Expanding Horizons: A Comprehensive Exploration of Robustness, Performance and Programmable Data Plane Routing in Next Generation Data Centers.

Mohamad Al Adraa

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

December 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mohamad Al Adraa**

Entitled: **Expanding Horizons: A Comprehensive Exploration of Robustness, Performance and Programmable Data Plane Routing in Next Generation Data Centers.**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________ Chair
Dr. Abdelhak Bentaleb

______________________ External Examiner
Dr. Nizar Bouguila

______________________ Examiner
Dr. Abdelhak Bentaleb

______________________ Examiner
Dr. Sandra Céspedes

______________________ Supervisor
Dr. Chadi Assi

Approved by ______________________
Joey Paquet, Chair
Department of Computer Science and Software Engineering

__________ 2023

____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Expanding Horizons: A Comprehensive Exploration of Robustness, Performance and Programmable Data Plane Routing in Next Generation Data Centers.

Mohamad Al Adraa

In recent years, data center networks have garnered significant attention, rapidly scaling up to meet the demands of the explosive nature of current applications. One notable facet driving this expansion is the pivotal role these networks play in advancing artificial intelligence (AI) and machine learning (ML). As applications like natural language processing models (e.g., OpenAI's GPT series) and image recognition algorithms for autonomous vehicles continue to evolve, data center networks provide the essential computational infrastructure required for the training and deployment of these sophisticated AI and ML models. Lately, significant efforts have been dedicated to enhancing the performance of data center networks, particularly in comparison to the often performance-lagging standard Clos-based topologies like Fat-Tree. One of the approaches for performance improvement is to use alternative data center network topologies. Consequently, researchers explored topologies based on Expander Graphs (EGs), such as Jellyfish, Xpander, and STRAT, where they exploited the sparse and incremental nature of these new topologies. This thesis focuses on investigating the STructured Re-Arranged Topology (STRAT) as a potentially robust and efficient design for next-generation data centers. To benchmark STRAT's performance against the well-known Expander data centers, a robustness framework based on geometric and connectivity-based metrics, along with throughput metrics, is adopted. The findings reveal that STRAT outperforms well-known Expander architectures, positioning them as promising alternatives that surpass the performance of present Clos-based topologies. Moreover, such observations are validated through extensive flow and packet level simulations, demonstrating STRAT's superior performance as compared to other Expanders.

Moreover, the evolution of modern network technology has witnessed a transformative shift with the advent of programmable switches, marking a paradigmatic leap in the realm of data center networks. The programmable data plane of ASIC switches, a cornerstone of this technological advancement, has emerged as a pivotal catalyst for unprecedented innovation and efficiency in data center networks. Its versatility becomes evident in diverse applications, such as employing ML for network classification, enabling dynamic routing mechanisms to achieve line-rate speeds, and implementing In-band Network Telemetry (INT) for enhanced network visibility at a granular level. These applications underscore the transformative power of the programmable data plane, transcending traditional limitations and ushering in a new era of adaptability and performance in data center networks. Building upon this foundation, this thesis introduces a novel routing algorithm that is meticulously prototyped on the BMv2 virtual programmable switch, leveraging the expressive capabilities of the P4 programming language. This implementation serves as a tangible demonstration of the intersection between routing strategies, Expander-based topologies, and the programmable data plane. Notably, the novel routing algorithm showcases superior performance improvements over traditional Equal-Cost Multi-Path (ECMP) algorithm, affirming its potential as a promising solution for harnessing the abundant path diversity inherent in the Expander next-generation data center topologies.

Acknowledgments

Praise be to Allah the Almighty, the Lord of all worlds, for bestowing upon me the strength, guidance, and perseverance throughout this academic journey.

My deepest gratitude to my supervisor, Prof. Chadi Assi. I could not reach this day without his guidance and insightful ideas. His dedication to excellence and commitment have been a constant source of inspiration.

I extend my gratitude to Pelekhaty Vladimir and Frankel Michael at Ciena for their invaluable assistance, mentorship, and collaborative spirit. Their expertise and support have played a pivotal role in the success of this research.

I would like also to thanks the committee members for their effort and time on reading my thesis and providing me with valuable comments and suggestions.

Special appreciation to my parents, Yassine and Nada, my brothers Ahmad and Bahaa, my sisters Bayan and Nour, my aunt Naima, and my fiancé Israa, for their endless love, support, and encouragement. Their love has been my anchor, and their belief in my abilities has fueled my determination.

A special thanks to Al-Madinah Center and the brothers there for their kindness and support. Being in that community, and sharing moments of peace became the thing that I won't ever forget.

In conclusion, I would like to thank everyone who has helped, no matter how small, to make this happens.

Contents

List of Figures

List of Tables

List of Acronyms

ASIC	Application-Specific Integrated Circuit.
DC	Data Center.
DCN	Data Center Network.
EE	Edge Expansion.
FCT	Flow Completion Time.
FT	Forwarding Table.
LR	Last Resort.
MCF	Multi-Commodity Flow.
ML	Machine Learning.
OSP	Off Shortest Path.
OSP1	Off Shortest Path (shortest path hops + 1).
OSP2	Off Shortest Path (shortest path hops + 2).
P4	Programming Protocol-Independent Packet Processors.
PDP	Programmable Data Plane.
PINGing	Packet Initiated Network Generation.
RRG	Random Regular Graph.
RT	Routing Table.
SDN	Software Defined Network.
SP	Shortest Path.
STRAT	STructured Re-Arranged Topology.
ToR	Top of Rack.

Chapter 1

Introduction

1.1 Background and Motivations

In recent years, the Data Center (DC) market has experienced exponential growth, reflecting a shift in technological landscapes and an increased demand for online services. According to recent report [4], the DC market had a value of 5.5 billion in 2019, and this figure is predicted to soar to 20 billion by 2026. This surge is indicative of the escalating demand for online applications, computer architecture, and algorithms in the post-Moore's law era [5, 6]. The unprecedented growth of online services, encompassing on-demand video delivery, storage, social networks, cloud computing, and financial services, necessitates a proportional expansion of DCs [7, 8, 9]. To accommodate this demand, networks have evolved into mega-DCs, comprising hundreds of thousands of servers interconnected by high-speed links [9]. Recognizing the challenges posed by the scale and siting complexities in metropolitan areas, major DC operators like Amazon, Facebook, and Google have transitioned to multi-DC regions [7, 8]. These regions consist of several relatively large DCs (typically 5–20) situated a few tens of kilometers apart, forming a cohesive network within a designated region [7]. However, this scaling up of DCs introduces additional capital and operational expenditures (CA-OP-EX) [10], making the management and operation of DCs increasingly intricate.

The complexity is further heightened by the advent of generative AI, such as Large Language Models (LLMs), which demand extensive datasets, often in the order of hundreds of gigabytes or more. The interaction between LLMs and Data Center Network (DCN) places substantial demands on network bandwidth, necessitating advancements in DC infrastructure to accommodate the escalating data loads and computational intensity of modern applications.

To address these challenges and meet the performance requirements of next-generation DCs, the architecture must be resilient to network failures, deliver low latency, and support high bandwidth with diverse traffic patterns across servers [11]. In the pursuit of achieving these goals, a multitude of topology designs have been proposed and discussed in the literature.

1.1.1 Data Center Topologies

In the realm of DC topologies, two predominant structural categories have emerged: Clos-based topologies and Expander-based topologies. These architectural paradigms play a crucial role in shaping the connectivity and efficiency of modern DCs.

Clos-Based Topologies

Clos-based topologies represent a class of network architectures characterized by a hierarchical structure. Networks following Clos-based designs are deployed in rigidly structured configurations, with examples including popular topologies like Fat-Tree, BCube, and DCell [12, 13, 14].

- **Fat-Tree [12]:** Fat-Tree topology consists of a k switch port count, Point of Delivery (POD) with three-layer structure in the form of binary tree. The first layer comprises of $(k/2)^2$ core switches through which DC communicates with the outside world. The second layer consists of $k/2$ aggregation switches per POD that distributes bandwidth between the low and top layers. The third layer is the access layer consists of $k/2$ Top of Rack (ToR) switches per POD connected to the servers. Notably, a Fat-tree is designed to support full bisection bandwidth between the servers, ensuring that half of the servers can simultaneously communicate with the second half at full capacity.

- **BCube [13]:** BCube topology is characterized by a recursive structure. Each layer consists of BCube blocks, and each block has a fixed number of servers and switches. The structure is defined by parameters such as the number of layers and the number of servers and switches in each block.

- **DCell [14]:** DCell introduces a recursive structure with interconnected switches and servers. The number of layers in a DCell determines the depth of recursion. In a DCell with k ports per switch, each layer consists of k switches, and the number of layers defines the overall structure.

The key feature of Clos-based topologies lies in their hierarchical organization, enabling efficient routing and resiliency. While these topologies offer fault tolerance, their inherent hierarchy can pose challenges in terms of horizontal scalability, flexibility and adaptability to dynamic workloads. For example, in a Fat-Tree topology, adding a single server to accommodate rising service demands necessitates the addition of an entire POD comprises of k switches to preserve the full bisection bandwidth of the topology; however, it results in an increase in capital expenditure (CapEx).

Expander-Based Topologies

In contrast to the hierarchical nature of Clos-based topologies, Expander-based topologies present a flat and more streamlined architecture. In this design, a single layer of ToR switches directly connects to servers, simplifying the network structure and reducing the number of switches.

Initially, all Expander networks inherit from Random Regular Graph (RRG), a class of graphs characterized by being d-regular (all nodes exhibit same degree), where $d \geq 3$, and the product of the number of nodes and d is even. RRG has found extensive applications in diverse fields such as error-code correction, distributed systems, and more recently, in data center design [15]. In various

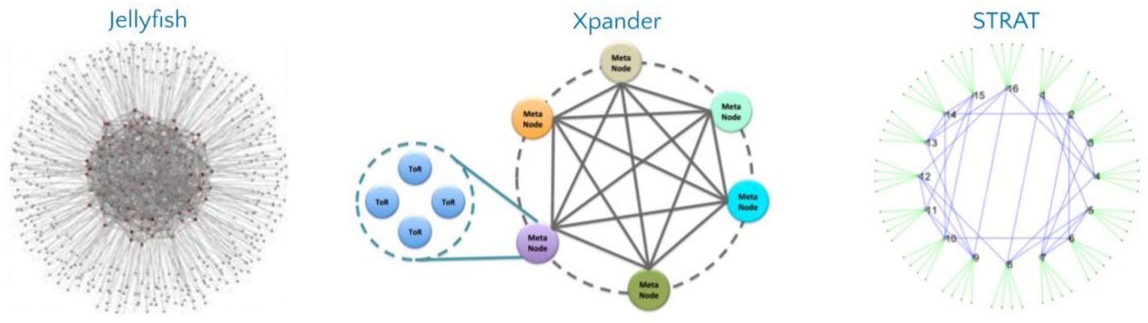


Figure 1.1: Illustrative example of Expander topologies [1], [2], [3].

practical scenarios, RRG demonstrates notable properties, particularly a high Edge Expansion (EE).

EE refers to the presence of a significant number of edges connecting a subset of nodes to the remaining nodes in the graph. This characteristic is indicative of a graph with a small average shortest path, translating in a network context to low-latency connectivity. The utilization of RRG as a foundation for Expander networks underscores its effectiveness in constructing networks with desirable properties, essential for applications requiring low-latency communication and high connectivity.

The following is a brief overview of the three next-generation expander topologies.

- **Jellyfish [1]:** (Fig.(1.1) is a RRG consisting of V switches with M ports each, [1]. d ports on each switch are arbitrarily connected to other switches, and the remaining $M - d$ ports are tied to the servers. This topology can be easily extended by randomly removing edges, after which, the new switch will be connected to the free ports that were previously attached to the removed edges. It has been shown that Jellyfish supports 25% more servers at full capacity compared to Fat-tree; a percentage that may increase with scale [1].

- **Xpander [2]:** (Fig.(1.1) is embodied by RRGs similar to Jellyfish, [2], but it is more cabling friendly. Mainly, the nodes are grouped into clusters called *meta-nodes*, where there are no connections between nodes in the same cluster. This means it has a lower wiring complexity than Jellyfish. Moreover, Xpander is constructed, using a low-complexity algorithm, in a deterministic manner that eliminates the probability of unreliable performance present in Jellyfish. Specifically, Xpander adopts the 2-lifting algorithm proposed in [16], and [17] to expand the topology to the desired number of nodes. Then, it rewires the links between *meta-nodes* to improve the *EE* of the graph. The results in [2] show that Xpander matches the performance of Jellyfish in terms of throughput and outperforms Fat-tree while using roughly $80 - 85\%$ of the switches.

- **STructured Re-Arranged Topology (STRAT) [3]:** (Fig.(1.1) is a novel, high efficiency and low diameter flat DC topology with V switches and fixed-radix ToRs, proposed in [3]. STRAT belongs to the family of Expander graphs, and can be defined as a well-connected d-regular graph with the lowest possible average distance and diameter between its nodes. To construct a STRAT-based network, an empty network graph is initiated, and then through a process of adding nodes and edges while preserving to have the lowest average distance possible a STRAT network is created. Similarly, an RRG network graph can be initiated and then in a process of changing edges connectivity, a new low average distance STRAT is built. Finally, the STRAT network is optimized by applying sophisticated algorithms (*e.g.*, Genetic Algorithm, [18] with Simulated Annealing, [19]) to minimize or maximize different graph metrics including average shortest path, diameter, *EE*, among others. Therefore, STRAT reduces the randomness in building Expanders (*e.g.,* Jellyfish, Xpander), which leads to a more efficient and optimized network. Moreover, to resolve the wiring problem, STRAT can be clustered and deployed with optical patch panels, accounting for zero power and a tiny cost. The work of [3] shows that this topology displays a substantial advantage compared to Clos-networks, since it offers 40% hardware savings with 60% higher throughput.

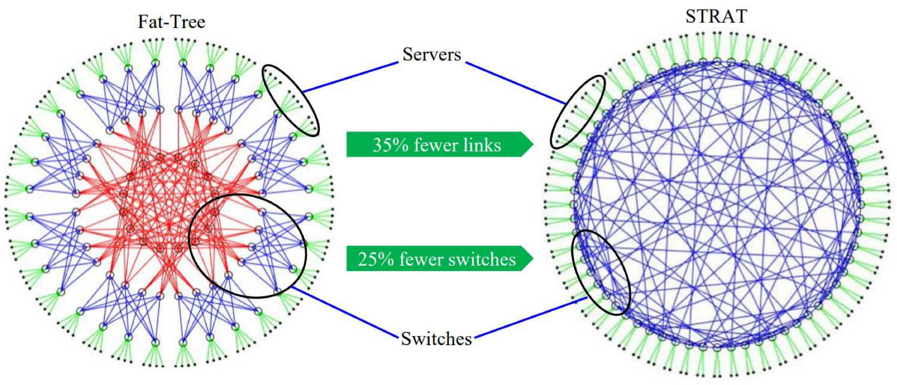


Figure 1.2: Comparison of Fat-Tree and STRAT topologies for a network supporting 128 servers.

Clos-Based vs. Expander-Based

Motivated by Expander graphs' sparsity and high connectivity, Expander-based topologies have been proposed to tackle the challenges of the currently deployed DCs in terms of scalability, resiliency, and energy efficiency [20]. Intuitively, these Expanders' characteristics (high *EE*, low average shortest path, low diameter, etc.) prevent traffic bottlenecks and ensure high network throughput. Moreover, Expanders provide many disjoint paths between nodes, making the network more resilient to failures. All of these network based intuitions are proven in Chapter 2 of this thesis. Furthermore, recent proposals demonstrated that Expander DCs provide near-optimal throughput under one-to-one uniform traffic and outperform Clos networks, [21], [22].

Figure 1.2 illustrates a comparative example between Fat-Tree and STRAT. Both topologies support 128 servers, with STRAT requiring 64 switches (8 ports each) and 192 links, while Fat-Tree necessitates 80 switches (8 ports each) and 256 links. Furthermore, STRAT exhibits a 2.4 average shortest path with a maximum hop of 3, contrasting with Fat-Tree's 3.7 average shortest path and maximum hop of 4. This not only underscores the efficiency of STRAT over a Clos topology but also emphasizes its ability to achieve the same server support with a reduced number of switches and links, translating to significant cost savings in DC setups. The smaller average shortest path and diameter in STRAT further highlight its superior performance in resource utilization and network connectivity, ultimately contributing to lower latency in data transmission and high bandwidth support as number of hops between any two nodes is relatively small.

1.1.2 Programmable Data Plane

The evolution of programmable networks commenced with the advent of Software Defined Network (SDN), a paradigm that decoupled the network control plane from the underlying infrastructure. SDN introduced a centralized and software-driven approach to network management, offering unprecedented flexibility and programmability. In the continual pace of advancement, distributed SDN has also been proposed in DCs to avoid a single point of failure. In this approach, many SDN controllers are distributed across the data center, enhancing resiliency and ensuring continuous network operation in case of failure. One application of SDN is to make forwarding decisions and by adopting the OpenFlow protocol it installs forwarding rules into the switch. As SDN matured, the focus shifted towards achieving programmability not only at the control layer but also within the data plane itself.

This shift led to the emergence of Programmable Data Plane (PDP), a significant advancement beyond traditional fixed-function Application-Specific Integrated Circuit (ASIC). Prior to 2016, data planes were confined to rigid, predetermined algorithms within ASICs, rendering them as black-box entities to network administrators. The turning point occurred with the introduction of the Barefoot Tofino programmable chip in 2016 by Intel.

One of the most renowned programming languages for data planes is P4, which stands for

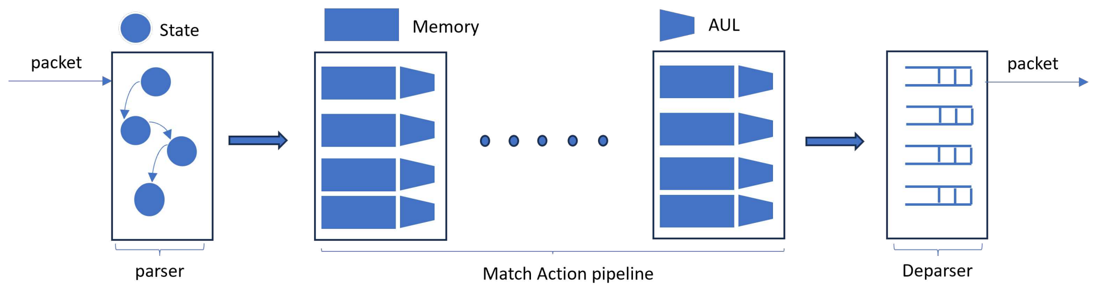


Figure 1.3: PDP architecture, PISA abstraction model.

Programming Protocol-Independent Packet Processors (P4), [23]. P4 is a language meticulously crafted to grant network administrators unparalleled flexibility in crafting data plane-specific codes. It is distinguished by two crucial features: target and protocol independence, [24]. Essentially, P4 enables the customization of packet processing behaviors without being confined to specific communication protocols or underlying hardware architectures. These characteristics underscore P4's pivotal role in revolutionizing programmable network environments.

The programmability inherent in data planes offers a multitude of advantages, ranging from customized packet processing to enhanced processing speed and reduced power consumption, [24, 25]. To illustrate PDP architecture, in Fig. 1.3 consider the PISA architecture that consists of a parser, Match-Action Pipeline, and a deparser, [26].

- Parser: A finite state machine extracts packet headers, excluding the payload as the payload insertion occurs at the end, ensuring efficient processing. For example, tofino utilizes approximately 18 parsers per port for enhanced header resolution and parallelism.

- Match-Action Pipeline: It forms a parallel processing pipeline. Operating on a hit/miss basis, it determines actions based on lookup key outcomes. It gets populated by the control plane, and it provides control over packet header manipulation. A Match-Action pipeline comprises of 12 to 20 units based on the data plane architecture.

- Deparser: Consisting of flow control statements to reassemble packet headers and payload before transmitting to the egress port, it is essential for coherent packet structure reconstruction

The significance of PDPs extends prominently to DCs. In the dynamic and complex environment of DCs, programmability plays a pivotal role in:

- Customizing Applications: Tailoring packet processing for specific applications within the DCs environment. For instance, a DC-specific network protocol can be implemented in the PDPs for better routing efficiency without requiring any changes in the application layer.
- Adapting to Workload Changes: Contributing to forwarding tables scalability by updating their entries locally in the switch based on network conditions and traffic pattern.
- Optimizing Resource Utilization: Efficiently deploying diverse networking functions, leading to the judicious use of hardware resources.

A concrete example of PDP usage in DCs is the dynamic creation of application-specific network services. For instance, a DC can deploy load balancing, firewalling, and telemetry services within a single programmable switch, optimizing resource usage and enhancing network efficiency. Moreover, the integration of Machine Learning (ML) for network classification and resource management further extends the capabilities of PDPs, [27, 28, 29]. ML algorithms can intelligently categorize network traffic, enabling adaptive and automated resource allocation based on real-time demand. This dynamic synergy between programmability and ML empowers DCs to achieve heightened operational efficiency, responsiveness, and adaptability to the evolving demands of modern applications.

Indeed, the move towards Expander-based topologies marks a departure from the traditional hierarchical structures seen in Clos-based topologies. The efficiency gains achieved by Expander DCs (see Fig. 1.2) make them a compelling area of research, addressing challenges posed by the growing scale and complexity of contemporary DCs. However, despite the promising performance of Expander DCs, a comprehensive analysis, especially in the case of STRAT against state-of-the-art Expanders like Jellyfish and Xpander, is notably absent from the existing literature. Furthermore, the robustness of Expander DCs remains an underexplored dimension, with robustness studies traditionally focused on multilayered topologies. Moreover, recognizing the vast potential of Expander DCs and leveraging the emerging of PDPs to empower DC efficiency, motivated the propose of an efficient routing protocol tailored to Expander DCs to exploit their power and characteristics.

1.2 Thesis Contributions

Building upon the motivating factors highlighted in the preceding section, this thesis makes significant contributions to the field through the following:

1.2.1 Comprehensive Performance and Robustness Analysis of Expander-Based DCs

In chapter 2, we investigate STRAT as an efficient Expander-based topology for next-generation DCs. Hence, motivating the exploration of its achieved robustness and throughput performance compared to the existing earlier-mentioned state-of-the-art flat topologies for the purpose of highlighting its key advantages. To the best of the authors' knowledge, this is the first in-depth study on the state-of-the-art Expander topologies specifically STRAT, Xpander, and Jellyfish, with major contributions that can be summarized as follows:

(1) An extensive robustness comparison is performed between STRAT, Jellyfish and Xpander using geometric and connectivity-based metrics. The study also examines how switch failures affect these metrics. The result shows the superiority of STRAT as a possible design for high performance DCs, as it achieves $\lesssim 13\%$ lower average shortest path, and $\lesssim 11\%$ higher spectral gap. Moreover, it is shown that STRAT remains more robust under random and targeted failures than Jellyfish and Xpander, and although STRAT's robustness decreases faster than that of the other topologies, it is never worse.

(2) A scalable Multi-Commodity Flow (MCF) optimization problem using only K-paths is formulated, with the aim of maximizing the minimum end-to-end demand throughput. The MCF optimization objective ensures some fairness among flows, making it more efficient than using k-shortest path routing. To this end, the throughput of STRAT, Jellyfish, and Xpander is analyzed as a key performance metric of DCs. Using the all-to-all traffic matrix, the analysis shows that STRAT outperforms other Expander-based topologies by approximately $6 - 7\%$

(3) An extensive flow and packet level simulations using Mininet [30] and Netbench [31] network simulators are conducted. The simulation result demonstrates that STRAT has better flow completion time $\lesssim 8\%$, indicative of enhanced latency and throughput, using various existing

routing in the literature (*e.g.,* ECMP[32], VLB[33], KSP[34], and HYB[22]) and under different traffic matrices (*e.g.,* all-to-all Facebook DC's workload, and ProjectToR rack-to-rack Microsoft DC's workload[22]) as compared to Jellyfish and Xpander.

1.2.2 Expander-Based Data Center Routing: A Programmable Data Plane Perspective

In Chapter 3, our focus shifts to the integration of Expander-based network features with the revolutionary capabilities offered by PDPs. The primary contribution of this chapter lies in the introduction of a novel routing algorithm tailored for Expander DCs and designed to operate exclusively within the data plane which enhances the efficiency of data packet forwarding, ensuring line-rate speed and responsiveness. This approach significantly reinforces the network's scalability, a critical factor in addressing the evolving demands of emerging Mega DCs. Notably, our work extends beyond theoretical propositions by practically implementing the new routing algorithm using P4 [24]. P4, as the predominant language in the realm of PDPs, facilitates a seamless integration that aligns with industry standards, underscoring the practical viability of our contributions. Indeed, this chapter lays the groundwork for a paradigm shift in Expander-based DC routing, combining the inherent advantages of Expander topologies with the programmability and efficiency of PDPs to meet the complex challenges of modern DCs environments.

1.3 Thesis organization

This thesis embarks on a journey through the intricate landscapes of Expander-based network topologies and the transformative realm of PDP routing algorithms. In Chapter 2, unfolds as the cornerstone, unveiling the STRAT architecture's efficiency and robustness through a meticulous comparative study. Chapter 3 pivots to the revolutionary domain of PDP routing algorithms, introducing a novel approach tailored for Expander DCs. Finally, Chapter 4 synthesizes key findings and contributions, and outlining future research directions.

Chapter 2

Comprehensive Performance and Robustness Analysis of Expander-Based Data Centers

2.1 Overview and Motivation

2.1.1 Legacy Data Centers

The development of DCs has been primarily driven by the creation of new DC topologies that are scalable, low-latency, and resilient. The most well-known DC topology is the Fat-tree, which has a hierarchical structure. Fat-tree can be built in two different ways, such as a two-layer Fat-tree (leaf and spine) adopted by Facebook [35] and a three-layer Fat-tree (edge, aggregation, and core) proposed by Google [12]. F10 [36] is a more robust version of the three-layer Fat-tree that rewires the redundant links in the core layer for increased resiliency. VL2 [37] is based on the three-layer Fat-tree with slight modifications to the upper layer to have complete bipartite graph between aggregation and core layers. DCell [14], BCube [13], and MDCube [38] are server-centric topologies that use servers as both switching and endpoint devices. These topologies can be built in a recursive manner, duplicating blocks of switches and servers to achieve the desired topology.

2.1.2 Reconfigurable Data Centers

Optical Circuit Switching (OCS) is a cutting-edge technology that offers high bandwidth to accommodate the demands of ML and big data applications. This new technology opens the door to low-latency and high-bandwidth reconfigurable DCs. c-Through [39] is based on the three-layer Fat-tree, with all ToR switches directly connected to an OCS for fast host routing. Helios [40] is another reconfigurable DC consisting of two layers (leaf and spine), with a combination of electrical and optical switches in the spine layer. Helios uses the max-min fairness problem to distribute traffic across paths. Flexspander [41] is a novel reconfigurable DCN topology which incorporates OCSs into the network architecture to enhance its performance. Flexspander is composed of different PODs, each of which forms a well-designed expander graph. The interconnections between the pods are established through the OCSs, allowing the network to dynamically adjust to predicted traffic patterns and improve communication efficiency.

2.1.3 Expander Data Centers

While Legacy DCs, epitomized by topologies like Fat-tree, have been foundational in providing resiliency, they grapple with challenges posed by scalability and dynamic workloads. The hierarchical structures inherent in these designs may limit their adaptability to fluctuating computational demands. On the other hand, reconfigurable DCs, leveraging technologies like OCS, showcase promising features for high bandwidth and low latency. However, the practical implementation of such reconfigurable architectures is still an area of active research and experimentation. As the industry seeks solutions that balance adaptability, performance, and scalability, Expander-Based DCs emerge as a compelling paradigm for the next generation of DCs. These DCs, encompassing designs like Jellyfish, Xpander, and STRAT, introduce innovative topologies that exhibit robustness, efficiency, and the ability to dynamically adjust to varying traffic patterns. In the following sections, we delve into a comprehensive exploration of Expander-Based DCs, elucidating their performance and robustness advantages.

In the context of Expanders' performance, several metrics were adopted to verify their performance [42, 43, 11]. The authors of [42] derived an upper bound for the throughput of the homogeneous topologies and demonstrated that Jellyfish (RRGs) throughput complies with this bound under a random permutation traffic matrix. Additionally, they showed that RRGs can be used in heterogeneous topologies in order to improve the overall DC performance. [43] studied the throughput of Xpander in terms of the demand-completion-time metric. The authors validated that Xpander achieves a throughput close to one when using a single permutation matrix, which is not the case when the traffic matrix is a collection of permutation matrices. The authors of [11] affirmed that bisection bandwidth is not a sufficient metric to study the performance of DCs. In their study, they showed that Expanders can achieve full bisection bandwidth in places where they lack full throughput, and this raises many questions about prior research on the DCs' performance.

Furthermore, several existing work studied Expanders' performance compared to current DC networks [44, 22, 21, 45]. The authors of [45] demonstrated that even though Expanders are static topologies (not re-configurable), yet flexible enough to provide network capacity compared to the Fat-tree network. Through flow evaluation and discrete packet simulation, they showed that with $67.5-80\%$ of comparable cost, Expanders were capable of outperforming Fat-tree in all traffic scenarios. The work of [22] verified that Xpander beats Fat-tree under different skewed traffic matrices using a combination of ECMP, flowlet switching and valiant load balancing. In [21], the authors compared the performance of DRing (simple ring topology), Expanders (RRGs), and leaf-spine topologies. They considered two different routing schemes including ECMP and shortest-union, a practical routing algorithm for flat networks, and they determined that DRing and Expanders have lower Flow Completion Time (FCT) on different scales and under various traffic matrices than leaf-spine network. Last but not least, in [44] the authors developed a framework to benchmark the throughput of network topologies, including Fat-tree, Longhop, Slim Fly, and Jellyfish.

2.2 Robustness of Data Centers Topologies

DCs are expected to provision/support highly elevated service availability (*e.g.*, from five to six nines) regardless of their susceptibility to failures and/or saturation; particularly when operating under high data traffic loads. Failures typically manifest themselves in three main forms, namely: *i*) hardware failures (*e.g.*, physical links, switches, etc), *ii*) software failures (*e.g.*, protocols, applications, etc), and, *iii*) human errors (*e.g.*, falsified input parameters, erroneous configurations, etc) [20]. In this regard, emphasis is directed towards the fact that failure prevention is remarkably less costly than failure recovery and much more efficient from the perspective of network operability and functionality. The literature displays many definitions for the term "*network robustness*" (*e.g.*, [46]), though herein, network robustness refers to the resiliency of the network performance, following physical network component failures; precisely switch and/or link failures.

In light of the above discussion, several studies have systematically explored DC network robustness [47, 46, 48, 49, 8]. For instance in [47], the authors investigated how failures (switches, servers, and links) affect Clos-based topologies, modelled as graphs, in terms of connectivity and path length. For the same topologies, the authors of [46] applied a combination of the classical network robustness metrics, such as diameter, elasticity, and heterogeneity, under various failure scenarios. They also introduced a deterioration parameter, which measures a metric's degradation after network failure. The work of [48] conducted theoretical and simulation-based robustness analysis, where they showed that BCube and DCell exhibited better resiliency than Fat-tree. In [49], a robustness study was carried out on the multilayered topologies to reflect the influence of failures on connectivity. Therein, the metric (μ-A2TR) identified how hard it is to break the network into different subnetworks given a specific failure type and topology. Finally, the work of [8] examined how failures affect the traffic routed in a multilayered topology and showed the gain that redundancy of aggregation switches and links may provide in the case of failure.

2.3 Robustness Analysis of Expander Data Centers

Failures in DCs often lead to catastrophic consequences if not timely and adequately handled/recovered where recovery costs can be remarkably high especially that recovery is required to be seamless. Recently, Rogers Communication experienced a disastrous outage because of routers ill-configuration that stopped Internet services for roughly 2.25 million subscribers and brought down emergency calls and some banking services, [50].[1] Moreover, around 60 autonomous cars developed by Cruise experienced a server outage causing a traffic blockage in San Francisco, [51]. Unarguably, driven by the possibility that such severe scenarios may arise, designing robust networks that would take into account such failures and counteract them is a must. In this regard, this section is dedicated to laying out an extensive analytical framework that quantifies network robustness in terms of various metrics for Expander DCs. This framework will assist the reader in understanding the impact of physical failures through providing in-depth and in-breadth insights into the occurrence of such failures. This framework starts by presenting the system model which explain how each DC topology is modeled.

2.3.1 System Model

A DC network topology is modeled herein as a graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes representing the switches, and the set of edges that connect these switches, respectively. Thus, the considered topology consists of $V = |\mathcal{V}|$ switches, and $E = |\mathcal{E}|$ links. Also, we assume that each switch is equipped with M ports. Since Expanders used herein are d-regular graphs, each switch is connected to d other switches while the remaining $M - d$ switch ports are switch-to-server connections. $\boldsymbol{\beta} \in \mathbb{R}^{V \times V}$ is the adjacency matrix of G, with:

$$\beta_e = \begin{cases} 1 & \text{if link } e = (i,j) \in \mathcal{E}, \\ 0 & \text{otherwise.} \end{cases}$$

In the following, the capacity of link $e = (i, j)$, for each $e \in \mathcal{E}$, that connect nodes i and j, for all $i, j \in \mathcal{V}$, is denoted by $C(e)$ and satisfies $C(e) \geq 0$. The maximum bandwidth of traffic traversing link e should not exceed $C(e)$. For the sake of clarity, let $\mathcal{F} = \{f^1, f^2, \ldots, f^{\mathrm{N}}\}$ be the set of flows in the network. Each flow $f \in \mathcal{F}$ is defined by the tuple $f^n = (s^n, d^n, b^n)$, where s^n, d^n and b^n are the source, destination, and the demand of the flow f^n, respectively, for all $s^n, d^n \in \mathcal{V}$. $\mathcal{F}$ constitutes a general traffic matrix that can represent all-to-all, random shuffling, or any type of

[1]Canadian mobile carrier.

skewed traffic patterns. It follows that $\sum_{n=1}^{\mathrm{N}} f_e^n \leq C(e)$, where f_e^n represents the portion of flow f^n traversing the edge, $e = (i, j)$, where $f_e^n \leq b^n$. For the purpose of this study, Jellyfish/Xpander-based network instances are generated by feeding a Python-based script employing NetworkX (see [52]) with high-level input parameter values such as the number of switches, number of servers, and switches' port count. To this end, Jellyfish and Xpander topologies are established based on the algorithms provided in [1], [2], respectively. On the other hand, STRAT (see [3]) is generated using an in-house framework. The experiments are performed over five different topologies, starting with a small-scale network and moving to a large-scale network unless stated otherwise. Since Jellyfish, Xpander and STRAT belong to the same family of graphs, identical number of links, switches and servers is used for all them.

2.3.2 Robustness Metrics

What follows is a brief summary of the adopted structural robustness metrics utilized in conducting network robustness analyses:

• *Average Shortest Path Length*, (ASPL), denoted by $\overline{L}_{sp}$ is the sum of all path lengths between every source-destination pair in the network averaged by $V \times (V-1)$. In general, a network with small $\overline{L}_{sp}$ tends to be more robust [53]. In this regard, it has been proven that the lowest $\overline{L}_{sp}$ for d-regular graphs is:

$$\overline{L}_{sp}^{*} = \frac{\sum_{a=1}^{\mathrm{b}-1} a \times d \times (d-1)^{a-1} + b \times R}{V-1}, \tag{1}$$

where

$$R = V - 1 - \sum_{a=1}^{\mathrm{b}-1} d \times (d-1)^{a-1} \geq 0, \tag{2}$$

and $\boldsymbol{b}$ is the largest integer satisfying (2).

• *Diameter*, D, is the number of hops corresponding to the longest path's length among all shortest paths between each source-destination pair. Generally, networks with a small diameter are more likely to be able to withstand failure occurrences, [54]. Noting that the lower bound of a d-regular graph is $\lceil \log_d V \rceil$ [2], where $\lceil x \rceil$ is the ceiling of x.

• *Spectral Gap,* $\Delta\lambda$*,* is the difference between the largest and the second largest (in absolute value) eigenvalues of the adjacency matrix. Usually, high *EE* implies high spectral gap, which indicates the goodness of a graph [55]. Therefore, $\Delta\lambda$ is used to approximate the expansion of d-regular graphs, since computing the *EE* of d-graphs is NP-hard, [56]. High value of $\Delta\lambda$ relates to better resiliency in the network [57].

• *Algebraic Connectivity,* λ_2*,* is the second smallest eigenvalue of a graph's Laplacian matrix. λ_2 indicates the difficulty to break the network into different sub-networks. Specifically, a graph with a high value of λ_2 infers that the topology is more resilient against switch and link failures [57].

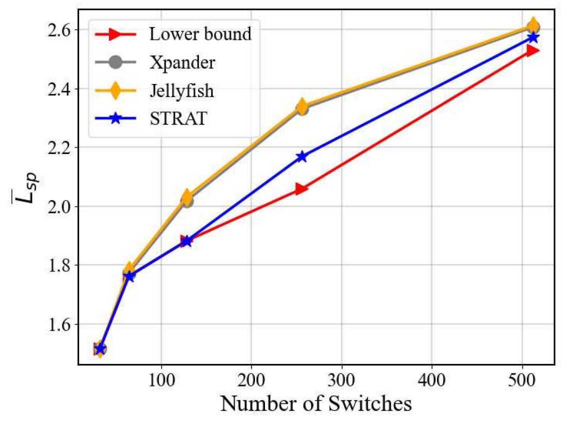
• *Betweenness Centrality* is the number of shortest paths traversing a switch (σ_1) or a link (σ_2) normalized by $2/((V-1)(V-2))$. In fact, betweenness centrality quantifies the influence that a component (switch and/or link) incurs on the network. In other words, whenever a node and/or edge with a large betweenness fails, it has the potential to disrupt the network as a whole. Thus, it is evident that a network with smaller nodes' and edges' betweenness will likely be less disrupted by failures, [57].

In essence, the above classical metrics are based on the concepts of graph theory. In other words, these metrics study the structural properties of a network without introducing switch or link failures. In this regard, *Deterioration*, DT is a dynamic metric that quantifies the impact of various types of failures on the network robustness. DT evaluates the degradation of a metric MT (*e.g.*, $\overline{L}_{sp}$ and λ_2) under various failure percentages, [46]. This metric reflects the QoS of many important network measures such as delay, jitter, and packet loss. Let δ_0 be the value of δ when the network does not have any failure, and δ_i be the value of δ at i percent of switch and/or link failure. Then DT can be computed as:

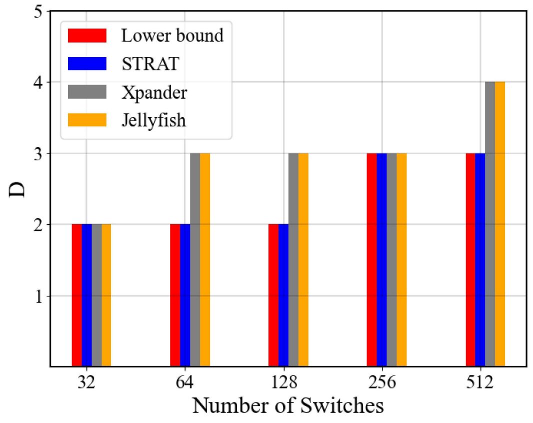
$$DT = \left| \frac{1}{\delta_0} \left(\frac{\sum_{i=1}^{\mathrm{n}} \delta_i}{n} - \delta_0 \right) \right| \tag{3}$$

where $|x|$ is the absolute value of x.

In fact, failures in DCs can be classified based on their cause into random and target failures, [53]. Unexpected (random) failures are caused by hardware (switch or link), congestion, or human



(a) Average Shortest Path Length



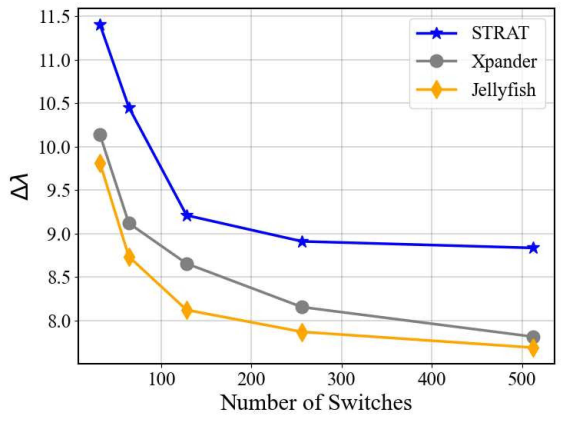
(b) Diameter

Figure 2.1: Comparison of Average Shortest Path Length and Diameter VS. the number of switches.

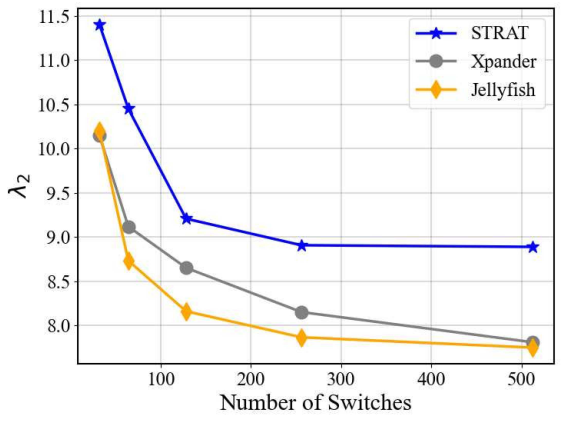
errors. This kind of impairment occurs indiscriminately. On the other hand, target failures result from external attacks directed to some of the network's switches and/or links. For example, failure of nodes or edges with high betweenness centrality may lead to higher performance degradation than nodes or edges with lower betweenness centrality.

2.3.3 Numerical Evaluation

In this section, extensive numerical simulation is conducted to evaluate the robustness of STRAT, with both Jellyfish and Xpander adopted as benchmarks. In this simulation environment, five different network sizes are considered with a fixed switch degree of $d = 15$. In fact, the robustness metrics described earlier are used herein, i.e., average shortest path length, diameter, spectral gap, algebraic connectivity, and betweenness centrality. Additionally, the deterioration of these robustness metrics under switch failures is investigated. Specifically, random and target failures are introduced on the network switches within a range of 0 to 12% in a step of 2%. In both types of failure, it is ensured that the network is represented as a new graph $\hat{G}(\hat{\mathcal{V}}, \hat{\mathcal{E}})$, where $\hat{G} \subseteq G$, remains connected after the failures. On a side note, reported results are averaged over 100 independent Monte Carlo runs in the case of random failure.



(a) Spectral Gap



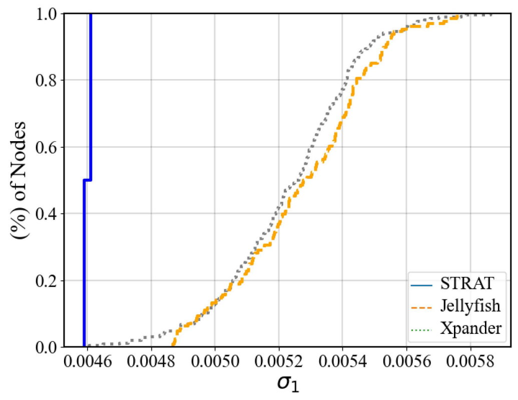
(b) Algebraic Connectivity

Figure 2.2: Comparison of Spectral Gap and Algebraic Connectivity VS. number of switches.

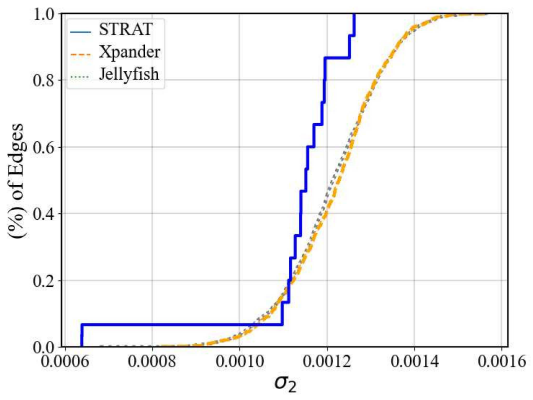
Robustness Performance Without Failure

Fig. 2.1a and Fig. 2.1b plot $\overline{L}_{sp}$ and D, respectively, while varying the number of switches. Particularly, Fig. 2.1a illustrates the gap between $\overline{L}_{sp}$ of each network (*i.e.*, STRAT, Jellyfish, and Xpander) and the lower bound in (1). The figure shows that all Expanders achieve the lower bound at a small scale network. However, as the network size increases, the optimality gap increases (*i.e.*, larger $\overline{L}_{sp}$). Moreover, it is evident that STRAT keeps attaining a better $\overline{L}_{sp}$ (smaller gap) than both Xpander and Jellyfish, because of the way STRAT is constructed. For example, at $V = 256$, the gap between the achieved $\overline{L}_{sp}$ of STRAT and the lower bound is only 5% compared to 13% for both Jellyfish and Xpander. Accordingly, traffic circulating through STRAT topology will experience shorter paths and lower latency towards the destination. As a result, STRAT will be less penalized by failures. Fig. 2.1b shows the diameter against the number of switches of the network. It is clear from the figure that STRAT has smaller diameter which is equal to the theoretical lower bound ($\lceil \log_d V \rceil$) while that of Jellyfish and Xpander are respectively within 1 hop from this lower bound. Therefore, STRAT will be more resilient to failures, and network flows will experience low end-to-end latency.

Fig. 2.2a and Fig. 2.2b illustrate the spectral gap, $\Delta\lambda$ and the algebraic connectivity, λ_2, respectively, while varying the number of switches. The figures show that both $\Delta\lambda$ and λ_2 are decreasing as a function of the number of switches. This decrease in $\Delta\lambda$ and λ_2 is expected since increasing



(a) Node betweenness CDF with V = 256 and d = 15



(b) Edge betweenness CDF with V = 256 and d = 15

Figure 2.3: Cumulative distribution function of node and edge betweenness.

the number of switches while maintaining the same degree, $d = 15$ in this example, involves fewer links needed to break the network. The figure also demonstrates that STRAT has better $\Delta\lambda$ and λ_2, exhibiting higher values, compared to Jellyfish and Xpander, which further confirms that STRAT is more resilient. Even though Jellyfish and Xpander achieve similar $\overline{L}_{sp}$ and D at small sized networks to that of STRAT, the latter manifests greater $\Delta\lambda$ and λ_2, which validates its robustness even for such networks. Furthermore, Xpander manifests slight improvement in $\Delta\lambda$ and λ_2 over Jellyfish. Hence, Xpander appears to be more robust, which can be attributed to Jellyfish's random nature.

Fig. 2.3 demonstrates the cumulative distribution function (c.d.f.) of the node and edge betweenness for network size, $V = 256$, (*i.e.,* $G(256, 1920)$). Fig. 2.3a vividly shows that STRAT has a much smaller maximum of node betweenness, which means that switches will not be as prone to overloading as much as in Jellyfish and Xpander. In other words, all nodes in STRAT have relatively similar node betweenness (0.0046), where every node is traversed by around 149 shortest paths. On the other hand, nodes in Jellyfish and Xpander display various levels of betweenness. An insignificant portion of the nodes exhibits a small betweenness (155 shortest path), whereas the rest of them are traversed by shortest paths ranging from 160 up to 190. Hence, for STRAT, traffic will be more spread out and there will be fewer potential bottlenecks in the network due to the fact that all the nodes host the same number of shortest paths. On the contrary, for Jellyfish and Xpander,

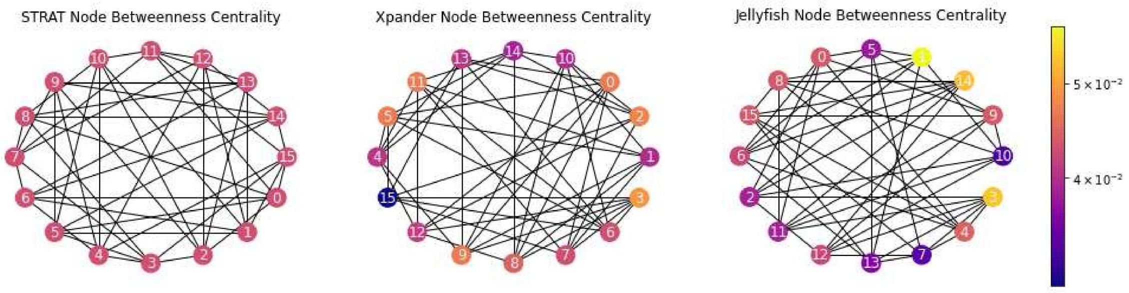

STRAT Node Betweenness Centrality
Xpander Node Betweenness Centrality
Jellyfish Node Betweenness Centrality
5×10^{-2}
4×10^{-2}

Figure 2.4: Node betweenness distributions with $V = 16$ and $d = 6$.

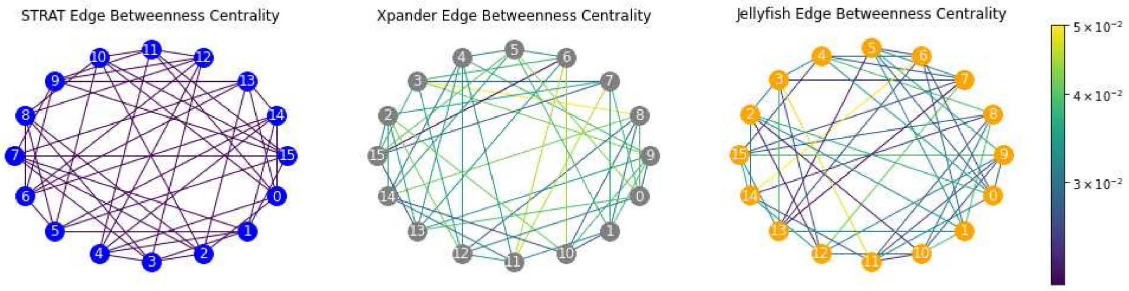

STRAT Edge Betweenness Centrality
Xpander Edge Betweenness Centrality
Jellyfish Edge Betweenness Centrality
5×10^{-2}
4×10^{-2}
3×10^{-2}

Figure 2.5: Edge betweenness distributions with $V = 16$ and $d = 6$.

nodes with high betweenness will experience more intense traffic leading to higher chances of network congestion. Another indication is that if a node fails in the latter the consequences will be more server because the network will be loosing a higher number of shortest path. Similarly, Fig. 2.3b shows that STRAT has smaller maximum of edge betweenness, which means that less number of shortest paths are attributed to the same link. Thus, traffic is more spread out over links enhancing the overall utilization of the network.

For more elaboration, Fig. 2.4 and Fig. 2.5 are Heat-Map representations of node and edge betweenness, respectively, for small network size, $V = 16$. The figures illustrate that all nodes and edges of STRAT exhibit nearly similar and low betweenness compared to Jellyfish and Xpander, which leads to the same intuition concluded from the 256 network's betweenness experiment. Moreover, these results reveal that target failures will cause severe performance degradation in Jellyfish and Xpander as most of their nodes and edges have large maximum betweenness leading to a bigger loss in terms of shortest paths compared to STRAT.

Table 2.1: Deterioration of robustness metrics under random switch failure.

N (%)	$\overline{L}_{sp}$ DT(%)			$\Delta\lambda$ DT(%)			λ_2 DT(%)		
	STRAT	Xpander	Jellyfish	STRAT	Xpander	Jellyfish	STRAT	Xpander	Jellyfish
2	2.18 (0.549)	2.337 (0.311)	2.347 (0.301)	8.570 (3.537)	7.914 (2.815)	7.643 (2.458)	8.579 (2.41)	7.831 (3.947)	7.617 (3.194)
4	2.193 (1.101)	2.345 (0.628)	2.354 (0.607)	8.339 (4.976)	7.687 (5.439)	7.431 (4.881)	8.287 (6.826)	7.531 (7.630)	7.363 (6.418)
6	2.205 (1.659)	2.361 (0.949)	2.352 (0.917)	8.110 (7.393)	7.459 (8.035)	7.223 (7.254)	7.985 (10.216)	7.111 (11.285)	7.233 (9.622)
8	2.217 (2.219)	2.360 (1.277)	2.368 (1.240)	7.883 (9.795)	7.236 (10.612)	7.016 (9.677)	7.680 (13.643)	6.946 (14.811)	6.849 (12.952)
10	2.229 (2.788)	2.368 (1.609)	2.376 (1.556)	7.656 (12.153)	7.018 (13.185)	6.802 (12.057)	7.365 (17.185)	6.661 (18.299)	6.595 (16.175)
12	2.241 (3.352)	2.376 (1.949)	2.385 (1.887)	7.431 (14.526)	6.800 (15.732)	6.598 (14.522)	7.0442 (20.794)	6.386 (21.676)	6.337 (19.464)

Table 2.2: Deterioration of robustness metrics under targeted switch failure in which switches with high betweenness are dropped out.

N (%)	$\overline{L}_{sp}$ DT(%)			$\Delta\lambda$ DT(%)			λ_2 DT(%)		
	STRAT	Xpander	Jellyfish	STRAT	Xpander	Jellyfish	STRAT	Xpander	Jellyfish
2	2.181 (0.571)	2.340 (0.444)	2.349 (0.426)	8.601 (2.473)	7.905 (3.039)	7.612 (3.257)	8.594 (3.365)	7.777 (4.606)	7.563 (3.673)
4	2.194 (1.188)	2.350 (0.864)	2.358 (0.799)	8.395 (4.802)	7.649 (6.175)	7.355 (6.524)	8.303 (6.646)	7.534 (7.593)	8.687 (6.786)
6	2.207 (1.799)	2.361 (1.337)	2.368 (1.223)	8.163 (7.435)	7.353 (9.813)	7.079 (10.025)	8.036 (9.645)	7.202 (11.666)	6.801 (13.557)
8	2.223 (2.501)	2.370 (1.731)	2.375 (1.561)	7.906 (10.349)	7.084 (13.114)	6.835 (13.128)	7.757 (12.779)	6.876 (15.659)	6.542 (16.849)
10	2.235 (3.061)	2.379 (2.107)	2.384 (1.935)	6.197 (12.657)	7.702 (16.155)	6.836 (15.926)	6.615 (15.902)	7.479 (20.991)	6.442 (21.237)
12	2.25 (3.748)	2.390 (2.582)	2.394 (2.344)	7.457 (15.444)	6.567 (19.454)	6.354 (19.241)	7.103 (20.129)	6.201 (23.939)	5.584 (29.027)

Robustness Performance Under Switch Failures

Tables 2.1 and 2.2 demonstrate the impact of switch failure on the structural properties of the network in terms of $\overline{L}_{sp}$, $\Delta\lambda$ and λ_2. The results interpret each metric and its deterioration under both random and targeted switch failures. Although the $\overline{L}_{sp}$ of STRAT deteriorates, in general, faster than that of Jellyfish and Xpander, it still exhibits lower values. This is because STRAT is mainly optimized toward minimizing the $\overline{L}_{sp}$, making it more sensitive. Similarly, since Xpander is optimized to enhance the spectral gap (equivalent to enhancing the algebraic connectivity), its spectral gap is more sensitive to failure compared to STRAT and Jellyfish. However, as shown in Table II (target failure), the deterioration rates of both Jellyfish and Xpander are higher than that of STRAT's in terms of $\Delta\lambda$ and λ_2. For example, at 10% failure, STRAT's deterioration rate is roughly 5% less than that of the two other topologies. This indicates that with increasing failure percentage STRAT proved to be less vulnerable when it comes to the loss of the available shortest paths.

2.4 Throughput Analysis

2.4.1 Throughput Maximization

Among the most famous DC routing algorithms are the ECMP, [32], and K-Shortest Path (KSP), [34]. Of the two, ECMP seamlessly exploits the paths in a traditional Clos-network, where many paths have the same cost (*e.g.* Fat-tree). However, it falls short when it comes to Expander DCs due to the path cost diversity. As shown in [2] and [1], KSP could utilize Expanders' paths more efficiently than ECMP. Still, KSP does not optimally benefit from this advantage of Expander DCs because, after all, it is bounded by K-paths, so it could partially exploit the variety of paths supplied by Expander DCs in general, STRAT in specific. To illustrate Expanders' rich throughput performance, a Multi-Commodity Flow (MCF) optimization problem is formulated with the objective of maximizing the fraction of flow demand, that each commodity can send to guarantee fairness between demands. The motivation is to determine the maximum throughput that Expander DCs can optimally achieve while efficiently using their path diversity. Let α be the variable that controls the fraction of demand of all flows in the network. Then, the optimization problem can be written as follows:

$$\mathcal{P}: \quad \max_{\mathbf{B},\alpha} \quad \alpha \tag{4a}$$

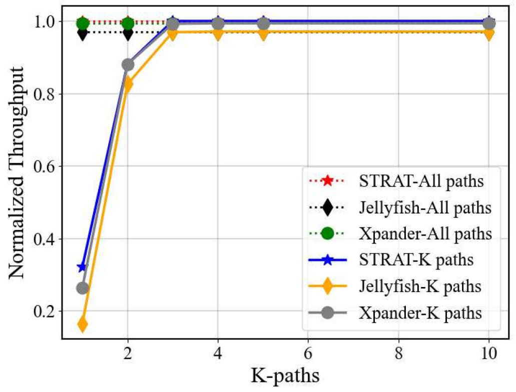
$$\text{s.t.} \sum_{n\in\mathcal{N}} b^n_{(j,i)} \leq C(j,i), \quad \forall\ (j,i) \in \mathcal{E}, \tag{4b}$$

$$\sum_i b^n_{(j,i)} \cdot \beta_{j,i} - \sum_i b^n_{(i,j)} \cdot \beta_{i,j} = \begin{cases} +\alpha\,.\,b^n & \text{if } j = d^n, \\ -\alpha\,.\,b^n & \text{if } j = s^n, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall j \in \mathcal{V} \tag{4c}$$

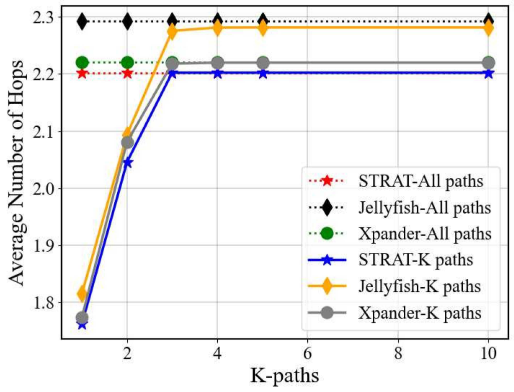
$$0 \leq b^n_{(j,i)} \leq b^n, \quad \forall (j,i) \in \mathcal{E}, \forall n \in \mathcal{N} \tag{4d}$$

$$0 \leq \alpha \leq 1. \tag{4e}$$

$\mathbf{B} = \{\mathbf{b}^1, \mathbf{b}^2, \ldots, \mathbf{b}^n, \ldots, \mathbf{b}^N\}$ is the matrix that represents the flow optimization variables, where $\mathbf{b}^n$ of size $\mathbb{R}^{2\times \mathrm{E}}$ is the vector of flow variables of commodity n. Variable $b^n_{(i,j)}$ represents the rate of the commodity n passing through edge (i,j). Constraint (4b) guarantees that the total traffic traversing through link (i,j) does not exceed the total link capacity. (4c) represents the flow conservation constraints. (4d) and (4e) represent the boundary constraints. $\mathcal{P}$ is a linear programming (LP) problem which was programmed using PuLP Python library [58] and solved using the CPLEX solver, [59]. However, problem $\mathcal{P}$ is not scalable by nature due to the large number of optimization variables included when All-paths are being used. Consequently, a path-based approach is applied, where only flow variables belonging to the first K-paths are incorporated in $\mathcal{P}$. This significantly reduces the total number of variables to be optimized. The K-paths based approach proves to preserve the optimality, increase the scalability, and decrease the time complexity of $\mathcal{P}$. To clarify, let $\mathcal{E}^n$ be the set of possible edges that demand n can traverse through, where $E^n = |\mathcal{E}^n| << 2 \times E$. Accordingly, the number of adopted variables is much lower than that of $\mathcal{P}$.



(a) All-to-All throughput V.S. the number of paths.



(b) Average number of hops V.S. the number of paths.

Figure 2.6: All-to-All throughput and average number of hops VS. K, with $V = 64$ and $d = 15$, in which the dashed lines correspond to the All-paths based solution.

2.4.2 Numerical Evaluation

In an attempt to show that the K-paths based optimization problem, with fewer optimization variables, guarantees a near-optimal solution, throughput was evaluated while varying the number of utilized paths assuming all-to-all traffic matrix (see Fig. 2.6). In all the conducted experiments, throughput has been normalized relative to the upper-bound predefined in [42] according to the following formula:

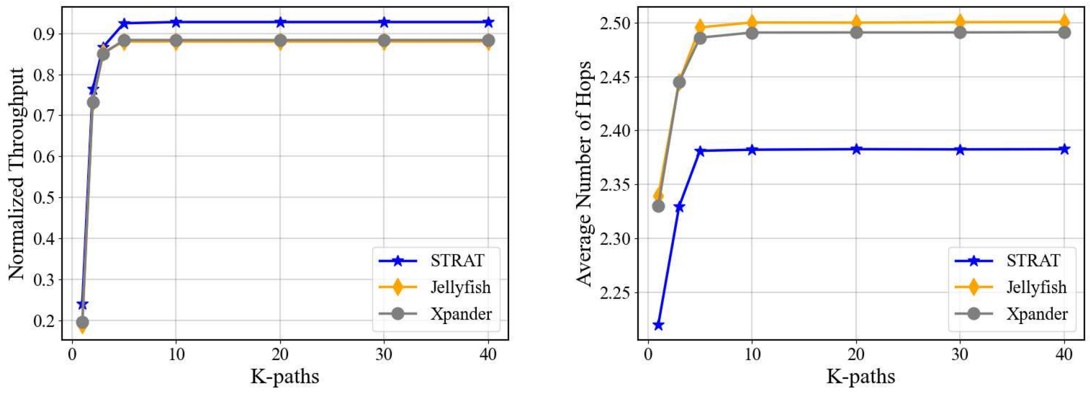
$$T^*_{(V,d,|\mathcal{F}|)} = \frac{V \times d}{|\mathcal{F}| \times \overline{L}^*_{sp}} \tag{5}$$

This upper-bound proved to reflect the throughput of d-regular graphs under a random permutation traffic matrix. Moreover, for $d \geq 13$, Equation (5) approximates the throughput upper-bound under all-to-all traffic [42].

For DC networks consisting of 64 switches ($V = 64$), Fig. 2.6a shows that the normalized throughput increases as more paths are employed, and it saturates at the optimal solution for all topologies (STRAT, Xpander, and Jellyfish) when K is large relative to V, (*i.e.,* $K = 5$). This is because a large K is enough to exploit the full diversity of topology while using MCF to distribute the traffic in the optimal way such that the optimal solution is almost attained.

On the other hand, Fig. 2.6b presents the average number of hops while varying K for the same

64 switches network. One can see that STRAT has an average number of hops lower than both Jellyfish and Xpander. This figure gives a similar intuition, as K increases, traffic will be spread out over more paths; therefore, the number of hops traversed by a flow will gradually increase such that the network throughput increases. It is worth noting that the trade-off between the number of hops and the maximum throughput plays a critical role in this behaviour. Remarkably, at low K (few optimization variables), the flows are constrained by a limited number of short paths that reduce both the average number of hops and the maximum throughput. On the contrary, increasing the number of paths (i.e., incorporating a larger number of optimization variables) provides a better degree of freedom which helps the network in achieving higher throughput.



(a) All-to-All throughput V.S. the number of paths. (b) Average number of hops V.S. the number of paths.

Figure 2.7: All-to-All throughput and average number of hops VS. K, with $V = 256$ and $d = 15$, in which these results are obtained using the K-paths based approach.

Fig. 2.7a. and Fig. 2.7b. present the same experiment for 256 switches network. However, in this case, it is observed that more K-paths are needed to reach the optimal solution. This behavior is due to the larger network size, which increases the number of paths between any pair of switches, leading to higher path diversity. In summary, these experiments show that considering a large enough K is sufficient to attain the optimal performance with lower complexity and computational time than when all paths are incorporated into the problem.

To further extend the analysis and highlight the scalability of the K-path approach, an additional experiment was conducted on a STRAT network with 512 nodes. Table 2.3 displays the All-to-All throughput, the relative margin of error between the throughput obtained by the algorithm and the

k	All-to-All throughput	Error (%)	Time
1	0.003484321	70.082483	0.11095
5	0.010461318	9.795759	5.87395
10	0.011461318	1.203100	26.34145
15	0.011461318	1.203100	32.20154

Table 2.3: Run time (in hours) of the K-paths based MCF under various values of K, for $V = 512$.

upper bound derived in Equation 5, and the time taken to solve the MCF problem for various values of K. The results show that as the value of K increases, the error percentage decreases. For instance, for $K = 1$, the algorithm achieves the highest error percentage at 70.08%, indicating a high margin of error between the obtained throughput and the theoretical upper bound due to the limited paths used by the algorithm. Interestingly, for $K = 10$ and $K = 15$, the same throughput value is achieved with a relatively small error rate of around 1.2% within a reasonable amount of time. This implies that for larger networks, where the All-path based approach would take very long time if it were to converge, a suitable trade-off can be found between K and relative error. In other words, one should select a suitable value for K such that the MCF problem can be solved within a reasonable time frame while still achieving a relatively small margin of error with respect to the upper bound.

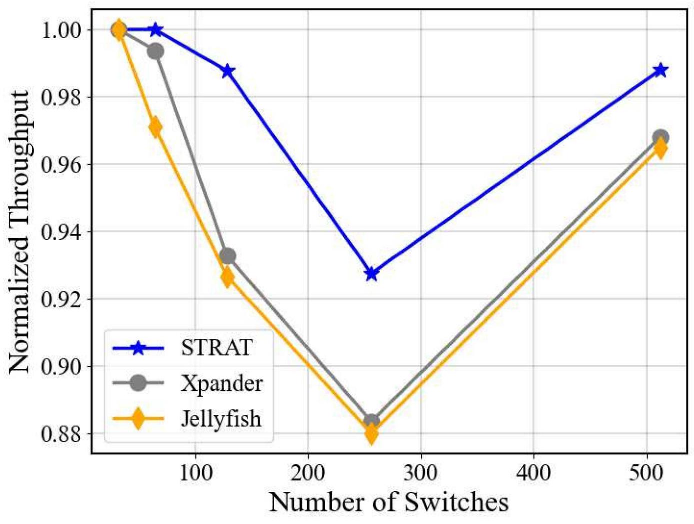
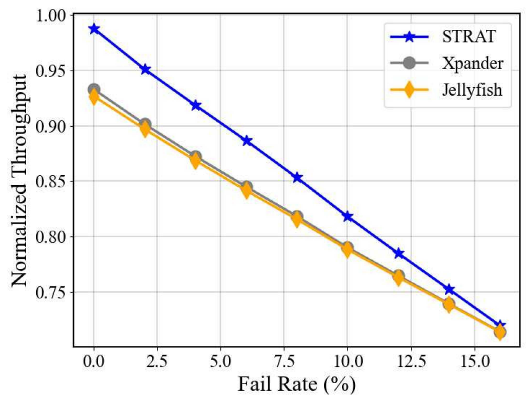
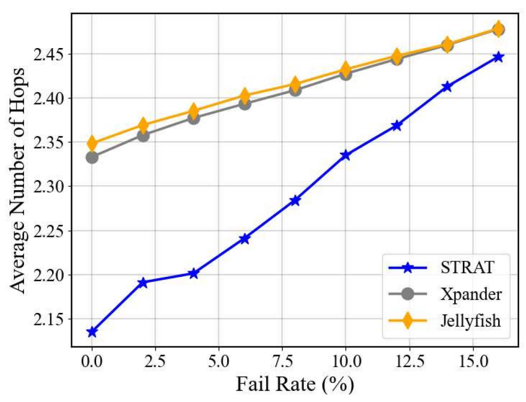


Figure 2.8: All-to-All throughput for different networks' sizes with fixed $d = 15$ using the K-paths solution.

Assuming the all-to-all traffic, Fig. 2.8 depicts the maximum throughput achieved by the various Expander DCs at different network sizes with a fixed degree ($d = 15$). Clearly, STRAT outperforms both Jellyfish and Xpander, which roughly attain identical throughput. Precisely, STRAT's is $6-7\%$ closer to the upper-bound. It is important to note that the reason behind the dip in the throughput (Fig. 2.8) is that, at $V = 256$ the upper bound loosens. This occurs because as V gets larger, $\overline{L}_{sp}$ jumps in steep increments, 20% increase in $\overline{L}_{sp}$ (Fig. 2.1a), which starts a new cycle for the bound (For more explanation, readers are referred to [42]).



(a) All-to-All throughput.



(b) Average number of hops.

Figure 2.9: Results for All-to-All throughput and average number of hops under link failure applying the K-paths solution with $V = 128$ and $d = 15$.

Fig. 2.9a presents the throughput's degradation under link failure. The fail rate along the x-axis signifies the percentage of the failed links from the total number of links present in the network. These links are intentionally chosen to be those with higher betweenness centrality to observe the impact they induce. Once the link failure percentage bypasses a certain threshold of $X\%$, the normalized throughput is recomputed. Under such failures, STRAT seems to degrade more gracefully than the other Expanders, which show very similar behavior. At the fail rate of 16%, the three networks converge because removing a large number of links from the network severely impacts the structural properties of networks, including $\overline{L}_{sp}$, $\Delta\lambda$, λ_2, and so forth. In other words, at that point, the topology will contain far less links compared to the initial number of links, resulting in many bottlenecks.

Similarly, Fig. 2.9b displays how STRAT still possesses fewer hops even when these failures are imposed. These observations align with the robustness results discussed in section 2.3. For clarity, introducing failure into the network impacts the graph's structural properties, resulting in a deteriorating throughput performance, which is proportional to the failure rate. Finally, these results indicate that STRAT allows for lower end-to-end latency despite the failures, which will be validated through Mininet emulator in the sequel.

2.5 Simulation Results

The deployment of a network in a real-world environment needs long-term planning, and the network's performance may vary due to the diversity and complexity of the protocols and algorithms to be deployed. Therefore, this work sets up a simulation framework to validate the results presented in previous sections, indicating that STRAT performs better than Jellyfish and Xpander. We opted to use Xpander and STRAT as our benchmark in section 2.5.2, having verified that Xpander and Jellyfish yield almost identical performance, thus eliminating any potential impact from the randomness state of Jellyfish and ensuring accurate performance measurements.

2.5.1 Mininet Experiments

Mininet emulator [30] along with the POX SDN controller [60] was used to carry out an HTTP file transferring experiment. As a control and data plane configuration, the OpenFlow protocol [61] was used with KSP as the routing algorithm[34].[2]

Taking into consideration Mininet's scalability limitations, and to guarantee that the generated results are trustworthy, all simulated networks are configured as follows: (1) The simulated topologies consist of 64 switches, (2) All links interconnecting the switches are of capacity 1 Mbps, while links between end hosts and ToR switches have the highest bandwidth (bounded by the CPU) to avoid server bottlenecks. The result for each simulation setting is averaged over 10 independent runs.

The experiment was run under various traffic workloads including random shuffle, random permutation, one-to-many, and skewed-random-shuffle. Each server hosts a multi-threaded HTTP

[2]The simulation was performed on an Ubuntu machine with 64GB of RAM, 12th Gen Intel(R) Core(TM) i9-12900.

Client-Server that supports transfer of files over TCP channels. After establishing a TCP connection, the end hosts start sending and/or receiving 1 MB and/or 8 MB files depending on the traffic workload.

Workloads	64 servers			128 servers		
	STRAT	Xpander	Jellyfish	STRAT	Xpander	Jellyfish
random shuffle	9.802	10.0	10.027	11.195	12.061	12.108
random permutation	9.987	10.273	10.692	13.43	13.796	13.826
skewed-random-shuffle	50.067	52.068	52.171	54.88	58.031	58.681
one-to-many	92.125	94.493	94.809	91.723	93.187	93.258

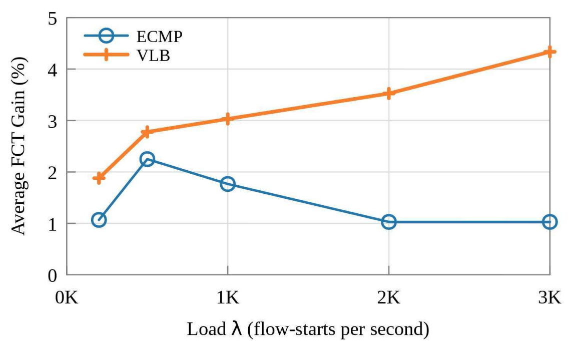
Table 2.4: Average FCT measured in seconds with $V = 64$ and $d = 15$ under two different topology settings.

Table 2.4 illustrates the average FCT, which is the sum of the completion time of all the flows circulating the network over the total number of flows therein ($|\mathcal{F}|$). Intuitively, a smaller average FCT implies a higher throughput and a lower end-to-end latency in the network. For the case of random shuffle and random permutation workloads, where all expander DCs exhibit near optimal performance, STRAT's average FCT is smaller than both Jellyfish's and Xpander's by $\lesssim 8\%$. Furthermore, as the workload gets skewer i.e., the traffic matrix approaches the worst-case scenario (all-to-all), STRAT still maintains its superior status over the two other Expanders by $\lesssim 7\%$. This observation aligns with the previous results where STRAT achieved higher throughput with a consistent network size of $V = 64$ (see Fig. 2.8) and continues to prove that STRAT operates at higher optimality than both Jellyfish and Xpander.

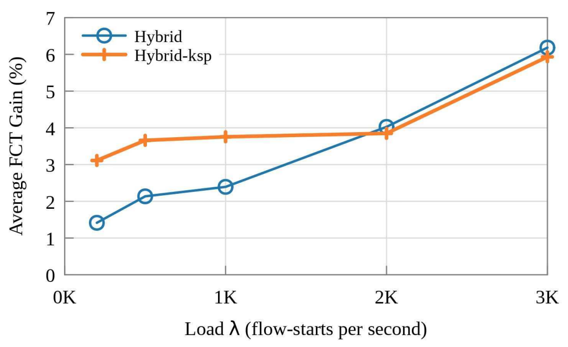
2.5.2 Netbench Experiments

To ensure consistent results in large-scale DCs, Netbench [31] simulator is being utilized to simulate Expander DCNs at scale. A set of exiting routing algorithms in the literature for DCs is employed such as ECMP[32], VLB[33], KSP[34], and HYB[22]. Moreover, DCTCP is used as the congestion control, since it shows better performance than TCP and is commonly deployed in todays' Intra-DCs, [62].

Fig. 2.10 illustrates the average FCT gain of STRAT over Xpander under all-to-all traffic conditions, with 93% of the 768 servers (6 servers per rack) being active. Flows are generated following a uniformly distributed form with a flowlet gap of 50 µs, which approximates Facebook's workload



(a) Average FCT gain of STRAT over Xpander using ECMP and VLB.



(b) Average FCT gain of STRAT over Xpander using HYB and HYB-ksp.

Figure 2.10: Average FCT gain of STRAT over Xpander under all-to-all traffic with $V = 128$ and 768 servers (6 servers per ToR).

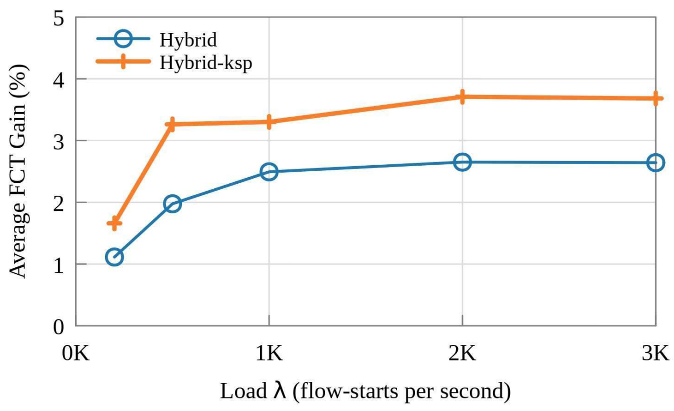
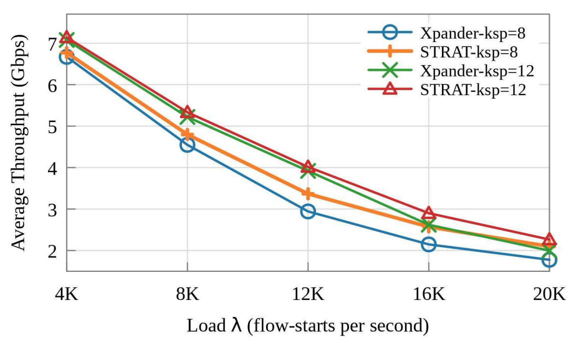


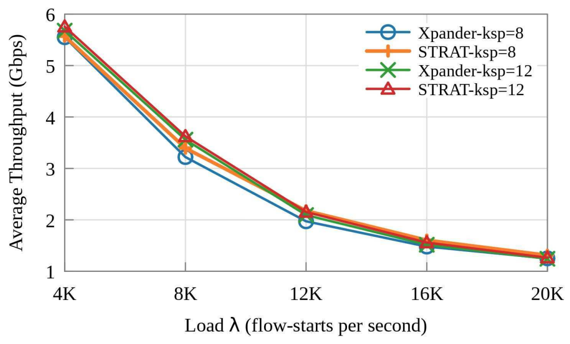
Figure 2.11: Average FCT gain of STRAT over Xpander under all- to-all traffic with $V = 256$ and 1024 servers (4 servers per ToR).

between some of the servers [63]. As shown in Fig. 2.10a, STRAT outperforms Xpander using two load balancing techniques, namely ECMP and VLB, despite the poor performance of ECMP on Expander DCs [1]. Additionally, Fig. 2.10b exhibits the performance gain of STRAT using a Hybrid routing approach, which is a combination of ECMP and VLB proposed in [22]. The Hybrid algorithm attempts to employ VLB routing when the traffic is routed between two adjacent racks, as ECMP will only use the direct connection to forward traffic where other paths are available which leads to throughput bottleneck. On the other hand, ECMP would perform better than VLB if traffic is being routed between two distant racks of servers because ECMP would use the shortest path along the way compared to VLB. Furthermore, Hybrid-ksp is a combination of ECMP, VLB, and

KSP. The Hybrid routing approach proves to better utilize the path diversity offered by Expanders, resulting in up to a 7% lower average FCT for STRAT compared to Xpander. Moreover, Fig. 2.11 shows that on larger-scale network with 256 switches and 1024 servers, STRAT still achieves up to 4% low average FCT as compared to Xpander.



(a) Average throughput achieved by STRAT and Xpander using KSP routing, while avoiding server bottleneck (ToR to ToR traffic).



(b) Average throughput achieved by STRAT and Xpander using KSP routing, with server bottleneck (server to server traffic).

Figure 2.12: Average throughput achieved by STRAT and Xpander under ProjectToR's rack-to-rack traffic with $V = 128$ and 1024 servers (8 servers per ToR).

Fig. 2.12 depicts the average throughput achieved by STRAT and Xpander as the number of flows generated per second increases. The traffic used in this experiment is heavily skewed, generated according to ProjectToR's rack-to-rack connection probabilities from a Microsoft DC [64]. KSP routing is used to assess the performance of Expander DCs, with different reasonable values of shortest paths (8 and 12) tested. To stress the topology further, ToR-to-ToR traffic is considered in Fig. 2.12a, to avoid server bottlenecks and focus more on the robustness of the topology itself. The results validate the findings in section 2.4, where STRAT achieves 7% to 8% higher throughput compared to Xpander, particularly under high traffic load. On the other side, the same topology is tested while server-to-server traffic is considered (Fig. 2.12b). Although server bottleneck degrades the performance of both STRAT and Xpander, STRAT still achieves better throughput.

2.6 Summary

This chapter serves as the cornerstone of our exploration, unveiling the STRAT topology and meticulously analyzing its efficiency and robustness. A robustness framework is proposed to model the DC topology as a network graph and study its structural characteristics and the failures consequences on the performance. Furthermore, a scalable throughput maximization optimization problem is formulated as a part of exploring the throughput, and then to practically validate these results, an extensive simulations are conducted on small and large scale networks. This chapter illuminates the distinctive advantages of STRAT, establishing its prowess as an efficient and resilient choice for next-generation DCs.

Chapter 3

Expander-Based DC Routing: A Programmable Data Plane Perspective

3.1 Overview and Motivation

The networking landscape has undergone a transformative shift towards programmability, notably in the data plane. Emerging from the foundations of Software-Defined Networking (SDN), the Programmable Data Plane (PDP), driven by technologies like the Barefoot Tofino chip, Broadcom Trident chip and the P4 programming language, offers unprecedented flexibility in packet processing.

3.1.1 Programmable Data Plane Applications

In 2023, Broadcom Inc.[1] unveiled its latest chip in the Trident family, the Trident 5-X12, fully-programmable using the open-source Network Programming Langauge (NPL). Notably, this advanced chip incorporates a neural network engine, NetGNT, introducing cutting-edge capabilities for next-generation telemetry and traffic management [65]. The programmability of this chip enables lightning-fast packet processing, reaching an impressive 16 Terabits per second. PDPs have

[1]Broadcom Inc. is an American multinational designer, developer, manufacturer, and global supplier of a wide range of semiconductor and infrastructure software products.

been employed in many domains, including but not limited to network security and network virtualization. [66] proposed a line-speed framework for federated learning. [67] implemented a layer 3 firewall in a programmable router using P4. [68] developed an embedding random forest algorihm in programmable switches to detect attacks in the network. [69], [70] proposed a framework for Network Function Virtualization within DCs, enabling the instantiation of Virtual Network Functions on a variety of platforms, including software switches and hardware devices such as ToR switches, SmartNICs, or FPGAs.

3.1.2 Routing Algorithms in Data Centers

While the PDPs initially targeted broad applications in DCs (see 1.1.2) and other areas (see 3.1.1), its impact on routing algorithms is a critical dimension worth exploring.

Equal-Cost Multi-Path Routing

Expander DCs, despite their inherent capabilities, face challenges with current routing algorithms, such as ECMP. While ECMP is an instrumental in enhancing load balancing and network redundancy, it exhibits limitations. It can result in uneven traffic distribution due to random flow hashing[71, 72, 73, 74], especially when there are few large flows. In essence, Expander DCs characterized by the diversity of path where not all path are equal cost paths, which make ECMP falls short on it.

State-Unaware Load Balancing Algorithms

Researchers have proposed state-unaware algorithms like Flare [75], LocalFlow [76], and DRILL [77]. Operating at each network hop, these algorithms use flowlet switching to route small bursts of packets independently over multiple paths. While offering simplicity, scalability, and hardware compatibility, they lack awareness of downstream congestion and path utilization, potentially leading to suboptimal load balancing. Moreover, their reliance on inter-packet gaps to define flowlets makes them susceptible to variations in network conditions, traffic patterns, and transport protocols.

Congestion-Aware Load Balancing

In response to the limitations of state-unaware algorithms, congestion-aware systems like CONGA [78] and HULA [74] emerged. CONGA utilizes custom switch ASICs to monitor congestion levels across paths and shares data with other switches via specialized packets. Each switch maintains a congestion feedback table for destination ToR routing decisions. While innovative, CONGA has significant drawbacks. It heavily consumes memory for path information storage, which can be costly and inefficient. Since it relies on remote feedback, it can cause latency-related inaccuracies. It also hinders innovation since it demands customized ASICs for implementation. HULA was introduced to overcome these limitations, leveraging programmable data planes. This method tracks congestion on the best path to a destination through neighboring switches while employing periodic probes to collect network utilization information. HULA was primarily tailored for a Fat-tree topology, which inherently supports only one ECMP group. This design demonstrates HULA's inability to fully exploit the diversity of paths present within Expander-based networks.

3.2 Packet Initiated Network Generation (PINGing) - Proposed Routing Algorithm for Expander Data Centers

Building upon the insights gained from the challenges and limitations of existing routing algorithms in Expander DCs, we introduce a novel routing algorithm that leverages the PDPs' capabilities.

Our proposed algorithm aims to address the shortcomings identified in current routing strategies. By harnessing the power of PDPs and capitalizing on the diverse paths provided by Expander DCs, our algorithm seeks to optimize load balancing, and enhance network efficiency.

In the following sections, we delve into the key design principles, functionalities, and expected benefits of our proposed routing algorithm. Through this contribution, we aim to provide a valuable addition to the evolving landscape of routing strategies tailored specifically for the unique characteristics of Expander DCs. In here, we consider the case where all network links can support bi-directional communications.

3.2.1 Constructing Shortest Path Routing Tables

During the initialization phase, we build the Routing Table (RT) by performing Packet Initiated Network Generation (PINGing) algorithm. Every network node starts with all its routes to potential destinations through all interfaces set to an initial value of infinity and zeros for the routes to itself. This particular node, which ultimately serves as the destination for all other nodes, then announces its presence to each nearby neighbor by transmitting a PING packet. The PING packet contains only the node's name and a metric of zero, which can be hop count, delay, etc. All PINGs are processed with a "no reply" policy, meaning they are never resent on the same interface they received on. Subsequent to this, each node that receives a PING does the following:

(1) It registers the incoming interface of the PING.

(2) It increases the delivered metric by the metric required to reach the sender node.

(3) Optionally, in the context of Option 1 and Option 2 mentioned below, the update process adjusts the stored metric by considering the relationship between the incremented metric and the currently stored metric.

(4) If the stored metric is updated, it re-sends the PING with the incremented metric; otherwise, it terminates the process.

The algorithm offers various options that balance the speed of convergence and the inclusion of Off Shortest Path (OSP) entries in the RT. These options are as follows:

(1) Option 1: The algorithm stores the increased metric for the destination and re-sends an updated PING to nearby neighbors if the new metric is smaller than the metrics associated with all other interfaces. It also resends if the new metric is the same as the existing one but arrived through an alternative interface. If neither conditions is met, the PING is terminated at that node, and nothing is updated. This method ensures the creation of RT entries for guaranteed Shortest Path (SP), with possible OSP entries established later by consulting the distance vectors (DVs) of adjacent neighboring nodes. Any remaining healthy interfaces can be used as a last resort for forwarding packets, rather than dropping them.

(2) Option 2: Similar to option 1, this approach stores the incremented metric for the destination and resends the PING with the updated metric to nearby neighbors. However, in this case, it only resends if the new metric is smaller than the metric associated with the receiving interface (not all interfaces as in option 1). If this condition is not met, the PING is terminated at that node, and nothing is updated. This results in tables with both guaranteed SP and OSP entries, eliminating the need to consult DVs of adjacent neighboring nodes. Remaining healthy interfaces can be used as a last resort for forwarding packets.

(3) An extension of option 2 involves providing reliable metric entries to all remaining Last Resort (LR) interfaces by enforcing loop-less PING propagation. This allows for the selective forwarding of packets through LR interfaces based on their now reliable metric values.

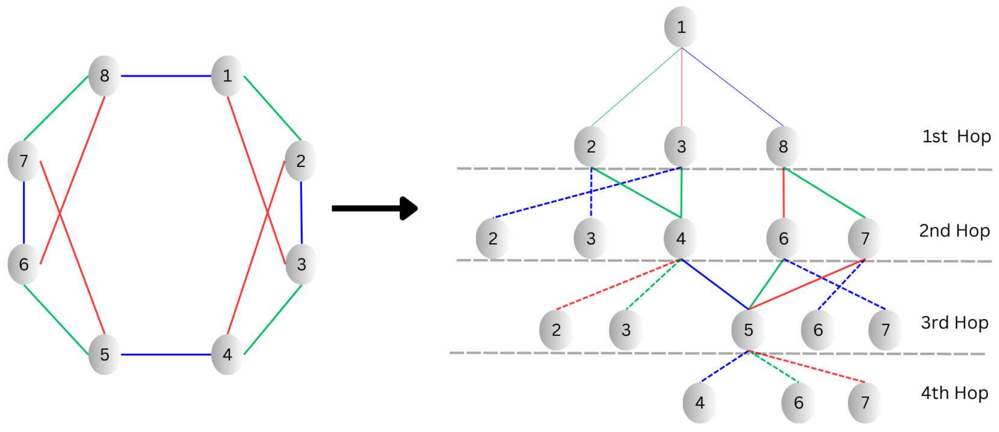


Figure 3.1: 8 nodes Expander network with PING propagation for node 1

To demonstrate the first option of our PINGing we will apply it on an 8-node Expander network featuring a diameter of 3 (Fig. 3.1), which we will then compare to the results of the second option on the same topology to show the similarities. In this network, each node has three interfaces, corresponding to three links connecting them to their adjacent neighboring nodes. The links are color-coded as follows: interface 1 is blue, interface 2 is green, and interface 3 is red. For this example let the metric be the distance-only metric, for simplicity. We will also be fully focusing on the PING initiated by Node 1.

D		S1	
1	0	0	0
2	∞	1	∞
3	∞	∞	1
4	∞	∞	∞
5	∞	∞	∞
6	∞	∞	∞
7	∞	∞	∞
8	1	∞	∞

Figure 3.2: First Hop Node 1 RT

D		S1	
1	0	0	0
2	∞	1	∞
3	∞	∞	1
4	∞	2	2
5	∞	∞	∞
6	2	∞	∞
7	2	∞	∞
8	1	∞	∞

Figure 3.3: Second Hop Node 1 RT

D		S1	
1	0	0	0
2	∞	1	∞
3	∞	∞	1
4	∞	2	2
5	3	3	3
6	2	∞	∞
7	2	∞	∞
8	1	∞	∞

Figure 3.4: Third Hop Node 1 SP RT

D		S1	
1	0	0	0
2	∞	1	2
3	∞	2	1
4	4	2	2
5	3	3	3
6	2	4	4
7	2	4	4
8	1	∞	∞

Figure 3.5: OSP RT Option 1

D		S1	
1	0	0	0
2	5	1	2
3	5	2	1
4	4	2	2
5	3	3	3
6	2	4	4
7	2	4	4
8	1	5	5

Figure 3.6: OSP RT Option 2

After the initial hop, the PING packet reaches the nearest adjacent neighboring nodes of its source node. This immediately generates true SP metrics for the PING source node as a future destination for the RTs of these neighboring nodes. In our case, The PING initiated from Node 1 proceeds to nodes 2, 3, and 8 via interfaces 2, 3, and 1, respectively, generating entries of value 1 in the initial row of the RT (Fig. 3.2). Following this, every PING-receiving node propagates the PING message, featuring updated metrics, through all their operational interfaces, excluding the one through which it originally received the message ("no reply"). After the second hop, both the RT updates to entries of value 2 and the termination of PING transmissions occur in accordance with the Option 1 algorithm. For instance, when the PING initiated from node 1 reaches node 4 via interfaces 2 and 3, and nodes 6 and 7 through interfaces 3 and 2, it updates their initial row entries in the RT to 2 (Fig. 3.3). Conversely, when the PING reaches nodes 2 and 3, it conveys metrics worse than their pre-existing values and is consequently terminated at nodes 2 and 3 marked with dashed-lines (Fig. 3.1).

Upon completing the third hop, which aligns with the network's diameter, all updates are ultimately set to values of 3. When the PING from node 1 reaches node 5 through all three interfaces, it updates the initial row of the node's RT to values of 3 (Fig. 3.4). However, the PING is terminated at nodes 2, 3, 6, and 7 for conveying metrics that are less favorable than the pre-existing values.

Lastly, upon reaching the fourth hop, all PING transmissions are terminated. This termination occurs because all the SP RTs are fully established within the same number of hops as the network's diameter, which is three. Consequently, all fourth hops are marked as terminal with dashed lines.

3.2.2 Constructing Off Shortest Paths RTs With Already Established Shortest Paths

We can acquire OSP RTs while SP RTs are already established. The entries within the tables obtained are categorized into two distinct types.

(1) The first type, Off Shortest Path (shortest path hops + 1) (OSP1), pertains to the interface connecting to neighbor S2 of the source node S. It is applicable when S2's SP metric to destination M(S2→D) does not exceed the metric from node S to destination M(S→D) plus the metric M(S2→S) to reach S2 from S. OSP1 entries in the OSP RT ensure that packets forwarded through this OSP1 interface can reach the destination using only the SP Forwarding Table (FT). The discovered OSP1 entry is the sum of M(S→S2) and M(S2→D).

(2) The second type, Off Shortest Path (shortest path hops + 2) (OSP2), relates to the interface connected to neighbor S4 from the source node S. It is relevant when S4's SP metric to destination M(S4→D) is the same as the metric from node S to destination M(S→D) plus the metric M(S4→S) to get from S4 to S. However, this only applies if there is another alternative SP path from neighbor S4 to destination D with the same metric M(S4→D). The OSP2 entry is essentially the sum of M(S→S4) and M(S4→D).

Fig. 3.5 shows the presence of SP RT entries (highlighted in green) alongside OSP1 and OSP2 entries (in blue and red, respectively). This highlights the noteworthy decrease in infinities within the OSP RT.

Just like the OSP RT of Option 1, the OSP RT of Option 2 (Fig. 3.6) was derived without the need to consult the DVs of neighboring nodes. The only trade-off is that it may involve a couple of extra hops, though the exact number depends on the network's specific topology.

Option 2's OSP RT differs from Option 1's OSP RT because it replaces infinite metric values with finite ones. All entries related to SP, OSP1, and OSP2 in OSP RTs are the result of intrinsically loopless PING propagation, making them perfectly accurate. The smallest possible loop under a "no reply" policy would add 3 hops to any loopless PING trajectory, leading to a higher metric than OSP2. However, LR entries in OSP RTs are susceptible to PINGs looping along the way. Although the OSP RTs of Option 1 and Option 2 express LR interface metrics differently their functionality in terms of both RTs and FTs remains completely identical.

3.2.3 PathPort to Prevent Network Loops

Both OSP types, OSP1 and OSP2, prevent immediate looping back to the node. However, potential forwarding loops can still arise further along the path. The industry standard Time to Live (TTL) mechanism is used to discard packets caught in loops by setting a pre-set limit on the number of allowed hops, a better approach is to prohibit them from ever entering the loop by ensuring they do not visit any node more than once on their way to the destination. It is achieved by providing the packet with a *PathPort* which records the path already taken to the destination and prohibits packet forwarding through any interface connecting to a previously visited node. The approach is more complex but improves overall performance in individual packet forwarding.

3.3 Forwarding Protocol

After having the RTs constructed by PINGing, we use the forwarding protocol depicted in the Algorithm 1 which outlines the process of forwarding packets within the data plane and how we utilize the full capabilities of Expander DC.

Upon receiving a packet at a node, the algorithm first checks if the destination server is hosted

Algorithm 1 Forwarding Algorithm

```
1:                                        ▷ Set up variables to hold the best port values
2:  PortOut ← 99                          ▷ Port on which the packet will be forwarded
3:  Qout ← 99                             ▷ Minimum Queue Occupancy
4:  HopOut ← 0                            ▷ Number of hops to the final destination for a given port
5:  mQc ← 8                               ▷ Maximum number of packets that can be hold by the network queue
6:  for each port index $pi$ do
7:                                        ▷ Check if port is connected to a previously visited node/switch
8:      if portIsNotConnectedToVisitedNode($pi$) then
9:          if getHops($pi$) $>$ HopOut then
10:                                       ▷ Advance to the next ECMP group
11:             if PortOut $==$ 99 then
12:                                       ▷ No valid port so far
13:                 Port ← getPort($pi$)
14:                 Q ← getQSize(Port)
15:                 if Q $<$ Qout and Q $\leq$ mQc then
16:                                       ▷ Valid Port
17:                     Qout ← Q; Hops ← getHops($pi$);
18:                     HopOut ← Hops; Port ← getPort($pi$);
19:                     PortOut ← Port
20:                 end if
21:                                       ▷ No valid port found yet
22:             end if
23:                                       ▷ Valid port found, however advanced to next ECMP group - Keep port as is
24:         else
25:                                       ▷ We remain within the same ECMP group
26:             Port ← getPort($pi$)
27:             Q ← getQSize(Port)
28:             if Q $<$ Qout and Q $\leq$ mQc then
29:                                       ▷ Update port values as we found a better port
30:                 Qout ← Q; Hops ← getHops($pi$);
31:                 HopOut ← Hops; Port ← getPort($pi$);
32:                 PortOut ← Port
33:             end if
34:                                       ▷ Current port is not an improvement - keep previous values
35:         end if
36:     end if
37: end for
38: if PortOut $<$ 99 then
39:     markSwitchAsVisited()
40:     SendPacketOnPortOut(PortOut)
41: else
42:     DropPacket()
43: end if
```

on that node. If it is, the packet is promptly delivered. However, if the node does not host the server, it evaluates the available interfaces, excluding the ones connected to nodes in the Pathport and drops the packet if no interface is available. From these interfaces, the algorithm selects those with the best metrics. It then further refines its choice by opting for the least loaded interface. Using this selected interface, the packet is forwarded to the next node in the network, and the current node is marked as "visited". This algorithm shows how we leverage the diversity of paths, by checking every ECMP group for the best port, meaning a port that is available, has the lowest number of hops and with the least loaded queue to forward the packet through.

3.4 P4 Virtual Environment: Architecture and Setup

To bring our routing algorithm to life, we embarked on establishing a virtualized P4-enabled network environment (see Fig. 3.7). The architecture comprises the following key components:

- Topology File: An adjacency matrix detailing the DC topology, outlining nodes and connectivity. This file is parsed by the DCN Python module.
- DCN: A Python module specifically designed to simulate DC topologies, providing a robust testing and experimentation platform. It uses P4-Utils APIs to create the DC topology specified in the Topology File, and it loads P4-code into these P4-enabled software switches.
- P4-Utils: A Python library built on top of Mininet, providing P4-enabled features for Mininet software switches by integrating BMV2 P4 software switch capabilities.
- Mininet: A Python emulator utilizing Linux namespaces to create virtual switches with virtual links.
- BMv2 (Behavioral Model version 2): A C++ framework empowering developers to implement P4-programmable architectures as software switches. In our case, we utilized the SimpleSwitch module, resembling the architecture of the Barefoot Tofino chip.
- P4-Code: The actual P4 programming code implementing our routing algorithm. It defines

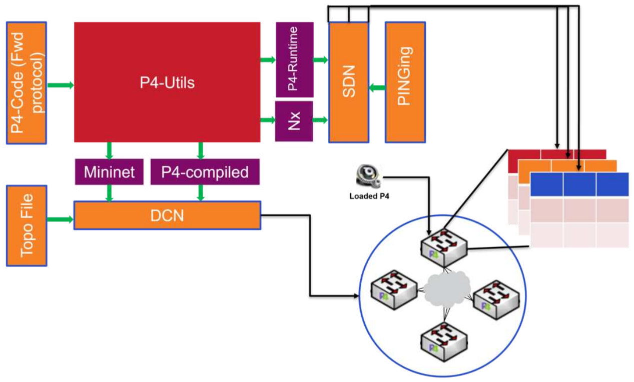


Figure 3.7: High level system architecture to implement PINGing algorithm in a P4 virtual environment.

packet processing behavior, including the parser, match-action pipeline, and deparse functionality.

- SDN (Software-Defined Networking): This centralized control and orchestration Python module house the PINGing algorithm. Leveraging Networkx, it manages the DC as a graph, facilitating the calculation of shortest paths across the topology. Subsequently, it utilizes P4-Runtime to install RTs to the BMv2 switch.

- Networkx (Nx): A Python library specializing in the creation, manipulation, and study of complex network structures. Networkx models the network created by P4-Utils, presenting a flexible approach to route calculations.

- P4-Runtime: Serving as the runtime environment for P4 programs, P4-Runtime facilitates communication between the control plane and P4-enabled devices. In this context, it installs RTs from the SDN to the BMV2 software switch.

- P4-Compiler: A compiler dedicated to translating high-level P4 programs into the low-level instructions executed by P4-enabled devices (BMV2).

By meticulously assembling these components, we created a robust and versatile P4-enabled network environment, facilitating the development, testing, and evaluation of our routing algorithm.

Moreover, it is worth mentioning that for the PathPort implementation, the fundamental modification involves augmenting the packet header at the source node with a dedicated field to encapsulate path information, which is then emitted at the destination node. This field serves as a repository to record the sequence of nodes that the packet traverses during its journey through the network. At each hop, as the packet progresses through intermediate points, the header is updated to include the id of the current node. Additionally, we set the TTL to a pre-defined value at the source node which allocates the maximum number of hops for the packet to traverse our network.

As our forwarding algorithm focuses on the least-loaded ports within the available ports with the best metrics, we utilized standard metadata, specifically emphasizing enqueue and dequeue parameters provided by P4. These metadata components serve a vital function in overseeing queue dynamics. The enqueue metadata enables tracking the influx of packets into the queues, offering visibility into the pace of packet arrivals. Conversely, the dequeue metadata facilitates monitoring the exit of packets from the queues, granting a comprehensive view of the processing and forwarding rates.

3.5 Simulation Results

3.5.1 P4-virtual Testebed Experiment

In conducting our experiments, we used a STRAT topology, comprising a network configuration that included 16 switches and 16 hosts, 1 host per switch. The chosen topology exhibited a diameter of 2, emphasizing the number of hops corresponding to the longest path's length among all shortest paths between each source-destination pair and an average shortest-path of 1.6.

We deployed both ECMP and PINGing on the topology. Employing different loads allowed us to gauge the performance and adaptability of the routing mechanisms under diverse conditions.

In Fig. 3.8, The logarithmic scale plot of drop ratios for the PINGing and ECMP routing algorithms reveals insightful trends in their performance across varying scenarios. Initially demonstrating low drop ratios, both algorithms exhibit effective packet handling when traffic is relatively low. However, as traffic increases, the PINGing algorithm consistently outperforms ECMP, maintaining a notably lower drop ratio. The logarithmic scale facilitates a clear visualization of the scaling behavior, emphasizing the exponential increase in drop ratio, particularly for ECMP. This is because ECMP doesn't use all the available paths provided by expander-based topologies. In contrast,

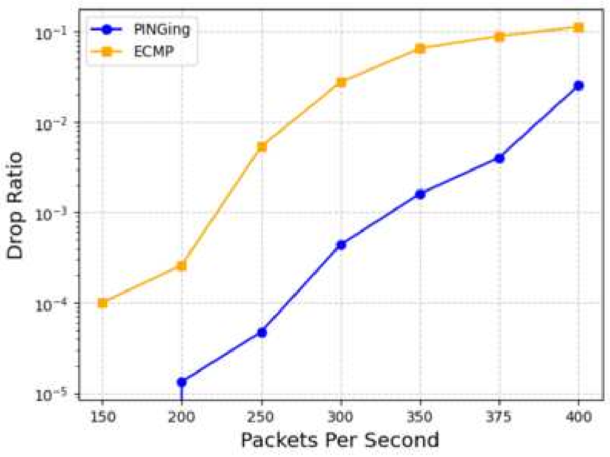


Figure 3.8: Comparison of ECMP VS. PINGing in terms of packets dropped ratio (in logarithmic scale).

PINGing achieves this by effectively using the diverse paths available in the network.

3.5.2 OMNET++ Experiments

To further demonstrate the potential of our routing algorithm and its scalability, we conducted tests using OMNET++ [79]. The experimentation involved evaluating the algorithm's performance on a STRAT with 256 switch and 1024 servers, with 4 hosts assigned to each switch.

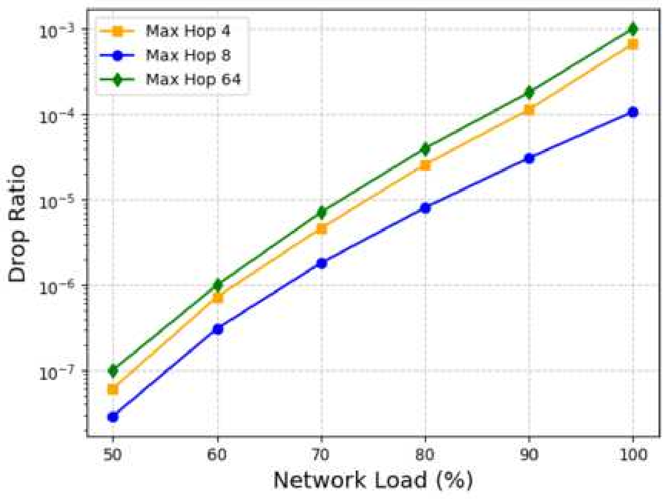


Figure 3.9: Drop ratio (in logarithmic scale) VS. network load using PINGing with different maxhop configurations for STRAT with $V = 256$, $d = 16$ and 1024 servers under uniform random all-to-all traffic matrix.

As illustrated in Figure 3.9, having a small max hop number (4) does not give the packet enough hops to reach its destination in the case of congestion. While having a high max hop number (64), the packet is just wondering around the network effectively creating artificial additional congestion which affects the overall network performance. The best approach is to give the packet few extra hops (8) additional to its longest shortest path which gives room to the packet to reach its destination without affecting other packets in the network.

Illustrated in Figure 3.10, the queue utilization across each port is depicted, showcasing its dynamic evolution in response to varying packet quantities. Notably, we can see that on average, the queue utilization maintains a consistent level despite the progressive increase in the number of packets. The majority of queues do not exceed an average depth of 2, demonstrating the algorithm's efficient utilization of diverse paths and its ability to route packets based on optimal metrics and the least-loaded queues. Notably, the orange line representing the identical average queue depth is the result of an equal distribution of traffic among these specific ports, meaning that load amongst these ports remains consistent.

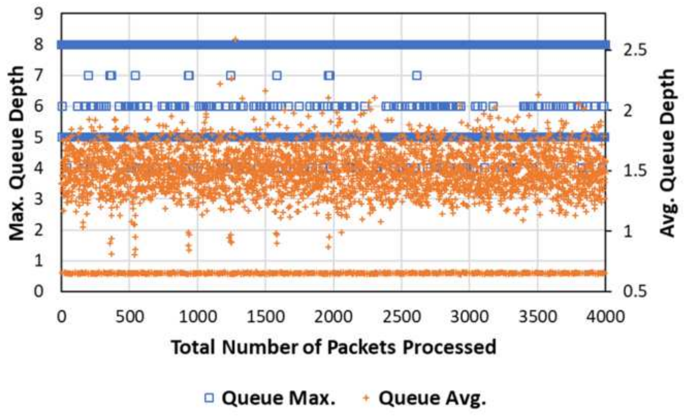


Figure 3.10: Maximum queue depth and average queue depth for STRAT with $V = 256$, $d = 16$ and 1024 servers under uniform random all-to-all traffic and 90% network load.

3.6 Summary

Building upon the insights gained in Chapter 2, we delve into the revolutionary realm of programmable data plane routing algorithms. In chapter 3, we introduce a new routing algorithm designed specifically for Expander DCs, emphasizing its operation within the data plane. The algorithm's efficiency in achieving better performance than ECMP and its compatibility with Expander architectures to harness the diversity of paths, showcasing a practical approach to DCN optimization.

Chapter 4

Conclusions and Future Works

4.1 Conclusion

In summary, this thesis presents a comprehensive exploration of network design and optimization for DCs, focusing on two key aspects. The first investigation centers around the STRAT topology, demonstrating its efficiency and robustness in comparison to state-of-the-art topologies such as Jellyfish and Xpander. The findings highlight STRAT's superior performance in terms of resilience to failures, overall network throughput improvement, and the interconnectedness of spectral gap, algebraic connectivity, and throughput. The study establishes a correlation between betweenness centrality and network resiliency, positioning STRAT as a competitive choice for rapidly expanding DCs. The demonstrated efficiency gains of STRAT not only enhance resource utilization but also underscore its adaptability to non-homogeneous networks and differently optimized DC sub-networks. The second facet of this thesis introduces a novel data plane-exclusive routing algorithm tailored for Expander DCs, achieving line-rate speed and unlocking the full potential of these architectures which is not harnessed by traditional routing algorithms that are not capable of fully using the diversity of paths provided by Expander DCs.

Collectively, the research presented in this thesis contributes to advancing the state of the art in DCN design. The findings from both investigations offer valuable insights into improving efficiency, resilience, and overall performance in the rapidly evolving landscape of DC technologies.

4.2 Future Works

As we conclude this study, several promising avenues for future research emerge, offering opportunities to expand upon the insights gained and address areas left unexplored. The following points outline potential directions for future work:

Reconfigurability in STRAT Topology

One intriguing avenue for exploration involves enhancing the STRAT architecture's adaptability. Introducing reconfigurability to STRAT could empower the network to dynamically allocate and redistribute network capacity, aiming for optimal utilization even under peak loads. Investigating the impact of reconfiguration mechanisms on network performance and resource utilization presents an exciting area for future study.

TCP Performance Analysis for PINGing Routing

The evaluation of STRAT's routing algorithm under Transmission Control Protocol (TCP) conditions offers valuable insights into real-world scenarios. Analyzing the out-of-order packets ratio and its influence on overall network performance provides a comprehensive understanding of how the proposed architecture interacts with widely used communication protocols. This analysis contributes to a more nuanced assessment of STRAT's applicability in diverse networking environments.

Physical Testbed Validation

To validate the robustness and effectiveness of the proposed routing algorithm, transitioning from simulation to a physical testbed becomes imperative. Implementing and testing the algorithm on physical network hardware allows for a more realistic evaluation of its performance, taking into account factors that might be challenging to capture in a simulated environment. This step would bridge the gap between theoretical simulations and practical deployment.

These proposed future works aim to build upon the foundations laid in this study, offering researchers and practitioners new avenues to explore, refine, and extend the presented concepts in the dynamic landscape of DCN research.

Bibliography

[1] A. Singla *et al.*, “Jellyfish: Networking Data Centers Randomly,” in *Proc. USENIX NSDI*, 2012, pp. 225–238.

[2] A. Valadarsky *et al.*, “Xpander: Towards Optimal-Performance Datacenters,” in *Proc. ACM CoNEXT*, 2016, pp. 205–219.

[3] M. Y. Frankel, V. Pelekhaty, and J. P. Mateosky, “Flat, Highly Connected Optical Network for Data Centers,” in *Proc. IEEE OFC*, 2019, pp. 1–3.

[4] Wadhwani, “Edge data center market analysis,” https://www.gminsights.com/industry-analysis/edge-data-center-market, Global Market Insights Inc., 2022.

[5] H. Ballani *et al.*, “Sirius: A Flat Datacenter Network With Nanosecond Optical Switching,” in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, 2020, pp. 782–797.

[6] C. E. Leiserson *et al.*, “There’s Plenty of Room at the Top: What Will Drive Computer Performance After Moore’s Law?” *Science*, vol. 368, no. 6495, p. eaam9744, 2020.

[7] V. Dukic *et al.*, “Beyond the Mega-Data Center: Networking Multi-Data Center Regions,” in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Architectures, Protocols Comput. Commun.*, 2020, pp. 765–781.

[8] P. Gill, N. Jain, and N. Nagappan, “Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications,” in *Proc. ACM SIGCOMM*, 2011, pp. 350–361.

[9] Q. Cheng *et al.*, “Recent Advances in Optical Technologies for Data Centers: a Review,” *Optica*, vol. 5, no. 11, pp. 1354–1370, 2018.

[10] S. Malla and K. Christensen, “A Survey on Power Management Techniques for Oversubscription of Multi-Tenant Data Centers,” *ACM Comput. Surv.*, vol. 52, no. 1, pp. 1–31, 2019.

[11] P. Namyar *et al.*, “A Throughput-Centric View of the Performance of Datacenter Topologies,” in *Proc. ACM SIGCOMM*, 2021, pp. 349–369.

[12] M. Al-Fares, A. Loukissas, and A. Vahdat, “A Scalable, Commodity Data Center Network Architecture,” *ACM SIGCOMM CCR*, vol. 38, no. 4, pp. 63–74, 2008.

[13] C. Guo *et al.*, “BCube: A High Performance, Server-Centric Network Architecture for Modular Data Centers,” in *Proc. ACM SIGCOMM*, 2009, pp. 63–74.

[14] C. Guo *et al.*, “Dcell: A Scalable and Fault-Tolerant Network Structure for Data Centers,” in *Proc. ACM SIGCOMM*, 2008, pp. 75–86.

[15] N. C. Wormald *et al.*, “Models of random regular graphs,” *London mathematical society lecture note series*, pp. 239–298, 1999.

[16] A. Marcus, D. A. Spielman, and N. Srivastava, “Interlacing Families I: Bipartite Ramanujan Graphs of All Degrees,” in *Proc. IEEE FOCS*, 2013, pp. 529–537.

[17] Y. Bilu and N. Linial, “Lifts, Discrepancy and Nearly Optimal Spectral Gap,” *Combinatorica*, vol. 26, no. 5, pp. 495–519, 2006.

[18] S. Mirjalili, “Genetic Algorithm,” in *Evolutionary algorithms and neural networks*. Springer, 2019, pp. 43–55.

[19] D. Bertsimas and J. Tsitsiklis, “Simulated Annealing,” *Stat. Sci.*, vol. 8, no. 1, pp. 10–15, 1993.

[20] W. Xia *et al.*, “A Survey on Data Center Networking (DCN): Infrastructure and Operations,” *IEEE Commun. Surv. Tutor*, vol. 19, no. 1, pp. 640–656, 2016.

[21] V. Harsh, S. A. Jyothi, and P. B. Godfrey, “Spineless Data Centers,” in *Proc. ACM SIGCOMM HotNets*, 2020, pp. 67–73.

[22] S. a. Kassing *et al.*, “Beyond Fat-Trees Without Antennae, Mirrors, and Disco-Balls,” in *Proc. ACM SIGCOMM*, 2017, pp. 281–294.

[23] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[24] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, “An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends,” *IEEE Access*, vol. 9, pp. 87 094–87 155, 2021.

[25] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, “A survey on data plane programming with p4: Fundamentals, advances, and applied research,” *Journal of Network and Computer Applications*, vol. 212, p. 103561, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804522002028

[26] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, “The programmable data plane: Abstractions, architectures, algorithms, and applications,” *ACM Comput. Surv.*, vol. 54, no. 4, may 2021. [Online]. Available: https://doi.org/10.1145/3447868

[27] Z. Xiong and N. Zilberman, “Do switches dream of machine learning? toward in-network classification,” in *Proceedings of the 18th ACM workshop on hot topics in networks*, 2019, pp. 25–33.

[28] X. Zhang, L. Cui, F. P. Tso, and W. Jia, “pheavy: Predicting heavy flows in the programmable data plane,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4353–4364, 2021.

[29] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, “Programmable switches for in-networking classification,” in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[30] B. Lantz, B. Heller, and N. McKeown, “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,” in *Proc. ACM SIGCOMM HotNets*, 2010, pp. 1–6.

[31] "ndal-eth/netbench: Packet simulator for data center network topologies, routing, and congestion control," https://github.com/ndal-eth/netbench.

[32] "RFC 2992 - Analysis of an Equal-Cost Multi-Path Algorithm," https://datatracker.ietf.org/doc/html/rfc2992.

[33] R. Zhang-Shen and N. McKeown, "Designing a predictable internet backbone with valiant load-balancing," in *Quality of Service–IWQoS 2005: 13th International Workshop, IWQoS 2005, Passau, Germany, June 21-23, 2005. Proceedings 13*. Springer, 2005, pp. 178–192.

[34] E. Q. Martins and M. Pascoal, "A New Implementation of Yen's Ranking Loopless Paths Algorithm," *4OR-Q J Oper Res.*, vol. 1, no. 2, pp. 121–133, 2003.

[35] "Introducing data center fabric, the next-generation facebook data center network - engineering at meta," https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/, (Accessed on 02/13/2023).

[36] V. Liu *et al.*, "F10: A fault-tolerant engineered network," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 399–412.

[37] A. Greenberg *et al.*, "Vl2: a scalable and flexible data center network," *Communications of the ACM*, vol. 54, no. 3, pp. 95–104, 2011.

[38] H. Wu *et al.*, "Mdcube: a high performance network structure for modular data center interconnection," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009, pp. 25–36.

[39] G. Wang *et al.*, "c-through: Part-time optics in data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 327–338.

[40] N. Farrington *et al.*, "Helios: a hybrid electrical/optical switch architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 339–350.

[41] M. Y. Teh *et al.*, “Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering,” *Journal of Optical Communications and Networking*, vol. 12, no. 4, pp. B44–B54, 2020.

[42] A. Singla, P. B. Godfrey, and A. Kolla, “High Throughput Data Center Topology Design,” in *Proc. USENIX NSDI*, 2014, pp. 29–41.

[43] C. Griner *et al.*, “Cerberus: The Power of Choices in Datacenter Topology Design-A Throughput Perspective,” *Proc. ACM POMACS*, vol. 5, no. 3, pp. 1–33, 2021.

[44] S. Jyothi *et al.*, “Measuring Throughput of Data Center Network Topologies,” in *Proc. ACM SIGMETRICS*, 2014, pp. 597–598.

[45] S. A. Kassing, “Static Yet Flexible: Expander Data Center Network Fabrics,” Master’s thesis, ETH-Zürich, 2017.

[46] K. Bilal *et al.*, “On the Characterization of the Structural Robustness of Data Center Networks,” *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 1–1, 2013.

[47] R. S. Couto, M. E. M. Campista, and L. H. M. Costa, “A Reliability Analysis of Datacenter Topologies,” in *Proc. IEEE GLOBECOM*, 2012, pp. 1890–1895.

[48] R. D. S. Couto *et al.*, “Reliability and Survivability Analysis of Data Center Network Topologies,” *J. Netw. Syst. Manag.*, vol. 24, no. 2, pp. 346–392, 2016.

[49] M. Manzano *et al.*, “On the connectivity of data center networks,” *IEEE Commun. Lett.*, vol. 17, no. 11, pp. 2172–2175, 2013.

[50] “Canadian Telco Rogers - DCD,” https://www.datacenterdynamics.com/en/news/.

[51] “Server Outage Causes Cruise Self-Driving Cars to Block Traffic - DCD,” https://www.datacenterdynamics.com/en/news/server-outage-causes-cruise-self-driving-cars-to-block-traffic/.

[52] “NetworkX — NetworkX Documentation,” https://networkx.org/.

[53] M. Manzano, E. Calle, and D. Harle, "Quantitative and Qualitative Network Robustness Analysis under Different Multiple Failure Scenarios," in *Proc. IEEE ICUMT*, 2011, pp. 1–7.

[54] A. H. Dekker and B. D. Colbert, "Network Robustness and Graph Topology," in *Proc. ACM ACSW*, 2004, pp. 359–368.

[55] S. Hoory, N. Linial, and A. Wigderson, "Expander Graphs and Their Applications," *Bull. Am. Math*, vol. 43, no. 4, pp. 439–561, 2006.

[56] N. Linial, E. London, and Y. Rabinovich, "The Geometry of Graphs and Some of Its Algorithmic Applications," *Combinatorica*, vol. 15, no. 2, pp. 215–245, 1995.

[57] W. Ellens and R. E. Kooij, "Graph Measures and Network Robustness," *arXiv preprint arXiv:1311.5064*, 2013.

[58] "Optimization with PuLP Documentation," https://coin-or.github.io/pulp/.

[59] "CPLEX Optimizer | IBM," https://www.ibm.com/analytics/cplex-optimizer.

[60] "Noxrepo/pox: The POX Network Software Platform," https://github.com/noxrepo/pox.

[61] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM*, vol. 38, no. 2, pp. 69–74, 2008.

[62] A. Mohammad *et al.*, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.

[63] R. Arjun *et al.*, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[64] G. Monia *et al.*, "Projector: Agile reconfigurable data center interconnect," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 216–229.

[65] B. Inc. (2023) Broadcom announces trident 5-x12, delivering next-generation programmable switching and routing. Accessed: December 2, 2023. [Online]. Available: https://www.broadcom.com/company/news/product-releases/61571

[66] Q. Qin, K. Poularakis, K. K. Leung, and L. Tassiulas, “Line-speed and scalable intrusion detection at the network edge via federated learning,” in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 352–360.

[67] P. Vörös and A. Kiss, “Security middleware programming using p4,” in *Human Aspects of Information Security, Privacy, and Trust: 4th International Conference, HAS 2016, Held as Part of HCI International 2016, Toronto, ON, Canada, July 17-22, 2016, Proceedings 4*. Springer, 2016, pp. 277–287.

[68] J.-H. Lee and K. Singh, “Switchtree: in-network computing and traffic analyses with random forests,” *Neural Computing and Applications*, pp. 1–12, 2020.

[69] T. Osiński, M. Kossakowski, M. Pawlik, J. Palimąka, M. Sala, and H. Tarasiuk, “Unleashing the performance of virtual bng by offloading data plane to a programmable asic,” in *Proceedings of the 3rd P4 Workshop in Europe*, 2020, pp. 54–55.

[70] T. Osiński, H. Tarasiuk, L. Rajewski, and E. Kowalczyk, “Dppx: A p4-based data plane programmability and exposure framework to enhance nfv services,” in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 296–300.

[71] Z. Zhang, H. Zheng, J. Hu, X. Yu, C. Qi, X. Shi, and G. Wang, “Hashing linearity enables relative path control in data centers,” in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Jul. 2021, pp. 855–862. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/zhang-zhehui

[72] K. Xi, Y. Liu, and H. J. Chao, “Enabling flow-based routing control in data center networks using probe and ecmp,” in *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2011, pp. 608–613.

[73] J.-L. Ye, C. Chen, and Y. Huang Chu, “A weighted ecmp load balancing scheme for data centers using p4 switches,” in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, 2018, pp. 1–4.

[74] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2890955.2890968

[75] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, p. 51–62, mar 2007. [Online]. Available: https://doi.org/10.1145/1232919.1232925

[76] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 151–162. [Online]. Available: https://doi.org/10.1145/2535372.2535397

[77] S. Ghorbani, B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Micro load balancing in data centers with drill," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIV. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2834050.2834107

[78] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 503–514. [Online]. Available: https://doi.org/10.1145/2619239.2626316

[79] A. Varga, "Omnet++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.