Title
Simulation to scale of the HELIOS system

Permalink
https://escholarship.org/uc/item/4q2890b0

Author
Subramanya, Vikram

Publication Date
2010

Peer reviewed|Thesis/dissertation
UNIVERSITY OF CALIFORNIA, SAN DIEGO

Simulation to Scale of the HELIOS System

A thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Computer Science

by

Vikram Subramanya

Committee in charge:

Professor Amin Vahdat, Chair
Professor Alex C. Snoeren
Professor George Varghese

2010
The thesis of Vikram Subramanya is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

________________________________________

________________________________________

________________________________________

Chair

University of California, San Diego

2010
DEDICATION

To my adorable parents Mr. P Subramanya and Mrs. K N Savithri, my beloved brother Vijay Subramanya, and my charming grandmother Mrs. P Sandhyavali.
A lot of times people think they’re crazy,
but in that craziness, we see genius.

And they are the ones we’re making our tools for.

—Steve Jobs, in 1997, heralding a Apple’s comeback from near-collapse
# TABLE OF CONTENTS

Signature Page .......................................................... iii  
Dedication ................................................................. iv  
Epigraph ................................................................. v  
Table of Contents ...................................................... vi  
List of Figures ......................................................... viii  
List of Tables .......................................................... ix  
Acknowledgements ..................................................... x  
Vita and Publications .................................................. xi  
Abstract of the Thesis ................................................ xii  

Chapter 1  Introduction ................................................ 1  
  1.1  Trends in Datacenter Networking ............................... 1  
      1.1.1  Movement towards Modular Datacenters .......... 2  
  1.2  Optical Networking and Circuit Switching ............... 4  
  1.3  HELIOS: Hybrid Electrical/Optical Switch Architecture . 6  
      1.3.1  Basis for HELIOS Solution: Key Features .... 6  
      1.3.2  Suitability of HELIOS for Datacenter Workloads . 7  
  1.4  Simulator for HELIOS ........................................ 8  
      1.4.1  Need for Simulations ............................... 8  
      1.4.2  Likely Challenges ................................... 8  
  1.5  Organization of the Thesis ................................ 9  

Chapter 2  Related Work ............................................... 10  
  2.1  Circuit Switching with Optical Interconnects ........... 10  
  2.2  Network Simulation .......................................... 12  

Chapter 3  Research Motivation for the Simulator ............... 14  
  3.1  Research Questions and Hypotheses ....................... 14  
      3.1.1  Pod Switch Scalability ............................ 15  
      3.1.2  Link Aggregation ................................... 15  
      3.1.3  Proportion of Core Packet Switches ............... 15  
      3.1.4  Bisection Bandwidth Utilization Rate .......... 16  
      3.1.5  Traffic Patterns ................................... 16  
      3.1.6  Running Times of the Simulator ................... 16
Chapter 4  System Architecture, Design and Implementation .... 17
  4.1  Topology and Experimental Testbed ............... 18
  4.2  Design: The Control Loop ........................ 19
    4.2.1  Software Components ........................ 19
    4.2.2  Load Generator ............................ 21
    4.2.3  Control Loop Steps ......................... 21
    4.2.4  Edmonds's Algorithm ....................... 23
    4.2.5  Observations: Phase-wise Utilization of Cycle Time 24
  4.3  Working of the Simulator ........................ 26
    4.3.1  Principles of HELIOS Network Simulation ... 26
    4.3.2  Simulation Methodology ....................... 27
  4.4  Control Flow and Implementation .................. 28
    4.4.1  Control Flow of the Simulator ............... 29

Chapter 5  Evaluation of HELIOS System Using the Simulator .... 32
  5.1  Methodology ..................................... 32
    5.1.1  Traffic Patterns ............................ 33
    5.1.2  Abbreviations, Symbols, and Terms ............ 34
  5.2  Validation: Comparison to the Ground Truth .......... 35
    5.2.1  Experiment 1: Stride traffic pattern ............ 36
    5.2.2  Experiment 2: Variation of stability .......... 37
    5.2.3  Explaining Simulator Inaccuracy ................ 40
  5.3  Simulation Results: Employing the Simulator for Prediction . ...... 41
    5.3.1  Pod Switch Scalability ....................... 41
    5.3.2  Link Aggregation ............................ 42
    5.3.3  Proportion of Core Packet Switches ............ 44
    5.3.4  Bisection Bandwidth Utilization Rate .......... 50
    5.3.5  Running Times of the Simulator ............... 54
  5.4  Summary: Research Findings ....................... 55

Chapter 6  Conclusion and Future Work ....................... 57
  6.1  Enhancing the Simulator Functionality ............... 57

Bibliography ........................................... 60
**LIST OF FIGURES**

| Figure 4.1: | HELIOS topology consisting of an array of electrical and optical core switches that interconnect a set of pods | 18 |
| Figure 4.2: | Experimental setup of 6-pod/5-host HELIOS testbed, along with description of the switches/hardware | 20 |
| Figure 4.3: | Control loop of HELIOS | 22 |
| Figure 4.4: | Time taken by various phases during a cycle in HELIOS | 25 |
| Figure 5.1: | Simulator vs. Testbed Comparison: Bisection bandwidth in a 4-pod/5-host HELIOS network | 36 |
| Figure 5.2: | Matrices showing post-reconfiguration Circuit Connection Maps at points A...F in Figure 5.1. Also shown are corresponding simulator *tick* and testbed *cycle* numbers | 38 |
| Figure 5.3: | Effect of variation of $N$ on Bisection Bandwidth for Stride traffic pattern | 42 |
| Figure 5.4: | Effect of variation of $N$ on Bisection Bandwidth for Random traffic pattern | 43 |
| Figure 5.5: | Effect of variation of $N$ on Bisection Bandwidth for Crawl Stride traffic pattern | 44 |
| Figure 5.6: | Effect of variation of $w$ on Bisection Bandwidth for Stride traffic pattern | 45 |
| Figure 5.7: | Effect of variation of $w$ on Bisection Bandwidth for Random traffic pattern | 46 |
| Figure 5.8: | Effect of variation of $w$ on Bisection Bandwidth for Crawl Stride traffic pattern | 47 |
| Figure 5.9: | Effect of variation of $p$ (with oversubscription on optical paths) on Bisection Bandwidth for Stride traffic pattern | 48 |
| Figure 5.10: | Effect of variation of $p$ (without oversubscription on optical paths) on Bisection Bandwidth for Stride traffic pattern | 49 |
| Figure 5.11: | Effect of variation of $p$ (with oversubscription on optical paths) on Bisection Bandwidth for Random traffic pattern | 50 |
| Figure 5.12: | Effect of variation of $p$ (without oversubscription on optical paths) on Bisection Bandwidth for Random traffic pattern | 51 |
| Figure 5.13: | Effect of variation of $p$ (with oversubscription on optical paths) on Bisection Bandwidth for Crawl Stride traffic pattern | 52 |
| Figure 5.14: | Effect of variation of $p$ (without oversubscription on optical paths) on Bisection Bandwidth for Crawl Stride traffic pattern | 54 |
LIST OF TABLES

Table 1.1: Differences between Optical/Circuit Switching and Electrical/-Packet Switching ........................................ 5
Table 5.1: Statistics for 4-pod/5-host Stride Pattern Experiment ........ 37
Table 5.2: Simulator vs. Testbed Comparison: Average bisection band-
width as a function of the host-level stability of traffic ........... 39
Table 5.3: Average Bisection Bandwidths and Utilization Rates for Simu-
lation Experiments .......................................................... 53
Table 5.4: Simulator Running Times for Stride Traffic Pattern .......... 55
ACKNOWLEDGEMENTS

First and foremost, I wish to thank my adviser Prof. Amin Vahdat, who has been much more than Master’s adviser for me. Amin has been a guide, a philosopher, and a friend – he has shown me the right path, rapped on my knuckles when I strayed away from it, and encouraged me to experience the sense of fulfillment that comes from a hard day’s work. Today, if I have a thesis that is coherent and provides research insight into the working of a system, it is because it has passed through the careful eyes of Amin.

I would like to thank my teammates in this research – Hamid Bazzaz, Nathan Farrington, George Porter, and Sivasankar Radhakrishnan. Each one has made a unique and significant contribution to the project. Without the near-constant support I have received from them – in the form of reuse of Siva’s code to George’s thesis reviews to Nathan’s push for graphs to Hamid’s succinct explanation of concepts – this thesis would not be in the current shape. My acknowledgments also go to the rest of the members of Amin Vahdat’s Triton Data Center Networking or DCSwitch group (http://dcswitch.sysnet.ucsd.edu/contact.html). I will sorely miss the lively discussions about the future of datacenter networking in our weekly meetings on Wednesday afternoons.

This thesis is, in part, based on the publication titled “Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers” [FPR+10] set to appear in ACM SIGCOMM 2010, New Delhi, India. The thesis author was a co-author of this paper.
VITA

2004-2008 Bachelor of Technology in Computer Engineering, National Institute of Technology Karnataka Surathkal (NITK), India

2006 Summer Research Student in Dept. of Computer Science, University of Otago in Dunedin, New Zealand

2007 Summer Project Intern in Autonomic Computing Group, IBM India Software Labs, Bangalore, India


2008 Research Assistant in Dept. of Computer Science and Automation (CSA), Indian Institute of Science (IISc), Bangalore, India

2008-2010 Master of Science in Computer Science, University of California, San Diego

2008-2010 Graduate Student Researcher (GSR) / Teaching Assistant (TA), University of California, San Diego

2009 Engineering Intern in Platforms Networking Team, Google Inc., Mountain View, CA

PUBLICATIONS


Within the space of a few years, IT companies with need for large-scale datacenters are considering alternatives to building traditional central datacenter facility. They have embraced a modular datacenter (also called pod, a smaller mobile datacenter) as the building block for reasons such as efficient cooling, power savings, mobility, easier deployment and administration. However, delivering scalable inter-pod bisection bandwidth remains a challenge. Current solutions that promise full bisection bandwidth between arbitrary pair of hosts can be expensive and power-intensive for interconnecting modular datacenters. Our recent work [FPR+10] proposed HELIOS, a scalable switch architecture for interconnecting modular datacenters using a hybrid core switch array consisting of electrical and optical switches. We show that HELIOS delivers a significant reduction in cost,
cabling complexity, and power while providing scalable on-demand bandwidth to the communicating pair of hosts.

In this thesis, we propose a TCP flow-based simulator for the HELIOS architecture. We describe the design/implementation, and validate the correct behavior of the simulator with the testbed execution of HELIOS prototype. We then evaluate the performance of HELIOS system against a variety of parameters – like scale (number of pods), link aggregation, and the composition of the core switch array – for differing communication patterns. The primary goal of this tool is to provide insights into how HELIOS might scale. Possible applications of this tool include network planning, searching newer optimizations, identifying bad application performance, planning VM migration such that the VMs better utilize HELIOS infrastructure.
Chapter 1

Introduction

1.1 Trends in Datacenter Networking

As more and more data are pushed into the cloud, the last few years have seen an explosion in the size of datacenters. Massive datacenters are commonplace in the industry these days. What is also notable is the remarkable change in perception towards datacenter sizes – a server farm of 200,000 servers, that can, be termed a “moderately-sized” datacenter today, constituted one of the largest ones in the world just two years back [Sha08]. Another trend contributing to this near-constant expansion of datacenters is that companies have realized that the key to cost-cutting lies in improving software intelligence rather than trimming their hardware portfolio. Google, owner of the world’s largest search engine, reached this conclusion as far back as in 2003 [BDH03]. Google Fellow Jeff Dean claims that “it is better to have twice as much unreliable, cheap commodity PCs than half as much reliable expensive ones” – the software has the capability to mask failures.

The network inside datacenter has been playing an increasingly central role as datacenters continue to expand. Large-scale Internet applications such as web search, social networks, and e-commerce typically require communication between hosts at different ends of the datacenter. It was estimated that a typical web search query on Google hits 700-1000 servers [Sha08]. The need for provisioning full bandwidth between any two arbitrary hosts has never been greater.
To provide high bisection bandwidth, the datacenter networking research community has explored various interconnect topologies – examples include fat trees [AFLV08], DCell [GWT+08], and BCube [GLL+09]. While these geometries may provide full bisection bandwidth, they come with significant costs in increased wiring complexity, number of switches, and the associated labor, administrative, and power costs.

1.1.1 Movement towards Modular Datacenters

Recently however, there has been a minor trend away from massive data centers. While all the reasons that favor huge datacenters are still valid, certain technical and non-technical issues have contributed many companies to see reason in having many smaller datacenters called modular datacenters [Ham07] instead of a few large ones. Technical issues working against large datacenters are the following:

- **Power consumption**: Large datacenters need massive infrastructure of cooling systems. As the datacenter grows, it becomes harder to focus the target area for cooling since the walls of the datacenter act as heat sinks, increasing the cost of cooling per square foot [Nor10].

- **Heat density limits**: There are limits on heat density within a room hosting datacenter imposed by the various operating equipments, machines, processors, and disks

- **End-to-end intra-datacenter latency**: As Ethernet does not scale to datacenter dimensions, the latency in communicating between pair of hosts across a large datacenter building at layer-3 can be significant.

- **Administrative burden**: Hundreds of thousands of machines have to be administered – fault isolation and replacement, name management, wiring complexity, increased labor costs at scale etc.

There are some non-technical issues as well that go against large datacenters: political and security constraints, taxation laws, real estate etc.
A modular datacenter, also called a *pod*, is a mobile, virtualized, and self-contained datacenter “in a box” which can be shipped to the client. It was designed for rapid deployment and energy efficiency. Vendors like HP, IBM and others offer to build larger datacenters by interconnecting such pods [PI09, Can09], each pod supporting up to a thousand servers. Companies with huge datacentric focus such as Google, Microsoft, and Amazon have built larger datacenters out of pods.

Modular datacenters were originally conceived to take advantage of the localised communication patterns within datacenters [Ham07], with the inter-pod communication kept minimal. However, that premise was short-lived. The challenges facing modular datacenters of today are listed below:

1. **Large-scale services**: Large web services like search and e-commerce are being migrated over to modular datacenters (because pods have become the granule to build larger datacenters). These services typically require substantial communication between pods.

2. **Remote communication patterns**: As communication patterns evolve, certain subsets of datacenter nodes across multiple pods may become tightly coupled [FPR+10]. For example, in Amazon’s virtual machine infrastructure, a design decision might be to keep storage (S3) and computation (EC2) servers in separate pods. This would necessitate sufficient bandwidth for inter-pod communication to use the infrastructure.

3. **Network knowledge of modularity**: Another key challenge is to make the network/switches aware of the modular nature of the topology. This will allow the network to leverage localization better than a topology-agnostic network.

Current modular datacenters provide (close to) full bisection bandwidth for intra-pod communication. However, ensuring adequate bisection bandwidth across arbitrary pair of pods without causing network bottlenecks still remains a significant challenge. Since the communicating set of nodes dynamically change, static provisioning of bandwidth between a certain set of nodes would not help.
Hence, the only option that ensures full bisection bandwidth is to build a non-blocking switch fabric for the entire datacenter. This being prohibitively expensive, the datacenter designers compromise on the bisection bandwidth to limit their costs. They resort to reducing the inter-pod capacity by some oversubscription ratio [cis07]. We believe that before this problem is adequately addressed, the push towards modular datacenters will be premature.

The widely held notion is that – to provision for dynamic “on-demand” bandwidth between arbitrary pairs of hosts, we have to allocate “full” bisection bandwidth between pods [AFLV08, GWT+08, GHJ+09]. In this thesis, we seek to dispel this notion. We believe delivering full bisection bandwidth at the scale of entire datacenter is not only prohibitively expensive and labor intensive, but might also be an overkill [WAK+09]. Traffic patterns in datacenters has gathered steam among research community of late [KSG+09, BAAZ09]. Full bisection bandwidth is rarely, if at all, required, as all-to-all communication is not the common case communication pattern in a datacenter. Analyses of traffic patterns for applications run in datacenters reveal that there is certain component of traffic that remains stable, even while other components are bursty. It would be a good idea to provision for enough bandwidth to this stable component of traffic.

1.2 Optical Networking and Circuit Switching

It is in this context that optical circuit switching with Wavelength Division Multiplexing (WDM) holds promise. Optical/circuit switching is known to have many advantages over electrical/packet switching – lesser power consumption, higher capacity of an optical port etc. However, optical switching is not widely used in traditional networks for the reason that reconfiguring circuits is time-intensive. Table 1.2 enumerates the key differences between optical circuit switching and electrical packet switching.

Since most of the Internet traffic is bursty, circuit switching is ill-suited. However, the stable component of traffic in datacenters provide a potential opening for the use of optical switches.
Table 1.1: Differences between Optical/Circuit Switching and Electrical/Packet Switching

<table>
<thead>
<tr>
<th>Optical/Circuit Switching</th>
<th>Electrical/Packet Switching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two hosts can communicate with each other iff an exclusive circuit is setup between them</td>
<td>If the two hosts are connected to a common switch or a network of switches, packets are switched/forwarded every hop until they reach the destination</td>
</tr>
<tr>
<td>Circuit setup and teardown operations are time consuming. Circuits need to be setup before communication begins</td>
<td>There is no overhead to initiate communication. Resembles plug-and-play</td>
</tr>
<tr>
<td>No processing overhead per packet as it passes through the circuit “pipe”</td>
<td><em>Store-and-forward</em> switching: Memory copying and header processing overhead per packet</td>
</tr>
<tr>
<td>There is no sharing of links between flows; hence, no contention for links</td>
<td>When there is contention for the link, <em>statistical multiplexing</em> determines the next packet forwarded</td>
</tr>
<tr>
<td>Eliminates up to two optical transceivers per pod switch uplink (fiber optic)</td>
<td>Requires two optical transceivers per pod switch uplink (fiber optic) to convert between electrical to optical signal</td>
</tr>
<tr>
<td>Power consumption: 0.24 W/port; Cost (w=4): $125/port [FPR+10]</td>
<td>Power consumption: 12.5 W/port; Cost: $500/port [FPR+10]</td>
</tr>
</tbody>
</table>
Wavelength Division Multiplexing (WDM): It is a technology which multiplexes multiple optical carrier signals on a single optical fiber by using different wavelengths of laser light to carry different signals. This allows for a multiplication in capacity of the optical link [Wik07].

1.3 HELIOS: Hybrid Electrical/Optical Switch Architecture

In [FPR+10], we present HELIOS, switch architecture using a hybrid of electrical and optical switches. In the paper, we describe a methodology to identify the subset of traffic best-suited for circuit switching. This solution combines the benefit of both technologies: packet switching for bursty patterns, and circuit switching for stable patterns of the workload. It also delivers similar performance as a fully-provisioned electrical switch for a variety of workloads at cheaper cost, power, and cabling complexity [FRV09].

1.3.1 Basis for HELIOS Solution: Key Features

HELIOS provides a novel approach to build a core network interconnect for a modular datacenter. It takes advantage of the following three features:

1. **Locality in communication patterns**: In HELIOS, if locality of communication emerges within a single pod, the modular architecture is designed to provide adequate bisection bandwidth. Instead, if there emerges a set of pods that communicate with each other frequently, HELIOS will weigh the benefit of setting up an optical circuit between those pairs of pods.

2. **Traffic stability**: HELIOS leverages the stability in traffic pattern, defined as: probability that a host $A$ communicating with another host $B$, continues to communicate with $B$ after a unit time. For stable flows, HELIOS weighs the benefits of allocating an optical circuit to these flowpaths.

3. **Link aggregation of optical uplinks**: Wavelength Division Multiplexing
(WDM) technology (explained in Section 1.2) allows multiple optical uplinks to be tied into a single high-capacity uplink that connects to the Core Circuit Switch. A single optical port can carry many multiples of 10 Gbps of traffic in each direction. This ability lies at the heart of HELIOS’s suitability for core interconnect in datacenters – it helps bring down the cost per optical port (relative to an electrical port), incremental cost to drive 10 Gbps of data through an optical port, and power consumption.

1.3.2 Suitability of HELIOS for Datacenter Workloads

Many typical applications run in datacenters exhibit data locality and traffic stability. We also believe that full bisection bandwidth, when provided at the packet gradualarity, will never be fully utilized. Our belief is informed by the following reasons:

1. **Other bottlenecks**: Network is not the only bottleneck; CPU, disk I/O, and synchronization are potential bottlenecks too. [BAAZ09] explains that network has an ON-OFF presence: hosts transfer data for a while, then go to other activities like computation or disk I/O, and then resume communication again.

2. **Limited set of communicating partners**: In scientific applications that maybe run in datacenters, the communication patterns can be characterized by hosts talking to only a few other hosts [BBH+05]. This introduces data locality.

3. **Batched delivery**: Other jobs like MapReduce optimize their use of network resources with batched delivery of data. Network comes into play just before the start of map and reduce phases to move the data over to the mapper and reducer nodes. Buffering is provided at the source nodes to enable batched communication [WAK+09]. This enhances traffic stability.

For applications that do not exhibit data locality and traffic stability at all, HELIOS performs badly as such flows, without the requisite circuits, default to the smaller number of Core Packet Switches running with large oversubscription ratios.
1.4 Simulator for HELIOS

In this thesis, we present a TCP flow-based simulator for HELIOS system. This acts as an important tool to analyze/predict the performance and suitability of HELIOS. Since we are limited to our relatively small-sized testbed, simulation is the only viable alternative to gain insight into the behavior of the system at scale.

1.4.1 Need for Simulations

Advantages of using simulator are common to other network simulations as well:

1. **Reduced cost**: Including economic, labor (no wiring required), power, and hardware.

2. **Substantial sharing of code**: The simulator shares the circuit scheduler library with the testbed execution.

3. **Quick test iterations**: New scheduler algorithms could be plugged in and tested quickly.

Specific to HELIOS, simulation plays a critical role before any deployment of the testbed. A number of parameters such as link aggregation factor, and the proportion of packet switches in the core layer have to be tuned by trial and-error in accordance to knowledge of the traffic pattern, as well as the size of the testbed. Tuning to the best value would involve inspection, trial-and-error, and insight into the working of the system (see Chapter 4).

1.4.2 Likely Challenges

We foresee a number of potential challenges for the hybrid switch model.

1. **TCP reordering upon circuit reconfiguration**: When the circuit is configured/reconfigured between pods A and B, the flow that previously used the Core Packet Switch would now be forwarded through the circuit.
This could cause the earlier-sent packets from electrical path to arrive later than those on optical path causing reordering at the destination host.

2. **Centralized Topology Manager**: We employ a central Topology Manager that oversees the routing tables in pod switches, and the circuit reconfiguration in the Core Circuit Switches. [MPF+09] explains the tradeoff between the complexity of a fault-tolerant (but complex) distributed protocol versus fault-prone (but simple) centralized solution. For example, Google’s modern storage [GGL03] and data processing systems [DG04] employ a centralized controller at the scale of tens of thousands of machines. The authors opine that simplicity trumps every other concern in designing datacenter networks.

### 1.5 Organization of the Thesis

This thesis is organized as follows:

- In Chapter 2, we cite research works related to network simulation in general and HELIOS in particular.

- In Chapter 3, we open with a set of research questions and challenges that motivate our research.

- Chapter 4 describes the architecture and design of the simulator. We also share brief details about the implementation.

- In Chapter 5, we evaluate the simulator by varying input parameters like number of pods, link aggregation factor, and proportion of Core Packet Switches for different traffic patterns. We summarize our findings.

- Chapter 6 concludes with a list of possible enhancements for the simulator in the future.
Chapter 2

Related Work

In this chapter, we compare our work to prior research work in optical switching and network simulation.

2.1 Circuit Switching with Optical Interconnects

There has been a deep research interest into how to combine circuit and packet switching to get the benefits of both.

More than a decade back, [NML98] proposed ATM switching in which long flows were assigned circuits when the benefits outweighed the circuit establishment overhead. They discarded the end-to-end connection of ATM, and integrated ATM hardware with IP. The packets are switched by ATM hardware before it is forwarded by IP software. When hardware IP routing became commonplace [CRIB02], this work proved to essentially duplicate the IP functionality at the ATM layer. Our work, which is more specific for datacenter deployments, differs in two ways: (a) HELIOS does not mimic circuits in packet switching – we actually do employ circuit switching (through optical interconnects) in addition to packet switching. (b) We do not require any modification in the TCP/IP protocol stack in the hosts or IP forwarding in the pod switch – when a circuit is available, switching over circuit path overrides packet path, but does not modify IP forwarding (over packet path) itself.

For the High Performance Computing applications, [BBH+05] proposes a
hybrid electrical optical interconnect. The motivation of their work is similar to ours – HPC applications have stable to very-stable inter-processor communication patterns. The paper proposes a two-network interconnect, optical and electrical, and claims to save costs by using fewer optical transceivers. This work is close to HELIOS in principle. However, the distinction lies in how the circuits are established – between end-hosts or pod switches. In their work, hosts will have to make the switching decision between electrical and optical paths. This would involve modification of host operating system. In contrast, HELIOS performs these activities at the pod switch level, obviating the need to modify end hosts. We take advantage of aggregation available in large datacenter topologies by pushing the circuit decision-making to the granularity of pods.

The closest to our work is perhaps the recent paper that treats datacenter as a router so that any host-to-host path is reconfigurable [WAK+09]. It similarly proposes a hybrid electrical and optical switch. The authors consider end hosts as part of the switching framework, which compete for circuits. They buffer data for individual flows at their output queues, so as to merit the assignment of a new circuit. The differences between their system and HELIOS are listed here:

1. **Host modification**: It is instructive that they treat the entire data center as one large virtually output-queued router – implementing their approach requires a “fundamental network re-design with substantial pre-optical queueing at hosts.” [WAK+09]. Our worldview is that modification of end hosts operating system is not an acceptable solution to the datacenter designers – it is too cumbersome to manage, and custom kernel loses the advantage that comes with running a globally-compatible vanilla OS.

2. **Increased latency**: The per-pod in-kernel buffers at the host are expected to queue traffic until a circuit is established. This approach of queuing at hosts can introduce significant latency for traffic patterns that have smaller stability values. These queues drain slowly over the oversubscribed packet switches, or quickly at line rate once an optical circuit has been established.

3. **Computation of traffic matrix**: [WAK+09] estimate traffic matrix by
observing the queue lengths across the hosts. We instead query the flow counters in the pod switches to estimate the traffic matrix.

4. **No link aggregation**: The authors do not take advantage of link aggregation offered by Wavelength Division Multiplexing (WDM) of optical ports at the pod switch level, in contrast to our solution.

Another effort that advocates a hybrid datacenter architecture is [KPB09]. It proposes addition of on-demand wireless *flyways* to cater to the hotspots in addition to the oversubscribed base network. The difference with HELIOS is that their base network is electrical, while ours is optical. On the parameters of power and price per port, optical switches do better than electrical switches. Wireless takes the adjunct network status in their work, and handles the “extra” traffic from the electrical network. The adjunct network in HELIOS is electrical which handles the traffic for those pairs of pods that do not have circuits. Comparing electrical and wireless, wireless is more expensive, and provides lesser bandwidth. However, [KPB09] certainly can be more easily integrated with the existing datacenter facilities with electrical switches than HELIOS.

### 2.2 Network Simulation

Network simulation has long been a topic of immense interest to the networking research community. Part of the reason is that deploying real hardware to scale is prohibitively expensive and time-consuming, which has led us to explore alternatives like simulators.

The popular discrete event network simulators are *ns-2* [IH09] and *OM-NeT++* [VH08]. Both simulate packet-switching networks for TCP/IP, and involve processing per packet. In contrast to these simulators, our simulator is based on [AFRR+10] which simulates TCP flows. While we lose some accuracy for being unable to simulate TCP characteristics like timeouts, we gain simplicity and speed of simulation.

The goal of [WBSS01] is the same as ours in this thesis – to simulate a Optical WDM network. *OWns* is implemented on top of *ns-2*. Since *ns-2* provides
a packet-switching framework, they implement an abstraction of logical topology for circuit-based switching. They simulate at the level of packets, which also makes their simulator slow. Our simulator works at the flow level.
Chapter 3

Research Motivation for the Simulator

The primary goal of the simulator is to model/predict the performance and behavior of Helios system across a wide range of demand characteristics and scale, while remaining accurate with respect to performance properties. The purpose of this thesis is to ensure that the use of this simulator is both practical and insightful for network planners and operators.

Of particular significance is modeling Helios behavior at scale accurately; this tool could form the basis for important decisions for/against deployment in the absence necessary hardware. Hence, we seek answers to the question: “Can we reliably run Helios at scale maintaining similar performance levels, while exploiting the network resources just as effectively as with a smaller testbed?” In particular, the following related questions merit our research.

3.1 Research Questions and Hypotheses

In this thesis, we use the word “performance” to refer to the average bisection bandwidth measured as the sum of bisection bandwidths of individual uplinks to the core switches. Following is a list of research challenges that we aim to tackle in this thesis.
3.1.1 Pod Switch Scalability

How does the performance scale with number of pods $N$? Is it linear? That is our hypothesis, since the number of uplinks (hence the capacity) will increase with each addition of a pod switch by a constant amount.

3.1.2 Link Aggregation

It is important that we use the optimal link aggregation factor $w$, as higher $w$ will save us of the precious Core Circuit Switch ports as well as the associated fiber costs. Provision for link aggregation could rank as the single most-important benefit of using circuit switches over packet switches.

How will $w$ affect the performance (for a given $N$)? We hypothesize that performance will probably not improve with higher $w$ for regular traffic patterns, and will probably not degrade for stride traffic pattern. The benefit of a larger $w$ could be in cost and power savings – which are outside the scope of this thesis.

3.1.3 Proportion of Core Packet Switches

Our inquiry is into the possible effects on performance of increasing the number of Core Packet Switches $p$ (for given $N$ and $w$). Here, we investigate two cases:

- **With Oversubscription**: Keeping the total number of pod switch uplinks fixed – by increasing oversubscription on optical paths (i.e. uplinks to Core Circuit Switches).

- **Without Oversubscription**: Keeping the number of circuit switches fixed – so there is no oversubscription on optical paths (i.e. uplinks to Core Circuit Switches).

Our hypothesis is that as the number of packet switch paths increase, performance improves, to say nothing about the increased cost for more packet switches.
3.1.4 Bisection Bandwidth Utilization Rate

An Ideal Crossbar Switch (ICS) is a non-blocking switch in which every host connected to it has a dedicated path to every other host. As a consequence, no two flows have to compete with each other to share bandwidth on an uplink. The bisection bandwidth of an ICS with $N \times H$ ports also represents the maximum available bisection bandwidth for a Helios topology of $N$ pods with $H$ hosts each.

Next, we consider how pod scalability and link aggregation affect utilization? Utilization can be measured and compared for traffic patterns that saturate the pod switch capacities for the duration of the experiment. We define the Utilization Rate for an experiment as:

$$\text{Utilization Rate} = \frac{\text{Average Bisection Bandwidth}}{\text{Maximum Achievable Bisection Bandwidth}} \quad (3.1)$$

3.1.5 Traffic Patterns

Lastly, what is the dependence of performance on the underlying traffic pattern? The common feature among the patterns we consider in Subsection 5.1.1 is that they try to saturate the pod switch capacities for the entire duration of the experiment.

3.1.6 Running Times of the Simulator

How does the running time of the simulator vary with the increase in number of pods? Does it scale linearly? Prior knowledge of running times of the simulator might prove useful for employing this tool for large scale simulations. While the simulator running time does not convey any information about the actual testbed run directly – provides useful information about how the simulator could be employed in real world settings. For example, hypothetically if the simulation for a minute’s worth of testbed execution could be performed in a few seconds, the simulator tool could see wide adoption before any decision-making. On the other hand, if the simulation were to take a few hours or days, we could employ the tool only for high-level initial studies.
Chapter 4

System Architecture, Design and Implementation

In this chapter, we present the architecture of the simulator and describe our design. Later, we explain the implementation decisions we took. This chapter is organized as follows:

1. **Topology and Experimental Testbed**: We describe the topology of a scalable HELIOS testbed, and introduce its dimensions. We also show a block diagram of the testbed and hardware installed for our experimental setup.

2. **Design: The Control Loop**: We briefly describe the various software components in the design. Further, we present an overview of the steps involved in the Control Loop of HELIOS system.

3. **Working of the Simulator**: We describe the principles on which the simulator is based, as well as the simulation procedure at a high level. Also, we elaborate on the methodology followed to simulate each step in the control loop.

4. **Control Flow and Implementation**: We present the pseudo code of the control loop and discuss the implementation details of the simulator.
4.1 Topology and Experimental Testbed

Figure 4.1: HELIOS topology consisting of an array of electrical and optical core switches that interconnect a set of pods

Figure 4.1 presents the architecture of HELIOS system. A series of $N$ pod switches (referred to as ‘pods’) form the *Edge* and *Aggregation* switch layers (Layer-1 and Layer-2 respectively) in a typical three-tier data center network architecture [AFLV08]. Each pod has $H$ hosts/servers connected through copper downlinks. The uplinks from the pods go to the Core switches.

The *Core* switch layer (Layer-3) consists of a mixture of electrical and optical switches. Determining the optimal proportion is a research challenge as enumerated in Subsection 3.1.3. A pod is connected to all Core Packet Switches through an uplink each. The remaining uplinks are grouped together into sets of
$w$ uplinks each, to form Superlinks. A superlink connects a pod to a Core Circuit Switch. $w$ is called the link aggregation factor; the link aggregation of optical links is achieved using Wavelength Division Multiplexing (WDM) technique [FPR$^+10$] through optical multiplexers situated at the egress ports of a pod switch.

Figure 4.2 depicts the architecture of our small testbed installed for the purposes of current research [FPR$^+10$]. Our testbed consists three 24-port Fulcrum Monaco 10 GigE packet switches, which are partitioned into two virtual pods each. We also employ a 64-port Glimmerglass optical switch, which is partitioned into three virtual Core Circuit Switches. Connected to the pod switches are hosts with 10 GigE interfaces.

## 4.2 Design: The Control Loop

### 4.2.1 Software Components

HELIOS system consists of three major software modules [FPR$^+10$]:

1. **Topology Manager** ($TMgr$): This process runs on a dedicated server as a daemon. Its chief responsibility is to reconfigure the network topology in response to dynamically changing inter-pod communication patterns. It continuously estimates the true inter-pod traffic demand; modifies the topology if it deems beneficial. In order to measure the current state of the network and estimate likely demands, it executes the control loop as detailed in Subsection 4.2.3.

2. **Pod Switch Manager**: This process on the pod switch is responsible for initializing the pod switch hardware, managing the flow table and interfacing with $TMgr$.

3. **Circuit Switch Manager**: This accepts the circuit reconfiguration requests from $TMgr$, and issues the corresponding commands to Glimmerglass optical switch in a non-blocking fashion.
**Figure 4.2:** Experimental setup of 6-pod/5-host HELIOS testbed, along with description of the switches/hardware
4.2.2 Load Generator

The experimental results could be insightful only when we have the ability to specify various patterns of traffic as input. For testbed execution, we employ our Load Generator tool, which essentially takes as input a list of flows between pairs of hosts with start and end time. During each cycle, this configuration file is queried for flows which start (or end) at that instant; a new flow is initiated (or an existing flow is terminated) at the source host. The placement of flow in the data plane is handled by the entry installed in the flow tables of pod switches by the TMgr.

4.2.3 Control Loop Steps

The control loop refers to one iteration/cycle of the system during which each software component performs its functions, and coordinates with other components. Figure 4.3 shows the interaction of various software components. The control loop consists of the following steps [FPR+10]:

1. **Measurement of traffic matrix**: Pod Switch Managers communicate to the TMgr the traffic seen by their hosts. TMgr aggregates the traffic matrices from pods into a global traffic matrix with inter-pod traffic. This happens once every ‘sampling period’ (referred to as ‘scheduling rate’ in Subsection 4.2.5)

2. **Demand estimation**: The traffic matrix obtained from the above step could be easily skewed by the bottlenecks in the network. When the network is bottlenecked, this matrix reflects the artifacts of the topology rather than the underlying host-to-host demand. Hence, we estimate the inter-host demand by computing the min-max fair share [AFRR+10] that TCP flows would achieve in a steady state assuming a non-blocking switch. The flow traffic rates from earlier step is modified to reflect the fair share of egress links received.

3. **Computation of new topology**: This step is also called Circuit Scheduling.
After the $TMgr$ estimates the traffic demand matrix for a new cycle, it has to decide how to allocate the circuits to support that demand. **Given the traffic demand matrix $TM$ and connectivity matrices, we aim to maximize the amount of traffic through Core Circuit Switches.** The steps for constructing the bipartite graph $G$ with sets $A$ and $B$ are as follows [FPR+10]:

- Map each pod $i$ in the topology to two vertices $i_a$ and $i_b$ in $G$, one in each set.
- For each $i$ and $j$ where $TM_{ij} > 0$ (inter-pod demand), add an edge from $i_a$ to $j_b$ with a weight of $\min TM_{ij}, C_{pc}$ where $C_{pc}$ is the capacity of Pod Switch to Core Circuit Switch link.
We now seek a solution to maximize traffic through the circuits/optical network. This problem corresponds to maximum weight matching problem for graph $G$. The edges of the resultant subgraph represents the allocated circuits between pods. We employ Edmonds’s algorithm for the same explained in Subsection 4.2.4, which is known to have a time complexity of $O(|V||E| \log |V|)$.

4. **Pre-notification of circuits DOWN**: This step alerts the pod switches that some circuits are about to get reconfigured, so they can gracefully switch paths for their existing flows. Without this alert, there is risk of existing traffic being blackholed through the circuits being reconfigured. When TCP loses entire window worth of packets, it heads to Slow Start phase dropping the throughput to zero.

5. **Topology change**: The circuits get reconfigured during this time. While the reconfiguration itself is quick, this is typically accompanied by associated sanity checks on circuit switch ports which takes longer.

6. **Notification of circuits UP**: Once the circuits are UP, $TMgr$ will notify the pods, so they can modify their flow tables with newly available routes. Some of the packets may be delivered out-of-order at the receiver during rerouting, but this is not significant enough to affect average bisection bandwidth.

### 4.2.4 Edmonds’s Algorithm

[Edm65] In this subsection, we describe Edmonds’s algorithm [Edm65] that provides a solution for maximum weight matching problem. Given a bipartite graph $G$ with sets $A$ and $B$, we seek a matching subgraph $M$ such that each vertex of $G$ is incident with at most one edge in $M$, and $|M|$ is maximized. The algorithm is described in [Wik]:

**Notes on Algorithm 4.1**: Given matching $M$ of $G$, a vertex $v$ is “exposed,” if no edge of $M$ is incident with $v$. A path in $G$ is an “alternating path,” if its edges are alternately not in $M$ and in $M$. An augmenting path $P$ is an
Algorithm 4.1 Calculate $y = x^n$

**Input:** Graph $G$, initial matching $M$ on $G$

**Output:** Maximum matching $M^*$ on $G$

function find_maximum_matching($G, M$): $M^*$

$P ←$ find_augmenting_path($G, M$)

if $P \neq \emptyset$ then

    return find_maximum_matching($G$, augment $M$ along $P$)

else

    return $M$

end if

end function

alternating path that starts and ends at two distinct exposed vertices. A matching augmentation along an augmenting path $P$ is the operation of replacing $M$ with a new matching $M_1 = M \text{ExOR} P$. [Wik].

4.2.5 Observations: Phase-wise Utilization of Cycle Time

From our observation for 4-pod/5-host experiments, each cycle, or control loop, takes about 400 ms to complete on an average (with the exception of accompanied by circuit reconfiguration). We use this as the scheduling rate in our simulator – time after which the next call to circuit scheduler occurs. In this subsection, investigate the times taken by individual steps in the control loop. Figure 4.4 shows the dissection. We can distinguish four phases accordingly:

1. **Querying statistics (Phase-1):** For the first 100 ms of a new cycle, we query the rate statistics for every flow. At the end of this interval, we compute the average flowrate.

2. **Computation (Phase-2):** This consists of two sub-phases consuming 35 ms on average:

   - Demand estimation (20 ms on average)
Figure 4.4: Time taken by various phases during a cycle in HELIOS

- Edmonds's algorithm to find maximum weight matching (15 ms on average)

3. Circuit reconfiguration (Phase-3): This time is measured to be 165 ms on average (of which only about 25 ms is spent on actual reconfiguration of ports, and the rest is spent on sanity checking)

4. Wait time (Phase-4): 100 ms if no circuit reconfiguration occurs in that cycle. If it does, we wait for 1000 ms as a way to allow time for the flows to stabilize on the new paths.
4.3 Working of the Simulator

The simulation works at the time-granularity of a tick. A ‘tick’ refers to the smallest time duration that can be reliably measured and accounted for by the system. In our experiments, we set to 0.1 ms. Part of our challenge is to accurately estimate the number of ticks required by various events/phases during a (hypothetical) experiment.

4.3.1 Principles of HELIOS Network Simulation

The simulator works on the following principles of network simulation:

1. **Apportioning ticks for phases/flows**: From our observations of testbed runs (see Subsection 4.2.5), we extrapolate the number of ticks each phase clocks. We also express the duration of a flow in terms of ticks. The flow is said to be “active” during those ticks.

2. **Simulation of TCP behavior**: We simulate the Reno/NewReno variant of TCP for a flow assuming sawtooth behavior of Additive Increase Multiplicative Decrease (AIMD) after the initial Slow Start phase. Note that we do not model the latest TCP variants like BIC/CUBIC which might result in some loss of accuracy.

3. **Flow level simulation**: For the sake of simplicity, we simulate TCP behavior at the level of flows (and not at the level of packets). We acknowledge that packet-level simulation of TCP behavior results in better accuracy as TCP is a packet-based transport protocol. But simulating packets would add accounting overhead and complexity, since we would need to keep track of packet-flow mapping. Also, simulating packets at 10 Gbps would take prohibitively long time [AFRR^10]. On the other hand, flow-level simulation, while simple, is less accurate since we cannot model TCP characteristics like timeouts, packet loss in switch buffers etc.

4. **Estimation/aggregation of flow-level statistics**: During the life of a TCP flow, this simulation model gives us the flowrate, taking link capacities
into consideration. These flowrates from individual flows are aggregated periodically to compute the inter-pod traffic matrix.

4.3.2 Simulation Methodology

In this subsection, we explain how some of the key artifacts of HELIOS are simulated.

Traffic Patterns

We employ techniques very similar to that of Load Generator (see Subsection 4.2.2) in the simulator, except that there is no generation of physical traffic. Instead, we “create” a new flow on the tick it is due to start; we keep track of the number of ticks the flow has been active, and “terminate” it when it approaches its end tick. Creating a new flow involves placing it on a particular inter-pod path by proportionally reducing the flowrates of every other flow sharing that path, while terminating an existing flow involves increasing the flowrates of other flows.

Link Aggregation

As discussed earlier, link aggregation is realized by tying $w$ links with using a WDM multiplexer at the egress/uplink ports of pod switches. To implement this in the simulator, we have introduced the concept of *emulated network*. When $w > 1$, we emulate a network with $w = 1$ with the constraint that groups of circuits can only be allotted together. This naturally leads us to use the $w > 1$ parameters for the Edmonds algorithm, since we want the aggregated circuits to be allocated together.

Flow-level TCP Simulation

We simulate TCP Reno/NewReno at the flow-level with only basic features. Currently, the TCP simulator implements the three phases of TCP: slowstart, additive increase and multiplicative decrease. The *target flowrate* is updated each tick, and represents the highest flowrate the flow hopes to achieve given its current
TCP phase. The available capacity on the network is calculated as the lowest of unused capacities of all the links on the route of this flow.

1. **Slowstart Phase**: A flow, on creation, is set to slowstart phase with an initial flowrate of 1500 bytes/sec. The flowrate increases as a power of 2 till the available bandwidth.

2. **Multiplicative Decrease (MD) Phase**: When the target flowrate exceeds the available capacity, MD phase kicks in by dropping the flowrate by half.

3. **Additive Increase (AI) Phase**: When the available capacity on the flow route is sufficient to satisfy the target flowrate, the flow gets to its target flowrate. This portion of the plot is linearly increasing.

Other TCP characteristics like timeouts, retransmission delays, backoff can be simulated on a packet-level simulator, unless we have reliable statistical distribution of occurrence of such events in TCP flows.

### 4.4 Control Flow and Implementation

We have implemented the simulator and its components such as Hedera Demand Estimator and Edmonds’s algorithm in C language. In this section, we present Listing 4.4 with the pseudo code of the control loop of the simulator, and briefly elaborate of the steps involved in the control flow. The meaning of the identifier names used in this listing is self-explanatory.

Listing 4.1: Control loop of the simulator

```plaintext
for each tick:
    updateFlowrates(tick) // Simulate TCP
    createOrTerminateFlowsAsSpecified(tick)

if timeToSchedule(tick):
    // Query flow stats after the first 100ms of cycle
    calculateAvgFlowrates()
```
4.4.1 Control Flow of the Simulator

With reference to Listing 4.4, we describe the control flow and elaborate each step inside the loop that occurs once every simulation tick.

Update flowrates (line 2)

This step is responsible for updating the rates of all TCP flows every tick. The updating of rates is handled sequentially. For each flow, the following sub-steps are involved:

1. Compute how much data this flow has sent since last tick using the previously computed flowrate. Update the size of data of the flow left to be sent after
this tick’s worth of data is sent.

2. If this tick marks the beginning of new cycle (start of Query Statistics Phase-1), note the bytecount of the flow. This is used to calculate the average rate during that phase.

3. The new flowrates are calculated according to our discussion in Subsection 4.3.2. The rate increment/decrement depends upon which TCP phase the flow is currently at: slowstart, or additive increase, or multiplicative decrease.

**Create/terminate flows as specified (line 3)**

1. Check whether any flows need to be eliminated. A flow has to be removed if it has no bytes left to send, or the current tick is past the stop time specified in the configuration file. Also, free the bandwidth consumed by this outgoing flow along route along its path from source pod to destination pod.

2. Look for new flows that start

**On the scheduling tick (line 5)**

This tick corresponds to the point $B$ in Figure 4.4.

1. **Calculate average flowrates**: For the flowrate, we consider the average flow rate over the first 100 ms called the Query Statistics phase in the figure. At this step in the control flow, we calculate the average flowrate for the flow to be used for the rest of the current cycle.

2. **Schedule circuits**: This is the call to the circuit scheduler. It consists of two subphases: Demand Estimation and Edmonds’s algorithm as explained in Subsection 4.2.3. It first estimates the inherent demand of flows, and creates a new traffic “demand” matrix based on this knowledge of demands. It passes this traffic demand matrix along with the connectivity matrix of the circuit switches to the Edmonds’s algorithm. The resulting subgraph is the optimal set of circuits for the current demand patterns.
On the tick to pre-notify pods (line 12)

We encounter this tick at the point $C$ in Figure 4.4. This is a precaution to the pod switches of the impending changes in the circuit configuration. If the circuit connection map has changed, we do the following:

1. **Re-route flows after processing**: Since certain circuits are in line to be reconfigured, the affected flows have to be rerouted. This module checks which flows have to be rerouted due to the circuits that have gone down. New routes are found for such flows. The routing table is suitably modified to reflect the new routes.

2. **Reassign paths**: This method is called when routing table has changed. It oversees the utilization on the links. Once a flow is rerouted, its allocation is subtracted from the links on the old route, and add to the ones on the new route.

3. **Increase wait time before next schedule**: This was a small tweak introduced to prevent flapping of optical ports. Usually it takes some time (order of hundreds of milliseconds) for the traffic on the circuit switch to regain stability. If the next schedule is held within 100 ms, it might introduce noise in the scheduling behavior and results.

On the tick to notify pod switches (line 21)

This tick occurs at point $D$ in Figure 4.4. Here we notify the pod switches that the new circuits are now fully functional.

1. **Bring circuits back up**: This module recalculates routes for flows since the flows can now start using the circuits that have just come up. It also involves modification of the flow/routing table.

2. **Reassign paths**: The functionality of this module is same as explained in the above Subsubsection 4.4.1.
Chapter 5

Evaluation of HELIOS System Using the Simulator

In this chapter, we evaluate the scalability of the HELIOS system using the simulator described in the previous section. We also endeavor to find answers and validate hypotheses to the research questions raised in Section 3.1

5.1 Methodology

In our approach to evaluate the system, we follow the classic validate-predict model of evaluation – (a) validation of the correct behavior of simulator with smaller scale inputs, and (b) prediction of behavior at larger scale. Elaborating further, the two steps are:

1. Validation of simulator for HELIOS testbed behavior: For the executions of HELIOS which we have observed, we verify whether the simulator can reproduce similar numbers under similar input configurations. Here, we cite the results from [FPR+10]. We use the difference between the average bisection bandwidths for the experiment duration as a metric to measure the simulator’s “distance” from the testbed’s reality.

2. Prediction of HELIOS behavior at scale: Once we validate that the simulator’s performance figure is within acceptable margins of the testbed’s,
we use the simulator to predict how the system might behave with respect to varying parameters like number of pods, the link aggregation factor, proportion of Core Packet Switches, and the traffic patterns.

5.1.1 Traffic Patterns

In this chapter, we consider four saturating traffic patterns for our evaluation. By “saturating” we mean that every host constantly tries to send and receive data at its line rate, provided the system were to allow for it. We describe the patterns here:

1. **Stride**: This is a periodic pattern of traffic in which all hosts in a source pod send to, and only to, corresponding hosts in destination pod for a specific duration called stride interval [AFRR+10]. In the next interval, the hosts in the source pod pick a different destination pod to communicate with. There shall always be one-to-one relationship between source and destination i.e., during a stride interval:
   - No two source pods shall pick the same destination pod
   - No source pod shall communicate with more than one destination pod
   - A host shall only send to a single other host

   In our implementation, pod $i$ sends to pod $(i + 1) \mod N$.

2. **Random**: This is a periodic pattern of traffic in which a source host talks to a random destination host outside its own pod for a specific interval [AFRR+10]. And the process of picking the destination host repeats every interval until the experiment is completed.

3. **Crawl Stride**: This is a variation of Stride pattern in that the shift in communication is gradual – one host at a time. Let source pod $A$ and destination pod $B$ be communicating in this interval. At the beginning of a next stride interval, one host from $A$ (say $A_1$), will begin communicating with its corresponding host from new destination pod $C$ (say $C_1$), while the rest
of A’s hosts continue talking to their counterparts in B. A1’s old counterpart B1 will start communication with a new destination host D1. In the interval after this, A2 will talk to C2, while B2 goes to D2. This process repeats every interval until the experiment duration. Formalizing the rules, during any stride interval:

- A source pod shall have not more than two destination pods
- A destination pod shall not receive incoming traffic from more than two source pods
- A host shall only send to a single other host

5.1.2 Abbreviations, Symbols, and Terms

Provided below is a glossary with some of the abbreviations, symbols, and terms frequently encountered in this section:

- CCS: Core Circuit Switch
- CPS: Core Packet Switch
- ICS: Ideal Crossbar Switch i.e. a switch with has all-to-all links between its ports
- BBW: Bisection Bandwidth
- N: Number of pod switches
- H: Number of hosts per pod switch
- w: Number of uplinks per pod switch
- w: Link aggregation factor, i.e. the number of uplinks from a pod switch tied together using Wavelength Division Multiplexing (WDM) technique. These uplinks (also called a superlink), therefore, utilize only one port of the CCS (albeit with a capacity w times that of an individual uplink)
- c: Number of CCS’es
• $p$: Number of CPS’es

• For all the configurations of the simulator, the following relation between $u$, $c$, and $p$ holds:

$$u = (w \ast c) + p \quad (5.1)$$

• *Oversubscription*: The HELIOS system is said to be oversubscribed w.r.t. optical paths if the total capacity of the CCS-facing uplinks on the pod switches is less than the total capacity of the host-facing downlinks. Mathematically, the condition for oversubscription of a pod is: $H > w \ast c$

• *Stability*: It is defined as the average time after which the source pod shifts some or all of its originating traffic to a new destination pod. As the name indicates, it measures the duration during which the inter-pod traffic remains stable.

### 5.2 Validation: Comparison to the Ground Truth

Our task in this section is to verify the correctness of the simulator – to show that the simulator behaves in a manner that *approximately* corresponds to the testbed’s behavior. At the very outset, we concede that the simulator does not ‘mimic’ the testbed *exactly*; rather, our goal in this thesis has been to keep the simulator design simple enough, so it can serve as a good approximation tool for the testbed performance.

We go about this task of verification by considering the performance of the testbed observed in [FPR+10]. We then see how the simulator’s predictions compare against them. The behavior of the simulator is deemed correct if its predictions fall within an acceptable margin of error (See Subsection 5.2.3). We run the following two experiments on both testbed and simulator:

1. Stride traffic pattern (in a 4-pod/5-host configuration)

2. Variation of stability (for traffic patterns such as Stride, Random, and Crawl Stride in a 6-pod/5-host configuration)
5.2.1 Experiment 1: Stride traffic pattern

The first experiment is to evaluate by inspection the point-to-point proximity of the bisection bandwidth for testbed run and the simulation (see Figure 5.1). For this, we use the testbed run from [FPR+10] for 4-pod/5-host Stride pattern. The experiment duration is 60 s, with the stride interval set to 10 s.

![Figure 5.1: Simulator vs. Testbed Comparison: Bisection bandwidth in a 4-pod/5-host HELIOS network](image)

We also observe that both simulator and testbed arrive at similar decision about circuit reconfiguration events at the same instants. While one would expect this as the simulator and HELIOS prototype share significant amounts of code, we can explain the similar bisection bandwidths observed by the similar scheduling decisions made. Note that a Circuit Switch Program event occurs at the beginning
of every stride interval, except at \( t = 10 \) s, as labeled \( A \ldots F \) in the figure. In Figure 5.2, we show the post-reconfiguration Circuit Connection Maps corresponding to these points, with the most recently reconfigured circuits in bold. It is instructive to see that the Simulator Connection Map (\( SimCM \)), despite being initialized differently, quickly converges to the Testbed Connection Map (\( TbCM \)) at Cycle 0 of the testbed. All subsequent decisions of the simulator agree with the testbed scheduler.

Table 5.2.1 shows that simulator’s prediction of average BBW is within 4.6% of testbed’s performance.

Table 5.2.1: Statistics for 4-pod/5-host Stride Pattern Experiment

<table>
<thead>
<tr>
<th>System configuration</th>
<th>( N = 4, H = 5, w = 1, p = 1, u = 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. BBW from Testbed Run</td>
<td>116.83 Gbps</td>
</tr>
<tr>
<td>Avg. BBW from Simulation</td>
<td>111.46 Gbps</td>
</tr>
<tr>
<td>Inaccuracy of the Simulator</td>
<td>-4.6% over Testbed</td>
</tr>
<tr>
<td>Maximum achievable BBW</td>
<td>200 Gbps (Ideal Crossbar, 10 Gbps links)</td>
</tr>
<tr>
<td>Utilization rate (over Ideal Crossbar)</td>
<td>58.41% (Testbed); 55.73% (Simulator)</td>
</tr>
</tbody>
</table>

5.2.2 Experiment 2: Variation of stability

We now compare the average BBW of testbed runs and simulations for a variety of stability values and traffic patterns. We seek to understand why simulator might work well with certain stability values and not others.

From Table 5.2.2, we note that the simulator’s accuracy is particularly inaccurate for lower stability values. We observe that lower the stability, higher the volatility in TCP behavior. Since the simulator does not account for TCP timeouts and exponential backoff (see Subsection 5.2.3) which affect low-stability flows heavily, the simulator generally overestimates the BBW i.e. for stability values of 3 s and 5 s. This is due to the delay introduced at the physical layer.

\(^1\)Maximum achievable BBW for 6-pod/5-host HELIOS (Ideal Crossbar @ 10 Gbps per link) = 300 Gbps
Initialization: \( SimCM_{\text{tick}=0} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \\ 1 & 2 & 3 & 0 \end{bmatrix} \)

Point A: \( SimCM_{\text{tick}=2546} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \end{bmatrix} = TBCM_{\text{cycle}=0} \)

Point B: \( SimCM_{\text{tick}=205096} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \end{bmatrix} = TBCM_{\text{cycle}=88} \)

Point C: \( SimCM_{\text{tick}=304946} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 \end{bmatrix} = TBCM_{\text{cycle}=126} \)

Point D: \( SimCM_{\text{tick}=404796} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 \end{bmatrix} = TBCM_{\text{cycle}=164} \)

Point E: \( SimCM_{\text{tick}=504646} = \begin{bmatrix} 2 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 \end{bmatrix} = TBCM_{\text{cycle}=202} \)

Point F: \( SimCM_{\text{tick}=604496} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 0 \end{bmatrix} = TBCM_{\text{cycle}=241} \)

**Figure 5.2**: Matrices showing post-reconfiguration Circuit Connection Maps at points A…F in Figure 5.1. Also shown are corresponding simulator tick and testbed cycle numbers
Table 5.2: Simulator vs. Testbed Comparison: Average bisection bandwidth as a function of the host-level stability of traffic

<table>
<thead>
<tr>
<th>Stability (s)</th>
<th>Testbed Run (Gbps)</th>
<th>Simulation (Gbps)</th>
<th>Accuracy (%age difference)</th>
<th>Simulator Utilization Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic pattern: Stride</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>83.26</td>
<td>154.31</td>
<td>85.34%</td>
<td>51.44%</td>
</tr>
<tr>
<td>5</td>
<td>115.01</td>
<td>162.21</td>
<td>41.04%</td>
<td>54.07%</td>
</tr>
<tr>
<td>10</td>
<td>139.15</td>
<td>167.73</td>
<td>20.54%</td>
<td>55.91%</td>
</tr>
<tr>
<td>15</td>
<td>152.99</td>
<td>169.76</td>
<td>10.96%</td>
<td>56.59%</td>
</tr>
<tr>
<td>20</td>
<td>150.78</td>
<td>170.33</td>
<td>12.97%</td>
<td>56.78%</td>
</tr>
<tr>
<td>Traffic pattern: Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>99.27</td>
<td>126.89</td>
<td>27.82%</td>
<td>42.30%</td>
</tr>
<tr>
<td>5</td>
<td>117.11</td>
<td>130.41</td>
<td>11.36%</td>
<td>43.47%</td>
</tr>
<tr>
<td>10</td>
<td>132.11</td>
<td>135.06</td>
<td>2.23%</td>
<td>45.02%</td>
</tr>
<tr>
<td>15</td>
<td>142.87</td>
<td>135.20</td>
<td>-5.37%</td>
<td>45.07%</td>
</tr>
<tr>
<td>20</td>
<td>139.23</td>
<td>138.44</td>
<td>-0.57%</td>
<td>46.14%</td>
</tr>
<tr>
<td>Traffic pattern: Crawl Stride</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>144.82</td>
<td>132.87</td>
<td>-8.25%</td>
<td>44.29%</td>
</tr>
<tr>
<td>5</td>
<td>153.66</td>
<td>136.92</td>
<td>-10.90%</td>
<td>45.64%</td>
</tr>
<tr>
<td>10</td>
<td>156.77</td>
<td>138.18</td>
<td>-11.86%</td>
<td>46.06%</td>
</tr>
<tr>
<td>15</td>
<td>168.01</td>
<td>136.95</td>
<td>-18.49%</td>
<td>45.65%</td>
</tr>
<tr>
<td>20</td>
<td>179.16</td>
<td>145.52</td>
<td>-18.78%</td>
<td>48.51%</td>
</tr>
</tbody>
</table>
of pod switch ports as explained in point-2. As the stability increases, we see a marked improvement in accuracy.

5.2.3 Explaining Simulator Inaccuracy

There is a variety of reasons for inaccuracy to creep into our simulation. As explained in Section 5.2, we have favored a simpler design for simulator at the cost of some accuracy. We list some of the reasons:

1. **TCP timeouts**: Simulation is at the flow-level, and not packet-level. Hence, we believe typical TCP characteristics like timeouts and retransmission cannot be simulated at the level of flows. The absence of this can be noted in the Figure 5.1 as well – the simulator’s curve is smoother than the testbed’s during a stride interval.

2. **PHY layer delay of pods**: Our Fulcrum Monaco pod switches are not designed for situations where there is frequent flipping of port state. Each time a port comes up, the switch control does an elaborate sanity check at the PHY layer of the port to diagnose the cause for the port going down, as well as to ensure better uptime in the future. This introduces massive delays in the order of seconds. Unfortunately, this does not work well with our use case since our system requires the ports to frequently flip. Simulation does not take into account this delay, which is why it overestimates the performance for lower stride values like 3s and 5s.

3. **Exponential backoff**: On the testbed, we noticed that some flows went into exponential backoff after several timeouts. They take longer to recover. This behavior is not accounted for in the simulator as it does not implement timeouts currently. This explains the slower ramping up of flows on testbed after a stride shift in the figure.

4. **Glimmerglass circuit reconfiguration time**: We observed that a call for circuit reconfiguration to Glimmerglass switch takes about 165 ms to return. The circuit remain down for about 25 ms within this 165 ms interval. For the
simulator, we have chosen 165 ms as the approximate circuit reconfiguration time. But on some occasions, we observed non-uniform reconfiguration times higher than this figure. On the hindsight, it turns out that our 165 ms could be optimistic. Currently, we are working to reduce this delay to about 25 ms.

5. **Non-uniform recovery of flows**: After a throughput drop, some flows recover slower than others. Recovery times cannot be predicted, as it is dependent on other flows in the system. This non-uniformity may contribute to skew simulation results.

6. **Host Synchronization**: While we assume hosts are synchronized in the simulator, this is rarely true in the testbed. Though the load generator tries to trigger all flows at the same time (in case of Stride pattern), clock skews contribute to unpredictable delays. Also, unlike the simulator, the HELIOS scheduler running on the Topology Manager is not synchronized with the hosts.

### 5.3 Simulation Results: Employing the Simulator for Prediction

#### 5.3.1 Pod Switch Scalability

We initiate our description of simulation results by varying the number of pods. How the HELIOS system scales with pods would likely determine the success of the effort, as HELIOS was designed to cater to the needs of large-scale datacenters.

In the following experiments, we vary \( N \), and fix \( H = 512, w = 4, p = 1, u = 513 \). We employ these configurations for different traffic patterns. Our expectation is that, for each traffic pattern, the average BBW should nearly double. It turns out to be the case as seen in the graphs shown in Figures 5.3, 5.4, 5.5. This validates our hypothesis in Subsection 3.1.1.
Figure 5.3: Effect of variation of $N$ on Bisection Bandwidth for Stride traffic pattern

5.3.2 Link Aggregation

We now turn our attention to the next parameter $w$, the link aggregation factor. In the following experiments, we vary $w$, fixing the rest: $N = 32$, $H = 512$, $p = 1$, $u = 513$. We employ these configurations for different traffic patterns.

Stride

As we hypothesized in Subsection 3.1.2, the Stride pattern is not affected by $w$ (Figure 5.6). This is because Stride pattern requires a pod (source) to be connected to only one another pod (destination) at any time, and all its hosts are guaranteed to talk to the hosts in the destination pod. Hence, irrespective of $w$, 

Figure 5.4: Effect of variation of $N$ on Bisection Bandwidth for Random traffic pattern

the same circuit connections are made.

Random

This analysis is most interesting since we begin to see clearly how the choice of a particular $w$ plays out in performance. As $w$ increases, the performance improves initially as seen in Figure 5.7. This is because, at lower values of $w$, the Edmonds's algorithm needs to make more decisions about the placement of circuit connections – and hence the higher chances of erroneous decisions. At some $w$ ($w = 16$ in our case), the benefit of this peaks, and higher $w$ ($w = 32$ in the figure) would deteriorate performance, as $w$ reduces the scope for scheduling decisions, and make the scheduling too restrictive.
Figure 5.5: Effect of variation of $N$ on Bisection Bandwidth for Crawl Stride traffic pattern

Crawl Stride

In the Figure 5.8, we see that different values of $w$ are performing better during different stride intervals. But we can see a pattern similar to that of Random – extremely high values of $w$ consistently perform worse than $w$ in the middle range.

5.3.3 Proportion of Core Packet Switches

Here, our focus turns to another parameter that potentially could impact the overall performance of HELIOS system – the nature of composition of the core switch layer. [FPR+10] stresses the need to arrive at a sweet spot for the proportion of CPS and CCS taking into account factors such as cost, and the sample space of
Figure 5.6: Effect of variation of $w$ on Bisection Bandwidth for Stride traffic pattern

applications expected to use the testbed (with their associated traffic patterns).

In the following experiments, we vary the number of CPS ($p$), and measure the bisection bandwidths during the lifetime of flows for different traffic patterns. As we note in Subsection 3.1.3, we partition our experiment space into two classes. This classification is based on how we vary $p$ within the constraints imposed by Equation (5.1):

1. **With Oversubscription**: As $p$ increases, we decrease $c$ to keep $u$ fixed. This introduces oversubscription on optical paths (to Core Circuit Switches).

2. **Without Oversubscription**: As $p$ increases, we increase $u$ to keep $c$ fixed. This ensures that there is no oversubscription on optical paths (to Core...
Figure 5.7: Effect of variation of $w$ on Bisection Bandwidth for Random traffic pattern

Stride

1. With Oversubscription: For Stride traffic pattern, it is clear from Figure 5.9 that increased oversubscription on optical paths degrades performance. As we increase electrical paths at the cost of optical paths, we allow for lesser share of circuits (or link aggregations for $w > 1$) between each pair of communicating pods. This consequently reduces the scheduling flexibility – since there are lesser number of circuits to be scheduled, the scheduler might decide to reduce the share of circuits/link aggregations for a pair (for reasons of fairness or otherwise), though the pair has the ability to fully
utilize those *stolen* circuits. This results in a smaller average BBW overall.

2. **Without Oversubscription**: On the other hand, increasing electrical paths without causing oversubscription on optical paths does not seem to have any effect on average BBW of Stride pattern (as deduced from Figure 5.10). This is along the expected lines, as HELIOS will strive to use optical paths when they are available over electrical paths.

**Random**

1. **With Oversubscription**: Figure 5.11 seems to convey that the proportion of CPS and CCS will not have a marked impact on the performance for a
Figure 5.9: Effect of variation of $p$ (with oversubscription on optical paths) on Bisection Bandwidth for Stride traffic pattern

Random communication pattern. The likely explanation for this behaviour is that, as long as there is *some* path to carry traffic between communicating pods, whether the path itself is electrical or optical is immaterial. Further experiments may be required to reliably conclude whether Random traffic pattern always does better with more Core Packet Switches.

2. **Without Oversubscription**: As we have come to expect from experiments with non-oversubscribed optical links, we do not see any performance benefit to having more electrical links when there are sufficient optical links to cater to the demand. Figure 5.12 shows same performance for various values of $p$. 
Figure 5.10: Effect of variation of $p$ (without oversubscription on optical paths) on Bisection Bandwidth for Stride traffic pattern

Crawl Stride

1. **With Oversubscription**: Not very different from Stride, the performance for Crawl Stride pattern deteriorates (shown in Figure 5.13 as we increase electrical paths at the cost of optical paths – for similar reasons as outlined in Subsubsection 5.3.3.

2. **Without Oversubscription**: In this case, the performance does not significantly improve on the average, we do notice that increased number of uplinks as a result of more electrical paths could be put to use on some occasions (for example, in the third stride in Figure 5.14). We can conclude that having more uplinks never hurts performance (to say nothing about the increased
Figure 5.11: Effect of variation of $p$ (with oversubscription on optical paths) on Bisection Bandwidth for Random traffic pattern cost).

5.3.4 Bisection Bandwidth Utilization Rate

In this subsection, we tabulate the average BBW and utilization rates for the above experiments. Utilization Rate, as defined in Equation (3.1) measures the effectiveness of utilization of the available BBW by HELIOS. The baseline comparison is with the BBW of an Ideal Crossbar Switch (ICS) of the same configuration, which is also the maximum achievable BBW.

For all our saturating traffic patterns, BBW of ICS is given by Equation 5.2.
Figure 5.12: Effect of variation of $p$ (without oversubscription on optical paths) on Bisection Bandwidth for Random traffic pattern

Max. Achievable BBW =BBW of ICS with $N \times H$ ports

\[ =N \times H \times \text{link capacity} \]
\[ =N \times H \times 10 \text{ Gbps} \]

In Table 5.3.4, we have bolded the highest utilization rates for each traffic pattern – the ones that exceed the next highest by more than 0.1%. We infer some trends about utilization rates when the variables are as follows:

- $N$: Even as the size of the testbed doubles, the utilization rates do not vary radically. We also observe that $N = 32$ is consistently performs well across the board.
Figure 5.13: Effect of variation of $p$ (with oversubscription on optical paths) on Bisection Bandwidth for Crawl Stride traffic pattern

- $w$: For Stride and Crawl Stride, values $w = 1 \ldots 8$ perform similarly. Higher values $w = 16, 32$ clearly perform worse. Random, on the other hand, peaks at $w = 16$, and drops sharply for higher values.

- $p$, $c$: Stride and Crawl Stride patterns have better utilization with a smaller $p$ and a larger $c$. There is not much effect on Random pattern.

- $p$, $u$: The rates are too close to call. It appears that increased number of packet switches is not causing any significant benefit for any pattern. When cost is factored in, it might be best to go with lesser number of packet switches.
Table 5.3: Average Bisection Bandwidths and Utilization Rates for Simulation Experiments

<table>
<thead>
<tr>
<th>Variable Parameter</th>
<th>Stride</th>
<th>Random</th>
<th>Crawl Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg. BBW (Gbps)</td>
<td>Util. Rate</td>
<td>Avg. BBW (Gbps)</td>
</tr>
<tr>
<td>$N$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10049.78</td>
<td>49.07%</td>
<td>8146.62</td>
</tr>
<tr>
<td>8</td>
<td>20113.10</td>
<td><strong>49.10%</strong></td>
<td>15891.70</td>
</tr>
<tr>
<td>16</td>
<td>40186.00</td>
<td>49.06%</td>
<td>31757.37</td>
</tr>
<tr>
<td>32</td>
<td>80370.23</td>
<td>49.05%</td>
<td>65048.74</td>
</tr>
<tr>
<td>$w$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>80386.02</td>
<td>49.06%</td>
<td>62948.77</td>
</tr>
<tr>
<td>2</td>
<td>80382.95</td>
<td>49.06%</td>
<td>63722.94</td>
</tr>
<tr>
<td>4</td>
<td>80370.23</td>
<td>49.05%</td>
<td>65048.74</td>
</tr>
<tr>
<td>8</td>
<td>80336.22</td>
<td>49.03%</td>
<td>67269.76</td>
</tr>
<tr>
<td>16</td>
<td>80188.17</td>
<td>48.94%</td>
<td>67796.77</td>
</tr>
<tr>
<td>32</td>
<td>80188.17</td>
<td>48.94%</td>
<td>50343.43</td>
</tr>
<tr>
<td>$p, c$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>80370.23</td>
<td><strong>49.05%</strong></td>
<td>65048.74</td>
</tr>
<tr>
<td>5</td>
<td>80275.60</td>
<td>49.00%</td>
<td>65045.74</td>
</tr>
<tr>
<td>9</td>
<td>79969.95</td>
<td>48.81%</td>
<td>65032.61</td>
</tr>
<tr>
<td>$p, u$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>80370.23</td>
<td>49.05%</td>
<td>65048.74</td>
</tr>
<tr>
<td>2</td>
<td>80379.41</td>
<td>49.06%</td>
<td>65048.74</td>
</tr>
<tr>
<td>4</td>
<td>80395.99</td>
<td>49.07%</td>
<td>65048.74</td>
</tr>
<tr>
<td>8</td>
<td>80425.29</td>
<td>49.09%</td>
<td>65048.74</td>
</tr>
</tbody>
</table>
5.3.5 Running Times of the Simulator

Table 5.3.5 shows the run times of the simulator for various values of $N$. To ensure uniformity of the environment for comparing running times, we ran all the experiments on an Intel Xeon machine with 8 processors at 2.83 GHz each, configured with 4 GB free memory. Each process was scheduled on a CPU core of its own. The parameter values for the experiments are as follows: $N = 4, 8, 16, 32$, $H = 512$, $w = 4, 8$, $p = 1$.

From Table 5.3.5, it is clear that the run time varies linearly as $N$, and is independent of other variables like $w$. Further, we analyzed the effect of varying $H$ on run time, which was also seen to follow linear dependence. Hence, we deduce that the running time of the simulator varies linearly as the product $(N \times H)$.
Table 5.4: Simulator Running Times for Stride Traffic Pattern

<table>
<thead>
<tr>
<th>$N$</th>
<th>$w = 4$</th>
<th>Factor Increase Over Previous</th>
<th>$w = 8$</th>
<th>Factor Increase Over Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Running Time</td>
<td>Factor Increase</td>
<td>Running Time</td>
<td>Factor Increase</td>
</tr>
<tr>
<td>4</td>
<td>6564 s (1.823 hr)</td>
<td>-</td>
<td>6587 s (1.830 hr)</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>15515 s (4.310 hr)</td>
<td>2.364</td>
<td>15802 s (4.389 hr)</td>
<td>2.399</td>
</tr>
<tr>
<td>16</td>
<td>34957 s (9.710 hr)</td>
<td>2.253</td>
<td>34154 s (9.487 hr)</td>
<td>2.161</td>
</tr>
<tr>
<td>32</td>
<td>75356 s (20.932 hr)</td>
<td>2.156</td>
<td>75159 s (20.878 hr)</td>
<td>2.201</td>
</tr>
</tbody>
</table>

5.4 Summary: Research Findings

In this section, we summarize the findings from our evaluation. We seek answers the questions we raised in Section 3.1 from the above results.

1. **Pod Switch Scalability**: Our hypothesis in Subsection 3.1.1 that the performance will linearly increase with increase in number of pods is validated. As $N$ doubles, the average BBW also doubles for all traffic patterns we consider.

2. **Link Aggregation**: On this parameter, we conclude that $w$ doesn’t affect the performance of Stride pattern which agrees with our hypothesis to begin with. However, for other patterns like Random and Crawl Stride, our graphs do not indicate any one value of $w$ (except the high value of 32) consistently performing much better or much worse than others. Very high value of $w = 32$ performs the worst as the scheduling flexibility is drastically reduced. $w = 1$ performs reasonably well for all patterns.

3. **Proportion of Core Packet Switches**

   - **With Oversubscription**: As we increase the number of CPS’es at the cost of circuits, we find that the performance consistently degrades. The reason is that the electrical path is unused when there exists a circuit between the communicating pod pairs. It so happens that for every traffic pattern in our experiments, the pod pairs have some circuits already.
Hence, any reduction of circuits will result in drop of throughput, but will not result in traffic on the electrical path.

- **Without Oversubscription**: As we increase the number of CPS'es without affecting the number of circuit paths, the performance remains consistent. Hence, addition of new electrical paths between pod pairs when they are already connected to each other through circuits will not affect the performance. This runs counter to our hypothesis in Subsection 3.1.3.

4. **Bisection Bandwidth Utilization Rate**: With regard to the utilization rate for various traffic patterns, we conclude that $N$ does not affect it, while $w$ and $p$ do. Lower values of $w$ would result in better utilization rates. When $p$ is increased at the cost of optical paths, utilization rate decreases.
Chapter 6

Conclusion and Future Work

In this thesis, we have presented a simulator for Helios system that models the behavior and performance of Helios at scale. First, we validated the simulator with the performance figures from the testbed execution. Later, we simulated the system performance at scale against a variety of traffic patterns and input parameters like number of pods, link aggregation factor, and proportion of Core Packet Switches. We believe that the simulation results presented in this thesis represents an important addition to our understanding of the working of Helios system.

The key contributions of this thesis include the following:

1. Making a sound case for a hybrid for electrical and optical core interconnect for modular datacenters.

2. Architecture and design of a simulator that predicts the performance of such an interconnect at scale.

3. Validation of the simulator with the testbed execution, and simulation at scale by varying various input parameters and traffic patterns.

6.1 Enhancing the Simulator Functionality

In this section, we explore the avenues for the future work on the simulator.
1. More accurate TCP modeling: The simulator models TCP at the flow-level, hence losing out on some of the characteristics of TCP that are more straightforward to model on packet-level simulation (as are modeled in network simulators like ns-2, OMNeT++). Examples of missing features include TCP behavior on timeouts, packet loss at the switch buffer, loss of an entire window of packets etc. It might be possible to model these on a flow-level simulator as well by developing a probability distribution model of the occurrence of events such as timeouts on the testbed.

2. Richer traffic patterns: It will be insightful to simulate with a variety of other traffic patterns, i.e. one-to-many, many-to-one to name a few. We can also explore the behavior with multiple traffic patterns and/or varying stability values among different subsets of hosts simultaneously. Another interesting extension to the load generator would involve the ability to specify dependency between the flows. An example of such a specification would be: flow B begins after flow A ends (without knowing the end time of flow A in advance).

3. Randomization of flows: Currently the simulator flows are synchronized – flows start and end at the very tick specified in the load generator configuration file. But, time synchronization of flows cannot be expected in the testbed. A small random variation of start/end times of flows will break the synchrony between flows to make the simulator more realistic.

4. Validation at scale: In this thesis, we validate the simulator predictions with results from our small-sized 4-pod/6-pod testbed. While this maybe suffice to show the correct baseline working of the simulator, some validations at scale would prove beneficial to make the simulator numbers more reliable. Obviously, building a larger testbed is not feasible; simulator is intended to substitute it. Hence, we have to build theoretical models for TCP and system behavior at scale.

5. Speeding up through parallelism: The simulation of TCP is more time-consuming than the other components such as Demand Estimator and Ed-
monds’s algorithm. Currently, updating the flowrates for each flow happens sequentially every tick. To speed up, we could explore how to parallelize the updation of flow rates. Intuitively, this is a hard problem since each thread must co-ordinate with others to ensure that the sum of all flowrates on a link does not exceed the link capacity.

6. **Emulation of HELIOS**: Finally, we should analyze the benefits of emulating the system at scale. Emulation could involve deploying real hosts and real traffic over emulated software routers acting as Core Packet/Circuit Switches. One likely gain is that TCP behavior will be more realistic.
Bibliography


