UNIVERSITY OF CALIFORNIA, SAN DIEGO

A High Performance Hybrid ToR for Data Centers

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

in

Computer Science

by

He Liu

2015

DEDICATION

To my fiancée and my parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisors, Professor Geoffrey M. Voelker and Stefan Savage. Thank them giving this opportunity to study in graduate school and become truly economically independent. Thank them for showing me in person that one can still contribute to this world in positive attitudes when there are a thousand reasons to keep a cynic view. Thank Professor Voelker for not giving me up when I was lost in the cold mountains.

Thank all the members in the project group for making REACToR happen. Thank Feng Lu, Rishi Kapoor, Malveeka Tewari, Alex Forenich and Sen Zhang for their collaborative efforts, and thank Professor Alex C. Snoeren, George Papen and George Porter for their insightful advising and being my thesis committee members.

I would also like to thank all my colleagues in UCSD and Berkeley for their invaluable help, assistance and collaboration: Kirill Levchenko, Neha Chachra, Tristan Halvorson, Ding Yuan, Danny Huang, David Wang, Qing Zhang, Andreas Pitsillidis, Peng Huang, Chris Kanich, Chris Grier, Hovav Shacham, Keaton Mowery, Michael Vrable, Tianying Xu, Jing Zheng, Jiaqi Zhang, Weiwei Xiong, Damon McCoy, Mark Felegyhazi, Brandon Enright, Nicholas Weaver, Christian Kreibich, Vern Paxon, Kai Wang, Ge Wang. Thank Brian Kantor, Cindy Moore, and Jennifer Folkestad for their professional help and support.

Finally, I would like to thank my fiancée, Jing Xu, for accompanying, supporting, and feeding me all these years. It is she and her awesome cooking that made our small rental apartment in a foreign country always feel like a sweet home.

Chapter 1, 2 and 3, in part, is a reprint of the material as it appears in the USENIX Symposium on Networked Systems Design and Implementation 2014. Liu, He; Lu, Feng; Forencich, Alex; Kapoor, Rishi; Tewari, Malveeka; Voelker, Geoffrey M.; Papen, George;

Snoeren, Alex C.; Porter, George. The dissertation author was the primary investigator and author of this paper.

Chapter 1, 2 and 4, in part, has been submitted for publication of the material as it may appear in the the conference of the ACM Special Interest Group on Data Communication 2015. Liu, He; Kapoor, Rishi; Tewari, Malveeka; Forencich, Alex; Zhang, Sen; Savage, Stefan; Voelker, Geoffrey M.; Papen, George; Snoeren, Alex C.; Porter, George. The dissertation author was the primary investigator and author of this material.

VITA

2009	Bachelor of Science, Tsinghua University, Beijing
2009–2015	Research Assistant, University of California, San Diego
2015	Doctor of Philosophy, University of California, San Diego

PUBLICATIONS

“Circuit Switching Under the Radar with REACToR”, He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014).* Seattle, WA, April 2014.

“Software Abstractions for Trusted Sensors”, He Liu, Stefan Saroiu, Alec Wolman, Himanshu Raj. *Proceedings of the 10th International Conference on Mobile Systems, Applications and Services (MobiSys 2012).* Low Wood Bay, Lake District, United Kingdom, June 2012

“Verilogo: Proactive Phishing Detection via Logo Recognition”, Ge Wang, He Liu, Sebastian Becerra, Kay Wang, Serge Belongie, Hovav Shacham, and Stefan Savage. UCSD Technical Report CS2011-0969, August 2011.

“Click Trajectories: End-to-End Analysis of the Spam Value Chain”, Kirill Levchenko, Andreas Pitsillidis, Neha Chachra, Brandon Enright, Mark Felegyhazi, Chris Grier, Tristan Halvorson, Chris Kanich, Christian Kreibich, He Liu, Damon McCoy, Nicholas Weaver, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage. *Proceedings of THE IEEE Symposium on Security and Privacy.* Oakland, CA, May 2011.

“On the Effects of Registrar-level Intervention”, He Liu, Kirill Levchenko, Mark Felegyhazi, Christian Kreibich, Gregor Maier, Geoffrey M. Voelker, and Stefan Savage. *Proceedings of the USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET).* Boston, MA, March 2011.

“Sora: high-performance software radio using general-purpose multi-core processors”, Kun Tan, He Liu, Jiansong Zhang, Yongguang Zhang, Ji Fang, and Geoffrey M. Voelker. *Communications of the ACM*, Januaray 2011.

“SAM: Enabling Practical Spatial Multiple Access in Wireless LAN”, Kun Tan, He Liu, Ji Fang, Wei Wang, Jiansong Zhang, Mi Chene and Geoffrey M. Voelker. *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking (Mobicom)*. Beijing, China, September 2009.

“Sora: High Performance Software Radio Using General Purpose Multi-core Processors”, Kun Tan, Jiansong Zhang, Ji Fang, He Liu, Yusheng Ye, Shen Wang, Yongguang Zhang, Haitao Wu, Wei Wang, and Geoffrey M. Voelker. *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Boston, MA, April 2009.

ABSTRACT OF THE DISSERTATION

A High Performance Hybrid ToR for Data Centers

by

He Liu

Doctor of Philosophy in Computer Science

University of California, San Diego, 2015

Professor Geoffrey M. Voelker, Chair
Professor Stefan Savage, Co-Chair

The potential advantages of optics at high link speeds have led to significant interest in deploying optical switching technology in data-center networks. Initial efforts have focused on hybrid approaches that rely on millisecond-scale circuit switching in the core of the network, while maintaining the flexibility of electrical packet switching at the edge. Recent demonstrations of microsecond-scale optical circuit switches motivate considering circuit switching for more dynamic traffic such as that generated from a top-of-rack (ToR) switch.

Based on these technology trends, this dissertation presents a prototype hybrid

ToR design called REACToR. REACToR combines 10-Gbps packet switching and 100-Gbps circuit switching, and appears to end-hosts as a 100-Gbps packet-switched ToR. REACToR synchronizes end host transmissions with end-to-end circuit assignments, and can react to rapid, bursty changes in the traffic from end hosts on a time scale of 100s of microseconds.

To service data center traffic demands effectively, REACToR needs to schedule the heavy bandwidth-hungry flows to the circuit switching network, and the small latency-sensitive flows to the packet switching network. To address this problem, this dissertation also presents a new switch scheduling algorithm called Solstice. Solstice minimizes the frequency of circuit reconfigurations to maximize circuit utilization when reconfiguration delay is not negligible. Evaluations also show that it can schedule data center traffic workloads effectively with practical computational overheads.

As a result, when using a REACToR hybrid switch with the Solstice scheduling algorithm, optical circuit switching extends to layers even closer to end hosts. Combined with a lower-provisioned electrical packet switch, the hybrid architecture services data center workloads with almost full bi-sectional bandwidth of high link rates like 100Gb/s. It provides network performance comparable to a full fat-tree that consists of electrical packet switches, but with much lower costs.

Chapter 1

Introduction

Modern data centers interconnect thousands of commodity servers with commodity packet-based network switches. It enables search, social networks, elastic cloud computing and many more “Big Data” applications. As these applications scale up to more users and data, the traffic demands they generate grow rapidly, pushing data center networks to use higher link rates.

However, designing scalable data center interconnects is already an extremely challenging problem, and it is only getting harder as the link rates move from 10 to 40 to 100 Gb/s and beyond. Unlike previous generational upgrades, upgrading to 100-Gb/s link rates fundamentally changes how data centers are wired. At 100 Gb/s, inexpensive copper cabling can no longer be used at distances greater than a few meters: virtually all cables other than those internal to an individual rack must be optical. If these cables interconnect electronic packet switches, they further require optoelectronic transceivers at both ends. Many popular packet-switched data-center topologies like multi-rooted trees [45] require large numbers of connections between racks. Hence, the dominant cost of these designs changes from those of the constituent packet switches to those of the transceivers mandated by the optical interconnects necessary to support the increased link speed [14]. These transceivers not only cost more to manufacture, but also consume power and produce great amounts of heat.

In contrast, if the switches internal to the network fabric are themselves optical, the need for transceivers can be significantly reduced. Researchers have previously proposed hybrid architectures consisting of a combination of packet switches and optical circuit switches managed by a common logical control plane [10, 14, 46]. Traditionally, however, their applicability has been limited by the delay incurred when reconfiguring the circuit switches, as traffic has to be buffered while waiting for a circuit assignment. Architectures based upon legacy optical circuit switches designed for wide-area applications are fundamentally dependent on stable, aggregated traffic to amortize their long reconfiguration delays. Therefore, their use has been restricted to either the core of the network [14] or to highly constrained workloads [46].

Researchers have recently demonstrated optical circuit switch prototypes with microsecond-scale reconfiguration delays [8, 29, 31], and showed that such a switch, when coupled with an appropriate control plane [40], has the potential to support more dynamic traffic patterns, extending the applicability of circuit switches to cover the entire network fabric required to interconnect racks of servers within a data center.

In this dissertation, I argue that by using a hybrid of 100-Gb/s optical circuit switching and lower-provisioned 10-Gb/s electrical packet switching, it is possible to service common data center workloads with almost non-blocking bi-sectional bandwidth, like full electrical packet switching fat-tree networks of 100Gb/s, but with fewer transceivers and hence much lower costs.

1.1 Hybrid Data Center Switches

Circuit switching alone is still inadequate to meet the traffic demands within a data center. To achieve acceptable link efficiency, a pure circuit switch needs to maintain circuit configurations stable in time durations that are at least an order of magnitude longer than the circuit reconfiguration delay. These long stability periods incur substantial delays to achieve efficiency (e.g., 61–300 μs to deliver 65–95% of the bandwidth of a comparable packet switch [40]), and the buffering required to tolerate such delays to support large number of active flows at 100 Gb/s is substantial. By integrating a certain level of packet switching, hybrid fabrics have the potential to address these shortcomings of pure circuit switching. However, existing hybrid switches use circuits that reconfigure in milliseconds, and are not capable of coping with the lack of traffic stability and aggregation present at the rack level of today's data center [3, 5, 27, 28].

This dissertation proposes a hybrid network architecture where its optical circuit switching penetrates the data center network to the top-of-rack (ToR) switches. Using the Mordia optical circuit switch [40] as a building block, I experimentally prototype the first hybrid network control plane that uses rapidly reconfigurable optical circuit switches to provide packet-switch-like performance at substantially lower cost than an entirely packet-switched network for targeted data center workloads. My hybrid network design consists of a 100-Gb/s optical circuit-switched network deployed alongside a pre-existing 10-Gb/s electrical packet switched network. In this model, ToR switches support 100-Gb/s downlinks to servers, and are "dual-homed" to a legacy 10-Gb/s electrical packet switched network (EPS) and a new 100-Gb/s optical circuit switched network (OCS).

My hybrid network design assumes that the traffic demand is highly skewed in the serviced data center. That is, if all the flows among the end hosts are sorted by their link bandwidths required, roughly the top 10% of flows in the data center will use more than 90% of the total link capacity, and the remaining 90% of the flows consume no more than 10% of the link capacity in the network. This skewness pattern is common in data centers. For example, Microsoft shows that it is very rare that more than five big flows coexist on a single end host in its data center [27]. With this workload assumption, my hybrid switch will efficiently service the long-lasting big flows with the optical circuit switches, and small flows, which consume less than 10% of the link bandwidth, with the lower-provisioned electrical packet switches.

1.2 Hybrid Switch Scheduling

One key challenge that this hybrid approach faces is to effectively schedule the bulk of the traffic with the higher-capacity—but slower to reconfigure—optical switching technology, while ensuring that the remaining traffic does not overdrive the under-provisioned packet network. While the potential cost savings that hybrid technologies could realize is large, their practical utility depends on the types of workload that can be effectively scheduled by the given switch design without incurring significant loss or delay.

For this new hybrid scheduling problem, I develop a hybrid switch scheduler called Solstice. It exploits the typical skewed pattern of data center traffic demand, and creates a small number of configurations with long durations to minimize the penalty for reconfiguration, leaving only a small amount of residual demand to be serviced by a low-speed (and, hence, lower-cost) unconstrained packet switch. Empirically, Solstice shows $O(N \log^2 N)$ time complexity, where N is the number of hosts that have traffic to schedule.

1.3 Contributions

In this dissertation, I present the design of a hybrid top-of-rack switch that combines an optical circuit switch and an under-provisioned electrical packet switch, along with a crossbar scheduling algorithm for this switch. Using this scheduling algorithm, the switch can service common data center network demand with performance akin to a high speed electrical packet switch but at much lower cost.

I argue that with optical circuit switches that can reconfigure in microseconds, it is possible to push the hybrid switch design to the top-of-rack layer. This hybrid design extends circuits to links closer to the end-hosts, and provides cheaper solutions for upgrading the server links to 100 Gb/s.

I design and implement the control plane for such a hybrid top-of-rack switch that connects an optical circuit-switching network and an under-provisioned electrical packet-switching network. It consists of three parts that are closely integrated: a centralized schedule controller that creates the traffic plan based on the demand, real-time flow controllers that synchronize network traffic with the circuit configuration changes, and end-host traffic classifiers that direct the traffic to either the circuit-switching or the packet-switching network.

I build a simulator for my hybrid switch, and carefully calibrate the simulator against the prototype implementation. Using this simulator, I extend the study on hybrid switching to large scale through simulated evaluations, and show the benefits of a hybrid switch design compared to a pure circuit design.

I define the circuit scheduling problem for hybrid switching, and distinguish the problem from previous crossbar scheduling problems: it is relaxed, where the demand that the optical circuit switch cannot service can be serviced by the under-provisioned electrical packet switch; it does not benefit from speed-ups, because optical links have no buffers for link speed transitioning; and the reconfiguration delay is non-trivial such that the scheduler should minimize the frequency of reconfigurations.

Specifically for this new hybrid scheduling problem, I design and implement a new scheduling algorithm. I observe and leverage the fact that the traffic demand in data center networks is often sparse and skewed, and hence it is possible to service in a small number of configurations during a scheduling period. My simulation results show that my algorithm empirically generates such schedules in $O(N \log^2 N)$ time, where N is the number of active hosts in the data center.

My results also show that, when the demand is skewed, my scheduling algorithm can service 90% of the traffic demand with the circuit links, even when the switch capacity is fully subscribed. Along with the help of the lower-provisioned packet switch, my hybrid scheduling algorithm provides almost full bi-sectional bandwidth. The resulting network performance of the hybrid network is hence comparable with a fully-provisioned bi-sectional electrical packet switching network.

1.4 Organization

The remainder of this dissertation is organized in the following manner.

Chapter 2 provides a general overview of previous data center switch designs that use optical circuit switches, and the motivation for introducing hybrid optical circuit switches for data center workloads.

Chapter 3 discusses REACToR, my hybrid optical circuit switch and its control plane design. Further, I evaluate the control plane performance of my design and demonstrate that it is able to reactively service data center network workloads.

Chapter 4 discusses the hybrid scheduling problem introduced by hybrid data center switches like REACToR. To address this problem, I present my scheduling algorithm, Solstice, and evaluate its efficiency on serving various data center network workload patterns.

Chapter 5 concludes the dissertation and discusses how the work in this dissertation could be further extended.

This chapter, in part, is a reprint of the material as it appears in the USENIX Symposium on Networked Systems Design and Implementation 2014. Liu, He; Lu, Feng; Forencich, Alex; Kapoor, Rishi; Tewari, Malveeka; Voelker, Geoffrey M.; Papen, George; Snoeren, Alex C.; Porter, George. The dissertation author was the primary investigator and author of this paper.

This chapter, in part, has been submitted for publication of the material as it may appear in the the conference of the ACM Special Interest Group on Data Communication 2015. Liu, He; Kapoor, Rishi; Tewari, Malveeka; Forencich, Alex; Zhang, Sen; Savage, Stefan; Voelker, Geoffrey M.; Papen, George; Snoeren, Alex C.; Porter, George. The dissertation author was the primary investigator and author of this material.

Chapter 2

Background and Motivation

This chapter provides background information about data center networks and motivates hybrid data center switches.

2.1 Data Center Networks Have Complex Topologies

Data centers are scalable computing infrastructures that interconnect large amounts of servers. Initially built from commercial personal computers and commodity Ethernet switches [11, 16], data center networks now consist of 100s of thousands or more blade servers and can provide tremendous computational power with low costs but high scalability. Using data centers, commercial companies gain the ability to process petabytes of data in parallel, and to service millions of concurrent user requests from all over the world. Moreover, these data centers use operating systems and software development stacks that are essentially the same ones that run on personal computers that people use on desktop workstations, further lowering the cost for developers to build data center applications.

With such great scalability at a feasible cost, data centers have gained more popularity and supported more applications. Hardware markets just for data center equipment have emerged. Specially-designed servers and equipment that are more powerful and more efficient have started to replace the traditional commodity servers in data centers [1]. The market have evolved data center servers with more processing units, faster storage, lower energy consumption, and higher network link rates. These server technology trends have correspondingly applied pressure on data center networks that connect these servers together.

Traditional data center networks use commodity Ethernet switches where each typically has 24 or 48 Ethernet ports [2, 16]. With these switches, data center networks ideally provide full non-blocking bi-sectional bandwidth — the best bandwidth service that a network can provide for the end hosts where each host has a fixed link rate. If a data center has such bandwidth, it means that as long as each end host sends and receives data at a rate at most the link rate in total, the network should always be able to satisfy all traffic demands from all servers without delaying any data transfer.

Mathematically, such traffic demand can be described as a matrix of demand traffic link rates, where each row represents a sending host, each column represents a receiving host, and each element in the matrix represents a traffic rate from the corresponding sending host to the receiving host. A satisfiable bi-sectional traffic demand is a matrix where the element sum in each row and column is no larger than the link rate at the input and output ports. A network that has full non-blocking bi-sectional bandwidth can service such a demand matrix without accumulating packets at the buffers of either the end hosts or the switches, where each end host can send and receive at the exact traffic rate equivalent to the corresponding element in the demand matrix.

Most commodity Ethernet switches only provide bi-sectional bandwidth for several 10s of ports, typically 24 or 48. To provide full bi-sectional bandwidth for thousands or more end hosts, the network has to cascade these switches into a topology that has higher connectivity, like a parallel tree or a high-degree multi-dimension mesh network.

For example, a fat-tree [2] cascades switches into a tree where the parallelism at the nodes on each layer is doubled when the layers approach the root of the tree. As a result, a k-layer fat-tree with $2n$-port switches can provide full bi-sectional bandwidth for $2n^k$ hosts , with $(2k-1)n^{k-1}$ switches and $2kn^k$ cables. As a concrete example, a 3-level fat-tree with 48-port switches can provide full bi-sectional bandwidth to 27,648 hosts with 2,880 switches and 82,944 cables. Data center topologies hence often lead to huge wiring complexity.

2.2 Higher Link Rates Must Use Optics

Traditional data centers started by using copper-based 100Mb/s Ethernet cables [11, 16], the same type of wires that people use on personal computers in everyday life. When data centers higher link rates such as 10Gb/s, 40Gb/s or even 100Gb/s are on the horizon, what should data center architects do? One option is to continue using copper-based links. However, with link rates higher than 10Gb/s, copper links cannot extend longer than several tens of meters, and 40Gb/s or 100Gb/s copper links are essentially multiple 10Gb/s or 25Gb/s cables combined together to provide high link rates in a parallel but costly and non-scalable fashion.

Another option is to use optical links that can support higher link rates and can reach longer ranges. However, if used with electrical packet switches, these optical cables have to connect to the these switches with optoelectronic transceivers at the ends of the cables. Each of these transceivers costs about 500 dollars on the market and consumes energy at about 240 mW [14]. When multiplying this cost by the large number of transceivers required in the network, using optical links with electrical switches in data centers presents a very expensive approach.

Alternatively, if the switches can also be optical themselves, the connections will no longer need the transceivers, and the cost of these transceivers can be saved.

2.3 Using Optical Switches in Data Centers

Ideally, to work with optical cables in data centers, one should use optical packet switches. However, although optical packet switches have been researched for about two decades, their expensive setup is still impractical to use in commercial use. One of the main technical problems is the lack of optical memories, and optical packets are hence hard to buffer [42]. As a result, optical switches favors circuit switching that does not require buffers, and has a feasible cost that commercial data centers can afford. Using 3D-MEMs and 2D-MEMs, these circuit switches reflect optical signals with mechanical mirrors that can reconfigure in milliseconds or even microseconds [40]. Using these optical circuit switches, data center researchers have proposed network architectures that partly replace the electrical switches at the aggregation layer with optical circuit shortcuts, like Helios [14] and c-Through [46], or even completely replace the interconnects among the top-or-rack switches with optical circuit links, like OSA [10].

These initial optical circuit networks uses circuits that can reconfigure in milliseconds, and can hence only deploy circuit links at the aggregation layer where traffic is either highly aggregated or highly stable. When two switches are connected with a circuit, the circuit works like a direct connected cable with no buffers. OSA [10] replaces the entire aggregation layer with circuits, and hence the routing and forwarding table must be setup correctly to provide full connectivity with possbily multiple hops, and the topology does not guarantee full bi-sectional bandwidth in general.

Helios [14] and c-Through [46] use a hybrid approach, where full connectivity is provided by an existing over-subscribed traditional aggregation layer of electrical packet switches, and the circuit links optimize bottleneck throughputs at the hot links among the switches. These hybrid approaches use circuits as a performance optimization for packet networks to provide cheaper interconnecting bandwidths. Similar approaches also appear

Table 2.1. Number of transceivers required *per upward-facing ToR port* for different network architectures. ([†]Presuming a 10-Gb/s packet network is already in place.)

Link rate	Full fat tree	Helios-like	Circuit-enabled ToR
10 Gb/s	$2-4$	$1-3$	N/A
100 Gb/s	4	3	1[†]

as reconfigurable wireless links in data center contexts [21, 26, 49].

These circuits are limited to serve as optimizations that augment the packet switches, or to serve only highly stable traffic, because the circuits take milliseconds or longer to reconfigure and are hence required to keep stable for seconds to achieve link efficiency. With this long scheduling period, a network controller has to reconfigure its topological view of the network and change the routing table correspondingly. The transport layer often needs to reestimate the link bandwidth due to the disruption of the topology change.

Recent research presents Mordia [12, 40], circuit switches that can reconfigure in 10s of microseconds, three orders of magnitude faster than previous ones. With this fast reconfiguration time, multiple transport layer flows that do not share the same destination can share the same circuit port in a TDMA fashion without sacrificing link efficiency and disturbing the transport layer. The switch reconfigures the circuits so fast that the circuits may be used by the nodes on demand without the need to change any forwarding table. This on-demand flexibility enables the switch to service traffic that is more dynamic, such as that found inside the ToR switches among the end hosts.

2.4 Optical Circuit Switching Saves Transceiver Costs

Extending optical circuit switching to end-hosts further saves the cost of transceivers for data center upgrades. As an estimation, consider data-center operators who want to upgrade an existing 10-Gb/s data center network—i.e., the part of the network that connects the top-of-rack switches together—to 100 Gb/s. Then, they need to upgrade the long cables into optical ones, with optical transceivers at the cable ends to connect to electrical switches. Table 2.1 shows the number of optical transceivers required for each upward-looking port of the ToR for three different network architectures. The first architecture is a fully provisioned 3-level fat-tree network [2]. If all of the links in the backbone network are optical, then this network requires four transceivers per upward port. In a Helios-like [14] architecture, an optical circuit switch is placed at the uppermost layer of the network, saving one transceiver per port as compared to the number used in a fat-tree network.

At 10 Gb/s, if the links between aggregation switches are short enough to be electrical, then transceivers may only be required between aggregation and core switches, potentially reducing the number of transceivers by up to three per port. At 100 Gb/s, however, while electrical interconnects may still be viable from an end host to a ToR (i.e., distances less than 5 meters), all connections from the ToR to the rest of the network are likely to be optical. Hence, either architecture will require a full compliment of optical transceivers. Moreover, to upgrade the network the operator will have to replace the existing 10-Gb/s transceivers with new 100-Gb/s transceivers.

A circuit-enabled ToR architecture, in contrast, deploys a 100-Gb/s circuit-switched optical network all the way down to the top-of-rack switches. As compared to the other two architectures listed in Table 2.1, it requires only one 100-Gb/s transceiver per upward-facing port of the ToR because optical circuit switching(OCS) does not use transceivers. As a result, for a fully provisioned three-level fat-tree network, if the per-port cost of the OCS used in this hybrid architecture is less than three times the cost of a 100-Gb/s optical *transceiver*, then the entire hybrid network will cost less than an equivalent 100-Gb/s packet-based network—even if the 100 Gb/s *switches* themselves were free. Larger networks require even more transceivers per end host: a five-level network requires eight transceivers to support each upward facing port, making the economics of this architecture even more compelling. Over-subscribed networks will use fewer transceivers in the core network, but the scaling trends are still applicable.

Where the example here uses 100-Gb/s circuits, the actual link rates for which a hybrid ToR architecture will be cost competitive with a fully provisioned or over-subscribed packet-switched network depend on market trends. Many OCS architectures are based on MEMs devices and can easily support link rates in excess of 100 Tb/s per port. The mirrors are typically reflective from approximately 1.3 μm to 1.6 μm, which corresponds to a bandwidth of approximately 400 THz. For this kind of device, the cost per optically switched bit is decreasing and is fundamentally inversely proportional to link rate. While the costs per switched bit of optical transceivers and packet switches are also decreasing, the rate of decrease is much slower. These trends imply the cost per switched bit will eventually become comparable at some link rate. What is less clear is the precise link rate when this crossover point will occur and the economic viability of a data-center network that supported such a link rate.

2.5 Data Center Networks Can Be Bufferless

One common concern on using circuit switching in data center networks is that circuit switches are inherently different from packet switches because they do not have buffers. This difference simplifies the switch architecture and fundamentally enables the switching in optics, but also enforces a network abstraction different from traditional Ethernet packet switching.

Traditional Ethernet switches use buffers to absorb congestion, and use packet drops to signal TCP when congestion appears so that the transport layer can converge to transmissions at the bottleneck link rate. This mechanism is necessary for wide-area networks, where network devices might be many hops away, connect in unknown or even changing topologies, and belong to different owners who might not cooperate on network management. With feedback from the buffer drops, TCP can converge to the bottleneck link rate without tight synchronization among different servers in the network. In the Internet, a common rule of thumb has been that at least a delay-bandwidth product is necessary to support TCP effectively [24].

However, in data center networks, all the machines in the network often operate under cooperating controllers, and tight synchronization becomes feasible. In fact, on the contrary, using traditional congestion control mechanisms might lead to sub-optimal link efficiency. For example, Appenzeller *et al.* [4] challenged the TCP buffer size assumption for core switches, and argue that for links carrying many TCP flows, less buffering is necessary. In the data center, Alizadeh *et al.* propose modifications to TCP that, along with appropriate switch support, can reduce the amount of buffering required down to a single packet per flow [3]. Other network technologies have also been created that reduce in-network buffering, including Myrinet [7] and ATM [32]. Numerous proposals for entirely bufferless "network-on-chip" (NoC) networks have been proposed [36], including hybrid NoC networks that also leverage packet switches [25].

All these studies show that data center networks often operate better with shallow or even zero buffers. Theoretically, if the network has a controller that can synchronize all the transmissions on all end hosts, no buffers would be required on the switches, and all congestion control can be enforced only by back pressure on the end hosts. Hence, it is plausible that data center can use buffer-less circuits as its main network switching technology, servicing most of the traffic in a TDMA fashion. In fact, work has shown that even packet switching networks can use TDMA to provide better network performance in data centers [39].

2.6 General Optical Circuit Scheduling is Hard

To tightly synchronize the traffic and deliver good network performance, circuit switching requires a circuit scheduler. In circuit switching, each input port can connect to only a limited number of output ports at a time. In this dissertation, I assume that the number of output ports per input node can connect is always one. Under this limitation, when multiple input ports want to connect to the same output port, and the circuits cannot satisfy the demand all at the same time, the scheduler decides which port connects first. Because the scheduler also determines the proportion of time each port is connected, a circuit scheduler also serves the role of a congestion controller.

The problem of circuit scheduling has been studied for decades [33, 34, 35], and circuit scheduling algorithms are even widely used inside packet switches. Many packet switches interconnect ports with electrical crossbar chips inside them, which are essentially equivalent to a set of circuits, but surrounded with large packet buffers in addition. These packet switches also often use similar types of circuit scheduling algorithms to provide non-blocking bi-sectional bandwidth at the crossbars.

A circuit scheduling problem for N end hosts can be formally described as decomposing the traffic matrix into a linear combination of permutation matrices. A permutation matrix is a matrix of dimension $N \times N$ where each element is either 0 or 1 and each row or column has exactly a single 1. Each permutation matrix represents a synchronized circuit configuration where a circuit connects an input port and an output port when the corresponding element in the matrix is 1. The coefficients in the linear combination for each permutation represent the proportional time duration that the configuration lasts in the schedule within a short scheduling window. The sum of the linear combination of the matrices should be no smaller than the traffic matrix at every element position in the matrix, which implies that the schedule can fully serve the traffic demand. The sum of the coefficients represents the scheduling window. When the scheduling window is one or more magnitudes smaller than the time granularity that the transport layer or the application layer need to discern between normal and error conditions, the circuit switch presents itself as a perfect crossbar that provides non-blocking full bi-sectional bandwidth.

Scheduling a circuit switch is hence a decomposition problem that searches for a set of permutation matrices and coefficients to cover the traffic matrix. For a switch of N end hosts, there are $N!$ possible permutation matrices as decomposition candidates, and exhaustively iterating all these candidates is an inefficient approach.

Some schedulers use algorithms based on maximum weighted matching [17], because finding a permutation matrix with the largest weight sum in a matrix is equivalent to searching a maximum weighted matching in a bipartite graph, which has algorithms that run in polynomial time. However, searching for the maximum weight matching does not lead to the best schedule, because such greedy matchings tend to pick a permutation with the maximum total weight, but the elements that this permutation covers might vary from very big ones to very small ones, and possibly some zeros. This variation makes assigning the coefficient—the time duration of the circuit configuration—difficult. To pick a meaningful time duration, the coefficient must be a positive number. However, assigning a circuit to an element that is zero in the matrix might be a waste of resources, because it takes a precious proportion of time in the scheduling window where a circuit could be assigned to other non-zero elements that share the same row or column with that zero element.

To fix the drawbacks of greedy algorithms like the ones that use maximum weighted matching, the scheduler needs to first "stuff" the traffic matrix into a doubly stochastic matrix by increasing the elements. A doubly stochastic matrix is a matrix which rows and columns all have the same sum. The scheduler would then use the decomposition of the stuffed matrix as the resulting circuit schedule. The stuffing here does not increase the maximum sum of each row and column, and the resulting schedule guarantees that it will service the origin traffic matrix because the original matrix is strictly smaller than the stuffed one, and the total time need to service the matrix remains unchanged. Further, after stuffing the decomposing matrix into a doubly stochastic one, it guarantees that there always exists a permutation matrix that can cover the input matrix, where each element that the permutation covers is positive if the input matrix is non-zero. Searching for such a permutation matrix is equivalent to finding a (non-weighted) perfect matching in a bipartite graph, where each edge in the graph corresponds to a non-zero element in the matrix. Existing algorithms can solve such perfect matching by searching augmented paths iterating over the input ports one by one [22]. After picking a permutation matrix, the scheduler will choose the minimum element among the ones that this permutation matrix covers in the input matrix and use its value as the coefficient for this circuit configuration. The served demand by the configuration can now be removed from the input matrix, where the input matrix remains a doubly stochastic matrix but with smaller sums for each row and column. This decomposition procedure repeats until the input matrix becomes zero. For each iteration in this procedure, at least one element in the matrix (the minimum one that the permutation covers) will fall to zero, and hence the decomposition will complete in $O(N^2)$ iterations for certain as it will clear all the elements. This algorithm is the standard Birkhoff-von-Neumann (BvN) decomposition algorithm [6], which generates perfect circuit configurations in polynomial time.

When decomposing a traffic matrix for a circuit switch or a packet switch crossbar in practice, a polynomial time algorithm often still takes too long to complete. Instead, faster approximating algorithms like iSLIP [33] are used. As a trade-off, these approximating algorithms require the crossbar to transmit at a speed often twice as fast as the link rate at the input and output ports to provide full bi-sectional bandwidth. In other words, they require a crossbar that has at least $2\times$ speedup.

Scheduling for optical circuit switches is different from traditional circuit scheduling for electrical packet switches in two aspects. First, optical signals are hard to buffer and also hard to run with speed-ups on the circuits. This restriction means that approximating algorithms are not suitable. Second, an optical circuit takes a non-trivial amount of time to reconfigure from one port mapping to another. Although the reconfiguration time is short enough to TDMA several transport-layer flows, frequent circuit reconfiguration can still lead to serious inefficiency on link utilization. This limitation means that the decomposition coefficient for each permutation matrix needs to be as large as possible, or in other words, the reconfigurations in a particular scheduling window should be as few as possible. This additional optimization goal further complicates the algorithm design where traditional BvN no longer generates the best solution. In fact, with $O(N^2)$ configurations, the link efficiency that the schedule provides is often unacceptable because a scheduling window is often not long enough even for merely reconfiguring the circuits for $O(N^2)$ times.

For optical circuit switches, the scheduling problem is hard in general, yet optical circuit switching cannot provide an efficient network without a good circuit schedule. Fortunately, in data centers, many traffic demands have common properties that favor circuit switching, and hence make it possible to schedule such traffic demands with circuits efficiently.

2.7 Data Center Traffic is Skewed and Bursty

Date center traffic often has characteristics that favor optical circuit switching. Studies of data-center traffic show that the traffic demand inside a data center is frequently concentrated, with a large fraction of the traffic at each switch port of a top-of-rack switch destined to a small number of output ports [27]. Such locality is not surprising, as

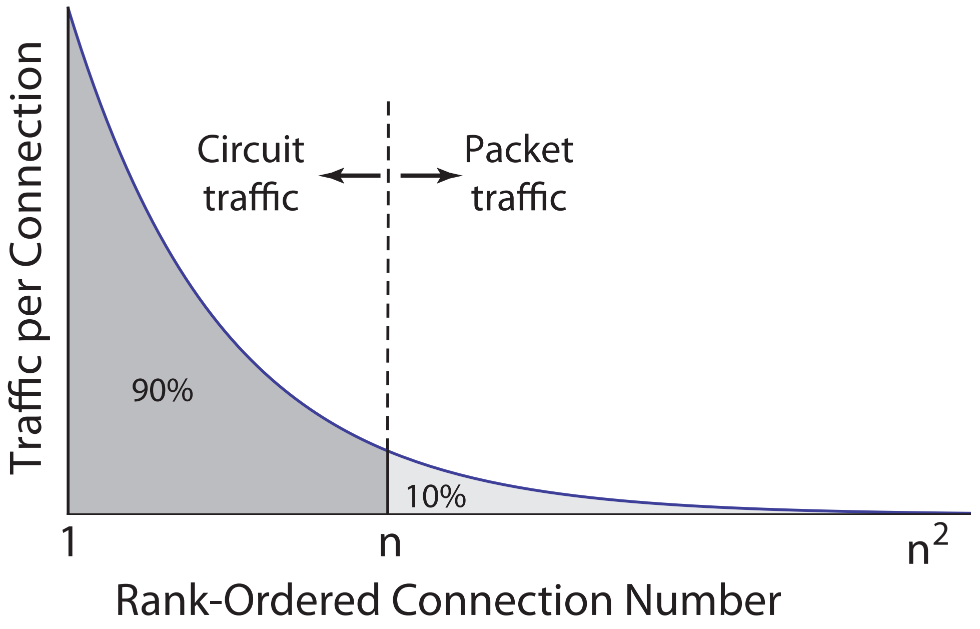


Figure 2.1. Rank-ordered traffic for each of the n^2 elements of a demand matrix, for which most of the traffic (e.g. 90%) is carried in a few ($O(n)$) flows.

application programmers and workload managers frequently use knowledge about the location of end hosts to coordinate workloads and minimize inter-rack traffic. Based on these empirical observations, researchers proposed hybrid architectures that classify the traffic into big flows that are long-lived and bandwidth-hungry, and small flows that are short-lived and often latency-sensitive. This observation can be expressed in terms of the n^2 rank-ordered elements of the demand matrix for a network that connects n nodes. Figure 2.1 shows an example where 90% of the inter-ToR traffic is carried by only $O(n)$ flows. In such settings, the demand matrix is frequently both sparse and stable [13, 21, 26], and this kind of traffic demand is generally suitable for a large port-count optical circuit switch.

While rack-level coordination can lead to bursty traffic at the upward looking ports of a top-of-rack switches, this dissertation carries this assumption one step further. Where previous network designs that use circuit switching focus on the core of the network, the design in this dissertation critically depends upon *individual hosts* being able to fill circuits assigned to them with data, which in turn depends on hosts transmitting groups of packets to the same destination ToR at fine time scales.

Previous work by Rishi Kapoor, *et al.* [28] verified this assumption. It measured individual flows, at microsecond granularity, emanating from a single host under a variety of workloads. It showed that host mechanisms, such as TCP segmentation offloading in the NIC and generalized segmentation offloading in the operating system, cause traffic to frequently leave the NIC in bursts of 10s to 100s of microseconds. In Section 3.3.1, I expand upon this analysis to show that circuit switching these flows can further enhance this behavior while not disturbing the transport protocol. For regimes in which circuit switching does not affect the transport performance of an end host, I say that its flows are "flying under the radar".

Because data center traffic is bursty and skewed at the ToR level, each row or column in the demand traffic would only has $O(1)$ number of big elements. If the circuit switch only service these large elements, the circuit scheduling problem is significantly simplified, can be solved in much shorter time.

Where the circuits cannot efficiently service rest of the elements in the traffic matrix that are small, these small elements fit well into a lower-provisioned packet switch which already exists in many data centers. And this service division of serving the traffic matrix leads to the hybrid network design that this dissertation proposes.

2.8 Hybrid Circuit Switches for Data Centers

In summary, data center networks benefit from using optical circuit switches from cost savings on the transceivers. Using circuit switching also matches the shallow buffer requirement in data centers networks. Observing that much data center traffic is skewed and bursty, I propose a hybrid switching architecture in this dissertation. It aims to service all the elements in the traffic matrix, providing almost non-blocking full bi-sectional bandwidth. It schedules the big elements with the circuit switch. Because in data center scenarios, these elements often form a sparse traffic matrix, the circuit matrix now can be scheduled and serviced by the circuit switch efficiently with high link utilization. The remained traffic can be serviced by the lower-provisioned packet switch with low latencies. By serving the big flows on the circuit links and the small flows on the packet switches, the overall hybrid architecture presents itself as a high-speed packet switching network for data centers, with performance comparable to a full bi-sectional bandwidth electrical packet switching network, but with much lower costs.

This chapter, in part, is a reprint of the material as it appears in the USENIX Symposium on Networked Systems Design and Implementation 2014. Liu, He; Lu, Feng; Forencich, Alex; Kapoor, Rishi; Tewari, Malveeka; Voelker, Geoffrey M.; Papen, George; Snoeren, Alex C.; Porter, George. The dissertation author was the primary investigator and author of this paper.

This chapter, in part, has been submitted for publication of the material as it may appear in the the conference of the ACM Special Interest Group on Data Communication 2015. Liu, He; Kapoor, Rishi; Tewari, Malveeka; Forencich, Alex; Zhang, Sen; Savage, Stefan; Voelker, Geoffrey M.; Papen, George; Snoeren, Alex C.; Porter, George. The dissertation author was the primary investigator and author of this material.

Chapter 3

REACToR: The Controller

This chapter presents a hybrid network architecture, REACToR. Its design consists of a 100-Gb/s optical circuit switch (OCS) network deployed alongsied a pre-existing lower-provisioned 10-Gb/s electrical packet switch (EPS) network, and provides 100-Gb/s packet-switch-like performance at substantially lower cost than an entirely packet-switched network.

REACToR's design is based on two key insights. The first is that it is impractical to buffer incoming traffic bursts from each end host within the ToR's switch memory. For a traditional in-switch time-division, multiple-access (TMDA) queueing discipline, this architecture would require a dedicated input buffer for each potential circuit destination. Given the unpredictable nature of the end-host network stack [28], these buffers would likely need to be quite large.

Instead, REACToR buffers bursts of packets in low-cost end-host DRAM memory until a circuit is provisioned, at which point the control plane explicitly requests the appropriate burst from each end host using a synchronous signaling protocol that ensures that the instantaneous offered load matches the current switch configuration. Because each REACToR is dual-homed to an EPS, the control plane can simultaneously schedule the latency-sensitive traffic over the packet switch. The packet switch can also service unexpected demand due to errors in demand estimation or circuit scheduling.

The second insight is that if circuit switching is sufficiently fast, then delays due using flow-level circuit-switched TDMA at the end-host network stack will not degrade the performance of higher-level packet-based protocols; in a sense the circuit switch will "fly under the radar" of these end-host transport protocols. As technology trends enable faster OCS reconfiguration times, this hybrid architecture blurs the distinction between packets and circuits. By combining the strengths of each switching technology, a hybrid network can deliver higher performance at lower cost than either technology alone, even at the level of a ToR switch.

We evaluate our design for a 100/10-Gb/s OCS/EPS hybrid network using a scaled-down 10/1-Gb/s hardware prototype that supports eight end hosts. The prototype consists of two FPGA-based REACToRs with four downward-facing 10-Gb/s ports each. Both REACToRs connect to the Mordia [40] microsecond OCS and a commodity electrical packet switch. The circuit switch supports a line rate of 10 Gb/s while the packet switch is rate limited to 1 Gb/s to enforce a 10:1 speed ratio. End hosts connect to our prototype using commodity Intel 10-Gb/s Ethernet NICs that we synchronize using standard 802.1Qbb PFC signaling.

Our experiments show that our REACToR prototype can provide packet-switch-like performance by delivering efficient link utilization while reacting to changes in traffic demand, and that its control plane is sufficiently fast that changes in circuit assignment and schedule can be made without disrupting higher-level transport protocols like TCP. Using simulation of more hosts, we also illustrate the large benefits that a small underprovisioned packet switch provides to a hybrid ToR relative to a pure circuit ToR. We conclude that REACToR can service published data-center demands with available technology, and can easily scale up to make effective use of next-generation optical switches and 100-Gb/s hosts by reusing an existing 10-Gb/s electrical packet-switched network fabric.

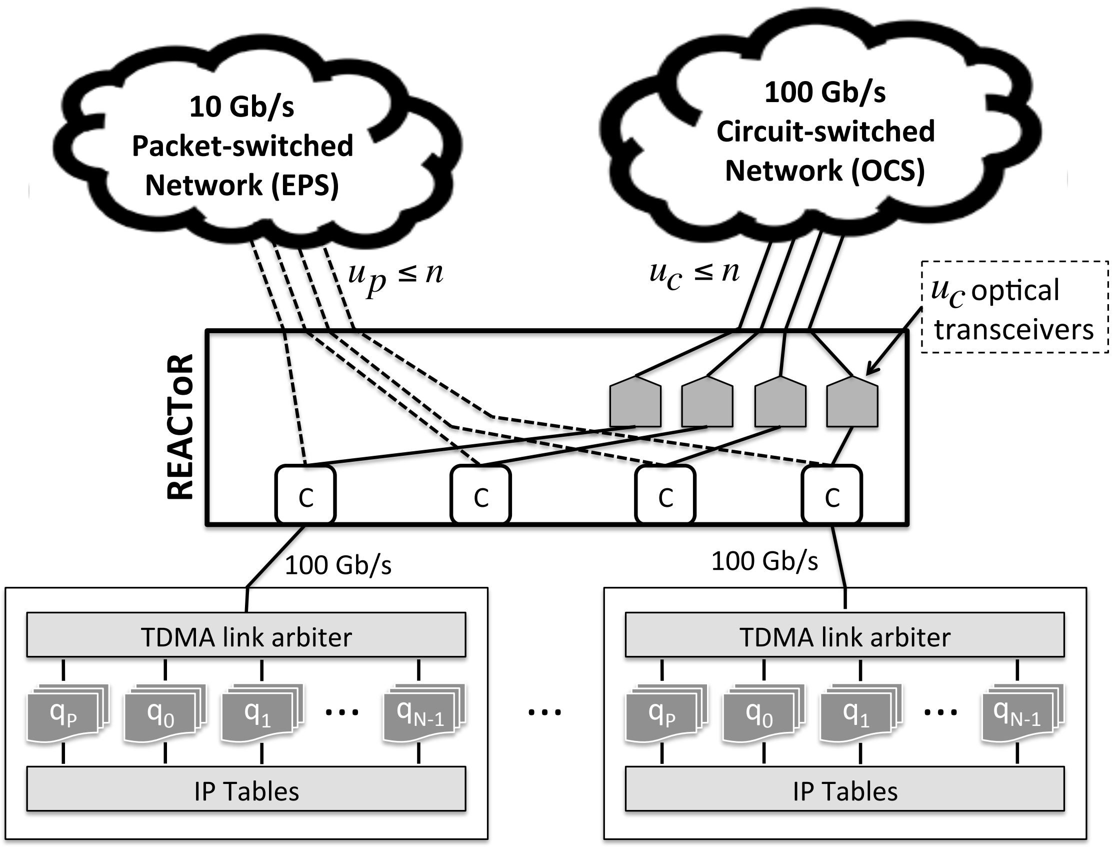


Figure 3.1. 100-Gb/s hosts connect to REACToRs, which are in turn dual-homed to a 10-Gb/s packet-switched network and a 100-Gb/s circuit-switched optical network.

3.1 Design

A REACToR-enabled data center consists of N servers grouped into R racks, each consisting of n nodes. We assume that a preexisting 10-Gb/s packet-switched network is already deployed within the data center. Overlaid on top of this packet-switched network is an additional 100-Gb/s circuit-switched network. At each rack is a hybrid ToR called a REACToR, which is connected to the packet-switched network with $u_p \leq n$ uplinks and is connected to the circuit-switched network with a separate set of $u_c \leq n$ uplinks. The packet-switched network supports $R \times u_p$ ports, and the circuit-switched network supports $R \times u_c$ ports. Each REACToR has n downward-facing 100-Gb/s ports to its n local servers. In this dissertation, we consider the fully provisioned case where $u_p = u_c = n$; however, additional cost savings are possible when either or both of u_p and u_c are less than n. Our architecture is agnostic to the particular technology used to build the circuit-switched fabric, but, given technology trends, we presume it is optical.

Referring to Figure 3.1, an (n, u_p, u_c)-port REACToR consists of n downward-facing ports connected to servers at 100 Gb/s, $u_p = n$ uplinks connected to the packet-switched network at 10 Gb/s, and $u_c = n$ uplinks connected to the 100-Gb/s circuit-switched network. At each of the n server-facing input ports, there is a classifier (labeled 'C' in the figure) which directs incoming packets to one of three destinations: to packet uplinks, to circuit uplinks, or through an interconnect fabric to downward-facing ports to which the other rack-local servers are attached. There is no buffering on the path to the packet uplinks, as buffering is provided within the packet switches themselves. There is also no buffering on the path to the circuit uplinks; instead, packets are buffered in the end-host where they originate. When a circuit is established from the REACToR to a given destination, the REACToR explicitly pulls the appropriate packets from the attached end-host and forwards them to the destination.

REACToR relies upon a control protocol to interact with each of its n local end-hosts to: (1) direct the end host to start or stop draining traffic from its output queues (which we refer to as *unpausing* or *pausing* the queue, respectfully), (2) set per-queue rate limits, (3) provide circuit schedules to the end-host, and (4) retrieve demand estimates for use in computing future circuit schedules. We first motivate the need for this functionality by describing the various other aspects of REACToR's design before detailing the host control protocol in Section 3.1.4.

3.1.1 End-host buffering

Each end-host buffers packets destined to the REACToR in its local memory, which is organized into traffic classes, one per destination ToR, with an additional class for packets specifically destined for the EPS (e.g., latency-sensitive requests). Each traffic class has its own dedicated output queue (i.e., $\{Q_0, Q_1, ..., Q_{N-1}\}$), with an additional queue for the EPS class, Q_P, as shown in Figure 3.1. At any moment in time, the REACToR can ask an end host to send packets from at most two classes: one forwarded at line rate to an OCS uplink (or local downlink port), and another forwarded to an EPS uplink. This latter class of traffic must be rate limited at the source NIC to conform to the link speed of the EPS to prevent overdriving the EPS. In the reverse direction, the EPS may emit packets into the REACToR at its full rate to a particular downward-facing port. Because that downward-facing port could potentially be shared by incoming line-rate circuit traffic heading to the same destination, REACToR must further ensure that the circuit traffic is sufficiently rate-limited so that there is enough excess capacity to multiplex both flows at the destination. Hence, end hosts will be directed by REACToR to similarly rate-limit traffic classes destined to the OCS at the source NIC, but at much higher rates. Further details on rate limiting are provided in Section 3.1.3.

Today, end-host NICs support modest amounts of buffering, on the order of a few megabytes. However, it is not organized in a way that can be directly used to support circuits. NICs partition their buffers into a small set of 8 to 64 transmit queues, which the OS uses to batch and store packets waiting to be sent. The scheduling policy for these queues is typically built into the NIC (e.g., round robin), so the actual transmit time of individual packets is outside the control of the OS.

To achieve high circuit utilization in REACToR, the NIC needs the ability to send data for a particular circuit destination to the ToR as soon as a circuit becomes established, and to fill that circuit continuously until it is torn down. At any one time, each circuit uplink within a REACToR is exclusive to a particular source port (attached end host), so efficiency degrades any time that source has no data to send. Thus, packets headed to the same circuit destination (i.e., remote host) should be grouped together within a host's memory, so that when a circuit to that destination becomes available, that group of packets can be sent from the NIC to the REACToR at line rate.

Within each host, we define a traffic class per destination host, and task the OS with classifying outgoing packets into the appropriate class based on, e.g., the destination IP address. REACToR then uses the host control protocol to pause and unpause end-host queues. In this model, the role of the OS and of the NIC changes somewhat: rather than the OS "pushing" packets to the NIC buffers based on queuing policies in the host, the NIC is responsible for "pulling" packets from the host memory into the NIC buffers according to the circuit schedule just in time to transmit them to the connected circuit. (We note that the NIC design advocated by Radhakrishnan *et al.* [41] would be especially well suited for this model.)

Demand estimation. Over a short time scale (i.e., 100s of µs, depending on the size of the NIC buffers), the occupancy of these traffic classes defines the imminent end-host demand because the packets in these queues have already been committed to the network by the OS. It is possible to query the OS, the application, or even a cluster-wide job scheduler to form longer-term demand estimates. For example, Wang *et al.* [46] use TCP send buffer sizes as estimates of future demand. Our prototype uses a demand oracle. In any case, the circuit scheduler uses these demand estimates to determine a set of future OCS circuit configurations.

3.1.2 Circuit scheduling

To make effective use of the capacity of the circuit switch, REACToR must determine an appropriate schedule of circuit switch configurations to service the estimated demand over an accumulation period W. This task is the responsibility of a logically centralized, but potentially physically distributed, circuit scheduling service, which implements a hybrid circuit scheduling algorithm. This service collects estimates of network-wide demand, in the form of an $N \times N$ matrix D. The service computes a schedule, P_k, of m circuit switch configurations, which are permutation matrices and their corresponding duratinos ϕ_k. A permutation matrix is a matrix of 0s and 1s in which each row and column has and only has a single 1.

The number m of configurations that comprise the schedule is constrained because each circuit configuration requires a finite reconfiguration time δ, during which time no data can be forwarded over the circuit switch. When δ is large with respect to W, it is more efficient to use fewer configurations. When δ is small with respect to W, more configurations can be used. Including this reconfiguration delay, the duration of the schedule is constrained by the length of the accumulation period so that $\sum_{k=1}^{m} \phi_k + \delta m \leq W$. The goal of the scheduling algorithm is to maximize $\min(D, \sum_{k=1}^{m} P_k \phi_k)$ subject to these constraints.

Obviously, if the switch introduces a reconfiguration delay, then it is impossible to service fully saturated demand at line rate. Existing research in constrained scheduling has focused on switches that run faster than the link rate, with the ratio of the switch rate to the link rate called the speedup factor. These algorithms [15, 30, 48] produce a variable-length schedule which is dependent on the actual reconfiguration delay.

Hybrid networks in general, and REACToR in particular, do not use a speedup factor. Instead, REACToR uses the lower-speed packet switch as a way to make up for the reconfiguration delay and any scheduling inefficiency. This “back channel” is a key distinction between REACToR and traditional blocking circuit scheduling because REACToR continues to service a subset of flows over the EPS when circuits are not available, thereby increasing support for latency sensitive workloads.

We will further discuss the selection and evaluation of an circuit switch algorithm in Chapter 4; in this chapter we compute the schedule offline using a variant of existing constrained switching algorithms based on a predetermined demand matrix D. Any schedule computed for use in REACToR, however, is subject to a number of constraints.

Class constraints. To ensure the offered load can be effectively serviced by the ToR, REACToR imposes a number of constraints on the set of queues that can be unpaused at any particular time.

First, the dedicated EPS queue (Q_P) is always unpaused but rate-limited to at most 10 Gb/s, providing the host with the ability to send latency-sensitive traffic directly to the EPS at any point in time.

Second, at most one additional queue can be unpaused at any one time for transmission at (near) link rate (i.e., 100 Gb/s). When such circuit-bound (or rack-local) traffic arrives at an input classifier in the REACToR, it is directly forwarded to the appropriate circuit uplink (or downward-facing port) without any intermediate buffering.

The third constraint is that, if a queue is unpaused for link-rate transmission in the current scheduling period, then it should never be unpaused for transmission to the EPS. This constraint serves two purposes: it prevents the EPS from being burdened with high-bandwidth traffic better served by circuits, and it gives that traffic class additional time to accumulate demand so that the circuits are highly utilized.

Fourth, any traffic class which is not assigned to a circuit (or downward port) during a scheduling period is instead remapped to the EPS, meaning that any packets in that class’s queue are routed to the EPS uplink. Finally, all of the queues corresponding to

EPS-bound traffic (i.e., both the dedicated EPS queue and and any classes not scheduled for a circuit in this period) must be rate limited such that the sum of their limits is less than or equal to the EPS link rate (e.g., 10 Gb/s).

3.1.3 End-host rate limiting

At any given time, each of the REACToR's downward-facing server ports can transmit data from two sources: a circuit from a single source established through the OCS (or rack-local connection) fabric, and traffic from any number of sources forwarded through the EPS. At each downward-facing port there is a multiplexer which performs this mixing. When the sum of bandwidth from the EPS (B_{EPS}) and OCS (B_{OCS}) exceeds the rate of the REACToR port (B_{ToR}), then without intervention, $(B_{\text{EPS}} + B_{\text{OCS}}) - B_{\text{ToR}}$ traffic would be dropped. To prevent such drops, and to ensure high overall utilization, we rely on end-host rate limiting.

The first way that we use end-host rate limiting is to ensure that in steady state, $B_{\text{EPS}} + B_{\text{OCS}} \leq B_{\text{TOR}}$. Since the OCS is bufferless, the multiplexer gives priority to packets arriving from the OCS because otherwise they would have to be dropped. Assuming a REACToR with a 100-Gb/s OCS and 10-Gb/s EPS as an example, each end host would rate limit its circuit-bound traffic in the range of 90–100% of the link capacity to leave sufficient head room for the EPS traffic, based on the estimate of EPS demand in the current schedule. Each time that a set of configurations for a scheduling period is computed, a rate limit is also computed per configuration, reflecting the estimated load from the EPS. Note that this estimate need not be perfect, and in fact we expect the EPS to absorb inaccuracies in scheduling, demand estimation, and rate limiting. For each scheduling interval, the associated rate limits are computed and sent to each end-host via the host control protocol.

The circuit rate limit also serves a second purpose, which is providing statistical

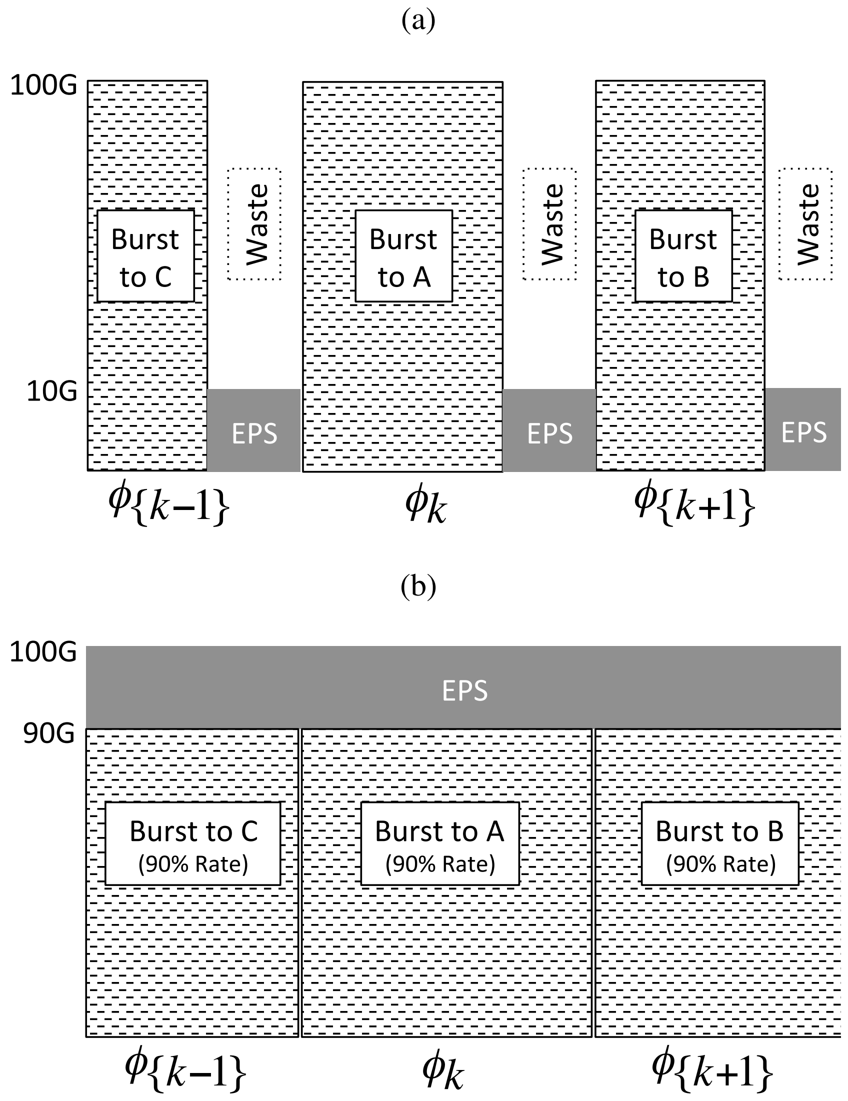


Figure 3.2. Rate limiting prevents bursts from the OCS from starving the EPS, which would otherwise be unable to make full use of each circuit-switch configuration interval ϕ_k. In both cases, the circuit-switched traffic achieves 90 Gb/s during each interval.

(a) When a circuit-switch configuration interval ϕ_k begins, queued traffic forms bursts which saturate the link during the first part of the configuration, leaving capacity for EPS traffic at the end of ϕ_k; since the EPS runs at a fraction of the line rate, it cannot efficiently use the remaining time. (b) By rate limiting circuit traffic, the EPS can spread its traffic out over the entire configuration interval.

multiplexing at the downward-facing REACToR port. Underpinning the design of REACToR is the assumption that on short time scales, traffic emanating to a single destination is bursty. Each burst by definition consists of a number of packets sent back-to-back. From the point of view of the REACToR port multiplexer, this means that, absent other controls, during the first portion of a given circuit-switch configuration interval ϕ_k, the entire port's bandwidth would be dedicated to servicing a single burst of traffic from the OCS. Thus, any packets originating from the EPS would be delayed until the end of ϕ_k. Figure 3.2 (a) shows a pictorial representation of this behavior. The challenge that arises is that the line rate of the EPS is presumed to be lower than the REACToR port speed and the OCS. Hence, the open region at the end of ϕ_k can only be filled with packets at the rate of the EPS (e.g., 10 Gb/s) instead of the OCS (100 Gb/s). Thus, for this example, the region at the end of ϕ_k only gets 10% utilized since the EPS can only drive 10% of the outgoing port bandwidth.

Instead, REACToR seeks to ensure that the circuit traffic is spread out across ϕ_k by limiting it to less than full line rate (e.g., 90 Gb/s of a 100-Gb/s link). Rate limiting over time allows the EPS-serviced traffic to be multiplexed on REACToR's downward-facing ports at a uniform rate across all configuration intervals ϕ_k, enabling the entire interval to be utilized by both circuit traffic and packet traffic. By setting circuit rate limits in the end host, as described above, the traffic headed to the circuit is paced to the appropriate rate. Figure 3.2 (b) shows the resulting treatment of circuit and packet data within that same configuration interval ϕ_k.

3.1.4 REACToR host control protocol

An instance of the REACToR host control protocol runs between each end-host and its REACToR switch. REACToR uses the protocol to retrieve demand estimates collected by end-hosts, to set per-queue rate limits, as described above, and to convey impending schedules to the end host from the circuit scheduler. These functions are relatively straight forward. In this section, we examine the fourth use of the host control protocol: managing end-host traffic classes and buffering. The key to achieving efficient use of the hybrid network is being able to drain the appropriate classes with fine-grained precision at the right times. We now describe the host control protocol that achieves this precision.

Overview: To ensure reliable transmission, we cannot reconfigure the OCS until all incoming circuit traffic has ceased, since the OCS is unable to carry traffic during the time δ when it is being reconfigured. While classifiers on each REACToR input port can shunt all traffic to the EPS nearly instantaneously, in general we would like to ensure that almost all circuit-bound traffic has been paused before reconfiguring the OCS. Otherwise, a massive queue would build up at the EPS at the end of each schedule. To avoid this buildup, we leverage the 802.1Qbb Priority Flow Control (PFC) protocol to pause traffic at the end host. Each traffic class in the end host corresponds to a PFC class.[1] At the end of each schedule, for each attached host, the REACToR first sends a PFC frame to pause the traffic class destined for the current schedule's circuit (if any). Note that PFC frames are selective, so traffic destined to the EPS will continue to flow while the OCS is being reconfigured. Once inbound circuit traffic has ceased, the OCS can be reconfigured. After reconfiguration, the traffic class corresponding to the next schedule's circuit can be enabled by a PFC unpause frame.

Performance: The overall speed of the control plane is bounded by the speed at which REACToR can pause and unpause traffic classes buffered at the end hosts. Because the PFC frame must be both received and processed at the NIC before traffic stops, there will be some delay between when the controller wants to pause traffic and when the traffic

[1]Although the current PFC specification is limited to eight frame priority levels, it is possible to reuse classes across schedule periods by recoloring.

finally stops arriving at the incoming ports at the REACToR. To quantify this delay, we extended the classifiers on our prototype to timestamp all incoming packets and mirror these timestamped packets to a collection host. We then measured the time from when the classifier sends a PFC frame to a host until it stops receiving packets from that host.

We measured the minimum (maximum) delay on an Intel 82599-based 10 Gb/s NIC as 1,014 (2,188) ns, with the actual delay varying as a function of PFC offset, meaning that if the PFC frame arrives more than 185.6 ns after the start of the current frame, the NIC will generate an additional frame before pausing, likely due to pipelining within the NIC implementation. Once the OCS has established a circuit and is ready to receive traffic, the REACToR needs to restart traffic for the newly connected destination by sending another PFC frame. The measured 'on' delay (i.e., from when the configuration is started by the transmission of a PFC frame unpausing the traffic) ranges between 1.2 µs and 1.3 µs. From the 'off' delay measurement, it is clear that we can hide the first microsecond of delay by sending the PFC frame before we actually want the traffic to stop, but it may take an additional 1.3 µs for all ports to cease sending. There is one additional source of delay: a port may be busy sending an outgoing packet at the moment the classifier wishes to send the PFC frame. This delay is bounded by the 1500-byte MTU in our prototype, leading to a worst-case combined delay of approximately 2.5 µs, which is the lower bound of the speed of the control plane achievable in REACToR with 10-Gb/s end hosts, a 1500-byte MTU size, and the 802.1Qbb implementation on our NIC.

3.2 Implementation

To evaluate our design, we have implemented two prototype 4-port 10-Gb/s REACToRs (shown in Figure 3.3) using two FGPAs, a Fulcrum Monaco 10-Gb/s electrical packet switch, and the Mordia microsecond OCS [40]. Mordia is 24-port reconfigurable

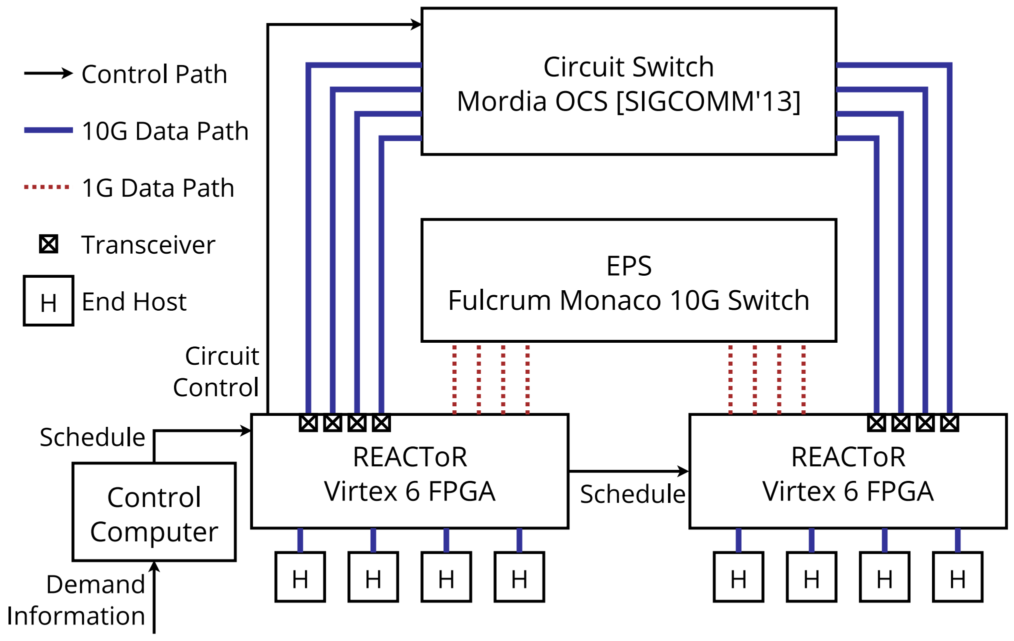


Figure 3.3. The prototype REACToR network.

OCS built from six 4-port “binary MEMs” wavelength-selective switches, with an average reconfiguration delay of $\delta = 12\mu s$, which includes the physical switching time of the MEMs devices and the time to reinitialize the attached 10-Gb/s transceivers. Thus, our REACToR prototype supports the same 10:1 bandwidth ratio described earlier, but at 10 Gb/s (OCS) and 1 Gb/s (EPS) rather than 100/10 Gb/s.

Each REACToR is implemented with a HiTech Global HTG-V6HXT-100GIG-565 FPGA development board as shown in Figure 3.4. Each FPGA board supports 24 ports of 10-Gb/s I/O. The circuit scheduling service runs as a user-level process on a dedicated Linux-based control server, and transmits schedules to the FPGA via a dedicated 10-Gb/s Ethernet connection. In our implementation, the end hosts are servers equipped with Intel 82599-based NICs. The end hosts classify traffic according to the destination using the Linux `tc` facility. The classifier on the FPGA selectively enables or disables packets to a given destination using the IEEE 802.1Qbb priority-based flow control standard, which supports eight flow classes. We use seven of these classes to

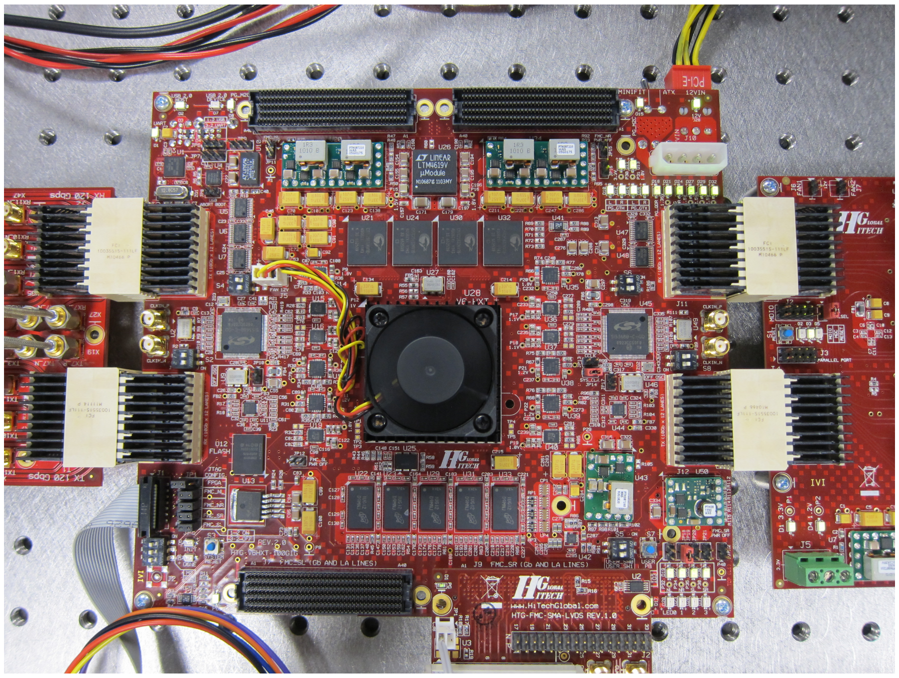


Figure 3.4. A HiTech Global HTG-V6HXT-100GIG-565 FPGA development board used in REACToR

correspond to the *n* circuit destinations reachable from a REACToR, and the eighth is reserved for the EPS-dedicated class.

At each switch reconfiguration, the controller on the FPGA updates the OCS and enables the corresponding end-host traffic classes using 802.1Qbb PFC frames. The controller also configures the classifiers so that they forward the appropriate line-rate flow to the circuit uplink, and forward the remaining traffic to the EPS.

Circuit switch characterization: The average reconfiguration delay for the Mordia switch is approximately 12 µs, with a maximum observed delay of 14.84 µs (as shown in Figure 3.5). The transceivers we use vary in their “lock” time, necessitating setting a more conservative reconfiguration delay. This variance is an engineering artifact of our hardware and is not fundamental; the IEEE 802.3av (10G-EPON) specification, for instance, calls for a 400-ns lock time. Except as noted, in the experiments that follow, we configure REACToR to assume a 30-µs reconfiguration time which, contained within

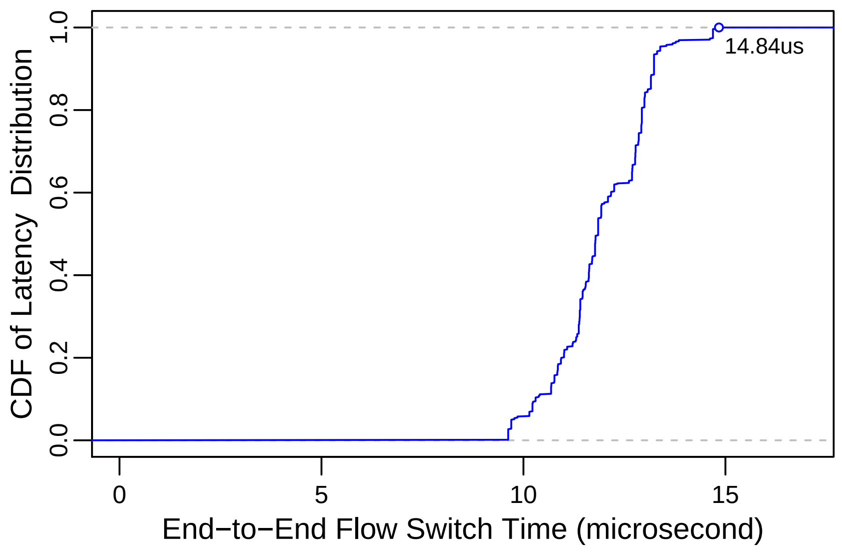


Figure 3.5. Observed end-to-end circuit switch reconfiguration delay δ.

at least a 160-µs configuration period, delivers at least 81% link efficiency.

REACToR host control protocol: To tightly time synchronize the attached hosts, REACToR sends the schedule to each attached host using two UDP packets. The first packet contains the impending schedule for the upcoming 3-ms period, whereas the second packet indicates the start of the 3-ms time period, serving as a precise periodic heartbeat. End hosts receive these packets in a kernel module via the `netpoll` kernel APIs, which reduces the delay in acting on them to less than 15 µs.

3.3 Evaluation

In this section, we evaluate the performance of our REACToR prototype implementation. We first show that, with buffering and scheduling packets at end-host NICs, circuit-switching does not negatively impact TCP throughput. Second, we show that the REACToR can dynamically update and switch schedules of many flows without impacting throughput. Third, we show that REACToR can serve a time-varying workload that consists of multiple high- and low-bandwidth flows, promoting flows as appropriate from the packet-switched fabric to the circuit-switched fabric. Finally, we use simulation to illustrate the large benefits that a small underprovisioned packet switch provides to a hybrid ToR.

To generate arbitrary traffic patterns, we implemented a Linux kernel module based on `pktget` [37] that can send MTU-sized UDP packets at arbitrary rates up to line rate. When the module is sending, it runs on a dedicated core and each packet it sends has a sequence number. At the same time, the module also serves as a traffic sink that receives UDP traffic via the `netpoll` kernel interface, and records the sequence number and source address of packets. For packet timing measurements, we configured the FPGA to generate a record for each packet that captures the source, destination, and a timestamp with 6.4-ns precision. The prototype sends these records out-of-band to a collection host using one of the 10 Gb/s ports of the FPGA, which we then process offline.

3.3.1 TCP under TDMA scheduling

In Section 2.7, we described how application flows exhibit intrinsic short-term correlated bursts as a consequence of the NIC trying to efficiently use the link. We therefore consider how flows behave in a hybrid fabric where a circuit scheduler pauses flows at the host while they wait for an assigned circuit and unpauses them when the circuit is established. While its flow is paused, an application may generate additional packets, increasing the size of its burst when its flow is eventually unpaused and thereby more efficiently use its circuit. However, the increased latency and latency variation induced by pausing and unpausing flows may detrimentally impact the transport protocol (e.g., TCP) or the application itself.

To study the impact of circuit scheduling on TCP throughput, we generate stride

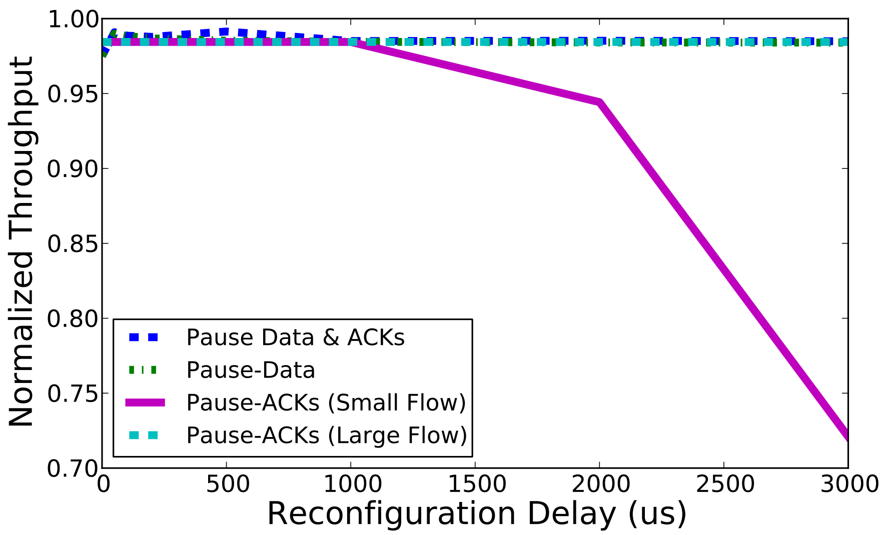


Figure 3.6. Effect of pausing/unpausing data/ACK packets on TCP throughput.

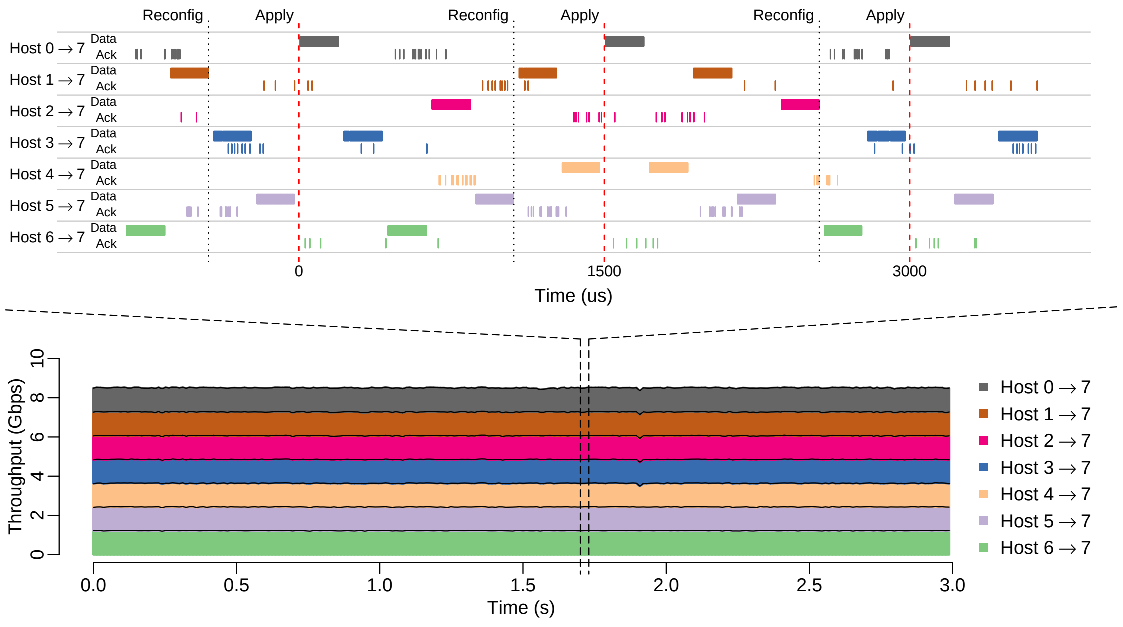


Figure 3.7. All-to-all workload with circuit configurations changing every scheduling period.

workloads where a single host sends to another host, and at the same time sinks a TCP flow from a third host. First we consider the case where we pause and unpause a bi-directional circuit, i.e., pause both data and TCP ACKs at the same time. Next we consider the case where we pause the data in the flow, but allow ACKs to return unimpeded (e.g., via the EPS). Finally, we consider the case where we pause the ACKs, but enable data packets to transmit unimpeded.

Figure 3.6 shows the resulting normalized throughput when varying the reconfiguration delay δ for a stride workload with eight hosts. In the first case, the normalized throughput of uni-directional and bi-directional circuits is close to ideal, showing that pausing data packets on the end hosts does not affect throughput for pause lengths considered by REACToR. When pausing only the ACKs, we find that there are two regimes to consider. During slow start ('Small Flow'), pausing ACKs decreases the overall throughput of the flow—up to 30% for 3-ms delays. For shorter delays (e.g., $\leq$1 ms) there is no detectable effect for pausing ACKs. Once the flow leaves slow start ('Large Flow'), there is no effect on throughput regardless of the reconfiguration delay.

These experiments consider the effect of circuit scheduling on TCP traffic in the absence of packet loss. In practice, packets may be lost for a variety of reasons. We repeated the experiments above where each end host drops packets uniformly at random with a configurable drop probability. While TCP throughput suffers as expected with increasing drop rates, the difference in performance with and without circuit scheduling (e.g., with and without issuing PFC pause frames) is insignificant for steady state loss rates up to 1%.

3.3.2 Switching "under the radar"

Next we evaluate the speed and flexibility with which REACToR can be reconfigured. We first run an all-to-all workload on eight hosts, where every host streams a TCP flow to each of the other seven hosts using all available bandwidth. To serve this workload, we load REACToR with a schedule of seven TDMA periods that fairly shares the links among all the flows. Each schedule period is 1.5 ms, within which each host sends and receives from each other host for 214.3 µs (including a 30-µs circuit reconfiguration delay) in each circuit configuration. We schedule all data packets via the circuit switch, and all TCP ACKs via the packet switch. We could use the same schedule for every period, but to further exercise our prototype we change the schedule so that hosts receive circuits in different permutations in each period.

Figure 3.7 shows three seconds of an all-to-all workload where flows start at the same time on the hosts. The bottom part shows the achieved throughput as reported by one of the hosts: the flows from the other seven hosts evenly split the available bandwidth. Total TCP goodput received is 8.1 Gb/s, the maximum given the 86% duty cycle resulting from the 30-µs reconfiguration delay in a 214.3-µs circuit.

At the application level, the achieved TCP goodput maximizes network capacity and is stable over time. However, if we zoom in and look at the packet traces, as shown in the top part of the figure, we can see the fine-grained behavior of scheduling the flows on circuits. A control packet triggers a new schedule each period, which the controller sends to the REACToR during the previous period (at the time marked ‘Reconfig’) and the switch loads just before the new period starts (‘Apply’). The schedule partitions each period into seven circuit configurations, one for each of the seven hosts sending to the host we are observing.

At time offset zero, for instance, host 0 has the first configuration in the schedule. Its data packets arrive over the circuit it receives, and no other host can send data packets through the circuit switch to host 7. The second configuration schedules host 3, and so on. ACKs received at host 7 use the packet switch, and hence can overlap circuits scheduled for other hosts. (The flow assignments are asymmetric; when host 0 is sending to host 7

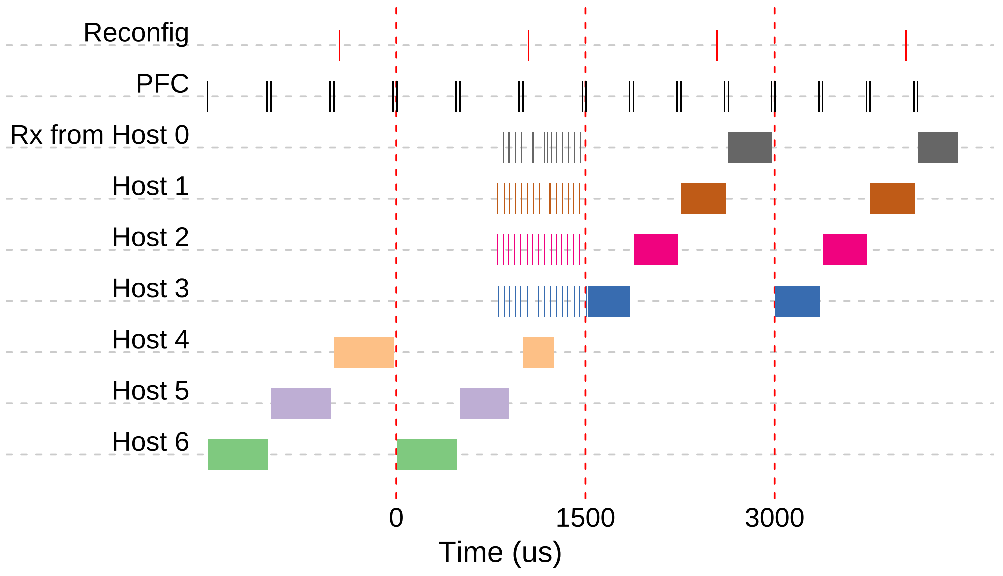


Figure 3.8. Changing the number and duration of configurations in scheduling periods.

at time zero, host 7 is sending to host 6 and receiving ACKs from it.)

This all-to-all workload does not vary demand over time. Given the frequency with which we can reconfigure the circuit switch, we can also serve time-varying workloads by serving different workload demands under different scheduling times with different numbers of configurations and circuit assignments.

We use another experiment to demonstrate this flexibility. We divide the eight hosts into two groups: G_A consists of hosts 0–3, and G_B hosts 4–7. We then generate traffic among the hosts using two workloads. The first is a group-internal all-to-all, where each host streams UDP packets to the other three hosts in its group at the maximum possible rate. To serve this workload, REACToR uses a schedule that has three configurations in a scheduling period. The period lasts 1,500 µs, and each configuration lasts 500 µs (including a 30-µs reconfiguration delay). The second is a cross-talking all-to-all workload where each host in G_A streams to all the other four hosts in G_B, and vice versa. For this workload, REACToR uses schedules with four configurations. These scheduling periods also last 1,500 µs, but each configuration lasts 375 µs (again including a 30-µs reconfiguration delay).

In the experiment, we change from the group-internal to the cross-talk workloads midway through, loading the REACToR with correspondingly different schedules. Figure 3.8 shows the incoming packets to host 7 around the workload transition time. We controlled the experiment so that the workload changes at an inconvenient, but more realistic, time for REACToR: *during* a scheduling period, at time 750 µs on the graph. REACToR's schedules commit the switch based on predicted demand, and workloads are apt to change their demand independent of when REACToR can conveniently accommodate them. At this workload transition, REACToR is halfway through its scheduling period and packets already queued at the first three hosts continue to arrive via circuits. Overlapping these flows, the other four hosts start sending packets to host 7. These hosts do not have circuits, so the packets arrive via the EPS at a much lower rate.

At the end of its committed scheduling period (time 1,500 µs), REACToR can then react to the workload transition and schedule circuit configurations that match the workload. At this time, host 7 changes from receiving packets in 500-µs configurations, scheduled round-robin from hosts 4–6, to receiving packets in 375-µs configurations from hosts 0–3.

In summary, these experiments demonstrate the speed and flexibility with which REACToR can reconfigure its circuit schedules given a known demand. Applications achieve their expected goodput at a high level, while individual flows are paused and released at fine time scales when their circuits are scheduled. Further, REACToR can adjust the circuit schedule to adapt to changes in application behavior and demand.

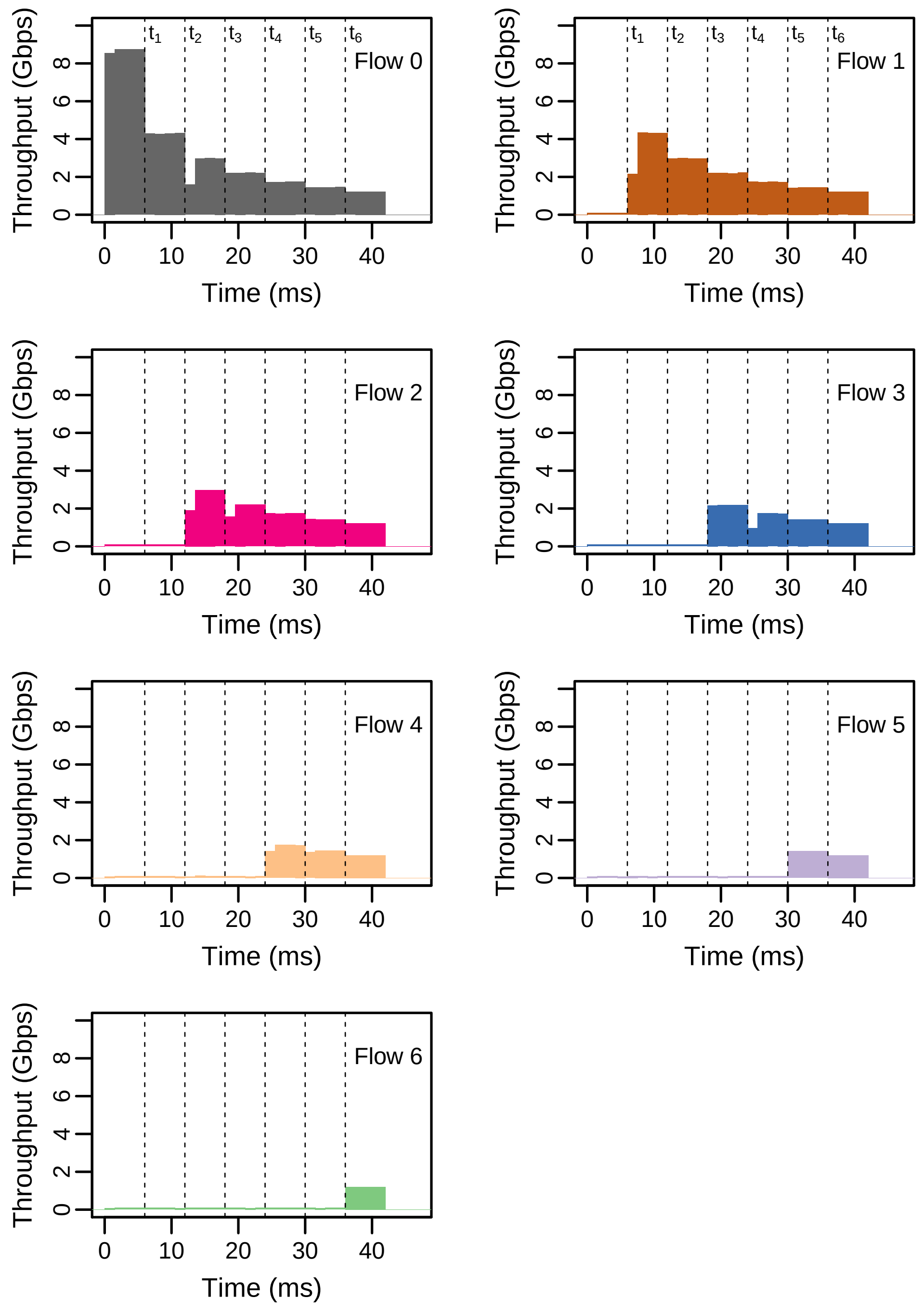


Figure 3.9. Goodput achieved for a time-varying workload of three flows to a single end host.

3.3.3 Time-varying workloads

Next we show that REACToR can dynamically serve rapidly changing traffic demands and efficiently move flows from the EPS to the OCS.

In this experiment, we vary the number of high bandwidth and low bandwidth flows among hosts at small timescales. The workload pattern is again all-to-all among eight hosts, which we observe from the perspective of one of the hosts and its seven incoming flows. Initially one of the flows is a high-bandwidth flow sending at full demand and served on the circuit switch, and the other flows are lower bandwidth flows (each paced at 96 Mb/s) served on the packet switch. At time t_1, one of the low bandwidth flows changes to a second high bandwidth flow — representing a dynamic shift in application behavior — and needs to be served on the circuit switch. At each subsequent time step, another lower bandwidth flow becomes high bandwidth and transitions from the EPS to the OCS.

Figure 3.9 shows the throughput achieved by each of the flows. Initially, the high bandwidth flow has exclusive use of the circuit switch. At each time step t_i, another flow transitions from low to high bandwidth and REACToR promotes it from the EPS to the OCS. Each time, all high bandwidth flows then adjust to fairly share the link bandwidth to the receiving host. In each case, REACToR seamlessly handles the shift in traffic demands.

Note that a flow might send at a lower rate during the first 1.5 ms scheduling period. This happens when the schedule changes and a flow is served earlier in this period than the previous one. As a result, the queue buffer does not yet have enough enqueued packets to fully utilize the link. The queue buffer will be built up starting from the second scheduling period, and the flow will fully utilize the link again.

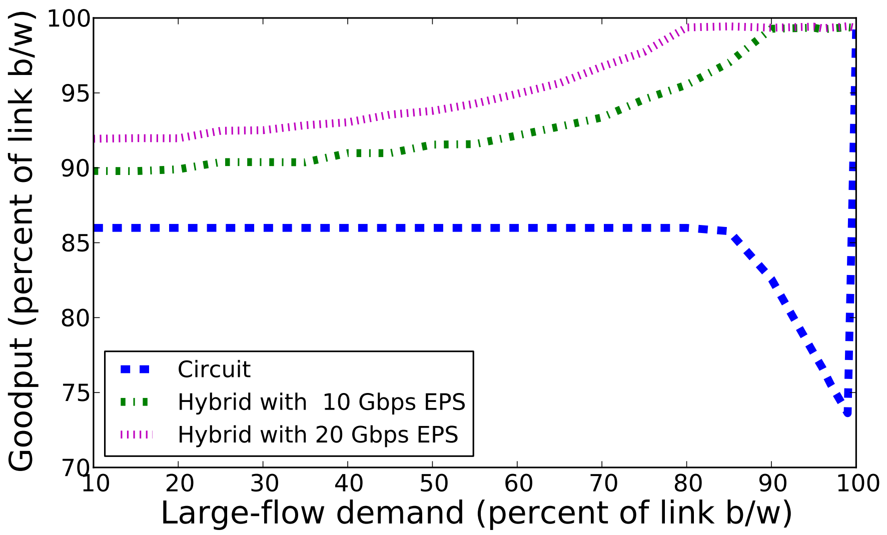


Figure 3.10. Performance of a circuit switch ToR and REACToR in different workload regimes.

3.3.4 Large benefits from a small EPS

As a final step, we illustrate how an underprovisioned packet switch in REACToR substantially relaxes the constraints of a pure circuit switch. In particular, we show how the ability to offload small flows to the packet switch enables REACToR (1) to maintain high circuit utilization and high workload goodput under our workload assumptions, and (2) to support full simultaneous endpoint connectivity for small flows.

We use simulation for these experiments to evaluate behavior beyond the constraints of our testbed. The simulator models a single REACToR switch, including the behavior of end hosts with NIC buffers, a circuit switch with switching overhead, a packet switch with buffers, and the circuit scheduler from Section 3.1.2. We validated the simulator output using our prototype: for workloads involving eight or fewer hosts, flow goodput calculated by the simulator always had errors less than 1% of flow goodput measured on the prototype implementation.

Maximizing circuit utilization. Using the simulator, we explore the performance regimes of a single hybrid ToR-like REACToR at rack scale. For comparison, we simulate 64 end hosts connected first to a pure circuit switch and then to a REACToR switch via 100-Gb/s links. We compare with a circuit switch, not because we expect it to perform ideally well, but because it helps illustrate how a REACToR switch performs. In this experiment, each host j sends traffic to its neighboring twenty-one hosts $j+1$ through $j+21$, one flow per host. The total offered demand across all twenty-one flows is 100 Gb/s. The flow from j to $j+1$ is a "large" flow whose demand D we vary up to the full 100 Gb/s. The other twenty "small" flows have equal demands dividing the remaining $(100-D)$-Gb/s bandwidth equally. For scheduling circuits, each configuration has a duration of at least 40 µs (including reconfiguration delay), the scheduling period is 3000 µs (at most 75 configurations), and the reconfiguration delay is 20 µs (hence each configuration has at least 50% utilization). For REACToR, we simulate a 100-Gb/s circuit switch and a packet switch internally, where the packet switch is 10 Gb/s or 20 Gb/s.

Figure 3.10 shows the results for this experiment. The *x*-axis shows the demand of the large flow from each host as a percentage of link rate (100 Gb/s), and the *y*-axis shows the goodput of the ToR given the offered workload. We show three curves, one for a pure circuit switch and two for REACToR, with the curves overlapping at points. We note, of course, that a fully-provisioned packet switch as the ToR could switch this workload at full rate.

The lowest curve shows the results of using a pure circuit switch for the ToR, with the right-most point of the curve as the ideal case for a circuit switch. Hosts send all of their traffic in the large flow at 100 Gb/s (small flows have zero demand). In this case all of the flows can take full advantage of a circuit when the switch schedules one for them: each flow has data to transmit during their entire allocation in the circuit schedule. Once the small flows start to have a non-zero demand, though, there is a cliff in circuit switch performance. The demands to the other hosts, although small, are all non-zero;

as a result, the switch schedules each small flow a circuit to carry its traffic. But the small flows do not have the traffic demand to fully utilize their circuit allocations, leaving them under-utilized. As the larger flow decreases in demand moving to the left, and the smaller flows correspondingly increase, the circuit switch performance improves as the small flows are able to utilize their allocations better. Once the small flows are able to fully use their circuits (when the large flow demand is at 87%), the pure circuit switch performance levels off. At this point, the lower goodput of the circuit switch is entirely due to reconfiguration delay overhead.

In comparison, the middle curve shows the performance of a hybrid ToR-like REACToR with a 100:10 capacity ratio. Between 90–100 Gb/s for the large flow ($<$ 10 Gb/s combined for the small flows), REACToR performs just like a packet switch because the combined demands of the small flows go through REACToR's packet switch while the large flows go through REACToR's circuit switch. This regime represents REACToR's ability to efficiently switch traffic that does not have good burst behavior. As long as the combination of those flows fits within the EPS "budget", REACToR has the performance of a 100-Gb/s packet switch using a combination of a 100-Gb/s circuit switch and a 10-Gb/s packet switch.

Below 90 Gb/s, REACToR performance gradually and gracefully degrades as the combined demands of the small flows exceed the 10-Gb/s per-host rate of REACToR's packet switch; notably, it avoids any discontinuities in performance. REACToR then needs to schedule an increasingly larger portion of small flow demand on the circuit switch. REACToR goodput will decrease as a combination of imperfect utilization of circuits when assigned to small-flow demand, and additional reconfiguration delays for those circuit assignments.

Note that for this curve the circuit and packet switches had a 100:10 capacity ratio. There is nothing fundamental about this choice. A network using REACToR switches

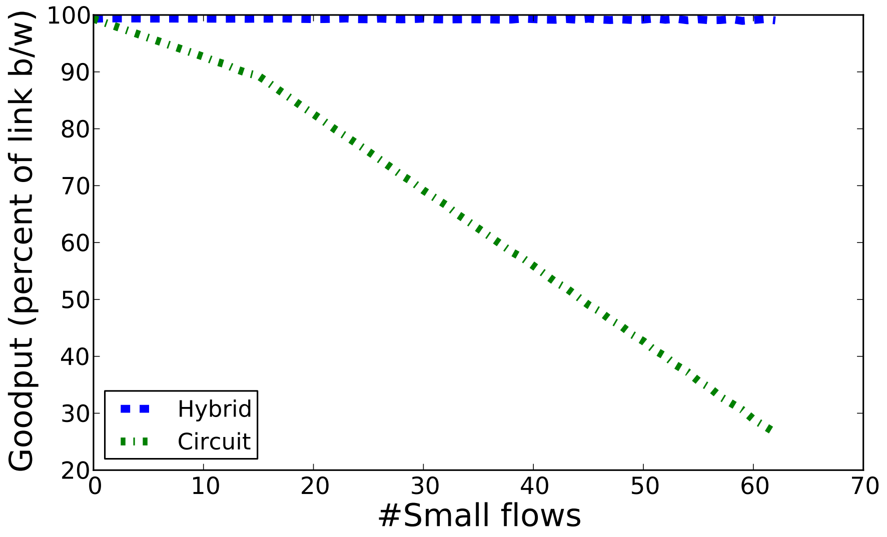


Figure 3.11. Performance of a circuit switch ToR and REACToR as a function of the number of small flows.

could tailor this ratio to balance cost and workloads: networks with more shorter-burst flows can deploy more EPS resources at higher cost, or vice versa. In terms of Figure 3.10, more EPS resources shift the point of 100% goodput for REACToR to the left, as shown by the top-most curve corresponding to an internal 20-Gb/s packet switch in REACToR.

Endpoint connectivity. In addition to maintaining high circuit utilization, the underprivisioned packet switch also enables REACToR to support many simultaneous flows between endpoints in the tail of the workload distribution. To illustrate this point, we perform one last experiment focusing on the number of simultaneous small flows between distinct endpoints in the network. In a network of 64 hosts, we represent the aspect of the workload well matched to circuits using one single large flow consuming 90% of the capacity: an ideal case for a pure circuit switched network. We then evenly split the remaining 10% among n small flows, where n varies between 1 and 64.

Figure 3.11 shows the goodput of the ToR (percentage of offered demand serviced by the ToR) as a function of the number of small flows for this experiment. At $n = 1$, both the hybrid and pure circuit ToRs perform the same on the trivial single large flow.

The bottom curve shows the pure circuit ToR goodput in the presence of small flows. Goodput steadily decreases because the circuit switch has to assign circuits to every flow. As the number of small flows increases, the demand in each flow decreases; circuit durations decrease, but the rate of reconfigurations correspondingly increases. Hence the pure circuit ToR becomes increasingly less efficient.

The top curve shows the hybrid ToR performance. By construction, its internal packet switch can satisfy the bandwidth demands of the small flows and therefore efficiently handle the full endpoint connectivity of the workload. If the total demand of the small flows comprising the tail of the workload exceeds the capacity of the underprovisioned packet switch, then the performance of the hybrid ToR will trend towards the left-hand regime in Figure 3.10 (e.g., where the large flow demand drops below 90% with a 10G EPS).

3.4 Summary

Hybrid ToRs, such as REACToR, have the potential to enable scalable, high-speed networks by pairing the numerous advantages of optical circuit switching with comparatively underprovisioned packet switching. The key insight driving our work is that by moving the vast majority—but not all—of the buffering out of the switch and into end hosts, more scalable interconnect fabrics can be supported.

Practically speaking, this only works if 1) end hosts emit bursts of traffic to a given destination that are both predictable and of sufficient duration to fill OCS circuits, and 2) the hybrid scheduler operates at timescales that are invisible to the transport and applications running on the end hosts. In the first case, in-NIC buffering that historically has been used to drive line rate transmissions can be repurposed to stage impending data bursts, therefore fully using OCS circuits. In the second case, for a two-REACToR prototype, we have shown that we can schedule end hosts to make use of an OCS without negatively impacting TCP performance. A design challenge posed by interconnecting a large number of REACToRs is co-scheduling and synchronizing directly connected REACToRs to avoid the need for buffering on uplink ports. We will discuss the scheduling problem in the next chapter.

This chapter, in part, is a reprint of the material as it appears in the USENIX Symposium on Networked Systems Design and Implementation 2014. Liu, He; Lu, Feng; Forencich, Alex; Kapoor, Rishi; Tewari, Malveeka; Voelker, Geoffrey M.; Papen, George; Snoeren, Alex C.; Porter, George. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Solstice: The Scheduling Algorithm

This chapter presents our hybrid switch scheduler, Solstice. Solstice exploits the skewed nature of data center traffic patterns, and creates a small number of configurations with long durations that minimize the penalty for reconfiguration and leaves only a small amount of residual demand to be serviced by the low-speed (and, hence, lower-cost) unconstrained packet switch.

Our simulation results demonstrate that Solstice is highly effective at scheduling skewed demands like those found in data centers. In particular, we find that by servicing a larger fraction of the demand over the circuit switch, Solstice dramatically decreases the latency experienced by traffic served by the packet switch—reducing the 99th percentile by over two orders of magnitude when compared to existing approaches. Moreover, we show empirically that Solstice's running time is consistently $O(N \log^2 N)$ for these workloads, making it practical to implement in both existing and recently proposed hybrid switches.

4.1 Preliminaries

Before we begin, we precisely define the setting and notation that we will use in the remainder of this chapter. We model a single node in a hybrid network fabric, i.e., an abstraction of a single input-queued switch with N ports. This abstraction consists of two crossbar switches and the associated queues that are serviced by these two switches.

In REACToR and other recently proposed hybrid architectures [46], the first crossbar switch resides entirely within a standard packet switch, and forwards packets from that packet switch's input buffers to its output buffer (in the case of an input/output buffered design). As a packet switch, this first crossbar is effectively unconstrained—in other words, it has negligible reconfiguration time—and delivers a data rate of R_p per port. This switch services its internal queues based on an existing scheduling algorithm that we do not consider here. The second crossbar switch typically uses an alternate technology (e.g., optics or RF wireless). As compared to the electrical crossbar switch, this second crossbar has a significantly larger data rate $R_c \gg R_p$, but is presumed to be constrained, i.e., have a non-trivial reconfiguration penalty. Hence, a connection at this slower time scale can be viewed as a circuit, and we will refer to this crossbar as the circuit switch in the remainder of this chapter.

One important difference between our model and traditional switch designs is that we explicitly do not expect there to be any (significant) queueing at the outgoing ports.[1] This forecloses any crossbar that relies on a *speedup* factor to send traffic faster than the port link rate. However, it dramatically simplifies the implementation of a hybrid switch by enabling a bufferless, all-optical path through the circuit switch. In recently proposed hybrid architectures, the input queues are actually maintained at the end hosts. This detail does not impact our model. However, we require that the input queues support virtual output queueing (VOQ), so that traffic queued for each output port can be explicitly addressed at each input port.

Figure 4.1 illustrates our canonical model. We assume that incoming traffic is allowed to accumulate at input queues for some period of time, W, and is serviced during

[1]There are a variety of ways to multiplex the traffic from the circuit and packet switches onto the outgoing port, some of which require minimal queuing while others do not (see Section 4.7).

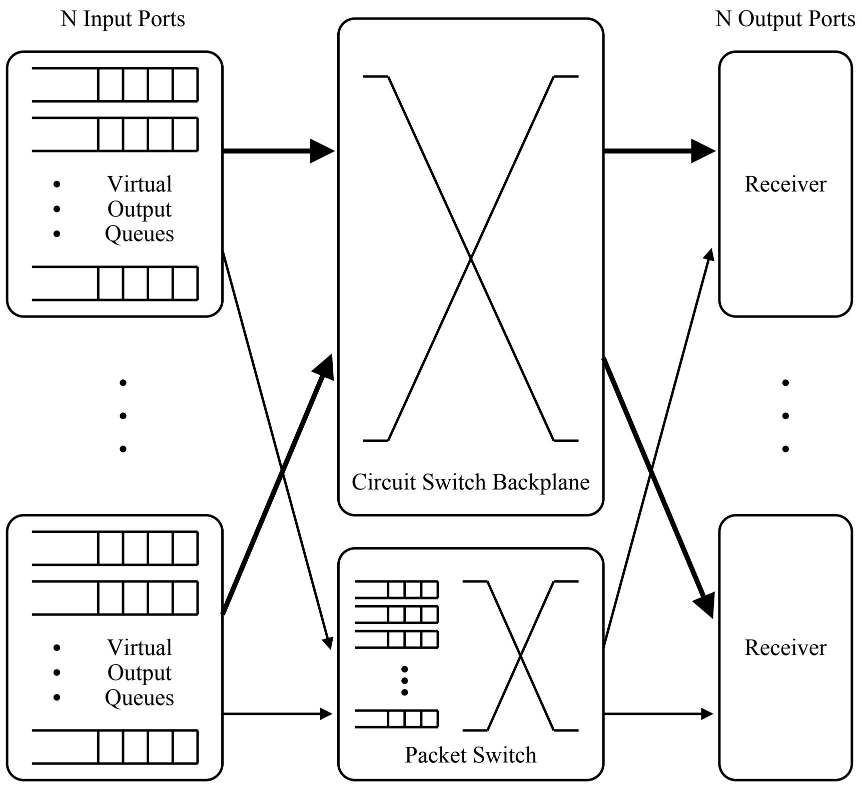


Figure 4.1. An illustration of our canonical hybrid switch architecture.

a later period. This time quanta serves as the basic unit of allocation; hence, we will variously refer to it as the accumulation period and schedule length. The demand at the end of any accumulation period can be expressed as a non-negative matrix, D, of size $N \times N$, where the rows are sources, the columns are destinations, and the elements are the corresponding demand from a source to a destination. In the general case, each port is simultaneously connected to both the packet switch and the circuit switch. At any point in time, some VOQs at each port may be drained by the packet switch, at most one queue may be serviced by the circuit switch, and others may be stalled. The circuit switch can connect any input queue to any output port, but no port may have multiple inputs or outputs (aside from their connection to the packet switch) in a single configuration.

The circuit configuration can be changed at the cost of a delay δ, during which time no data can be forwarded by the circuit switch. The packet switch, on the other hand, can service traffic at all times. To ensure high overall circuit utilization, each circuit configuration must be maintained for a relatively long period with respect to δ. For example, to service 90% of the link capacity over the circuit switch, the average duration of a configuration needs to be at least 9δ to amortize the reconfiguration delay.

The goal of a hybrid scheduling algorithm is, given a demand matrix D, to compute a schedule of m circuit switch configurations—permutation matrices—P_k, and corresponding durations, ϕ_k, so that the schedule is no longer than the accumulation period, i.e. $\sum_{k=1}^{m} \phi_k + \delta m \leq W$. (A permutation matrix is a matrix of 0s and 1s in which each row and column has and only has a single 1.) When the circuit switch implements the schedule, it services $S = \min(D, \sum_{k=1}^{m} \phi_k P_k)$ of the offered load, with the remainder directed to the packet switch. Clearly, the larger S can be, the lower the required capacity of the packet switch, R_p. In our model, the circuit switch forwards packets at the same rate as the input port line rate, R_c, while the packet switch forwards at a (much) lower speed (e.g., 1/10th that rate).

When δ is large, each uplink or downlink can only be shared among a limited number of flows with high efficiency. For example, if $\delta = 20$ µs, the average configuration duration needs to be 180‘µs to maintain 90% efficiency, and a schedule of e.g., $W = 3$ ms can have at most 15 such configurations. With 15 circuit switch configurations, each source can only send data over the circuit switch to at most 15 destinations, and likewise each destination can receive data from at most 15 sources during each schedule. Intuitively this long reconfiguration delay, restricts the use of hybrid network architectures to settings where demand is highly nonuniform. We will quantify this restriction in a later section.

4.2 Previous Studies

The crossbar switch scheduling problem has been studied for decades. In general, time is considered to be slotted (i.e., demand is quantized), and the basic approach—often referred to as time slot assignment (TSA)—decomposes an accumulated demand matrix into a set of weighted permutation matrices. Classical results [6] and early work on scheduling satellite-switched time-division multiple access (SS/TDMA) systems [23] show how to compute a perfect schedule, but the resulting schedules consist of $O(N^2)$ distinct configurations. While optimal for an unconstrained switch, the game changes once one begins to consider reconfiguration time.

When the switch introduces a reconfiguration delay, it is impossible to service fully saturated demand at link rate. Moreover, if the computed schedule is not perfect (i.e., some configurations are not fully utilized), additional inefficiency is introduced. Hence, researchers generally allow the crossbar to transmit packets faster than the incoming and outgoing ports can accommodate through the use of both input and output buffering. Much of the exiting work focuses on minimizing this so-called *speedup* factor (i.e., the ratio of the internal transfer rate to the port link rate).

The body of previous work tries to provide what is known as performance guaranteed switching [48], meaning they compute schedules that are stable [17] (i.e., they service 100% of the offered demand) and are bounded in length (so they introduce at most a fixed delay). Computing the optimal schedule in this regime is known to be NP-complete [19, 30], but there exist algorithms [19, 44, 47] that use the least possible number of configurations (N), which makes sense when the reconfiguration time is effectively infinite. When the reconfiguration time is non-negligible, but not extremely large, there is a need to balance the efficiency of each schedule with the total number of reconfigurations. DOUBLE [44] computes a schedule that always requires twice the minimum number of configurations. Further improved algorithms [15, 30, 48] take the actual reconfiguration delay into account, producing a variable-length schedule depending on the actual reconfiguration delay.

Unfortunately, practical limitations in implementing high-speed routers with large port counts restrict the complexity of the scheduling algorithms that can be employed—even for the unconstrained case. Hence, a variety of practical scheduling algorithms have been proposed that tradeoff throughput and delay for implementation complexity; among them, a well known one is iSLIP [33] which requires a $2\times$ speedup to maintain stability. Many of these algorithms perform poorly (i.e., introduce large delays) when the traffic demand is nonuniform, leading others to suggest using randomization to address the issue [17].

4.3 Motivation

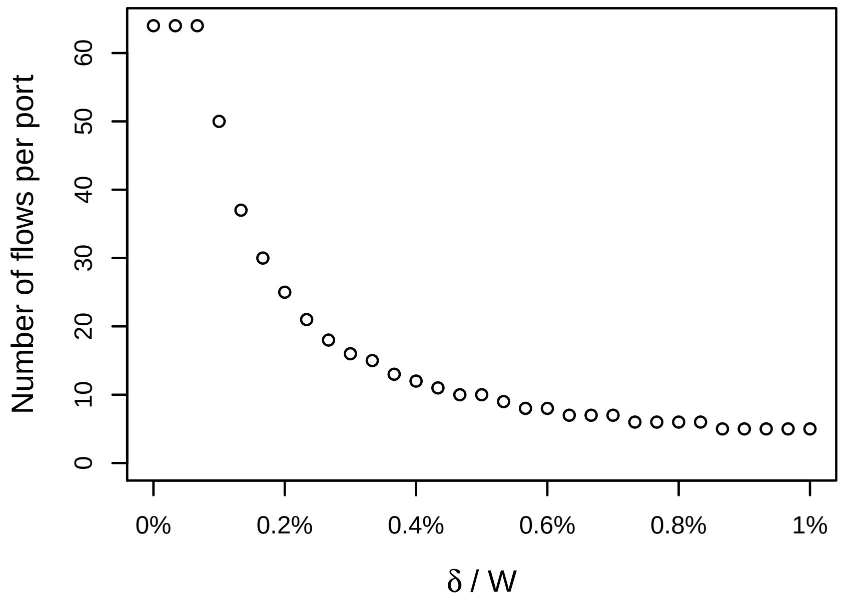
In contrast to traditional constrained switches, hybrid architectures eschew speedup, instead preferring to use a lower-speed packet switch as a way to make up for the reconfiguration delay and any scheduling inefficiency. Because existing scheduling algorithms focus on reducing the overall reconfiguration penalty—which can be avoided entirely on the packet switch—they produce sub-optimal schedules in this setting. Here, we motivate our work by considering the performance of existing algorithms in the absence of speedup—in other words, the crossbar can only forward traffic at the link rate of the input ports. To be fair, however, we ensure that the offered load can, in fact, be served without speedup; i.e., the demand matrix is not fully saturated. Yet, existing algorithms whose design is predicated on a speedup factor still service only a fraction of the demand.

For purposes of illustration, we consider a demand that consists of $N \times k$ flows, where each port has exactly k incoming and outgoing flows. Among the k flows on each port, 20% of them consume approximately 70% of the link rate, while the other flows equally share less than 30% of the link rate. In total, 98% of the link is requested, which means at most 2% of the time can be spent on reconfiguring the switch (when no speedup is allowed and the demand has to be satisfied). If each reconfiguration takes δ (expressed as a function of the scheduling interval), then k can be at most $\delta/0.02$, since each new destination requires a reconfiguration—i.e., a schedule that satisfies the demand can require at most k different configurations. We can easily generate such a demand by stacking k random perfect matching of flows together.

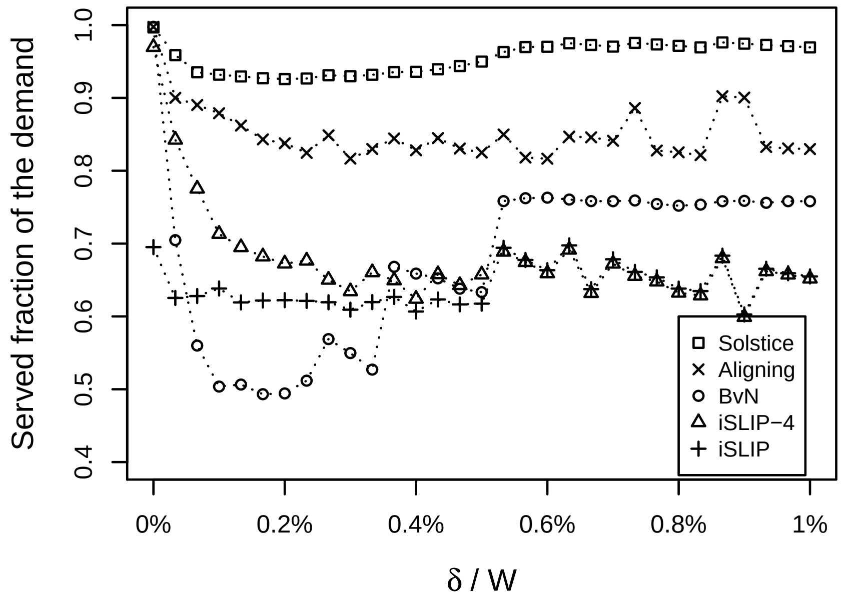
Figure 4.2(a) plots the number of admissible flows, k, in this construction as a function of δ. For the purposes of this example, we limit k to at most $N = 64$, which clips the left-hand portion of the curve. To make the problem more realistic, we randomly perturb each of the computed flow sizes by $\pm 0.3\%$ of the link speed while respecting capacity limitations; the resulting random demand matrix is therefore still satisfiable, but not as straightforward to decompose.

4.3.1 Utilization

Figure 4.2(b) shows the fraction of demand serviced by several published scheduling algorithms along with the algorithm described in this chapter, Solstice, as a function



(a)



(b)

Figure 4.2. A comparison of the performance of various scheduling algorithms as a function of the fraction of the reconfiguration delay, δ, over the accumulation period, W. Note that the y-axis does not start at zero.

of the reconfiguration time, expressed as a fraction of the accumulation time. In the figure, each mark is the mean link utilization of the algorithm serving 100 randomly generated demand matrices.

To begin, we plot the performance of perfect decomposition (known as Birkhoff von Neumann or BvN) and iSLIP. We consider two different versions of iSLIP, one that iterates only once, and another iterates four times for each slot. Because neither algorithm considers the reconfiguration penalty when computing the schedule, it is not surprising that they perform poorly. The figure shows that, for this particular demand and set of parameters, they each service less than 80% of the demand.

It is instructive to consider why the algorithms under-perform. The perfect matching approach searches among all non-zero elements of the demand matrix; the duration of each configuration is defined by the demand of the smallest element in the matching. When the elements of the demand matrix are of equal (or relatively quantized) size, the configuration durations are likely to be long. When the difference between demand elements is small, however, there is no bound on the minimum configuration duration: it may even output configuration times that are shorter than the reconfiguration penalty—resulting in less than 50% efficiency. iSLIP, on the other hand, computes a schedule with fixed configuration durations. In order to guarantee a minimum level of efficiency, iSLIP must use a configuration duration that is a reasonable multiple of the reconfiguration delay. In our examples, we set the configuration duration to be 9δ, resulting in a maximum utilization of 90%.

To deliver higher efficiency in the face of reconfiguration delay, existing schemes [30, 44, 47, 48] first "align" the demand matrix to a fixed quantum, thereby factoring the demand into a coarse matrix, D_c, and a fine matrix, D_r, where D_c is the portion that can be expressed in terms of the chosen demand quantum, and D_r contains the remainder. The appropriate demand quantum depends on how many configurations the algorithm seeks to use in the resulting scheduling. For example, the DOUBLE algorithm [44] always outputs a schedule of $2N$ configurations, using N configurations to serve D_c and another N to cover D_r.

More sophisticated algorithms [30, 48] take the reconfiguration delay into account when determining how many configurations to use in the schedule. Here, we approximate this class of algorithms by selecting the time quanta to be the smallest integer divisor of the scheduling period that is larger than 9δ, resulting in a duty cycle of at least 89.9% when serving the coarse demand matrix D_c. For now, we simply discard the residue, D_r, which is at most 10.1% of the demand. However, some fraction of D_r may be serviced by the schedule computed for D_c. Recall that in this particular experiment, the demand is concentrated in a small number of flows, so it is unlikely a schedule will completely starve some elements of the demand matrix: the demand is constructed so that each element is on the order of the computed time quanta. In order to exploit this opportunity we linearly scale up the durations of the configurations that serve the coarse matrix so that the total schedule duration is W.

4.3.2 Delay

If one attempts to service the non-scheduled demand D_r with a packet switch, it must be appropriately provisioned to avoid drops or excessive delay. In a typical REACToR setting, the packet switch is an order of magnitude slower than the circuit switch; i.e., it can handle at most 10% of the demand. In the scenario considered in Figure 4.2(b), only Solstice keeps the residual below 10%. Of course, even when the residual demand is satisfiable, any non-uniformity may lead to queuing within the packet switch.

As a concrete example, we compare Aligning and Solstice on a simulated testbed of 8 hosts. We consider a simple, uniform demand consisting at each node of one big flow that takes 85% of the link bandwidth and three small flows that, together, consume 10% of the link bandwidth. On top of that, we introduce a few small (less than 1% of the link bandwidth) latency-sensitive control packet streams to yield a total demand of 98.16% of optical switch capacity.

The Aligning scheduler services 87.5% of the demand on the optical switch, while Solstice handles 90.9%. This seemingly minor ($< 4\%$) improvement has a dramatic impact on the delay experienced by the latency sensitive flows that are part of the residual demand handled by the packet switch. In particular, under Aligning, the latency is 2.1 ms at the 90th percentile, 4.0 ms at the 99th percentile, with a maximum of 5.0 ms. In contrast, Solstice delivers a latency of 5 µs at the 90th percentile, 33 µs at the 99th, and at most 52 µs—a more than two-orders-of-magnitude reduction at the tail.

4.3.3 Implications

When the reconfiguration time is non-trivial, an aligning-based approach seems promising. Existing algorithms, however, make no assumptions regarding the demand matrix—in particular, they do not benefit from sparse demand matrices. They assume that it requires $O(N)$ configurations to cover the demand matrix. Our observation is that when reconfiguration times are significant, the demand matrix has to be sparse to be efficiently serviceable, and when the demand matrix is sparse, it does not need $O(N)$ configurations to cover the demand matrix; rather, it needs $O(1)$. Moreover, effectively scheduling the residue in general is a hard problem, requiring a similar number of configurations; it is likely far more efficient to simply pass it on to the packet switch.

4.4 The Hybrid Scheduling Problem

In this section, we formally describe the hybrid scheduling problem we seek to solve. We start by explaining the traditional constrained switch scheduling problem, and then enhance it to consider a hybrid switch.

4.4.1 Constrained switch scheduling

Traditionally, researchers have focused on scheduling a single switch crossbar with a fixed reconfiguration delay, δ. They start with a demand matrix D that represents the traffic that is queued during an accumulation period W. In order for D to be admissible, the diameter (i.e., the maximum of the row and column sums), $\phi(D) \leq W$. In this chapter, we will assume that D is always admissible.

Problem: Constrained. Given a matrix D, find a set of m permutation matrices $\{P_k\}_{1 \leq k \leq m}$, and a sequence of positive numbers $\{\phi_k\}_{1 \leq k \leq m}$ that maximize $S = \min(\sum_k \phi_k P_k, D)$, where:

$$\sum_{k=1}^{m} \phi_k + \delta m \leq W.$$

In this model, no traffic can be serviced during the reconfiguration delay, δ. Hence, without speedup, for any desired level of efficiency x the resulting schedule can consist of at most $m \leq (1-x)W/\delta$ configurations with an average duration $\overline{\phi_k} \geq \delta/(1-x)$. Existing algorithms typically assume some degree of speedup, s, and guarantee $sS \geq D$ [19, 44, 47].

4.4.2 Skewed demand

Our proposed hybrid architectures do not admit a speedup factor. Instead, we presume a certain degree of sparsity in the input demand. In particular, for a fixed N, when considering a single source port or destination port, the number of flows of non-trivial size that arrive during W is bounded by a constant, C; e.g., a study of a Microsoft data center [27] suggests that $C \approx 5$. Datacenters today might have larger C numbers, but C is likely to scale much slower than N. We capture this notion through what we call *skew degree*:

Definition: Skew Degree. Given a matrix D, define $\psi(D)$ to be the maximum number of non-zero elements in each row and column of D. Before considering the hybrid case, we first observe that skewed demand matrices are fundamentally easier to schedule.

Problem: Skewed (unconstrained). Given a non-negative matrix D with $\psi(D) \leq C$, find m permutation matrices $\{P_k\}_{1\leq k\leq m}$, and a sequence of positive numbers $\{\phi_k\}_{1\leq k\leq m}$, where

$$\sum_k \phi_k P_k \geq D \text{ and } \sum_k \phi_k \leq W.$$

To fully serve demand D, a schedule needs to have at most $\min(\phi(D), N^2 - 2N + 2)$ configurations [44] and at least $\psi(D)$ configurations. Algorithms for the constrained scheduling problem attempt to minimize the number of configurations, but do not make any assumption regarding the skew degree $\psi(D)$ of the input demand, so they produce schedules with on the order of $\phi(D)$ configurations, where $\phi(D)$ can be as large as N in general.

4.4.3 Constrained switching w/skewed demand

We now combine the assumptions on input demand with the scheduling constraints to arrive at the hybrid scheduling problem. Because it may be impossible to schedule all of the demand on the circuit crossbar, we relax the problem to that of maximizing the fraction of demand served. For this initial framing, we do not concern ourselves with the packet switch scheduling algorithm; we presume it is capable of servicing any admissible demand (given its reduced forwarding rate) without additional delay.

Problem: Hybrid scheduling (skewed & constrained). Given a matrix D,

Table 4.1. The constraints of various crossbar scheduling problems and the effectiveness of the best known algorithms to solve them. There are a variety of approximation algorithms for the constrained switching problem all based on the aligning approach we describe. Each algorithm delivers varying schedule efficiency depending on a number of parameters including the reconfiguration delay. Similarly, different algorithms have different running times.

Problems	Constrained	Skewed	Hybrid scheduling
Best Effort	$\surd$	$\times$	$\surd$
Reconfiguration Penalty	$\surd$	$\times$	$\surd$
Skewed Demand	$\times$	$\surd$	$\surd$
Suitable Algorithms	Aligning	BvN + RegMatch	Solstice
Schedule Efficiency	$\approx$ 50–80%	100%	$\approx$ 85%
Time Complexity	*varies*	$O(\phi(D)N)$	$O(N\log^2 N)$

where $\psi(D) \leq C$, find m permutation matrices $\{P_k\}_{1\leq k\leq m}$, and a sequence of positive numbers $\{\phi_k\}_{1\leq k\leq m}$ that maximize $S = \min(\sum_k \phi_k P_k, D)$, where:

$$\sum_k \phi_k + \delta m \leq W.$$

Because skewed demand matrices can be serviced with fewer configurations, the resulting schedules incur smaller reconfiguration penalties. Hence, the speedup required by existing algorithms for the constrained scheduling problem is also reduced.

We summarized these problems in Table 4.1, and will describe our algorithm, Solstice, for addressing this hybrid scheduling problem in the next section.

4.5 Solstice

Like previous work, we focus on decomposing the demand matrix D into two parts, a coarse matrix D_c and a fine matrix D_r [44]. However, unlike traditional constrained switch scheduling algorithms, which compute schedules for each matrix, we compute a schedule only for D_c and assume that D_r can be serviced by the packet switch.

This assumption allows us to ignore the number of schedules that might be required to service D_r on the circuit crossbar, and instead maximize the efficiency of the schedule for D_c—i.e., service the maximum demand on the circuit crossbar.

Solstice starts with a candidate factorization, $D = \hat{D}_c + \hat{D}_r$, by placing all extremely small demand elements, i.e., those less than some threshold (we use 2δ), into $\hat{D}_r$. The intuition is that elements with demand that is on the order of the reconfiguration penalty δ can never be scheduled efficiently on the circuit switch. It then attempts to construct an efficient schedule for $\hat{D}_c$ in a greedy fashion by considering circuit configurations of exponentially spaced durations. Once Solstice has considered configurations of all durations down to a minimum efficiency threshold, it sweeps any remaining unserved demand from $\hat{D}_c$ into $\hat{D}_r$.

Finally, because the duration of the schedule $\hat{S}$ computed to service $\hat{D}_c$ is unlikely to be exactly W, we uniformly scale the duration of each configuration in $\hat{S}$ to obtain a schedule that is precisely W in length. If we need to scale down, however, it is possible the resulting schedule contains configurations below our efficiency floor. In that case, we discard the configuration with the shortest duration (which obviously moves some demand from $\hat{D}_c$ back into $\hat{D}_r$) in the schedule and try to rescale the remainder. This process repeats until we arrive at a final schedule S that consists only of configurations whose durations are greater than the threshold and services $D_c \geq \hat{D}_c$ because the stretching process could result in configurations with longer durations than initially scheduled that may additionally accommodate some of the demand that was swept into $\hat{D}_r$ previously.

Algorithm 1 lists the pseudocode for Solstice. It proceeds by considering configuration durations of exponentially decreasing duration, starting with the first power of two greater than $W/2$ but less than or equal to W. One technically challenging aspect of Solstice is selecting the best permutation matrices for each configuration duration. Because it is an NP-complete problem, some previous algorithms compute approximate

```
input  : The demand: $D$, desired schedule length $W$,
         reconfiguration delay: $\delta$
output : $m$ configurations and durations: $\{P_k\}$, $\{\phi_k\}$
$D' \leftarrow$ TrimBelow($D$, $2\delta$)
$D' \leftarrow$ QuickStuff($D'$)
$T \leftarrow 2\delta$
while $T < W/2$ do
    $T \leftarrow 2T$
end
$k \leftarrow 1$
while $T \geq 2\delta$ do
    $D^\dagger \leftarrow$ TrimBelow($D'$, $T$)
    while $D^\dagger > 0$ do
        $P_k \leftarrow$ RandSlice($D^\dagger$)
        if $P_k$ is not null then
            $\phi_k \leftarrow \min\{D^\dagger[i,j] \mid P_k[i,j] = 1\}$
            $D' \leftarrow D' - \phi_k P_k$
            $D^\dagger \leftarrow D^\dagger - \phi_k P_k$
            $D^\dagger \leftarrow$ TrimBelow($D^\dagger$, $T$)
            $k \leftarrow k + 1$
        else
            break the inner while loop
        end
    end
    $T \leftarrow T/2$
end
$m \leftarrow k$
while $\sum \phi_k \neq W - m\delta$ do
    if $\sum \phi_k < W - m\delta$ then
        Linearly scale up all $\{\phi_k\}$ so that $\sum \phi_k + m\delta = W$
    else
        Linearly scale down $\{\phi_k\}$ so that $\sum \phi_k + m\delta = W$
    end
    $j = \arg\min_k \phi_k$
    if $\phi_j < \delta$ then
        Remove $P_j$ from the schedule
        $m \leftarrow m - 1$
    end
end
```

Algorithm 1: Solstice Algorithm

```
TrimBelow()
input  : $D$, $T$
output : $D'$
$D' \leftarrow D$
for each $D'[i,j]$ in $D'$ do
    if $D'[i,j] < T$ then
        $D'[i,j] \leftarrow 0$
    end
end
Return $D'$.
```

Algorithm 2: Trim below

results based on maximum matchings. However, we observe that maximum matching often leads to poor efficiency—especially when the demand is skewed and sparse, because a significantly larger element can easily mislead the matching to a sub-optimal configuration. Hence instead, we use perfect matching whose results are often more efficient. However, to produce these efficient configurations, we first need to transform the demand matrix through a process we call *stuffing*.

4.5.1 Stuffing

Stuffing is a heuristic that increases the weight of the matchings that can be obtained. Intuitively, an optimal configuration may under-utilize one or more of the port pairs, because the inefficiency in doing so is dominated by the benefits of serving the remaining port pairs in the configuration for a longer duration. Ideally, one would like to identify generalized permutation matrices, i.e., the optimal matchings over all possible configuration matrices of D (i.e., sub-matrices of permutation matrices that have $M \leq N$ ones). If the optimal configuration matrix has less than N ones, then the remaining dimensions can be "backfilled" by connecting linearly independent port pairs to make some use of the "left over" crossbar capacity. Unfortunately, identifying such optical configuration matrices is an open problem that has been considered in several

```
QuickStuff()
input  : $D$
output : $D'$
Calculate the sums of each row $\{R_i\}$
Calculate the sums of each column $\{C_i\}$
$\phi \leftarrow \max(\{R_i\}, \{C_i\})$.
$D' \leftarrow D$
for each $D[i, j] > 0$ in $D$ do
    $g \leftarrow \phi - \max(R_i, C_j)$
    $D'[i, j] \leftarrow D'[i, j] + g$
    $R_i \leftarrow R_i + g$
    $C_j \leftarrow C_j + g$
end
$i \leftarrow 1$
$j \leftarrow 1$
for $i \leq N$ and $j \leq N$ do
    while $R_i = \phi$ do
        $i \leftarrow i + 1$
        if $i \geq N$ then
            break the outer for loop;
        end
    end
    while $C_j = \phi$ do
        $j \leftarrow j + 1$
    end
    $g \leftarrow \phi - \max(R_i, C_j)$
    $D'[i, j] \leftarrow D'[i, j] + g$
    $R_i \leftarrow R_i + g$
    $C_j \leftarrow C_j + g$
end
Return $D'$.
```

Algorithm 3: Quick Stuffing

other contexts [20, 38, 43]. Solstice's stuffing heuristic in some sense "pre-fills" these elements which would be good candidate elements to add to an optimal matching of smaller dimension, but would otherwise not be included in a matching of dimension N.

Stuffing increases the elements of the demand matrix D until every row and column sum is the same as the diameter $\phi(D)$. Formally speaking, it takes a scaled doubly sub-stochastic matrix and turns it into a scaled doubly stochastic matrix. It is a necessary step for using perfect matching to find efficient optimal configurations, because, intuitively, when all the rows and columns have the same sum, any greedy perfect matching of non-zero weight will keep this invariant and always make optimal progress [9]. The methods for stuffing are many, but as long as it stuffs into a doubly stochastic matrix, perfect matching would always generate an optimal set of configurations when the configuration delay δ is zero. Previous work typically prefers stuffing the matrix uniformly to minimize the variance of the elements [9].

However, in hybrid scheduling the configuration delay δ is non-zero, and hence each configuration needs to pay a time penalty of δ out of the scheduling period W. Therefore, ideally, one would like to minimize the total number of configurations by identifying perfect matchings of larger weights. This optimization goal means that the stuffing heuristics should attempt to maintain a low skew degree $\psi(D)$ by preferring stuffing elements that are already non-zero or large, rather than uniformly. To find such stuffing, one could iterate over all the possible ways to stuff, but that would be computationally expensive. Instead, Solstice's stuffing algorithm, which we call QuickStuff (Algorithm 3), simply selects each of the non-zero elements of D in arbitrary order and maximizes them. Afterwards, if any rows or columns are not yet maximized, it picks the first zero element in each such row or column and places the remainder there.

QuickStuff runs in $O(CN)$ time where $C = \psi(D)$ is the maximum number of non-zero elements in each row (skew degree). Unfortunately, QuickStuff could output a matrix where $\psi(D') > C$, and in the general case, the skew degree of D' may approach N. For input matrices of low skew degree, however, our experience shows that the skew degree of D' is rarely substantially larger.

4.5.2 Slicing

```
RandSlice()
input  : $D'$
output : $P$
$B \leftarrow$ BinaryMatrixOf($D'$)
Build the bipartite graph based on $B$.
Set all rows and columns as unmatched.
for each column $j$ do
    for each row $i$ where $B[i, j] = 1$ do
        if row $i$ is unmatched then
            Match row $i$ with column $j$ break inner for loop
        end
    end
end
for each unmatched column $j$ do
    Perform a depth-first-search over the augmented graph, starting from column
    node $j$.
    During the search, when visiting the sub-nodes of a node, visit the sub-nodes in
    random order.
    if cannot find a path to an unmatched row then
        Return nil.
    else
        Flip the augmented path, so that column $j$ is matched.
    end
end
Return the matched nodes in the form of a permutation matrix $P$.
```

Algorithm 4: Random Slicing

Once the demand matrix has been stuffed, it remains to select a perfect matching (configuration) and a duration. We call the process of identifying a perfect matching to add to the schedule *slicing*, because it slices off demand a configuration at a time. Existing algorithms [9, 23] compute a prefect decomposition without concern to the number of configurations generated. Here, we are interested in finding a tradeoff between

$$\text{Demand Input:} \quad D = \begin{bmatrix} . & 13 & 10 & 70 & 4 \\ 1 & . & 14 & 12 & 69 \\ 65 & . & . & 12 & 14 \\ 15 & 33 & 2 & . & 11 \\ 12 & 7 & 3 & 1 & . \end{bmatrix}$$

$$\text{After first trimming:} \quad D' = \begin{bmatrix} . & 13 & 10 & 70 & 4 \\ . & . & 14 & 12 & 69 \\ 65 & . & . & 12 & 14 \\ 15 & 33 & 2 & . & 11 \\ 12 & 7 & 3 & . & . \end{bmatrix} \quad \text{After stuffing:} \quad D' = \begin{bmatrix} . & 14 & 10 & 70 & 4 \\ . & . & 17 & 12 & 69 \\ 71 & . & . & 13 & 14 \\ 15 & 70 & 2 & . & 11 \\ 12 & 14 & 69 & 3 & . \end{bmatrix}$$

$$\text{Slicing}, T = 32: \quad D^{\dagger} = \begin{bmatrix} . & . & . & 70 & . \\ . & . & . & . & 69 \\ 71 & . & . & . & . \\ . & 70 & . & . & . \\ . & . & 69 & . & . \end{bmatrix} \approx \begin{bmatrix} . & . & . & 69 & . \\ . & . & . & . & 69 \\ 69 & . & . & . & . \\ . & 69 & . & . & . \\ . & . & 69 & . & . \end{bmatrix}$$

$$\text{Residue:} \quad D' = \begin{bmatrix} . & 14 & 10 & 1 & 4 \\ . & . & 17 & 12 & . \\ 2 & . & . & 13 & 14 \\ 15 & 1 & 2 & . & 11 \\ 12 & 14 & . & 3 & . \end{bmatrix}$$

$$\text{Slicing}, T = 8: \quad D^{\dagger} = \begin{bmatrix} . & 14 & 10 & . & . \\ . & . & 17 & 12 & . \\ . & . & . & 13 & 14 \\ 15 & . & . & . & 11 \\ 12 & 14 & . & . & . \end{bmatrix} \approx \begin{bmatrix} . & 11 & . & . & . \\ . & . & 11 & . & . \\ . & . & . & 11 & . \\ . & . & . & . & 11 \\ 11 & . & . & . & . \end{bmatrix} + \begin{bmatrix} . & . & 10 & . & . \\ . & . & . & 10 & . \\ . & . & . & . & 10 \\ 10 & . & . & . & . \\ . & 10 & . & . & . \end{bmatrix}$$

$$\text{Residue:} \quad D' = \begin{bmatrix} . & 3 & . & 1 & 4 \\ . & . & 6 & 2 & . \\ 2 & . & . & 2 & 4 \\ 5 & 1 & 2 & . & . \\ 1 & 4 & . & 3 & . \end{bmatrix}$$

$$\text{Slicing}, T = 2: \quad D^{\dagger} = \begin{bmatrix} . & 3 & . & . & 4 \\ . & . & 6 & 2 & . \\ 2 & . & . & 2 & 4 \\ 5 & . & 2 & . & . \\ . & 4 & . & 3 & . \end{bmatrix} \approx \begin{bmatrix} . & . & . & . & 2 \\ . & . & 2 & . & . \\ . & . & . & 2 & . \\ 2 & . & . & . & . \\ . & 2 & . & . & . \end{bmatrix} + \begin{bmatrix} . & 3 & . & . & . \\ . & . & 3 & . & . \\ . & . & . & . & 3 \\ 3 & . & . & . & . \\ . & . & . & 3 & . \end{bmatrix} + \begin{bmatrix} . & . & . & . & 2 \\ . & . & . & 2 & . \\ 2 & . & . & . & . \\ . & . & 2 & . & . \\ . & 2 & . & . & . \end{bmatrix}$$

$$\text{Discarded:} \; D' = \begin{bmatrix} . & . & . & 1 & . \\ . & . & 1 & . & . \\ . & . & . & . & 1 \\ . & 1 & . & . & . \\ 1 & . & . & . & . \end{bmatrix}$$

Figure 4.3. An example Solstice execution with $W = 100$ and $\delta = 1$. Solstice initially computes a schedule with 6 configurations with a total duration of $69 + 11 + 10 + 2 + 3 + 2 = 97 > W - 6\delta$, so it must be linearly scaled down. In this example, durations must be integer, so the resulting schedule is the same, except the first configuration has a duration of 66.

an efficient decomposition (i.e., serving as much of the demand as possible) and limiting the number of reconfigurations required.

We select configurations in a greedy fashion, searching for configurations that can be fully utilized the longest so as to best amortize the reconfiguration penalty. (While an optimal schedule may in fact contain configurations that are not fully utilized, Solstice attempts to address this issue through stuffing, so at this stage we only schedule configurations for as long as they can be saturated by the stuffed demand.) This problem is known as the Max-Min Weighted Matching (MMWM) problem—which is distinct from the more common Maximum Weighted Matching (MWM) problem. MWM searches for the permutation matrix where the sum of all the included elements is maximized; some elements may be less than others, leading to inefficiency in the schedule. MMWM, on the other hand, searches for the permutation matrix where the minimum of all the picked elements is the maximum possible among all permutations.

In each iteration, Solstice looks for perfect matchings only among those demand matrix elements that are larger than a given threshold (see Algorithm 2), so the selected configuration can be scheduled for a duration larger than that threshold. An optimal MMWM algorithm would consider all $O(N^2)$ different thresholds; Solstice considers an exponentially spaced set of them.

Goel *et al.* show how to find a perfect matching in a regular matrix in $O(N \log N)$ time [18]. D is not necessarily regular, but we follow the same general methods to design an algorithm that takes $O(N \log^2 N)$ in common cases. For a given threshold, Solstice's slicing algorithm, which we call Random Slicing (Algorithm 4), selects a random perfect matching. If it cannot obtain a perfect matching, it returns failure.

4.5.3 Example

We now describe how Solstice operates on the example demand matrix shown in Figure 4.3. First, the input demand matrix D is trimmed by removing all elements strictly less than $2\delta = 2$. Because the resulting matrix, D', has maximum row sum 98, the matrix is then stuffed to obtain a matrix where each row and column sum is 98. This is the matrix D' that Solstice tries to schedule. In the first iteration, Solstice considers a minimum duration of 32 by extracting $D^{\dagger}$, the subset of elements of at least that size. In this example, $D^{\dagger}$ admits only one perfect matching with a minimum-sized element of 69, so the duration of the first configuration is determined to be 69. It substracts the demand from the stuffed matrix, D', and the algorithm continues to consider thresholds of exponentially decreasing size.

It is only when the threshold is reduced to 8 that D' admits another prefect matching. This time, there are two: the first perfect matching RandomSlice identifies has a minimum element of size 11, while the second matching it identifies has elements of size at least 10. After accounting for the demand serviced by both of these configurations, the residual demand matrix D' does not admit another perfect matching until the threshold is reduced to 2. At that point, Solstice is able to identify three more perfect matchings with durations 2, 3, and 2. The final residual is too small to be serviced by any configurations with duration of length at least $2\delta = 2$, so it is discarded.

Unfortunately, once the reconfiguration delays for all six configurations are accounted for, Solstice discovers that the computed schedule exceeds W. Hence, the schedule needs to be linearly scaled down. If we restrict durations to be of integer length, and round non-integer durations to the nearest integer, the resulting schedule simply shortens the duration of the first configuration.

4.6 Evaluation

In this section we evaluate the performance of Solstice under various workloads: where there is skew in demand between large and small flows (exploring the skew assumption), where the demand matrix becomes increasingly saturated with flows (exploring the sparsity assumption), and where source-destination pairs are more random and the flows compete for a fair share of the link bandwidth (exploring greater realism). We also include a validating experiment of running an implementation of Solstice on a hardware testbed, and a preliminary discussion of the time complexity of Solstice.

In each case we simulate a testbed of 64 nodes interconnected by a switch with an accumulation period W of 3 ms and a reconfiguration delay δ of 20 µs.[2] These values are easily supported by our prototype REACToR switch in Chapter 3 that employs the Mordia circuit switch [40]. We further consider two kinds of switches, a pure circuit switch and a hybrid switch; the hybrid switch also has a small packet switch with capacity 10% of the circuit switch. The schedules Solstice creates given an input demand matrix are the same for both kinds of switches, but the hybrid switch can use its packet switch to service residual demand.

4.6.1 Sensitivity to skew

Solstice is designed to take advantage of skew in demand across flows, tailoring its heuristics to schedule workloads consisting of a small number of flows with (relatively) large demands among a background of many more flows with small demands. We start by exploring the behavior of Solstice under such workloads, varying the degree of demand skew among a fixed number of flows.

We generate workloads that have a fixed number of flows per node: 2 are large and 10 are small. The large flows have equal demands that, when combined, are $96(1-x)\%$

[2]Referring back to Figure 4.2, these values correspond to a δ/W of 0.020/3 = 0.67% on those graphs.

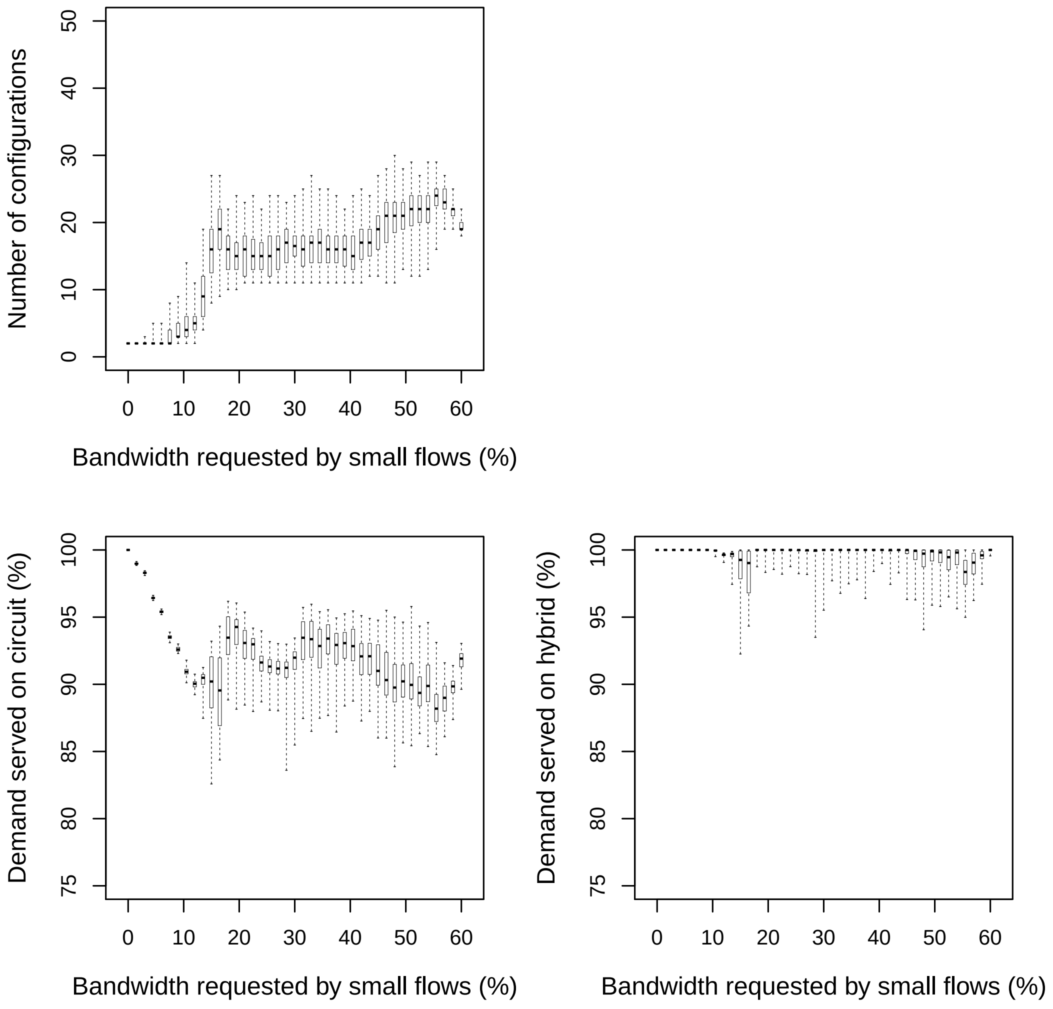


Figure 4.4. The number of configurations and the fraction of demand served when scheduling demand matrices of different skew using Solstice.

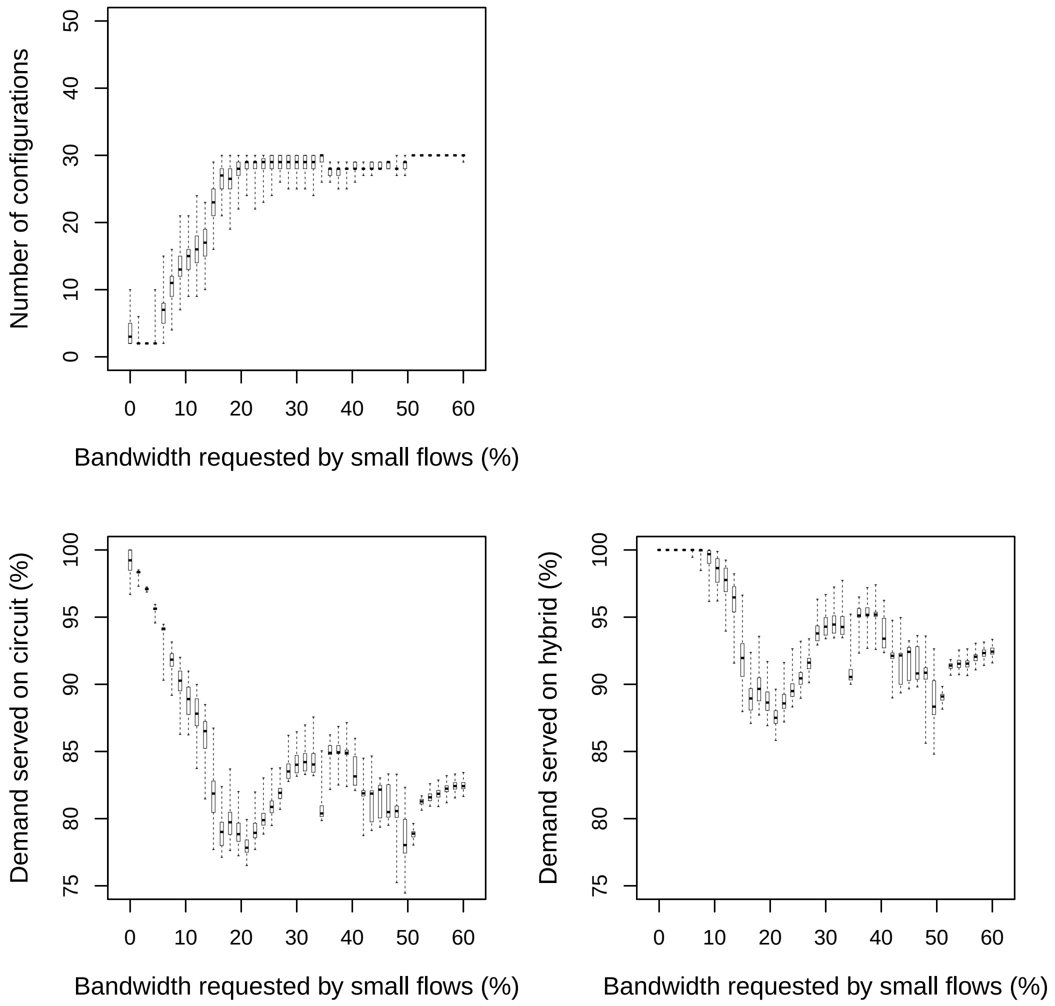


Figure 4.5. The number of configurations and the fraction of demand served when scheduling demand matrices of different skew using BvN with alignment.

of the link bandwidth, and the small flows have equal demands totaling 96x% of the link bandwidth, where x ranges from 0 (all demand is in the large flows) to 0.5 (half of the demand is in the small flows). To introduce noise, we perturb demands by adding $\pm 0.3\%$ of the link bandwidth to flows to avoid unrealistically easy decompositions of the demand matrix. Thus, for every value of x the expected link bandwidth required is 96% and the maximum is 99.6%; no link is oversubscribed. We vary x in 0.05 increments and simulate 100 random workloads at each increment.

For context, the workloads we evaluate have skew and sparsity characteristics commensurate with published data center workloads, yet have significantly higher traffic demands that stress the scheduler. Two one-hour traces from the University of Wisconsin [5] record traffic among 500–1000 servers. Binning this traffic into 3-ms windows and calculating traffic matrices, the maximum number of non-zero elements in a matrix are just 36 and 85 in the two traces, respectively. The links are mostly idle and, for any single host, the number of other hosts it contacts contacts with large flows in any 3-ms window is at most five.

Similarly, we also constructed a workload model based on the flow behavior and size distributions described by Alizadeh *et al.* [3] to represent a workload that combines query traffic that sends packets from one host to all other hosts, while at the same time containing background flows for other applications. Even scaling the workload to be five-times more dense, we observe just a maximum of seven concurrent large background flows per host in a 3-ms window (the paper shows at most four concurrent large flows per host in a 50-ms window). In total the query traffic consumes about 10% of the total link capacity of the switch, where the background flows use about 30%—far less than the workloads we consider here.

Figure 4.4 shows the results of these simulations using three graphs: (a) the number of configurations Solstice outputs in its schedule, and the fraction of requested demand that the schedule serves on (b) a pure circuit switch, and (c) a hybrid switch where a low-bandwidth packet switch can help with residue. Each value of k shows a candle graph denoting the 25th, median, and 75th percentile values across 100 runs as a box, and the minimum and maximum values at the extremes.

The results show three performance regions according to how Solstice treats the large flows compared to the small flows. The first region is ideal for Solstice: the large flows dominate the demand and Solstice creates schedules tailored to them. The small flows in this region are too small to schedule on the switch; they fall below the threshold τ and Solstice discards them from the demand matrix. The linear decrease in served demand with the pure circuit switch in Figure 4.4(b) reflects the increasing fraction of demand in the small flows that is ignored by Solstice. With the hybrid in Figure 4.4(c), served demand remains at 100% because small flow residue goes to the packet switch. (Since $\tau = 2\delta = 0.04$ ms, this scheduling behavior persists when demand across small flows is below $x = 0.04/3 = 13\%$ of total bandwidth.)

The second region, where the number of configurations remains flat, also represents a situation for which Solstice was designed. Here, the small flows rise above the trimming threshold τ and Solstice now also schedules them on the circuit switch. Recall, though, that Solstice partitions demands into exponentially decreasing durations. Hence, Solstice first generates schedules with long durations tailored to the large flows; once it schedules the large flows, it then turns to generating schedules for the small flows, which have correspondingly shorter durations. The serviced demand on the pure circuit switch remains high, with medians at 90% and the worst cases above 80%. A hybrid switch handles nearly all of the residue demands, with median serviced demands just below 100% and worst cases above 90%.

Finally, once the small flows together request x=40% of the link bandwidth, the small flows start to have demands that place them in the same demand partition as the large flows. As a result, Solstice schedules them in conjunction with the large flows. As x continues to increase, eventually all of the small flows are scheduled in the same iteration as the large flows. The number of configurations in the schedules correspondingly increase, and serviced demand decreases such that, even with the hybrid switch, not all demand can be serviced by Solstice's schedules.

To illustrate how a traditional algorithm might interact with the same workload, we simulated the same experiment using BvN with alignment (which factors the demand matrix into coarse and residue matrices) to schedule the demand. We use 100 μs for the alignment quanta so that each configuration results in at least an 80% duty cycle.

Figure 4.5 shows the results. The number of configurations is bounded by the number of alignments in the accumulation period W, which is 30 in this case (a 3-ms period divided by 100-μs alignments). Since BvN with alignment is not designed to minimize the number of configurations, it often generates the maximum of 30 configurations (whereas Solstice uses substantially fewer). Also, the served demand has two valleys when the small flows consist of roughly 25% and 41.5% of total bandwidth. These low utilization areas occur when the total demands of the small flows are in the middle of the alignment (25% is 1.5$\times$ the alignment, and 41.5% is 2.5$\times$). Served demand would drop even lower if there were more small flows.

4.6.2 Sensitivity to saturation

A workload with many flows of roughly equal demand represents a non-ideal case for Solstice because it cannot conveniently divide and conquer the workload according to demand. We evaluate this kind of workload next, in which flows are of nearly equal size and we add flows to increasingly "saturate" the demand matrix with demand elements.

To construct a demand that has k flows per source and destination pair, we randomly generate k perfect matchings. For each source-destination pair in the matching, we

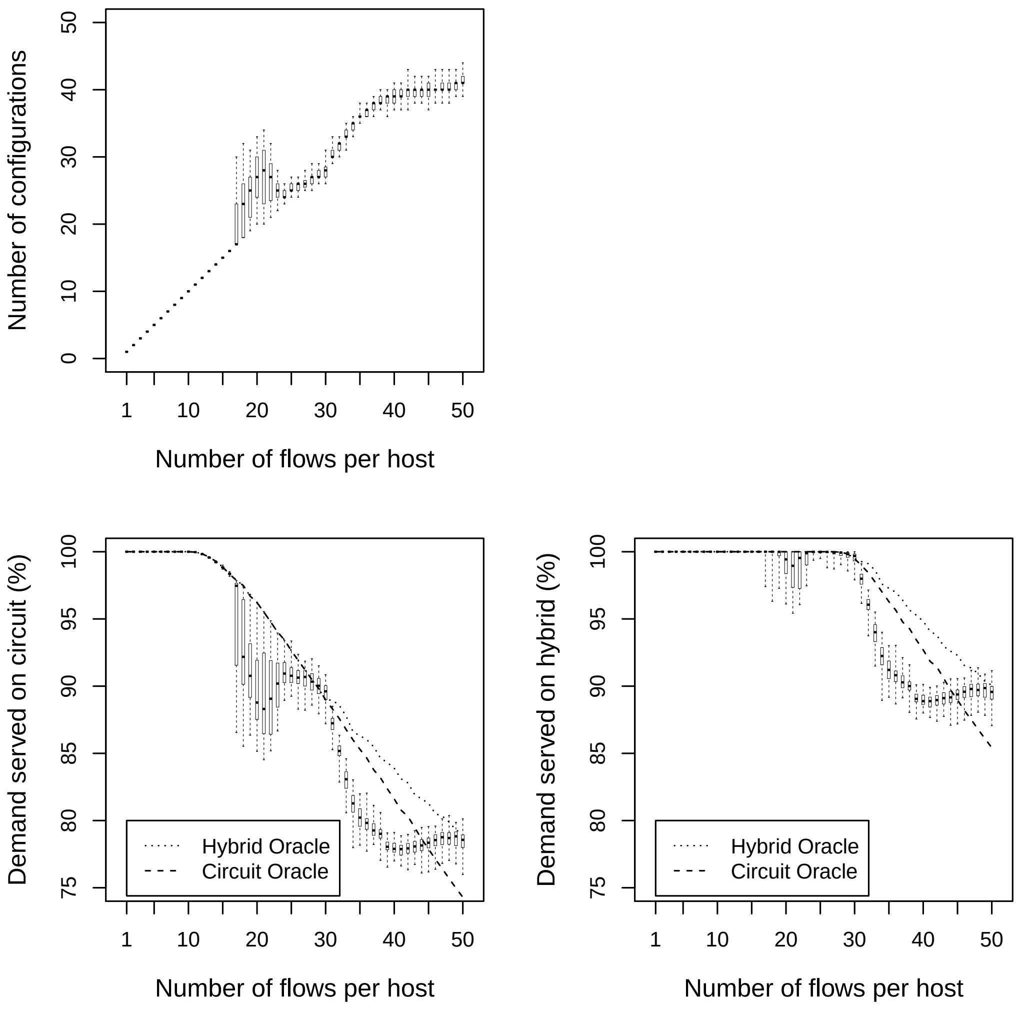


Figure 4.6. The number of configurations and the fraction of demand served when scheduling demand matrices of different saturation.

add a flow into the matrix that requests $90/k\%$ of the link bandwidth on the corresponding uplink and downlink; to introduce noise, we again perturb the demands by $\pm 0.3\%$. Hence the expected bandwidth required across all flows on a link is 90%, and the maximum is 99%; no link is oversubscribed.

We vary k from 1 to 60 flows to increasingly saturate the demand matrix with flows of roughly uniform size. With a reconfiguration delay δ of 20 μs, these values result in a potential link utilization of 95% when the demand is perfectly uniform and $k = 60$. For each value of k, we generate 100 random demand matrices.

Figure 4.6 shows the results of these simulations using similar graphs as in Figure 4.4. In addition, for reference we also show curves (dashed lines) that represent one form of optimal scheduling. Under the constraint that the switch will schedule *every* flow, causing the maximum number of reconfigurations, these lines show the performance of an optimal scheduler. Since in practice there are situations where it is better to not schedule a flow to reduce the number of reconfigurations, Solstice and other schedulers can do better. These reference lines show the regime of a good scheduler.

The results again show three performance regions as above, although for different reasons. The first region matches Solstice's assumptions, in which the number of flows is small and the demand matrix is relatively sparse (i.e., the matrix is sparsely composed of just "large" flows). In this region, Solstice serves the demand with the same number of configurations as the number of flows. Unlike in the skew experiment, there are no small flows and hence no residuals. Consequently, Solstice serves 100% of demand on both pure and hybrid circuit switches.

After 15 flows, the workload reaches an inflection point where Solstice starts allocating additional configurations for each flow. In this region, the combined effects of the noise added to the flows across a row or column makes it more difficult for Solstice to find an optimal schedule. Although the mean value of the noise is still 0, the standard deviation of the sums of the elements in a row or a column increases with the number of flows. In stuffing the demand matrix to be doubly stochastic, Solstice increases the elements in the rows and columns which together have the lowest combined demand. In an attempt to add skew to the demand matrix, Solstice increases only a small number of these elements, but with large increments. As a result, the stuffed demand creates sufficient skew that Solstice splits and schedules the larger elements across more than one configuration. Doing so diverges the schedule from the best plan, lowering demand served by a few percent for the pure circuit switch; demand served remains high with the hybrid switch, though, since the packet switch absorbs the residue.

Beyond 30 flows, the combined effects of the demand variance across the flows in a row or column continue to grow and Solstice fails to discover ideal schedules. In these cases, Solstice gives up scheduling all the flows, leaving 1–4 flows unserved per host. For a pure circuit switch, this behavior results in substantial unserved demand (20% in the worst cases). A hybrid switch does better in absorbing residual demand, but still 10% of demand remains unserved.

4.6.3 Serving random flows

Previous experiments construct demand matrices that are largely stacks of permutations matrices, which are highly coordinated as a result. This construction enables more precise control of the experiment, but is less realistic. To evaluate how Solstice performs with less coordinated demand, we construct a workload with demand skew similar to Section 4.6.1, but instead (a) the source-destination pairs are chosen randomly and (b) the large and small flows sharing a link compete for a fair share of the bandwidth in the spirit of TCP.

Figure 4.7 shows the results of Solstice with this randomized workload. Results are similar to those in Figure 4.4, where the workload was constructed from matrices

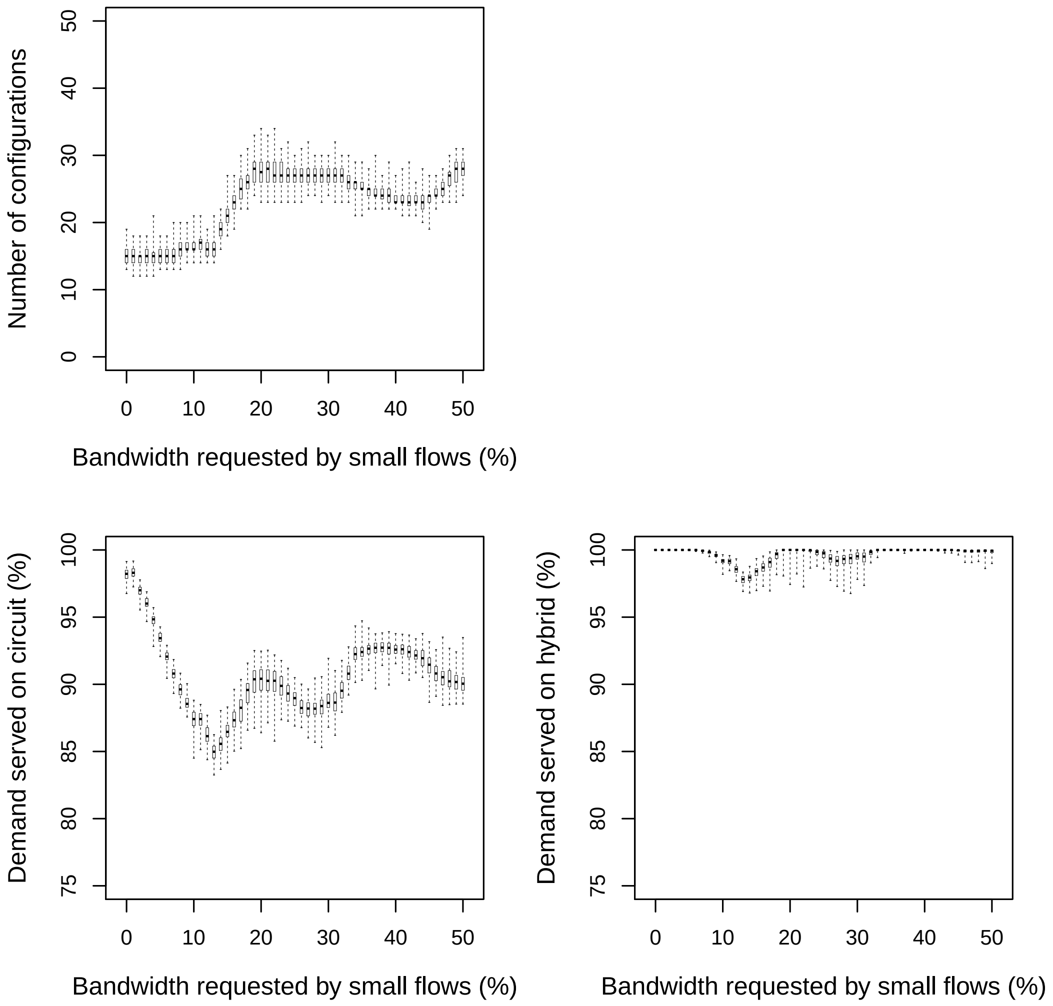


Figure 4.7. The number of configurations and the fraction of demand served when scheduling a randomized, fair sharing workload.

derived from perfect matchings. This similarity is reassuring in that adding more randomness and fair link sharing to the workload does not fundamentally change Solstice's performance.

4.6.4 Time complexity

Finally, we present an initial discussion of the time complexity of Solstice. It is straightforward to prove that the time complexity of the stuffing step is $O(N \log N)$ since it is dominated by the sorting step. The number of configurations in W is bounded by the minimum configuration duration for decomposing, which is a constant. The time complexity of the algorithm hence depends on the time complexity of the perfect matching algorithm.

The perfect matching algorithm has two parts. The first part randomly matches a subset of the nodes to bootstrap the matching, which always takes $O(N)$ time. The second part performs a random order depth-first-search over the augmented tree formed by existing matchings to match an unmatched node, and computation time is linear to the number of edges the search visits in total.

To estimate the time complexity of the perfect matching algorithm in practice, we experimented with large numbers of random permutation matrices and counted the total number of edges the DFS searches. These edge counts were bounded between $O(N \log N)$ and $O(N \log^2 N)$, which is encouraging. However, a formal analysis of Solstice conditioned on its assumptions remains future work.

4.6.5 Solstice on a hardware testbed

While we have explored the behavior of Solstice via simulation, we end by showing that an implementation of Solstice can schedule flows on a small hardware testbed. In Chapter 3, when we evaluated REACToR, we lacked a scheduler and so

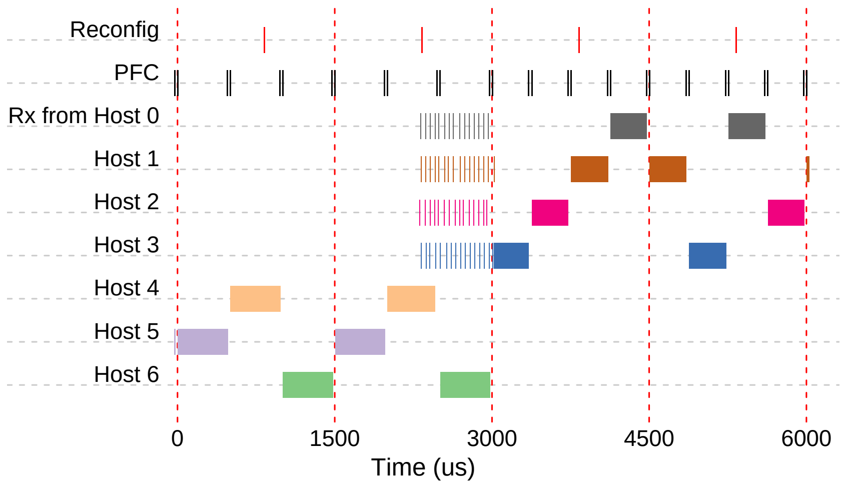


Figure 4.8. Running Solstice in real-time on our prototype testbed.

we precomputed schedules for all of the workloads running on the hardware prototype. Now we repeat one of the experiments from Section 3.3.2 (Figure 3.8) but, instead of precomputing circuit schedules, we use an implementation of Solstice to compute circuit schedules in real-time.

Figure 4.8 shows the incoming packets to host 7 at the workload transition time. We controlled the experiment so that the workload changes during a scheduling period. At the transition, REACToR is halfway through its scheduling period and packets already queued at the first three hosts continue to arrive via circuits. At the end of its committed scheduling period, REACToR receives the updated schedule that Solstice generates for the upcoming scheduling period, and schedules the new circuit configurations to react to the workload transition. In sum, the results when using Solstice to compute the schedules in real-time match the results when we had precomputed the schedules offline, showing that Solstice behaves as expected in practice.

4.7 Practicalities

While our simulations indicate that Solstice is effective at scheduling skewed demand over the model of a hybrid switch that we consider here, our experience implementing Solstice on a real prototype switch identified a number of practical constraints that, if formally addressed, would lead to an even more effective algorithm.

Port count/delay tradeoff. In our simple hybrid model, we assume that both the circuit and packet switches have the same number of ports. However, the technologies available today often present a tradeoff between port count and reconfiguration penalty. Optical circuit switches rely on two families of underlying technology to implement crossbar functionality, binary MEMs and planar wave guides, which currently scale to a modest number of ports. Similarly, hybrid networks using RF-based wireless circuits [21, 26, 49], when coupled with electronically steerable phased-array antennas, have limited separable degree angles.

Hence, when designing a scheduling algorithm it may be fruitful to consider hybrid switches where the circuit switch has fewer than N ports. Such a design can be thought of containing either an $M < N$ port non-blocking circuit crossbar, or an N-port blocking crossbar. In that case, the scheduling problem relaxes from trying to find permutation matrices of dimension N for each configuration to finding perfect matchings of sub-matrices of D of size M.

Circuit/packet crossbar co-scheduling. As currently formulated, Solstice leaves a residual for the packet switch to "take care of." Instead, another opportunity is to compute a deliberate schedule for the packet switch as well. Doing so could lead to not only cheaper hybrid implementations (by dispensing with the cost, heat, and power of the buffer memory) but potentially higher performance if one could jointly optimize the schedules of the two crossbars. Alternatively, a simplified version might still ignore the details of the packet switch, and instead focus on the properties of the residual demand it must (try to) service. An alternative algorithm might constrain the residual so that it is more easily (or predictably) handled by the packet switch. Another consideration is the order in which demand is presented to the packet switch. A more sophisticated alternative could reorder the schedule to control when the packet switch might service different queues to, e.g., minimize the overall waiting time.

4.8 Summary

The ever-increasing demand for low-cost, high-performance network fabrics in data center environments has generated tremendous interest in alternative switching architectures. We presented REACToR in Chapter 3, and in this chapter we develop a switching scheduler for it. We provide a general model of the problem, differentiate it from the classical constrained switching problems, and propose an initial algorithm, Solstice, that performs well for a class of demands that attempts to model traffic observed in a real data center. We also implement Solstice in our REACToR prototype hybrid switch that schedules traffic demand in real-time and we look forward to evaluating its performance with more realistic data center applications.

This chapter, in part, has been submitted for publication of the material as it may appear in the the conference of the ACM Special Interest Group on Data Communication 2015. Liu, He; Kapoor, Rishi; Tewari, Malveeka; Forencich, Alex; Zhang, Sen; Savage, Stefan; Voelker, Geoffrey M.; Papen, George; Snoeren, Alex C.; Porter, George. The dissertation author was the primary investigator and author of this material.

Chapter 5

Conclusion and Future Work

Parallel topologies like a fat-tree works well for current data centers, but it is expensive to directly upgrade these electrical packet switching networks into 100 Gb/s. A dominant part of the upgrade cost is the optical transceivers that connect the optical links and the electrical packet switches. Using optical circuit switches provides a lower-cost solution by using fewer transceivers, and recent optical switching technology that can reconfigure in microseconds enables these circuit switches to service more dynamic traffic demands like the ones that occur at data center top-of-rack switches.

Leveraging this technology trend, this dissertation presented REACToR, a new hybrid circuit switch design that combines an optical circuit switching network and a lower-provisioned electrical packet switching network that can service data center traffic with performance akin to a fully provisioned electrical packet switching network, but with much lower costs. It services multiple long-lasting flows using the optical circuit network in a TDMA fashion without disruptions on the transport layer, and services the small flows with the lower-provisioned electrical packet network, while presenting to end hosts as a fully-provisioned electrical packet network.

To efficiently utilize the optical circuits to service the long-lasting flows in the REACToR network for data centers, this dissertation also presented Solstice, a scheduling algorithm that minimizes reconfiguration frequency and hence maximizes circuit utilization and switching efficiency. Using Solstice, REACToR is able to reactively schedule the heavy, bandwidth-hungry flows to the optical circuit switching network, and the small, latency-sensitive flows to the electrical packet switching network.

REACToR extends optical circuit switching to a layer closer to the data center end hosts, and reduces the usage of expensive optoelectronic transceivers. Leveraging the skewness patterns in data center traffic demands, Solstice efficiently schedules the available link capacities that REACToR provides, and delivers bandwidth that is close to fully bi-sectional. With REACToR and Solstice I present a new approach that is cheaper than conventional ones when upgrading data center networks from 10 Gb/s to 40 Gb/s and to 100 Gb/s, but with comparable network performance.

5.1 Future Directions

I end this dissertation by discussing issues of the current design of REACToR and Solstice, and how these issues could be addressed in the future.

5.1.1 Shorten the reaction delay

Although REACToR is fast enough to “fly under the rader” on servicing long-lasting TCP flows, it currently still incurs several hundreds of microseconds to react to a traffic pattern change with a proper circuit. While this reaction latency is negligible to flows that last for seconds, it is non-trivial for a flow that only last for several 10s of microseconds or even shorter. As link rates upgrade to 40 and 100 Gb/s, flows of the same size will also take shorter time to transfer, and hence make the schedule reaction delay issue a more significant latency bottleneck.

The current mechanism calculates the schedule every scheduling period, where each period is about one millisecond long. This scheduling mechanism works well because it delivers the best possible link throughput. However, since data center network traffic is often bursty, it is likely that the number of active big flows will only be one or two at any time, especially when the link rates are high. As a result, if the scheduler can output schedules as a stream of reconfigurations in a just-in-time fashion rather than in a windowed fashion, it might shorten the service delays on reacting to demand changes. Unfortunately, this mechanism change requires logic changes over the entire control plane. The scheduler needs to take an input stream of demand update notifications (rather than a demand matrix), and continuously deploy a stream of circuit configurations to the circuit switch and the end-hosts within a much tighter real-time constraint. This change from windowed to streaming mechanism introduces challenges on both designing the algorithm and implementing the control-loop mechanism.

5.1.2 Work with TCP harmoniously

Working with upper-level transport protocols like TCP might also be a challenging open problem. REACToR essentially has two paths from each source host to each destination host, and each path has different available bandwidths. Because traditional TCP keeps track of the link state essentially as a single link, it works well when the link state is stable, but works sub-optimally when the link state is changing. As a result, when the scheduler migrates a flow from the circuit link to the packet link or vise-versa, the transport layer might not be able to respond to the sharp changing of the link state and cannot react quickly enough to fully utilize the available link capacity.

There are several possible ways to address the issue above. The end hosts could use multiple TCP flows to track the link state of both the circuit and the packet switching path, so that the states are distinguished. Or, the scheduler could smoothly increase or reduce the bandwidth allocation of a flow, and hence remove the sharp change on link state. Alternatively, the controller can send TCP the change in real-time by sending additional control packets or modifying TCP packet headers.

5.1.3 Validate the workload assumption

It is unclear what specific data center applications require from the network in terms of quality of service. REACToR's model might not fit the needs of all applications, especially when they scale up to higher link rates. For example, when the link rate increases by $10\times$, it is unclear if the applications tend to connect more hosts at the same time, or tend to communicate with the same number of hosts but with longer flows. A study about the scaling property of data center workloads would be informative, because it directly affects the efficiency of using circuit switching in higher link rate scenarios.

5.1.4 Bound the scheduling algorithm

The scheduling problem, though formally described, still lacks an upper-bound on the best possible link efficiency of scheduling a demand matrix in general. The lack of theoretic bounds makes it hard to evaluate if an algorithm is good enough.

The Solstice scheduling algorithm empirically shows a $O(N\log^2 N)$ time complexity for normal cases where N is the number of hosts, but it still lacks a formal proof about its worst-case time complexity. Also, in its current shape, when scaling it up to a more realistic data center scale with thousands or even hundreds of thousands of end hosts, even $O(N)$ time complexity will not be acceptable. The algorithm must be parallelized to be practical.

5.1.5 Synchronize the control plane at scale

Setting up buffer-less optical circuits requires tight time synchronization with the sending hosts. This real-time setup is a challenging task for a data center of thousands of machines spread over hundreds of square meters. Even synchronizing with signals traveling at the speed of light, it takes about 0.3 μs to send a signal from a control host to a machine that is about 100m away. This delay is equivalent to transmitting 2 MTU size Ethernet packets on 100 Gb/s links. It also means that when technology can push the reconfiguration delay of the circuits to nanoseconds, this synchronization signal propagation delay will be the dominant part of the system reconfiguration delay, which limits the type of traffic demand the circuit switching network can service.

To reduce this delay, the controller that sends these synchronization signals must also take the propagation delay into consideration as well, and it is likely that this propagation delay will be different for machines at different locations. Therefore, to configure the switches and the end hosts correctly, the propagation delay must be measured first. Ideally, the control plane system should measure these delays automatically, rather than let the administrator configure them manually. If the circuit switch is not a big monolithic switch, but consists of multiple smaller circuit switches, this issue also applies to the synchronization among these distributed switches.

Bibliography

[1] Open Compute Project. http://www.opencompute.org.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity, Data Center Network Architecture. In *Proc. ACM SIGCOMM*, August 2008.

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *Proc. ACM SIGCOMM*, August 2010.

[4] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing Router Buffers. In *Proc. ACM SIGCOMM*, October 2004.

[5] Theophilus Benson, Aditya Akella, and David A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. ACM Internet Measurement Conference*, November 2010.

[6] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–151, 1946.

[7] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[8] Nicola Calabretta, Roger Rueyo Centelles, Stefano Di Lucente, and Harmen J.S. Dorren. On the Performance of a Large-Scale Optical Packet Switch Under Realistic Data Center Traffic. *Journal of Optical Communications and Networking*, 5(6):565–573, June 2013.

[9] Cheng-Shang Chang, Wen-Jyh Chen, and Hsiang-Yi Huang. Birkhoff-von Neumann Input Buffered Crossbar Switches. In *Proc. IEEE INFOCOM*, March 2000.

[10] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping

Zhang, Xitao Wen, and Yan Chen. OSA: An Optical Switching Architecture for Data Center Networks and Unprecedented Flexibility. In *Proc. of 9th USENIX NSDI*, April 2012.

[11] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th ACM/USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, CA, December 2004.

[12] Nathan Farrington, Alex Forencich, George Porter, P.-C. Sun, Joseph E. Ford, Yeshaiahu Fainman, George C. Papen, and Amin Vahdat. A Multiport Microsecond Optical Circuit Switch for Data Center Networking. *IEEE Photonics Technology Letters*, 25(16):1589–1592, June 2013.

[13] Nathan Farrington, George Porter, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Hunting Mice with Microsecond Circuit Switches. In *Proc. ACM HotNets*, October 2012.

[14] 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.

[15] Shu Fu, Bin Wu, Xiaohong Jiang, Achille Pattavina, Lei Zhang, and Shizhong Xu. Cost and Delay Tradeoff in Three-Stage Switch Architecture for Data Center Networks. In *Proc. of 14th IEEE High Performance Switching and Routing*, July 2013.

[16] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, October 2003.

[17] Paolo Giaccone, Balaji Prabhakar, and Devavrat Shah. Randomized Scheduling Algorithms for High-Aggregate Bandwidth Switches. *IEEE Journal on Selected Areas in Communications*, 21(4), May 2003.

[18] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Perfect Matchings in $O(n \log n)$ Time in Regular Bipartite Graphs. In *ACM STOC*, 2010.

[19] Inder S. Gopal and C. K. Wong. Minimizing the Number of Switchings in a SS/TDMA System. *IEEE Trans. Communications*, 33(6), June 1985.

[20] J. Haglund and J.B. Remmel. Rook Theory for Perfect Matchings. *Advances in*

Applied Mathematics, 27:438–481, 2001.

[21] Daniel Halperin, Srikanth Kandula, Jitendra Padhye, Paramvir Bahl, and David Wetherall. Augmenting Data Center Networks with Multi-gigabit Wireless Links. In *Proc. ACM SIGCOMM*, August 2011.

[22] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. In *SIAM Journal on Computing*, July 1973.

[23] Thomas Inukai. An Efficient SS/TDMA Time Slot Assignment Algorithm. *IEEE Trans. Communications*, 27(10), October 1979.

[24] V. Jacobson and R. Braden. TCP Extensions for Long-Delay Paths. RFC 1072 (Informational), October 1988.

[25] Natalie Enright Jerger, Mikko Lipasti, and Li-Shiuan Peh. Circuit-Switched Coherence. *Computer Architecture Letters*, 6(1):5–8, July 2007.

[26] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. Flyways To De-Congest Data Center Networks. In *Proc. ACM HotNets*, October 2009.

[27] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The Nature of Data Center Traffic: Measurements & Analysis. In *Proc. ACM IMC*, November 2009.

[28] Rishi Kapoor, Alex C. Snoeren, Geoffrey M. Voelker, and George Porter. Bullet Trains: A Study of NIC Burst Behavior at Microsecond Timescales. In *Proc. ACM CoNEXT*, December 2013.

[29] Benjamin G. Lee, Alexandar V. Rylyakov, William M. J. Green, Solomon Assefa, Christian W. Baks, Renato Rimolo-Donadio, Daniel M. Kuchta, Marwan H. Khater, Tymon Barwicz, Carol Reinholm, Edward Kiewra, Steven M. Shank, Clint L. Schow, and Yurii A. Vlasov. Four- and Eight-Port Photonic Switches Monolithically Integrated with Digital CMOS Logic and Driver Circuits. In *Proc. OFC/NFOEC*, March 2013.

[30] Xin Li and Mounir Hamdi. On Scheduling Optical Packet Switches with Reconfiguration Delay. *IEEE Journal on Selected Areas in Communications*, 21(7), September 2003.

[31] Dan M. Marom. Switching Capacity of MEMS Tilting Micromirrors. In *Proc. IEEE Optical MEMS and Nanophotonics*, August 2012.

[32] James Martin, Katheleen Kavanagh Chapman, and Joe Leben. *Asynchronous Transfer Mode: ATM Architecture and Implementation*. Prentice-Hall, Inc., 1997.

[33] Nick McKeown. The iSLIP Scheduling Algorithm for Input-Queued Switches. *IEEE/ACM Transactions on Networking*, 7(2), April 1999.

[34] Nick McKeown, Venkat Anantharam, and Jean Walrand. Achieving 100% Throughput in an Input-Queued Switch. In *Proceedings of IEEE Infocom Conference*, March 1996.

[35] Adisak Mekkittikul and Nick McKeown. A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches. In *Proceedings of IEEE Infocom Conference*, March 1998.

[36] Thomas Moscibroda and Onur Mutlu. A Case for Bufferless Routing in On-Chip Networks. In *Proc. ISCA*, June 2009.

[37] Robert Olsson. pktgen the linux packet generator. *Proc. Linux Symposium*, July 2005.

[38] Quan-Ke Pan and R. Ruiz. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, 40:117–28, January 2013.

[39] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A Centralized "Zero-Queue" Datacenter Network. In *Proc. ACM SIGCOMM*, August 2014.

[40] 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 *Proc. ACM SIGCOMM*, August 2013.

[41] Sivasankar Radhakrishnan, Yilong Geng, Vimalkuma Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. SENIC: Scalable NIC for End-Host Rate Limiting. In *Proc. of 11th USENIX NSDI*, April 2014.

[42] Ireneusz Szcześniak. Overview of optical packet switching. In *Theoretical and Applied Informatics*, October 2009.

[43] M. Tandon, P.T. Cummings, and M.D. LeVan. Flowshop sequencing with non-permutation schedules. *Comp. & Chem. Eng.*, 15(8), 1991.

[44] Brian Towles and William J. Dally. Guaranteed Scheduling for Switches with Configuration Overhead. *IEEE Trans. Networking*, 11(5), October 2003.

[45] Amin Vahdat, Mohammad Al-Fares, Nathan Farrington, Radhika Niranjan Mysore, George Porter, and Sivasankar Radhakrishnan. Scale-Out Networking in the Data Center. *IEEE Micro*, 30(4):29–41, August 2010.

[46] 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*, August 2010.

[47] Bin Wu and Kwan L. Yeung. Minimum Delay Scheduling in Scalable Hybrid Electronic/Optical Packet Switches. In *Proc. IEEE GLOBECOM*, 2006.

[48] Bin Wu, Kwan L. Yeung, and Xin Wang. Improving Scheduling Efficiency for High-Speed Routers with Optical Switch Fabrics. In *Proc. IEEE GLOBECOM*, December 2006.

[49] 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 *Proc. ACM SIGCOMM*, August 2012.