that their devices did last longer, and are unsatisfied with their typical short lifespan [Ech16, Coo04, WTH15]. If there is a way to extend the usable lifespan of computers, it will address the eWaste problem while

reducing the monetary and environmental cost of computing devices.

Using datacenter networks, which are an increasingly available source of high amounts of computing power, computing work can be offloaded. Cloud computing services now support a significant amount of computing infrastructure [LWN+17], and are run on datacenter networks. Using similar methods to offload work from smartphones and laptops "into the cloud" could extend their lifetime. I will discuss this at length later in this chapter.

Why do devices get outdated? An inevitable scenario that forces a user to discard their older hardware is when the device becomes unable to readily access online resources. In this chapter I focus on web browsers, given their importance, especially for students. Internet browsers are an extremely elaborate and rapidly-evolving software domain. Major browsers release updates regularly in order to improve security and performance for their users. Updates also include additional features that websites can use to present and render more complex content. This can come at the cost of removing compatibility for older browser versions. Since many websites deprecate support for older browsers, users have needed to continuously upgrade their devices to stay apace with modern websites.

Further, mobile devices can become outdated at a hardware level if manufacturer of the device chooses to stop supporting it. Qualcomm officially supports updates for their mobile chipsets for only three years [Ama21]. Apple does not support iPhones more than five years out of date [App21]. The official "end of life" declaration from a smartphone manufacturer effectively eliminates any further usefulness it has for a typical consumer, who will not or cannot go out of their way to install alternative system software to extend the phone's usefulness.

Does Moore's Law help? The ending of Moore's law in the early 2000s fundamentally changed computing, and at first glance seems like it might make computers usable for longer, since CPU frequencies no longer increase at the rate they did prior. Unfortunately, available evidence shows the opposite, with the usable life of laptops decreasing from 2000 to 2005 [BS+17]. Further,

although CPU frequencies have not continued to increase, other resources such as memory, the number of CPU cores, flash storage capacity, and the prevalence of GPU units have provided computers with increased capabilities. Further, newer operating systems and device hardware offers new security primitives.

What about upgrades? What about repairing, upgrading, and extending computing devices in the field? In other arenas, repair is common, such as replacing a car's flat tire with a new tire instead of buying an entirely new car. Unfortunately, several trends in computing make repair and upgrade more infeasible. As previously mentioned, laptops, smartphones, and tablet computers rely on increasingly monolithic integrated subsystems and "systems on chip (SoC)" designs, which combine compute, graphics, storage, and even networking into a single chip that can only be replaced, not repaired. This is in contrast to pre-2000 era desktops, which consisted of a number of discrete components like sound cards, RAM modules, network cards, etc., which could all be independently upgraded or replaced. But even then, machines were often replaced rather than piecemeal upgraded over time. For these reasons, I largely rule out upgrading computing devices' hardware directly.

What about repairs? As mentioned above with respect to automobile tires, repairing damaged or defective components of very expensive equipment is commonplace. But for computers, it is unlikely that repair by itself will significantly address the needs I outline, since there is little evidence that durability is a significant factor in replacement decisions. The above-referenced studies show that the majority of devices are replaced either for non-technical reasons or to obtain new features/capabilities. When devices are eventually replaced, they are typically just as performant as the day they were manufactured. Further, as computers become more integrated, several sources of device failure are simply removed. For example, flash storage has largely replaced spinning hard drives, batteries are less likely to leak, and some laptops no longer rely on spinning fans for heat management. Smartphone screen repair remains popular, but nearly any

other damage to a smartphone requires replacing the entire device.

4.2.3 Background Summary

The increasing sophistication and resource requirements of modern websites, especially in the education technology space, render consumer computing devices prematurely obsolete, resulting in significant contributions to the eWaste problem and to financial burdens for students. If there is a way to extend the usable life of these devices, it will reduce their financial and environmental impact.

4.3 Browser Obsolescence

To understand the ability of older devices to use modern websites, I now study their compatibility to various "eras" of hardware and software, with configurations representative of a given year. Each configuration uses a device, operating system, and browser version that was released in the appropriate year I wish to investigate. Comparison between years reveals the trends of obsolescence for different devices over time. To focus my study, I target representative websites used by students in an undergraduate computer science program.

The websites I consider are Google Drive, Canvas (a learning management system), StackOverflow, Jupyter Notebook, and Piazza (a message board), which are all platforms frequently used by computer science undergraduate students. I wished to test against Blackboard, the learning management system I analyze data from in Section 4.4, but it unfortunately has been decommissioned at the authors' campus.

I leverage the BrowserStack[Bro11] online browser sandboxing platform to access each of these websites. BrowserStack provides a wide selection of smartphone hardware and desktop/laptop browser versions to the user dating back to more than eight years. Smartphone browsers are not emulated, and are run on live devices. The client is required to select the specific device

to use. I bin configurations into "eras" in time. Each era is represented by one of five fixed years, from 2012 to 2020, where I select device, operating system, and browser versions representative of what was up to date at the time. For each test, I complete a basic task on the website in question. I then determine the quality of the user experience, assigning it one of three outcomes: good (green), okay (yellow), and unacceptable (red). *Good* outcomes represent the website working as intended, *Okay* indicates minor issues such as incorrect CSS rendering or unsupported version error messages, and *Unacceptable* represents the website being unusable, due to intentional deprecation, HTTPS issues, or JavaScript errors.

4.3.1 Mobile Browsers

I evaluate both Android and iOS devices on BrowserStack, omitting apps for this study, though I acknowledge that apps do have a large user base (the Google Play store reports that Google Drive has 5 billion installations[Goo20], and my results in Section 4.4 show many app users as well). The mobile results are shown in Figure 4.1.

I see that there are definite limitations placed on users who are using older smartphones. Devices from the 2012 era are unusable in almost all cases. In these cases, HTTPS errors occur on StackOverflow, and the other two websites fail to render. Perhaps the most shocking result is that an iPhone 6 and iPhone 7 cannot access Canvas on nearly up-to-date browsers: the main window is non-functional. Older Android devices receive much better compatibility, but I do begin to see some issues with Canvas.

4.3.2 Desktop Browsers

The results for my study of desktop browsers are shown in Figure 4.2. These results solely look at the incompatibility of browser versions, which helps understand what failure modes occur in browsers.

My results reveal a very apparent trend that older browser software often has difficulties

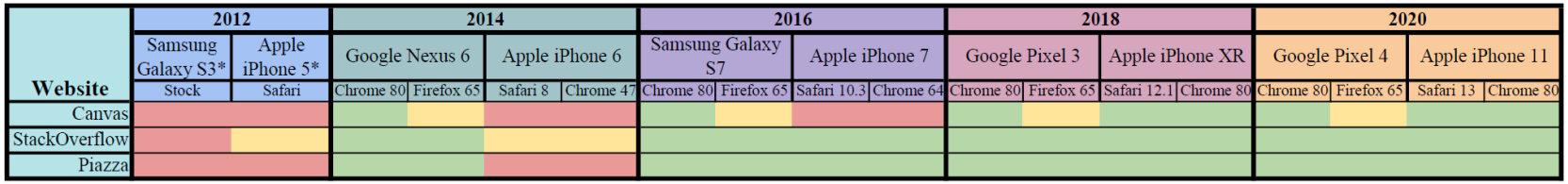


Figure 4.1: Mobile device/browser compatibility results via BrowserStack. '*' marks devices run via emulation. Browser versions used are the newest available for the given mobile OS.

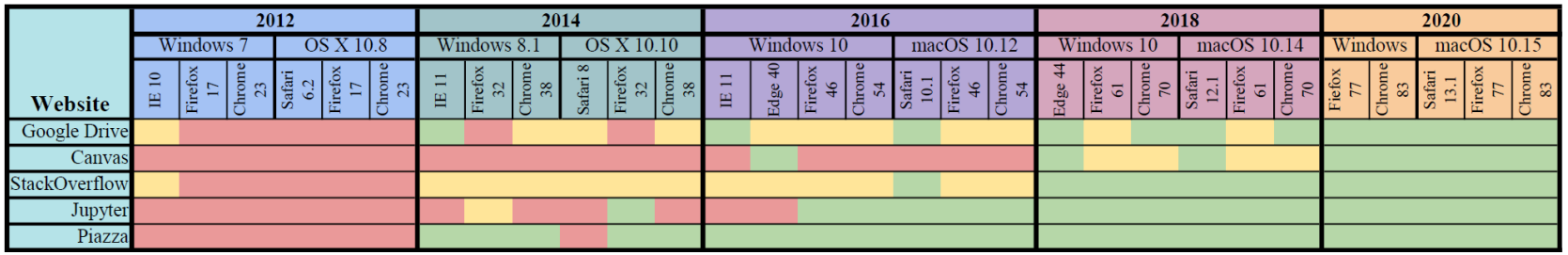


Figure 4.2: Emulated desktop browser compatibility results via BrowserStack.

supporting these websites, with 2012-era systems almost always failing. Even 2016-era software have significant issues, which is alarming given they are only four years outdated. Except for one case, every yellow cell is caused by either a CSS rendering error or a notification being given to the user that their browser is no longer supported. Google Drive and StackOverflow have frequent CSS issues that do not render the site unusable, but clearly impact the rendered appearance. Canvas gives an unsupported browser notification on every page that temporarily covers page elements for versions it deems are outdated. For 2012-era browsers, half of the errors are due to HTTPS negotiation failures; the browser software does not support the newer encryption schemes used by the site. For 2014-era and 2016-era browsers, cases of various JavaScript or rendering errors make websites unusable. These browsers are essentially inoperative across the spectrum of requirements a student may have if they wished to use these sets of websites in a class.

Browser version usage

It's clear from the above results that there are issues with using older browser versions on modern websites. However, it is unclear whether a user would be able to resolve these issues without needing to upgrade their hardware or needing an edge-cloud solution. One point of insight that I have been able to gather was to examine the browser version usage statistics gathered by StatCounter[Sta]. I analyze usage data for the month of May 2020, and found that a non-trivial 8.6% percent of users are using browser versions released on or before 2016. I compare against this baseline when studying browser usage data of of real students in the following section.

4.4 Student Browsers and Devices

Given my baseline of how older browsers and the devices that run them operate on websites used for coursework, I next move to analysis of real user data. I analyze seven years worth of access logs to Blackboard, a learning management system (LMS) commonly used on

the authors' university campus for both undergraduate and graduate courses. An LMS provides functionality for both instructors and students to post and review course material, submit and grade assignments, and more. Instructors that choose to leverage an LMS require students to use a device in order to access material and assignments hosted there.

Log frequency of the LMS is non-uniform due to a varying degree of usage between these dates, beginning in the fall of 2013 and ending in the spring of 2020. However, I am still able to observe some definite trends that reveal the limitations of software and the devices that run them among the student population. Teaching and administration staff are included in my dataset, but are not the vast majority of users. My dataset consists of over 165 million requests to the LMS web server. The use of this dataset was approved by the authors' campus for this study.

Every user in my dataset is first anonymized with a unique identifier. The information I base my study on are this identifier, access time, event type, and the user agent string given to the web server. Beginning in 2015 on iOS devices, and 2017 on Android devices, I begin to see user agent strings for the Blackboard mobile app, which includes a unique device identification string for a single app installation, and a more specific device model identifier. This allows for wider analysis of device upgrade rates among users.

For my purposes, I filter event types in my dataset to only include successful login and logout events to capture the range of usage of a device, and to filter out devices attempting to use the LMS unsuccessfully. Because my dataset is based around user agent strings provided to a web server, I cannot claim 100% accuracy as users may modify their user agent however they see fit (the authors have directly observed a nonzero but negligible number of user agents that were modified by a user).

All data is processed by user agent parsing software, and then inspected by a variety of assertions manually crafted by the authors. Assertion failures trigger a fallback routine that sends the user agent to a large-scale subscription user agent parsing service in order to provide verification of results. Mobile app agent strings are manually parsed using a key-value scheme

given as part of the agent string.

4.4.1 Browser usage

The most obvious and accurate information I can obtain from browser user agent strings is the versions of browsers themselves. Using the access date of the user and recording the browser and operating system information provided, I are able to obtain how dated a given access to the browser is for a given record. To more accurately survey my dataset, I group accesses by user and browser identifiers, and count only the latest entry from that user/browser pair. This means that I record the point where the browser was at its "oldest" and still used by the client to access the LMS.

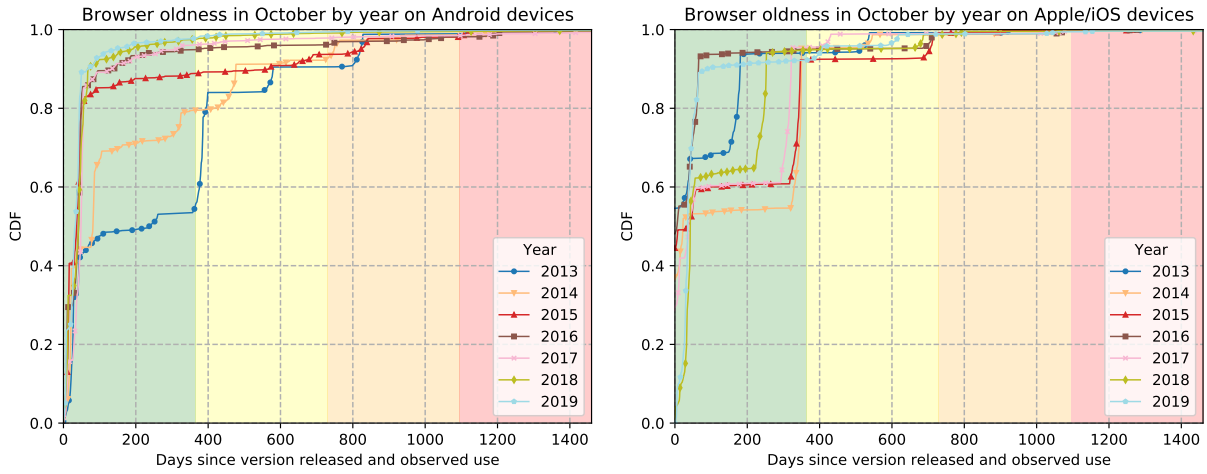


Figure 4.3: Browser age by year for Android (left) and iOS (right). Background colors denote yearly boundaries.

I create an "oldness" metric representing the number of days between when the version string of the browser given was released and when it was used by the client. I record this "oldness" metric over the period of each month in my dataset, i.e. each record indicates the oldest point that a browser was used by a given client within that one-month period. I plot this metric for a single month, October, over the different years of data that I have available. I select October as it is near the beginning of the academic year, and patterns across subsequent months within an academic

year were similar.

The oldness data of browsers used to access the LMS is given in Figure 4.3. I observe that effectively nearly 99% of all users access the LMS with a browser that has been released within a four year window. Only mobile results are shown, but the results for desktop devices (Windows and Mac) are similar.

There is no observable trend of browser age by year, except on Android, where there is a clear trend of browser versions becoming newer over time. I do not investigate why this is the case, but recognize that there is an obvious push for Android users to run newer software over time that has proven successful.

From this data I can conclude that nearly all users in most cases tend to use a browser that was released well within the past year. However, there is a non-trivial number of users that wish to run older software on their devices. Despite this, they still all typically fall within a four-year window, a trend that follows from my previous section.

Comparing this to the browser data I study in Section 4.3.2, there is a significantly lower percentage of students in my dataset that use a browser version four or more years out of date than on the internet at large. This may imply that the LMS system has upgrade requirements more aggressive than is typical for other web-based applications.

4.4.2 Device oldness

My dataset also provides significant insight into the trends of smartphone and tablet models used by students to access the LMS. Android devices frequently provided device information directly present in the user age or had characteristics unique to an individual model. Additionally, agents given from the the LMS mobile app provided device model information as well as a unique "device identifier" generated at the time of app installation. This identifier provided additional insight as to when a user changed which device they leveraged to access the LMS.

To support this data, I used multiple services to acquire and store device release dates,

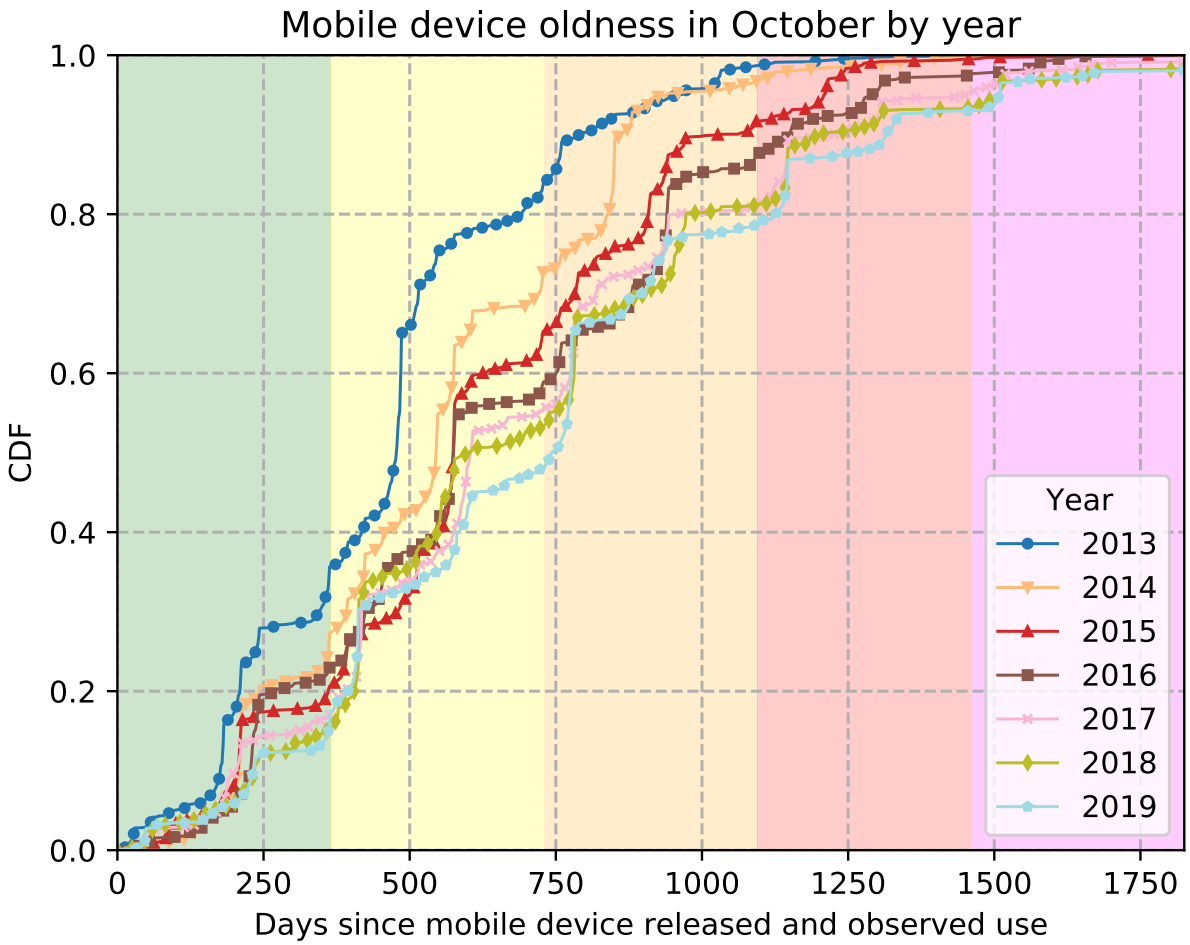


Figure 4.4: Device age at time of use across various years.

brands, and full model names. Device naming schemes ultimately proved to be extremely ephemeral and volatile, particularly for less popular devices such as low-cost and rebranded aftermarket devices. Manual inspection, human data entry, and secondary/tertiary verification of the dataset proved invaluable to solving these challenges, but the authors still cannot claim 100% accuracy for every device present in the the LMS dataset.

I repeat the "oldness" metric where I track the last time the device was used by a client within a month's time. The results are shown in Figure 4.4. I use a similar presentation as in the browser data shown previously.

I once again see that the majority of devices used were released within the past four years

from when it was used. There is a more significant tail of devices (¡5%) from users that have smartphones up to five to six years years old. The devices in the long tail shown are both from Android and Apple users; there is no clear distinction between device family for the long tail of users.

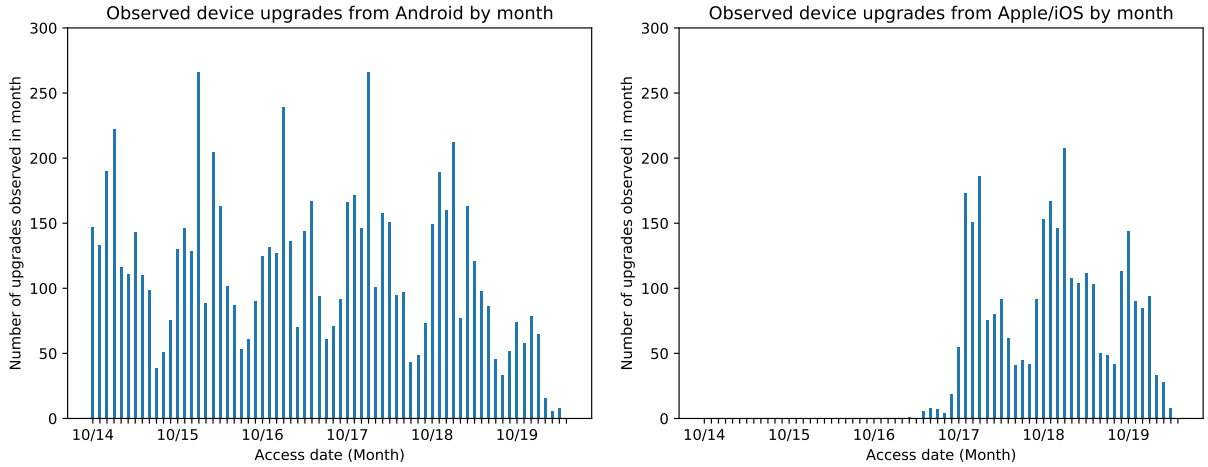


Figure 4.5: Device upgrades by month over time for Android (left) and iOS (right). iOS is limited to app installs, which began in 2017.

There is an observable trend that devices seem to grow slightly older over time. However, I posit that this is a result of the number of devices used on the LMS also growing over time, as more and more students became required to use it in order to interact with coursework. The number of entries in my dataset between 2013 and 2017 grows by approximately 1.6x.

4.4.3 Device upgrades

The last statistic I gather from my dataset is understanding how users upgrade their devices. Because these devices are used to access an essential academic service, they represent a case where a user is *required* to acquire a newer device. Of course, users may (and as shown, do) upgrade their devices well before they are outdated.

The LMS user data I study provides a significant amount of information, but does not give a direct signal for when a user upgrades their device. A user may simply use multiple devices, or

get a tablet or other device they use "on the side" of their primary smartphone. I create a fixed set of criteria I use when detecting an upgrade. In order for a device to count as an "upgrade" for a user, the following criteria must apply:

- The new device was released within the past year, *or* was newer than the old device by at least two years.
- The new device was released at least six months after the release date of the old device.
- The new device was first used within 90 days of the time the old device was last used.
- The old device stopped being used within 90 days from the first time the new device was used.
- The new device was last used at least a month after the old device stopped being used.
- To remove duplicate upgrade events (e.g. a user purchases two new devices), additional devices added or removed within two months of the first upgrade event are not counted.

I do not use a rolling monthly window as in the previous two subsections, but instead use the first time the new device is seen as the "time" of the upgrade event. Each user's entire set of devices is gathered and computed against the above requirements in order to detect eligible upgrade events. Device identifiers are used where possible, and my duplication filtering prevents counting two separate devices in the case where a user both installs the mobile app and accessed the LMS via a traditional web browser.

The number of upgrade events detected is shown in Figure 4.5. Each graph shows the number of times an Android or Apple device was discarded in the monthly window. I break down upgrade events into separate Apple and Android graphs, as Apple devices hide their model identifier in web browser user agents, whereas the mobile app provides an exact identifier of the device. Therefore identifiers for Apple devices begin only in the summer of 2017.

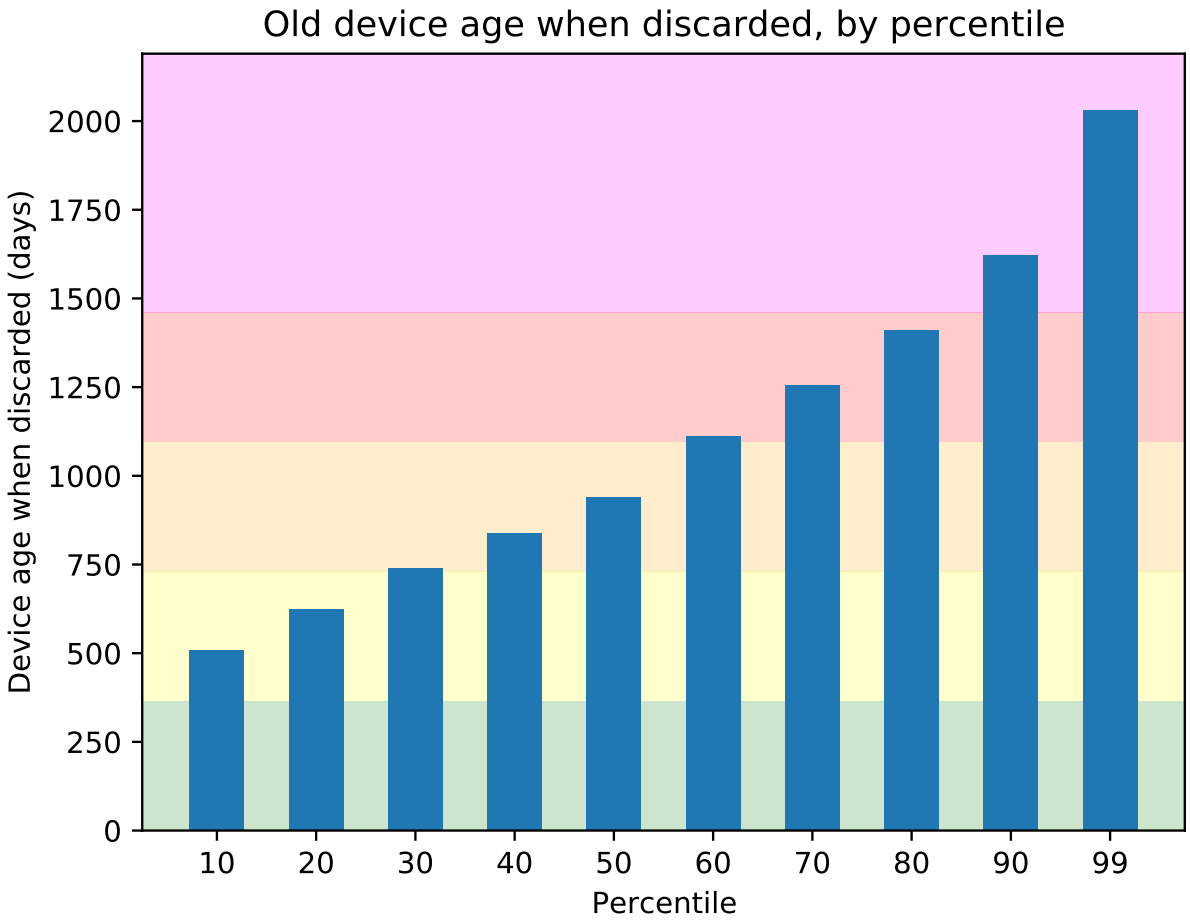


Figure 4.6: Ages of devices at time of upgrade, by percentile.

Interestingly, the pattern for the two device families is different, with Apple upgrades occurring towards the end of the year and Android occurring somewhat after the academic year begins, but with a spike at the start of the calendar year. I do not have insights into whether this coincides with other events, such as the release of a new flagship smartphone. However, I wish to primarily note that I see a fairly consistent pattern of upgrades across all years across device types (with the exception of the, 2019-2020 academic year, which has far less events than the other years due to the introduction and use of a new learning management system).

Next, I break down each upgrade event into percentiles by the age of the old device when it was upgraded. This will allow me to understand how different types of users upgrade their

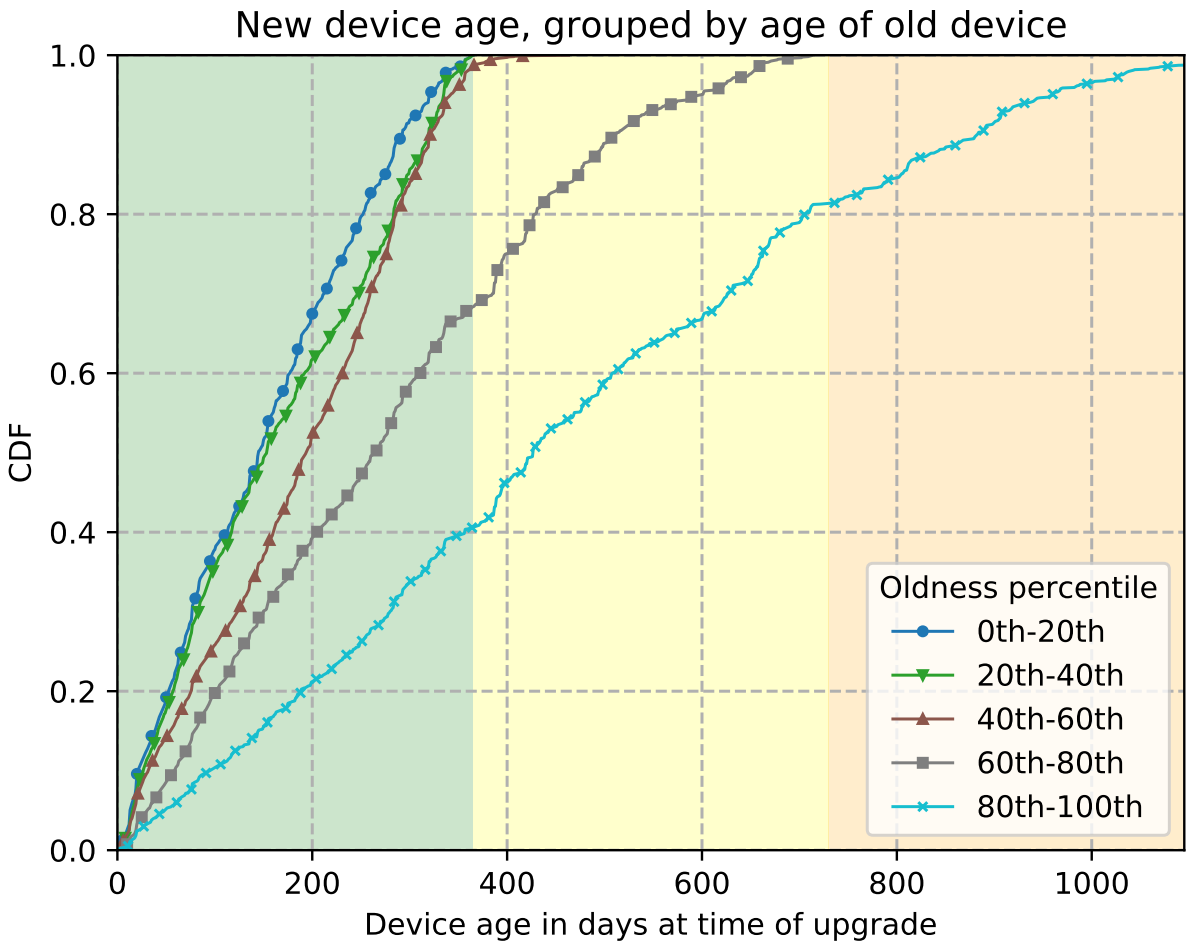


Figure 4.7: Age of new devices users upgraded to at time of first use, by oldness percentiles.

devices- for example, users who upgrade their device within a year of it first releasing are perhaps more likely to purchase a brand new device, rather than a device that may be somewhat dated. I then plot the age of the device these users upgraded to as a CDF. I also include the raw thresholds of old devices ages.

The percentile thresholds are shown in Figure 4.6. I immediately see that very few users actually upgrade their phone within a year- well under 10%. Additionally, there is a fairly linear growth between the most vigorous of upgrades and the least. My results show that almost all users upgrade their phone once it is four years out of date, although once again I observe a decent tail end of users (¡10%) hold onto their devices for roughly five years instead.

The upgrade newness CDF is shown in Figure 4.7. There is a clear trend that the users who discard their devices sooner are more likely to purchase a newer device. Note that the users who discard their phone within three years of its original release date all have roughly equivalent behavior when choosing a new device to purchase. It can be conjectured that the majority of users fall within a two to three year life cycle for their device, which is in line with information I present in Section 4.2.

However, the users from the 60th percentile onward clearly tend to purchase older devices, likely from a second-hand market. I observe that these users often discard their device when it is four years old, and then purchase a device that is three years old to replace it. Because of my previous results on device ages showing that phones are discarded after four years, these users are likely purchasing hardware that they will only find useful for just over a year before discarding it. This is a frightening trend: these users are purchasing a second-hand device that is then likely discarded within a year.

This implies that the second-hand market is not sufficiently extending the lifetime of devices: if it was, I would hopefully see some non-trivial number of students still leveraging these older phones. The reasoning for this is not obvious, but I can conclude that these users either cannot or do not wish to purchase newer devices.

4.5 Beating the Four Year Life Cycle

The common trend from all of my data is that roughly every four years, both software and hardware become obsolete. The financial burden on low income students and total e-Waste generated by devices suggests that smartphone lifetimes should be extended. However, it is not immediately obvious how to achieve this goal.

The cyclical nature of device manufacturing, use, and discard is a function of both consumer and producer. There are a vast variety of solutions that can increase the lifetime of

computing devices, reaching far beyond the scope of my work here. It is difficult to say whether focusing on producers or consumers to extend device longevity would be more impactful. There are recent efforts [Ama21] to develop longer-lasting devices from the producer side, and there are software solutions from consumers [Lin21] to support devices past this four year duration.

I believe that a more universal and transparent approach that involves neither the consumer or the producer could have the most impact. In this section, I describe my proposed approach to extending the lifespan of consumer computing devices through datacenter networks via cloud offload. I outline the benefits of cloud-offloaded applications. I then discuss previous cloud-offloaded web browsers intended for other purposes, finishing by proposing the requirements and functionality of a possible offload architecture for browsers with the goal of enabling students to access the web with legacy devices.

4.5.1 Cloud Offload for Applications

Cloud computing at the edge for radio networks has been in development for several years [TSM+17, LWN+17], and with the recent advent of large scale 5G network deployments, it has become an attractive target for cloud-based applications that require low latency. Interactive applications that run on a user's device are a great target for cloud offload. However, the requirements of offloading are different from application to application.

A more blunt solution for an outdated device may be to run remote desktop to a more powerful machine. However, I believe this is a poor solution. Remote desktop clients do not provide offline access to applications or files to the user, and do not take advantage of the client's hardware capabilities past simple video processing and keyboard/mouse I/O. Legacy client devices still can easily run applications such as text processors, presentation software, etc. for periods longer than four years, making them still suitable for many student needs.

For students, the core target application for cloud offloading is a web browser. Academic software with high compute requirements is an interesting target, but solutions for offloading

typically either already exist (e.g. remote compilation for programming projects), or are better suited to remote desktop environments even on the newest of student devices (e.g. complex engineering software such as AutoCAD).

4.5.2 Cloud-backed browsers

I continue to assume the environment of a student accessing educational resources remotely with a legacy device that is no longer supported. They need to access a set of educational websites and tools, as described in Section 4.3. In this context, any solution to achieve computer lifetime extension must meet the following requirements.

The user must be able to access modern websites and web-based applications. This includes the most recent versions of JavaScript, CSS-support, etc. While functionality is important, the user's experience must be performant, similar to the experience they would have received on modern hardware. Lastly, a typical student relies on a number of resources when doing school work, and so any solution must not consume an overly significant amount of resources on the target device.

Existing cloud-backed browsers

Development of a split-browser architecture has been explored for decades [FGBA96, FGG+98]. Recent efforts using virtual machines in cloud environments exist, but they primarily focus on providing client security rather than extending device lifetimes [Pat20]. They do provide insight into how a modern split-browser architecture on the cloud may be achieved.

One example is the recently developed browser isolation system from Cloudflare [Obe21], which uses a modern WebAssembly framework to provide a "remote browser" from the client to a cloud VM. The cloud VM performs all browser functions, and the client simply receives a network stream from the VM to interact with the rendered webpage. There is no offline functionality, which does not satisfy the requirements.

Other examples of recent cloud-backed browsers are Amazon's "Silk" browser [Ama20] and Opera's "mini" browser [Ope]. Silk targets performance as a primary objective and is more in line with the requirements, but still requires browsers to run all of the end-result web content locally, which does not solve the compatibility issue. Opera Mini solely performs webpage compression to save on networking overheads.

A cloud browser for legacy devices

Extending previous efforts of split browser architectures is likely the best way forward to create a cloud browser that fits the requirements. Using a remote VM in the cloud to create a fully-featured browser frame that the client interacts with is a strong solution that offloads all security and compatibility requirements off of the client, and has the potential to be useful for a large range of legacy devices.

In order to meet the offline requirement that is absent in the Cloudflare solution, there is a rather large development effort needed in order to translate page content into a safe and offline viewable format for each client. Original split browser designs from Fox et al. [FGBA96] could be useful in order to "distill" webpage information into a proper format.

A significant challenge with legacy devices using modern browsers lies in memory utilization. Browsers now use gigabytes of memory when a large number of tabs are simultaneously opened, and legacy devices may not be able to support this. A distilled webpage may allow a separate type of process to view the content, which may save on memory. Additionally, pages that are not currently active when the user is online can be cached in the cloud VM, creating further savings in memory used.

The challenges of 5G networking

My solution relies on the dual wins of 5G networking by providing students with access to a low-latency, high-bandwidth link to a nearby server in order to support a cloud-backed browser.

However, 5G is a new and rapidly developing technology that has its own challenges to overcome to make this a reality. The promises of modern 5G networks have yet to come to fruition.

One concern is that the environmental impact of 5G networking will offset the benefits that green-focused proposals like ours receive from it. There is significant effort in making new 5G deployments focused on reducing environmental impact at the power and antenna level [IRH+14], which alleviates some of this concern. However, it remains to be seen if the local edge datacenter deployments for these networks will receive a similar focus on reducing carbon emissions.

Additionally, while 5G networks are likely to become available at large university campuses and their students, it is unknown to what extent availability will benefit those in more remote areas, and if students on remote learning platforms will have access.

I believe solutions targeting 5G networks are still beneficial despite these issues. However, it is imperative that additional solutions for device longevity will need to be explored and implemented in order to achieve a zero-carbon future.

4.5.3 Future work

The approaches and concerns listed above are certainly not exhaustive, and there is an extremely large range of possible solutions for elongating the lifetime of computers. I hope that future work can examine and implement many such ideas and introduce new systems that promote a circular economy and aid in reducing eWaste.

In this work I primarily target one limited, specific type of user. There has already been work targeting other types of applications, such as those targeting GPU-related applications for videogame platforms. I expect to see additional work also examine many more types of consumers around the world.

Lastly, while work can be done individually by producers and consumers in order to increase device lifetimes, I believe that there needs to be work on how to change societal patterns as a whole regarding smartphones. The increasingly rapid cycle of device replacement cannot

solely be attributed to consumers' or producers' lack of ability to provide software or hardware solutions that increase device longevity. In order to create a holistic solution for a zero-carbon future, it will become necessary to create pressure via policy or market demand to change how the idea of "new" smartphones and other computing devices are viewed within the public consciousness.

4.6 Conclusions Regarding Smartphones and Datacenters

The cycle of device obsolescence has created worrying trends for students, who have in recent years have been required to use technology to access educational resources. The data I have analyzed shows that students are not exempt from the increased rates of software and hardware obsolescence. I observe an overall trend that roughly every four years, a student is required to upgrade their hardware and/or software. While many students can afford to purchase new devices, many experience financial hardship and cannot easily do so. Additionally, the frequent manufacturing and discarding of new devices increases the amount of generated eWaste in the world.

The observed trends in the LMS data I analyze point to the need for a solution for students to access educational resources without the need of purchasing a new device. The class of users that wishes to upgrade their devices less regularly and not purchase a brand new device shows that there is a definite need for a system that extends the lifetime of devices within a teaching environment. The previous solutions for cloud-based browser offloading via datacenter networks present an interesting solution space that I believe should be explored in order to provide sufficient extensions to product lifetimes for undergraduate students.

Chapter 4, in full, is a reprint of the work as it appeared in the Workshop on Computing within Limits (LIMITS 21). Rob McGuinness; George Porter, 2021. The dissertation author was the primary investigator and author for this paper.

Chapter 5

Conclusion

With digital infrastructure being an irreplaceable part of everyday life, it is imperative that the environmental impact of their creation, operation, and disposal be reduced to create a sustainable future. Datacenter networks are a very crucial component in modern computing. Datacenters are an important focus for aiding ecological efforts for at least two reasons.

First, future datacenter network designs can operate with less of an ecological impact. Using optical circuit switched networks will allow scaling datacenter networks to meet future demand without vastly scaling their operational energy requirements. However, there is still work to be done in providing hardware support to endhosts such that they can meet the precise transmission requirements emplaced by TDMA flow control. While endhosts can implement TDMA networking well at around 40 Gbps, new datacenter networks support are targeting 400 Gbps speeds.

Additionally, datacenters can aid reducing the impact of the embodied energy of smartphones, particularly in educational environments where mobile devices have become tethered to accessing educational resources. Smartphones have a fixed, limited lifespan for accessing online educational applications, and students are being required to purchase new phones to access

coursework. Datacenter networks can solve this by providing a platform for offloading application workloads so that smartphones will be operable for longer periods of time, reducing the impact of their embodied energy on the environment.

Datacenter networks have many uses beyond the scope of this dissertation. Additional research within other areas examining how future datacenters can operate with a lower environmental impact is necessary [All20]. New hardware designs will be required to not only implement TDMA networking, but holistically rework server architectures to solve the crisis of global warming.

Bibliography

[AFLV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[AFRR+10] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[AGM+10] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[AKE+12] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 19–19, Berkeley, CA, USA, 2012. USENIX Association.

[All20] Sustainable Digital Infrastructure Alliance. The roadmap to sustainable digital infrastructure by 2030, 2020.

[Ama20] Amazon. Amazon Silk documentation. https://docs.aws.amazon.com/silk/index.html, 2020.

[Ama21] Ron Amadeo. Fairphone suggests Qualcomm is the biggest barrier to long-term android support. https://arstechnica.com/gadgets/2021/03/the-fairphone-2-hits-five-years-of-updates-with-some-help-from-lineageos/, Mar 2021.

[App21] Apple Inc. Supported iPhone models. https://support.apple.com/guide/iphone/supported-iphone-models-iphe3fa5df43/ios, 2021.

[ASA+21] Anubhavnidhi Abhashkumar, Kausik Subramanian, Alexey Andreyev, Hyojeong Kim, Nanda Kishore Salem, Jingyi Yang, Petr Lapukhov, Aditya Akella, and Hongyi Zeng. Running BGP in data centers at scale. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 65–81. USENIX Association, April 2021.

[AWE19] Alexey Andreyev, Xu Wang, and Alex Eckert. Reinventing Facebook's data center network. *Facebook Engineering*, Mar 2019.

[AYK+12] Mohammad Alizadeh, Shuang Yang, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Deconstructing datacenter packet transport. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 133–138, New York, NY, USA, 2012. ACM.

[BDG+14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):8795, July 2014.

[BFG+17] CP Baldé, Vanessa Forti, Vanessa Gray, Ruediger Kuehr, and Paul Stegmann. The Global E-waste Monitor–2017, United Nations University (UNU), International Telecommunication Union (ITU) & International Solid Waste Association (ISWA), Bonn/Geneva/Vienna. *ISBN Electronic Version*, pages 978–92, 2017.

[Bla18] Blackboard Inc. Blackboard delivers worldwide growth. https://www.prnewswire.com/news-releases/blackboard-delivers-worldwide-growth-300398129.html, Jun 2018.

[Bro] Broadcom. High-density 25/100 gigabit ethernet StrataXGS Tomahawk ethernet switch series. https://www.broadcom.com/products/ethernet-communication-and-switching/switching/bcm56960-series.

[Bro11] BrowserStack. Browserstack. https://www.browserstack.com/, 2011.

[BS+17] CA Bakker, CSC Schuit, et al. The long view: Exploring product lifetime extension. https://www.oneplanetnetwork.org/resource/long-view-exploring-product-lifetime-extension, 2017.

[BVAV16] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS '16, pages 75–88. ACM, 2016.

[CAS21] Mauro Cordella, Felice Alfieri, and Javier Sanfelix. Reducing the carbon footprint of ICT products through material efficiency strategies: A life cycle analysis of smartphones. *Journal of Industrial Ecology*, 25(2):448–464, Mar 2021.

[CLGS16] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. HUG: Multi-resource fairness for correlated and elastic demands. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 407–424, Santa Clara, CA, 2016. USENIX Association.

[Coo04] T Cooper. Inadequate Life? Evidence of Consumer Attitudes to Product Obsolescence. *J. Consum Police*, 27:421–449, 2004.

[Cor20] International Data Corporation. Worldwide server market revenue grew 19.8% year over year in the second quarter of 2020, according to IDC. *IDC Media Center*, Sep 2020.

[CSS+12] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, and Xitao Wen. OSA: An optical switching architecture for data center networks and unprecedented flexibility. In *Proc. USENIX NSDI*, NSDI '12, April 2012.

[CWM+15] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, and Q. Dong. WaveCube: A scalable, fault-tolerant, high-performance optical data center architecture. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1903–1911, April 2015.

[CXW+16] Yong Cui, Shihan Xiao, Xin Wang, Zhenjie Yang, Chao Zhu, Xiangyang Li, Liu Yang, and Ning Ge. Diamond: Nesting the data center network with wireless rings in 3D space. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 657–669, Santa Clara, CA, 2016. USENIX Association.

[DB13] Jeffrey Dean and Luiz Andr Barroso. The tail at scale. *Communications of the ACM*, 56:74–80, 2013.

[Ech16] Fabián Echegaray. Consumers' reactions to product obsolescence in emerging markets: the case of Brazil. *Journal of Cleaner Production*, 134:191–203, 2016.

[Fac14] Facebook. Introducing data center fabric, the next-generation Facebook data center network. https://goo.gl/mvder2, Nov 2014.

[FGBA96] Armando Fox, Steven D Gribble, Eric A Brewer, and Elan Amir. Adapting to client variability via on-demand dynamic distillation. In *Proc. of the 7th ACM Inter. Conference on Architectural support for Programming Languages and Operating Systems*, 1996.

[FGG+98] Armando Fox, Ian Goldberg, Steven D Gribble, David C Lee, Anthony Polito, and Eric A Brewer. Experience with Top Gun Wingman: a proxy-based graphical web browser for the 3Com PalmPilot. In *Middleware 98*, pages 407–424. Springer, 1998.

[FGH+21] Andrew D. Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, Richard Alimi, Shawn Shuoshuo Chen, Mike Conley, Subhasree Mandal, Karthik Nagaraj, Kondapa Naidu Bollineni, Amr Sabaa, Shidong Zhang, Min Zhu, and Amin Vahdat. Orion: Google's software-defined networking control plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 83–98. USENIX Association, April 2021.

[Fou] The Linux Foundation. Data Plane Development Kit. https://dpdk.org/.

[FPR+10] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *Proc. ACM SIGCOMM*, August 2010.

[FSPP20] Alex Forencich, Alex C. Snoeren, George Porter, and George Papen. Corundum: An open-source 100-gbps nic. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 38–46, 2020.

[FWJ+13] Shu Fu, Bin Wu, Xiaohong Jiang, A. Pattavina, Lei Zhang, and Shizhong Xu. Cost and delay tradeoff in three-stage switch architecture for data center networks. In *Proc. IEEE High Perf. Switching and Routing*, July 2013.

[GFB+04] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In Dieter Kranzlmüller, Péter Kacsuk, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 97–104, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[GHJ+09a] 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*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[GHJ+09b] 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 Conference on Data Communication*, pages 51–62, Barcelona, Spain, 2009.

[GMP+16] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. ProjecToR: Agile reconfigurable data center interconnect. In *Proceedings of the ACM SIGCOMM Conference*, pages 216–229, Florianopolis, Brazil, 2016.

[Goo20] Google. Google Drive app. https://play.google.com/store/apps/details?id=com.google.android.apps.docs, 2020.

[GRRH17] Sara Goldrick-Rab, Jed Richardson, and Anthony Hernandez. Hungry and homeless in college: Results from a national study of basic needs insecurity in higher education. *Wisconson HOPE Lab*, 2017.

[GSG+15] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. Queues don't matter when you can jump them! In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 1–14, Berkeley, CA, USA, 2015. USENIX Association.

[GYBS20] Stewart Grant, Anil Yelam, Maxwell Bland, and Alex C. Snoeren. Smartnic performance isolation with fairnic: Programmable networking for the cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 681693, New York, NY, USA, 2020. Association for Computing Machinery.

[GZKS19] Boris Glinskiy, Yury Zagorulko, Igor Kulikov, and Anna Sapetina. Supercomputer technologies for solving problems of computational physics. *Journal of Physics: Conference Series*, 1392:012052, nov 2019.

[HJP+15] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. SoftNIC: A software NIC to augment hardware. Technical Report UCB/EECS-2015-155, EECS Department, University of California, Berkeley, May 2015.

[HRA+17] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 29–42, New York, NY, USA, 2017. ACM.

[Ins21] Instructure Inc. Our story. https://www.instructure.com/about/our-story, 2021.

[IRH+14] Chih-Lin I, Corbett Rowell, Shuangfeng Han, Zhikun Xu, Gang Li, and Zhengang Pan. Toward green and soft: a 5G perspective. *IEEE Communications Magazine*, 52(2):66–73, 2014.

[JAM+12] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, and Changhoon Kim. EyeQ: Practical network performance isolation for the multi-tenant cloud. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'12, pages 8–8, Berkeley, CA, USA, 2012. USENIX Association.

[JFR16] Xin Jin, Nathan Farrington, and Jennifer Rexford. Your data center switch is trying too hard. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 12:1–12:6, New York, NY, USA, 2016. ACM.

[Kam20] George Kamiya. Data centres and data transmission networks. *International Energy Agency*, Jun 2020.

[KBK+15] Barbara V Kasulaitis, Callie W Babbitt, Ramzy Kahhat, Eric Williams, and Erinn G Ryen. Evolving materials, attributes, and functionality in consumer electronics: Case study of laptop computers. *Resources, conservation and recycling*, 100:1–10, 2015.

[Kha19] Imran Khan. Greenhouse gas emission accounting approaches in electricity generation systems: A review. *Atmospheric Environment*, 200:131–141, 2019.

[KHK+16] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. HULA: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 10:1–10:12, New York, NY, USA, 2016. ACM.

[KNHM17] R. Kawashima, H. Nakayama, T. Hayashi, and H. Matsuo. Evaluation of forwarding efficiency in NFV-nodes toward predictable service chain performance. *IEEE Transactions on Network and Service Management*, 14(4):920–933, Dec 2017.

[KPAK15] Antoine Kaufmann, Simon Peter, Thomas Anderson, and Arvind Krishnamurthy. FlexNIC: Rethinking network DMA. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, HOTOS'15, pages 7–7, Berkeley, CA, USA, 2015. USENIX Association.

[KPB09] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. Flyways to de-congest data center networks. In *Proc. ACM HotNets*, HotNets '09, October 2009.

[KVS+17] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 281–294, New York, NY, USA, 2017. ACM.

[Lin21] LineageOS. https://wiki.lineageos.org/devices/, 2021.

[LLF+14] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. Circuit switching under the radar with REACToR. In *Proc. USENIX NSDI*, April 2014.

[LML+15] 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 *Proc. ACM CoNEXT*, 2015.

[LWN+17] Nguyen Cong Luong, Ping Wang, Dusit Niyato, Yonggang Wen, and Zhu Han. Resource management in cloud networking using economic analysis and pricing models: A survey. *IEEE Communications Surveys Tutorials*, 19(2):954–1001, 2017.

[MAV17] Muhammad Baqer Mollah, Md. Abul Kalam Azad, and Athanasios Vasilakos. Security and privacy challenges in mobile cloud computing: Survey and way ahead. *Journal of Network and Computer Applications*, 84:38–54, 2017.

[MDG+20] 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.

[MFK+20] William M. Mellette, Alex Forencich, Jason Kelley, Joseph Ford, George Porter, Alex C. Snoeren, and George Papen. Optical networking within the lightwave energy-efficient datacenter project [invited]. *IEEE/OSA Journal of Optical Communications and Networking*, 12(12):378–389, 2020.

[MGS20] Lokanath Mishra, Tushar Gupta, and Abha Shree. Online teaching-learning in higher education during lockdown period of covid-19 pandemic. *International Journal of Educational Research Open*, 1:100012, 2020.

[Mil13] Mark P Mills. The cloud begins with coal. *Digital Power Group*, 1, Aug 2013.

[MLD+15] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. TIMELY: RTT-based congestion control for the datacenter. *SIGCOMM Comput. Commun. Rev.*, 45(4):537–550, August 2015.

[MMR+17] 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, pages 267–280, New York, NY, USA, 2017. ACM.

[MSP+17] William Maxwell Mellette, Glenn M. Schuster, George Porter, George Papen, and Joseph E. Ford. A scalable, partially configurable optical switch for data center networks. *Journal of Lightwave Technology*, 35(2):136–144, 2017.

[MSWH16] Patricia Mahaffey, Dominick Suvonnasupa, Hayley Weddle, and Katie Hosch. Report on food and housing insecurity at UC San Diego. *UC San Diego Basic Needs Insecurity Committee*, Jul 2016.

[MT] Ltd. Mellanox Technologies. Mellanox sets new DPDK performance record with ConnectX-5. http://ir.mellanox.com/releasedetail.cfm?ReleaseID=1014514.

[Obe21] Tim Obezuk. Introducing Cloudflare browser isolation beta. https://blog.cloudflare.com/browser-beta/, Jan 2021.

[Ope] Opera. Opera Mini. https://www.opera.com/mobile/mini.

[Pan] Kantar World Panel. Kantar WorldPanel: Double Digit Smarphone Market Growth is over. https://www.kantarworldpanel.com/global/News/Double-Digit-Smartphone-Market-Growth-is-over.

[Pat20] Mehul Patel. Gartner report on remote browser isolation: Menlo security. https://www.menlosecurity.com/blog/gartner-report-on-remote-browser-isolation-menlo-securitys-continued-validation, Jan 2020.

[PCPS18] Supanan Phantratanamongkol, Fabrizio Casalin, Gu Pang, and Joseph Sanderson. The price-volume relationship for new and remanufactured smartphones. *International Journal of Production Economics*, 199:78–94, 2018.

[PHJ+16] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. NetBricks: Taking the V out of NFV. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 203–216, Berkeley, CA, USA, 2016. USENIX Association.

[POB+14] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 307–318, New York, NY, USA, 2014. ACM.

[PSF+13] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang-Chen Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating microsecond circuit switching into the data center. In *Proceedings of the ACM SIGCOMM Conference*, Hong Kong, China, August 2013.

[RGJ+14] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. SENIC: Scalable NIC for end-host rate limiting.

In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 475–488, Berkeley, CA, USA, 2014. USENIX Association.

[RM11] Barath Raghavan and Justin Ma. The energy and emergy of the internet. *Proceedings of the 10th ACM Workshop on hot topics in networks*, pages 1–6, Nov 2011.

[RZB+15a] 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*, SIGCOMM '15, pages 123–137, New York, NY, USA, 2015. ACM.

[RZB+15b] 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*, SIGCOMM '15, pages 123–137, New York, NY, USA, 2015. ACM.

[Sch20] Max Schulze. About the alliance: Our vision. https://sdialliance.org/about, 2020.

[SDV+17] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 404–417, New York, NY, USA, 2017. ACM.

[SKG+11] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 309–322, Berkeley, CA, USA, 2011. USENIX Association.

[SL15] James Suckling and Jacquetta Lee. Redefining scope: the true environmental impact of smartphones? *The International Journal of Life Cycle Assessment*, 20(8):1181–1196, 2015.

[SLLP09] Hari Subramoni, Ping Lai, Miao Luo, and Dhabaleswar K. Panda. RDMA over ethernet - a preliminary study. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–9, 2009.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[SOA+15] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle,

Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 183–197, New York, NY, USA, 2015. ACM.

[Sta] StatCounter. Statcounter global stats. https://gs.statcounter.com/.

[Suc19] Peter Suciu. Consumers balk at premium smartphone prices. https://www.technewsworld.com/story/85981.html, Apr 2019.

[Tay12] Michael B. Taylor. Is dark silicon useful?: Harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1131–1136, New York, NY, USA, 2012. ACM.

[TSM+17] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys Tutorials*, 19(3):1657–1681, 2017.

[UIU19] PACE UNEP, ILO ITU, and UNU UNIDO. A new circular vision for electronics time for a global reboot. https://www.weforum.org/reports/a-new-circular-vision-for-electronics-time-for-a-global-reboot, 2019.

[Uni21] University of California, San Diego. UC San Diego campus profile. https://ucpa.ucsd.edu/campus-profile/, 2021.

[Val82] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2), 1982.

[VPVS12] Bhanu Chandra Vattikonda, George Porter, Amin Vahdat, and Alex C. Snoeren. Practical TDMA for datacenter ethernet. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 225–238, New York, NY, USA, 2012. ACM.

[VSDS16] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 205–219, New York, NY, USA, 2016. ACM.

[VSG+10] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 205–218, New York, NY, USA, 2010. ACM.

[WAK+10] 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 *Proc. ACM SIGCOMM*, SIGCOMM '10, August 2010.

[WTH15] H Wieser, N Tröger, and R Hübner. The consumers' desired and expected product lifetimes. *Product Lifetimes And The Environment*, 2015.

[ZDM+12] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. DeTail: Reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 139–150, New York, NY, USA, 2012. ACM.

[ZLA+19] Irene Zhang, Jing Liu, Amanda Austin, Michael Lowell Roberts, and Anirudh Badam. I'm not dead yet! the role of the operating system in a kernel-bypass era. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, page 7380, New York, NY, USA, 2019. Association for Computing Machinery.

[ZZZ+12] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y. Zhao, and Haitao Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 443–454, New York, NY, USA, 2012. ACM.