Starfish: A Topology-Routing Co-Design for Small-Scale Data Centers

Anchengcheng Zhou, *Princeton University*
Vipul Harsh, *Conviva*
Sangeetha Abdu Jyothi, *UC Irvine*
Maria Apostolaki, *Princeton University*
Brighten Godfrey, *UIUC*

Abstract

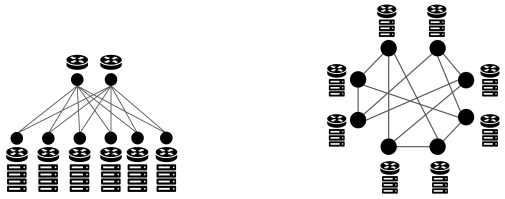
Most data centers today operate at scales much smaller than hyperscalers, *i.e.,* hosting only hundreds to a few thousand servers, yet their design has received disproportionately little attention in the literature. Leaf-spine is nearly ubiquitous in small-scale data centers, but presents a fundamental tradeoff: provisioning full non-blocking capacity is costly, while oversubscription reduces cost but under-serves bursty, skewed workloads. Recent expander-based designs could, in theory, offer better performance but face substantial deployment hurdles, including lacking a practical routing scheme.

We introduce Starfish, a topology-routing co-design tailored to small-scale data centers, that delivers high performance, fault tolerance, and ease of deployment. Starfish's topology, DRing, increases per-server egress capacity by spreading servers across all switches, reduces latency–even under failure scenarios–by exposing diverse near-shortest paths, and eases deployment by organizing switches into uniform blocks. Starfish's routing leverages structural properties of DRing and effectively adapts to traffic patterns using standard hardware and protocols. Notably, Starfish's routing also generalizes to expander-based topologies.

Evaluation shows that Starfish supports 56% more traffic on average than leaf-spine built with the same equipment across real-world traces. This demonstrates that small-scale data centers today can realistically achieve higher performance with their existing equipment. Starfish's routing also enables an expander-based design to support 52% more traffic on average than leaf-spine. More broadly, this work takes a first step towards a distinct design space for small-scale data centers.

1 Introduction

Most data centers are small-scale, consisting of tens to a few hundred racks and hundreds to a few thousand servers, and they power a large fraction of today's economy. This scale is prevalent in *(i)* enterprises' private data centers, which run about 45% of all enterprise IT workloads as of 2024 [81]; *(ii)* edge clouds, whose market size is projected to reach $29.6B in 2028 [59]; and *(iii)* Internet exchange points. Despite their prevalence and importance, small-scale data centers have received disproportionately little attention in the literature compared to hyperscale data centers operated by a handful of the largest service providers. Lacking further research, operators typically build leaf-spine networks [8], in which leaf switches (also top-of-rack (ToR) switches) and spine switches interconnect with a complete bipartite graph (Fig. 1a). To keep costs low, these data centers are often built with high oversubscription–sometimes higher than their large-scale counterparts [15]–and the leaf-to-spine bandwidth cannot sustain the traffic that servers could collectively generate, leading to performance degradation.



(a) Leaf-spine (indirect topology) (b) A direct topology

Figure 1: A direct topology can mask oversubscription. Leaf-spine (1a): 4 servers and 2 network links per rack (1/2 network link per server). Direct network (1b) built with the same hardware: 3 servers and 3 network links per rack (1 network link per server).

This paper investigates a simple question: Is there a topology design for small-scale data centers that can realistically deliver higher performance than leaf-spine while also offering strong fault tolerance and deployability?

At first glance, such a design may seem unlikely to exist. Known alternative topologies either suffer the same cost-oversubscription tradeoff as leaf-spine, or underperform at small scale, or face fault tolerance and deployment challenges. Leaf-spine, and more generally Clos networks [10], are indirect topologies [9]: some switches are only directly connected to other switches. They must either pay the cost of provisioning sufficient uplink capacity to stay non-blocking, which is expensive, or accept ToR oversubscription, which degrades perfor-

mance when servers transmit concurrently. Direct topologies avoid some of this tension by having all switches directly connected to servers: each ToR is attached to fewer servers and ToR switches can be used for both uplink and transit traffic. However, direct topology designs today fall short for other reasons. Dragonfly [46], common in high-performance computing, is designed for hundreds of thousands to millions of servers and is intentionally built with longer paths and long-distance global links. The long paths increase latency and bandwidth tax, leading to low performance, especially at small scales. Moreover, failures of global links cause disproportionate performance degradation [18]. Expander-based, low-diameter high-expansion topologies (*e.g.,* Jellyfish [78], Xpander [82], Slimfly [18]) promise high performance in theory but face potential deployment disadvantages *e.g.,* wiring complexity and difficulty expanding over time without excessive rewiring. Some topologies also enforce strict parameter constraints so that no feasible configuration exists at small scale [52]. More critically, beyond topology design, routing poses a major deployment barrier: there is no ready-to-deploy, high-performance routing solution for topologies other than multi-rooted trees like Clos. Topology defines the network's capabilities, which can only be realized under appropriate routing. Unfortunately, equal-cost multi-path (ECMP) uses only the shortest path, leaving two directly connected racks with only one single path to use in direct topologies. Other routing proposals either require special hardware (*e.g.,* optical [60] or programmable switches [12, 30, 45], flow control support [90]), demand end host modifications [16, 31, 38, 86], or fail to balance path length and path diversity and thus perform poorly under dynamic traffic patterns [19, 48, 50, 92, 95]. The lack of performant, deployable routing can deter small-scale data centers from adopting non-Clos-like topologies altogether. In fact, these observations may point to a trilemma: existing designs fail to simultaneously achieve performance, fault tolerance and deployability.

In this paper, we propose Starfish, a topology-routing co-design tailored to small-scale data centers that achieves high performance, fault tolerance and deployability. Starfish leverages the observation that at small scales, paths are naturally short, leaving some latency margin that allows one to use slightly longer paths to improve fault tolerance and deployability without noticeably affecting performance. Starfish's topology, DRing, distributes servers across all switches (*i.e.,* is direct), reducing rack-level oversubscription–to half of leaf-spine (Fig. 1b)–and alleviating congestion for bursty, skewed traffic. Moreover, DRing creates abundant equal-cost near-shortest paths, yielding strong performance under both skewed and uniform traffic and under random failures. Finally, DRing organizes switches into uniform blocks for ease of packaging, co-placement, wiring and expansion. Meanwhile, Starfish's routing leverages the correlation in DRing where a pair of racks that are more distant (*i.e.,* with longer shortest paths) also have more shortest paths available and vice versa, effectively balancing path length and path diversity. Starfish hedges against future traffic variations by spreading traffic across many paths and leaving link capacity headroom. Starfish's routing is deployable with off-the-shelf hardware and standard protocols. Notably, the routing generalizes beyond DRing to other topologies that satisfy the same correlation, providing a practical routing option even for expander-based topologies. For instance, Jellyfish [78] at small scale empirically satisfies this correlation too.

We evaluate Starfish in an htsim-based packet-level simulator [69] with real-world traffic traces from a university data center, a private enterprise data center [15] and three clusters at Meta [70]. Results[1] show that Starfish supports 56% more traffic on average across the five traces than a leaf-spine network constructed using the same equipment while keeping the tail latency at bay. Even Starfish with just traffic-oblivious weight assignment supports 33% more traffic than leaf-spine. Notably, Jellyfish, using Starfish's routing, supports 52% more traffic than leaf-spine too. Furthermore, Starfish is shown to be resilient to random link and switch failures and perform well at various sizes at small scale.

Overall, we see three key impacts of our work. First, to the best of our knowledge, this is the first work that demonstrates that small-scale data centers can achieve substantially higher performance than leaf-spine using existing enterprise networking gear. Second, our results highlight that *small-scale* topology design is an important research area distinct from *large-scale* design, where most recent work has focused. New designs delivering higher performance, fault tolerance and/or deployability may await discovery, and we hope this first work will lead to further efforts. Third, this work addresses a key deployment barrier for expander-based topologies by providing a high-performing, ready-to-deploy routing design.

2 Motivation

In this section, we highlight a key insight from the literature: performance, fault tolerance, and deployability seem to form a trilemma, as no known topology achieves all three simultaneously, and motivate the need for topology-routing co-design (§2.1). We then discuss why small-scale networks may offer an opportunity to sidestep this trilemma (§2.2).

2.1 Navigating the Design Trilemma

Topology designs form the foundation of data center interconnects and have long been a topic of research. A good topology should deliver *high performance, fault tolerance and deployability*. However, we observe that no existing work has so far managed to achieve all three, effectively creating a trilemma which we illustrate in Figure 2. We first motivate the three requirements before explaining how existing work falls short.

High performance. Data center traffic is diverse and challenging. Skewed, bursty traffic is increasingly common [15, 70] and diverse applications [14, 15, 32, 70] induce different bottlenecks

[1] Artifacts are available at https://github.com/AnnZhouCcc/Starfish.

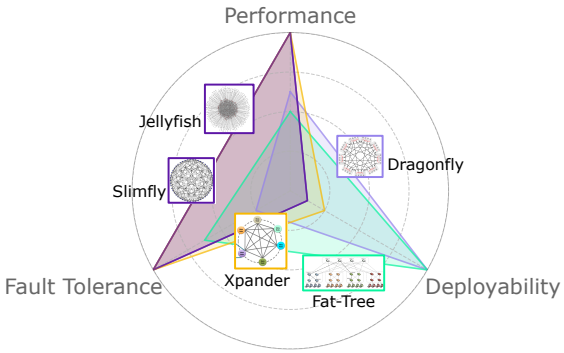


Figure 2: Existing work fails to achieve high performance, fault tolerance and deployability simultaneously. Instead, Starfish targets a specific point in the design space (*i.e.,* small-scale data centers) that sidesteps some of the tensions. Each colored triangle corresponds to a topology. Figure shown for illustration only (not to scale).

in the network [22, 46, 47]. The topology needs to support high throughput and low latency under any traffic. Short paths alleviate fabric congestion under uniform traffic and low oversubscription alleviates rack congestion under skewed traffic.

Fault tolerance. Failures are common in data centers [34]. Critically, the topology needs to retain its high performance even under random link and switch failures. A fault-tolerant topology typically exposes multiple similar-length paths for any pair of nodes such that taking out any link or switch will not cause disproportionate damage to performance.

Deployability. Construction, wiring and expansion complexity have been a barrier for adopting alternative topologies [62, 91] (in addition to the lack of practical routing schemes, which will be discussed later in the section). Hence, the topology needs to support flexible construction without excessive parameter constraints, and blocks of packaging and co-placement for ease of wiring and expansion [91].

To better understand how these requirements induce a design trilemma, let us consider some well-known topologies and investigate how they sacrifice at least one of the aforementioned desiderata. Clos [10, 55] and Clos-like topologies [6, 31, 57, 63, 75] have been widely adopted in data centers for deployability and fault tolerance. However, these tree-like topologies suffer from low performance, especially with skewed, bursty traffic (Fig. 14). This is because Clos is an indirect topology [9], where some switches are only directly connected to other switches, resulting in higher ToR oversubscription (Fig. 1). Nevertheless, direct topologies are not perfect either. For instance, Dragonfly [46], popular in high-performance computing [27], achieves better performance with skewed traffic but relies on "inter-group" global links that significantly impair its fault tolerance (Table 3 in [18]). Dragonfly also yields longer paths, increasing latency and network resources required to transmit per unit traffic (*i.e.,* the bandwidth tax) (Fig. 14). More recent topologies [18, 20, 51–53, 78, 82, 89] construct direct, expander-based topologies, which enable high performance and fault tolerance. However, they suffer from deployment disadvantages *e.g.,* wiring difficulties ([37], Fig. 9, 10 and Table 7 in [91]). §7.1 presents a more comprehensive literature review on topology designs that verifies our trilemma.

In practice, performance, fault tolerance, and deployability cannot be guaranteed by the topology alone, motivating the need for *topology-routing co-design*. Topology defines a polytope for what is possible, while routing determines which point on the polytope is actually realized. For some topologies, good routing is straightforward, *e.g.,* equal-cost multi-path (ECMP) works well in leaf-spine and other Clos networks because the topology's symmetry results in many equal-length shortest paths. But for other topologies, especially direct ones, routing is more challenging: each switch has servers attached, so each switch-to-switch link [A,B] results in a single shortest path between A's servers and B's servers. To utilize more than that one path, routing must exploit the topology's near-shortest alternative paths. Furthermore, routing must perform well for different traffic patterns–skewed or uniform–which have different preferences in path selection (more diverse, longer paths vs. fewer, shorter paths), and must effectively handle traffic variations over time [12, 60, 61, 74]. Finally, routing must rely only on standard hardware and protocols (*e.g.,* BGP, OSPF, VRF, TCP) for ease of deployment in small-scale data centers. In fact, we show in §7.2 that existing routing schemes fall short in addressing these challenges.

2.2 The Case of Small-Scale Topologies

While the tension among performance, fault tolerance and deployability is difficult to navigate in general, small-scale data centers present a unique opportunity of latency margin that allows us to sidestep some of the tensions. Specifically, paths are naturally shorter, and hence small-scale data centers could afford *a few* additional hops for *some* node pairs without incurring high average path lengths or degrading performance, creating a margin in path length that can be exploited for fault tolerance and deployability. This opportunity is of particular importance in light of the prevalence and critical role of small-scale data centers today [59, 81]. However, taking advantage of the path length margin is nontrivial. First, naively transplanting topologies designed for large-scale deployments to small scale does not suffice: some are simply infeasible at small scales due to parameter constraints (*e.g.,* PolarStar [52]), and others continue to face performance or operational challenges even after scaling down. Worse, some topologies create new challenges at small scales. For example, fat-tree retains shortest paths of up to 4 hops[2] at the small scale, consuming much of the margin that small-scale deployments could otherwise repurpose. Dragonfly faces similar challenges. Furthermore, it is not straightforward to resolve the trilemma

[2]The hop count includes only switch-to-switch hops and discounts server-to-switch hops.

with simple adaptations of existing topologies. Improvements along one dimension (*e.g.,* reducing oversubscription, shortening paths, or simplifying construction) often degrade another (*e.g.,* by increasing wiring complexity, reducing path diversity, or introducing inflexible structures). Therefore, small-scale topology design constitutes a distinct space of its own.

3 Overview

This section introduces Starfish's topology-routing co-design and highlights the insights behind it.

3.1 Starfish's Topology: DRing

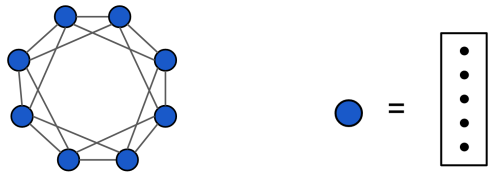
Topology construction. DRing is organized around a two-level structure: At the top level, we define a *supergraph* on m cyclically numbered vertices (supernodes), where vertex(i) is connected to vertices $(i+1)$ and $(i+2)$ (mod m) (Fig. 3a). At the lower level, each supernode contains n ToR switches (Fig. 3b), and every pair of ToR switches lying in adjacent supernodes has a direct link in the topology *i.e.,* each superlink connecting two supernodes is in fact n^2 ToR-to-ToR links.

Three characteristics allow DRing to achieve performance, fault tolerance and deployability simultaneously. First, servers are distributed across all switches, reducing ToR oversubscription to half of leaf-spine constructed with the same hardware (§4.1), hence alleviating congestion for bursty, skewed traffic. Furthermore, DRing connects every pair of adjacent supernodes with a complete bipartite graph, yielding many disjoint paths between the two supernodes. Meanwhile, the double-ring enhances distance-2 neighborhood connectivity, thus reducing path lengths and offering rich near-shortest routing alternatives. This symmetry and path diversity create opportunities for routing to improve **performance** for both skewed and uniform traffic, and stay **resilient** under random failures (§4.2). Finally, to address **deployability**, DRing organizes racks into natural units of co-placement and packaging, *i.e., blocks* [91]. This block-based design makes the cabling layout cleaner and allows links from a block to be bundled and neatly routed to another block (§4.3).

DRing specifically targets small to moderate deployments. Path lengths scale with the supergraph size, which in turn scales with the number of servers and switches. At large scale, long paths increase latency and bandwidth tax. But at small scale, paths remain short and DRing performs well (§4.4).

3.2 Starfish's Routing

To leverage these topology characteristics, we co-design Starfish's routing with the topology. Figure 4 illustrates Starfish's routing. As in prior work [17, 50], we split routing into two interrelated sub-problems: path selection and weight assignment. Starfish's routing exposes DRing's path diversity through a novel scheme, shortest-union-K (SU-K), that enables forwarding over all shortest paths and near-shortest



(a) DRing supergraph (b) DRing supernode

Figure 3: DRing construction: Each supernode in the supergraph (3a) consists of multiple ToRs (shown in 3b). Any pair of ToRs lying in neighboring supernodes are directly connected.

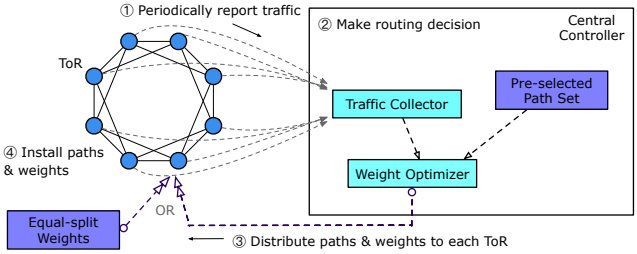


Figure 4: Workflow of Starfish's routing.

paths with a K-hop cap. A central controller periodically (at the granularity of tens of minutes) collects traffic demands from all ToRs, computes the weight assignment based on the SU-K path set, and distributes the weights back to the ToRs. Path weights default to an equal split in the event of controller failure or extremely high traffic variability.

Next we summarize how Starfish addresses the aforementioned routing challenges. First, SU-K leverages the correlation between the number of shortest paths and the shortest path length for a rack pair in DRing, and makes use of non-shortest paths *only when* the rack pair lacks sufficient shortest paths between them (§5.1), effectively controlling path length inflation while leveraging path diversity. Starfish also explicitly distributes traffic broadly when computing the weight assignment, creating link capacity headroom to accommodate future traffic variations (§5.2). Finally, SU-K is equivalent to shortest path routing on a transformed graph implemented with VRFs (Virtual Routing and Forwarding, a common feature on data center switches), ensuring loop-free operation. Starfish's routing is thus deployable using BGP on standard hardware (§5.1), with a basic coarse-grained SDN-style (or "intent-based") controller that is common today [1, 2, 7, 66, 75].

4 Topology Design

This section explains how DRing achieves high performance (§4.1, §4.2), fault tolerance (§4.2) and deployability (§4.3), and discusses impacts of scale and heterogeneous switches (§4.4).

4.1 Reduced Oversubscription

Oversubscription results in bottlenecks at the ToR, or *rack congestion*, when large volumes of traffic need to enter or leave a rack simultaneously, a common scenario under bursty or

skewed traffic. DRing mitigates rack congestion by adopting a *direct topology*, in which every switch is a ToR directly connected both to servers and to other switches (*i.e.,* network links). This design allows servers to be distributed across all switches instead of only a subset of them, as in the case of leaf-spine. This directness, illustrated in Figure 1, reduces rack oversubscription: each ToR has more network links per server because each ToR hosts fewer servers and can repurpose the freed ports as network links. The reduced oversubscription is especially beneficial for bursts where a rack has a lot of traffic to send in a short period of time and traffic is well-multiplexed at the network links (very few racks are bursting at any given point). The same benefit also applies to skewed traffic.

Next, we **formally quantify the oversubscription reduction in a direct topology** relative to leaf-spine. Consider a topology T and a direct topology $F(T)$ built with the same set of switches, servers and links. For every ToR, we define *Rack Oversubscription Ratio (OSR)* as the ratio of the server-facing downlink capacity to the network-facing uplink capacity (for simplicity, we assume uniform link capacity and uniform server distribution across all ToRs). We define $\alpha(T)$ as

$$\alpha(T) = \frac{OSR(F(T))}{OSR(T)}.$$

Intuitively, the ratio α captures the rack oversubcription reduction for direct networks, compared to the baseline topology.

We set leaf-spine as our baseline topology. Define $LeafSpine(x,y,a)$ with x leaf switches and y spine switches with a ports ($a \geq x > y$). Each spine is connected to x leafs; each leaf is connected to y spines and $a-y$ servers. We have $OSR(T = LeafSpine(x,y,a)) = \frac{a-y}{y}$. For the corresponding direct network $F(T)$,

$$OSR(F(T)) = \frac{\text{Server ports per switch in } F(T)}{a - \text{Server ports per switch in } F(T)}$$
$$= \frac{x(a-y)/(x+y)}{a-\big(x(a-y)/(x+y)\big)} = \frac{x(a-y)}{y(a+x)}$$
$$\text{Thus, } \alpha\big(T = LeafSpine(x,y,a)\big) = \frac{OSR(F(T))}{OSR(T)} = \frac{x}{a+x} \leq \frac{1}{2}.$$

α for leaf-spine is maximum when a takes the minimum value of x *i.e.,* when all ports are fully utilized, suggesting that the rack oversubscription of a direct network is *at least* reduced by half compared to leaf-spine. When $a = x$, the leaf-spine has $\alpha = \frac{1}{2}$, meaning that a direct network enables *up to* 2 times the throughput (or network capacity) of a leaf-spine when the bottleneck is at the ToRs[3]. Experiments in §6.2 confirm that a direct network can come close to having 2x throughput as the leaf-spine. Furthermore, observe that a can be expressed in terms of x *i.e.,* $a = bx$, and we have $\alpha = \frac{1}{b+1}$, meaning that α for a leaf-spine network is independent of x and y *i.e.,* the number of leaf and spine switches. Keeping the number of servers and number of switch ports constant, if a network has fewer spines and more leaves, the number of servers per rack is fewer but the aggregate uplink bandwidth at the ToRs is also smaller. These two factors cancel each other and hence, α remains constant. This implies that we could expect a direct network to *always* at least reduce rack oversubscription by half and achieve *up to* 2x more throughput than a leaf-spine built with the same hardware (the realized throughput gain depends on traffic pattern). §4.4 discusses how the performance remains strong with switches in a direct topology having different port counts.

[3]Note that $\alpha = \frac{1}{4}$ for a fat-tree (derived in Appendix 9.1), highlighting that fat trees are easier to outperform using a direct topology.

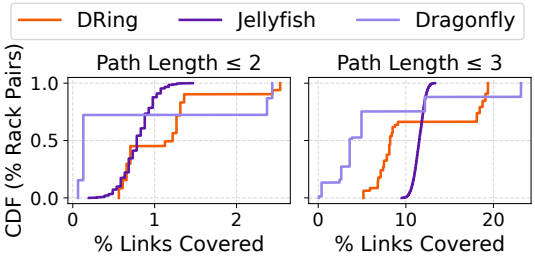


Figure 5: DRing (with shortest and near-shortest paths) makes use of more fabric capacity, providing better path diversity than Dragonfly and coming close to the high-expansion Jellyfish.

4.2 Path Diversity & Fault Tolerance

Path diversity. Shortest-path distances between rack pairs are critical for alleviating *fabric congestion*. In direct topologies, however, directly connected rack pairs have only one shortest path, so routing must exploit near-shortest alternatives to avoid hot spots and improve fault tolerance. Thus, a good direct topology must keep shortest paths short while also providing diverse near-shortest paths to utilize fabric capacity efficiently. DRing delivers this path diversity by incorporating complete-bipartite superlinks and a double-ring supergraph. Each superlink connects all ToRs in one supernode to all ToRs in the other, creating multiple disjoint one-hop paths, increasing path options between the two supernodes. The double-ring supergraph connects each supernode to four neighbors (rather than two in a simple ring), reducing inter-supernode distances and average path length. Figure 5 reports the percentage of links used by any rack pair with paths of length $\leq K, K \in \{2,3\}$, across three topologies built with the same equipment, a simplified approximation for path diversity, or how effectively shortest and near-shortest paths utilize the network capacity. DRing provides better path diversity than Dragonfly with shortest and near-shortest paths.

Failure tolerance. Failures are common in data centers. DRing is resilient to failures thanks to its ample near-shortest alternative paths and rack and link equivalence: racks (and their links) are structurally uniform, and thus removing any random link or switch will not cause a disproportionate impact on performance. Figure 6 reports the average shortest path length under random link failures across four topologies built

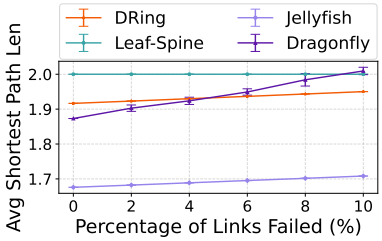


Figure 6: DRing's performance (approximated by average shortest path length) degrades slowly with random link failures, and exhibits minimal variance across trials due to link equivalence.

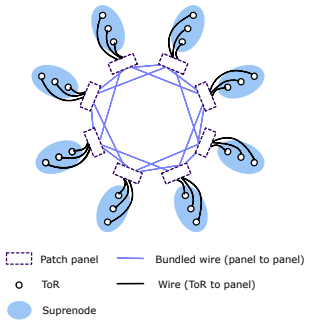
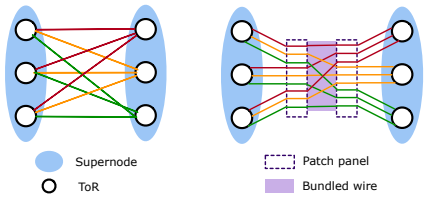


Figure 7: DRing's supergraph naturally defines a cabling scheme. Each supernode is assigned a patch panel and patch panels are connected with cable bundles identical to how supernodes are connected with superlinks.



(a) Logical superlink (b) Physical superlink

Figure 8: Two adjacent supernodes in DRing are connected by a superlink (7a). The cabling can be simplified by using two patch panels and a cable bundle (7b). For simplicity, we illustrate the case when $n=3$.

with the same equipment. DRing degrades slowly, indicated by the gentle slope, comparable to Jellyfish and markedly better than Dragonfly. Moreover, DRing exhibits minimal variance across trials due to link equivalence, indicated by the minimal error bars. In contrast, Dragonfly has large error bars, pointing to its heterogeneous link roles, where failures of global links are more damaging.

4.3 Wiring & Incremental Expansion

Wiring. Ease of wiring is critical for small-scale data centers to adopt a new topology, as it facilitates the deployment, maintenance and debugging. DRing incorporates local clustering and uniform blocks, and lands itself in a good position for a simple, straightforward cabling scheme (Fig. 7). First, local clustering refers to a way of organizing the topology where racks primarily connect to their local neighbors[4] with fewer long-distance links, which cuts down on the total cable length and makes layout more manageable. Moreover, supernodes form natural units of co-placement and packaging, or *blocks* as defined in [91]. Links from a block can be bundled and neatly routed to another block. More crucially, in DRing, all switches in the same supernode share identical connections to other supernodes. Therefore, for ToRs in two adjacent supernodes connected by a superlink, physically, we can use two patch panels to simplify the cabling (Fig. 8). All ToRs in each supernode are connected to their respective patch panel, and we just need a bundled wire to connect the two patch panels. In fact, as Figure 7 shows, the supergraph of the DRing is naturally also a cabling scheme. Each supernode has its own patch panel, to which all of its ToRs are connected using individual fibers, and we connect patch panels using cable bundles in the same way as superlinks connect the supernodes. Using cable bundles and patch panels significantly reduces deployment cost and operation complexity [91]. Furthermore, since DRing has a regular structure, the patch panels and cable bundles needed are largely uniform. Singular cable bundle types further reduce operation and cabling complexity [91].

[4]Neighbors are in the physical sense, not the graphical sense.

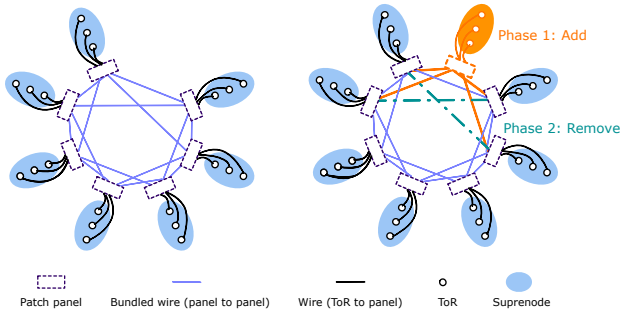


Figure 9: A DRing topology can be built incrementally by adding supernodes. A supernode can be added in a modular manner in two phases to minimize disruption.

Incremental expansion. The block-based design of DRing allows expansion by incrementally adding supernodes. Figure 9 illustrates how a supernode is added to a DRing topology originally with seven supernodes in a modular way. In phase one, we add the additional ToRs in the new supernode and wire the bundles accordingly (the elements in orange). In phase two, we remove the two green wire bundles. The two-phase process minimizes disruption to existing traffic and allows an initial DRing topology to be put into service first and be expanded over time.

4.4 Deployment Considerations

Impact of scale & high-radix switches. DRing performs particularly well for small-scale data centers, but its performance degrades when it has to increase the supergraph size to support more servers, which increases the average path lengths. When scaling a DRing topology by adding more racks to a supernode, the supergraph size remains unchanged, paths remain short and DRing remains high-capacity. However, it is not always feasible to increase the number of switches per supernode as we scale up DRing. Assume the maximum port

count on a switch is p, as restricted by hardware. If we have n switches per supernode, each switch in DRing connects to $4n$ other switches in neighboring supernodes, leaving only $p-4n$ ports for connecting servers ($n < \frac{p}{4}$). This means that we cannot keep scaling DRing by adding more switches per supernode as n is bounded by hardware constraints. At some point, we need to scale by adding supernodes, which increases the supergraph size and thus path lengths. As a result, DRing's performance deteriorates (see experiment results in §6.4). However, at small scale that is our focus, we can reap the benefit of DRing with a reasonably small supergraph size.

Notably, the emergence of high-radix switches (*e.g.,* NVIDIA Spectrum Ultra X800 [64], Broadcom Tomahawk 6 [21] with 512 ports) could make DRing performant at much larger scales. Specifically, higher port counts allow larger supernodes, which in turn keep the supergraph small and the average path length short even as the network size scales out.

Impact of heterogeneous switches. DRing can benefit from using some higher-radix switches, much like how leaf-spine can leverage higher-radix switches at the spine. One way is to distribute servers across all switches proportional to their port counts. This way, the corresponding DRing topology–and more generally any direct network–still reduces rack oversubscription by half (derivation in Appendix 9.2). Another way is to modify the vanilla DRing topology to make use of the additional ports. Consider a DRing topology in which each supernode contains x racks. Assume $2x$ higher-radix switches are available, each providing x more ports than the base switches. We can group the $2x$ higher-radix switches into two supernodes and use the additional ports to instantiate one more superlink between the two supernodes. As a result, racks in these two supernodes each gain x more network links.

5 Routing Design

Shortest-Union-K (SU-K) leverages structural properties of DRing and selects a set of paths for each rack pair to route traffic (§5.1). SU-K is deployable with off-the-shelf hardware and standard protocols (§5.1), and it already yields decent performance with simple equal-split path weights. To further extract the performance benefits out of DRing, we propose a weight assignment scheme that assigns path weights adaptively with traffic, based on the pre-selected SU-K-based path set (§5.2).

5.1 Path Selection: Shortest-Union-K

For direct topologies, the default shortest path scheme is valuable when traffic is very uniform. But using only shortest paths is insufficient in general. Directly connected rack pairs have only one shortest path, so routing must exploit near-shortest alternatives that the topology offers. However, using non-shortest paths necessarily increases average path lengths. This requires the path selection scheme to resort to non-shortest paths for a rack pair only when its shortest paths are limited.

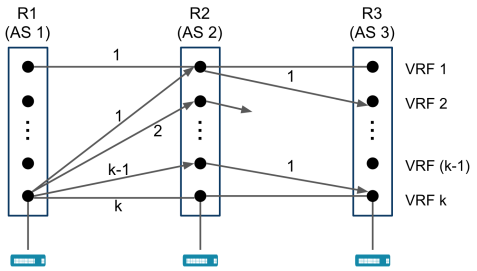


Figure 10: Shortest path routing in the VRF graph. Links are annotated with their costs. Links with arrows have different weights for forward and reverse links. Each router constitutes one AS for BGP. Not all connections are shown.

Theorem 1. *Consider a DRing topology in which each supernode contains n racks. For any pair of racks whose supernodes have p shortest paths of length h in the supergraph, there are* $p \cdot n^{h-1}$ *shortest paths of length h between the two racks.*

We identify a correlation in DRing between the number of shortest paths and the shortest path length, as stated in Theorem 1, meaning when $p \ll n$, a pair of racks with longer shortest paths also has more shortest paths between them and vice versa. Our path selection scheme, Shortest-Union-K (SU-K), leverages this correlation to identify when to use non-shortest paths for a pair of racks. Specifically, between two ToR switches R_1 and R_2, SU-K uses all paths that satisfy *either* of the following conditions: **(1)** The path is a shortest path between R_1 and R_2. **(2)** The length of the path is less than or equal to K. Effectively, SU-K employs non-shortest paths for pairs of racks that are close (*i.e.,* smaller h) and, hence, do not have enough shortest paths between them. Two racks that are distant from each other (*i.e.,* larger h) have sufficiently many shortest paths available between them to effectively load balance traffic and, hence, no extra paths are required.

SU-K is **implementable as a destination-based routing scheme with BGP and VRFs**, which are available in essentially all datacenter switches. We have prototyped SU-2 ($K=2$) in the GNS3 network emulator [3] on emulated Cisco 7200 routers. VRFs give us the power to virtualize a switch and partition the switch interfaces across the VRFs. We partition each router into K VRFs: (VRF 1, VRF 2, ..., VRF K). The host interfaces are assigned to VRF K. We use a unique AS number for each router and all VRFs on one router have that same AS number. For SU-K, K VRFs need to be configured at each router. For every directed physical connection from switch R1 to R2 in the topology (treating an undirected link as two directed links in opposite directions), we create the following virtual connections in the VRF graph:

1. (VRF K, R1) $\rightarrow$ (VRF i, R2) of cost i, for all i
2. (VRF (i-1), R1) $\rightarrow$ (VRF i, R2) of cost 1
3. (VRF 1, R1) $\rightarrow$ (VRF 1, R2) of cost 1

The cost of any other link not listed above is ∞. The costs can be set via path prepending in BGP. We simply use shortest

path routing in this VRF graph, which can be done via BGP (specifically eBGP). Figure 10 illustrates this design. This design is loop-free: There is no loop at the virtual-router level, since shortest path routing is used with the virtual routers. There is also no loop at the physical-router level, since all virtual routers from the same physical router are assigned the same AS and BGP does not admit any path that contains nodes belonging to the same AS. Theorem 2 states the correctness of this design, with the proof in Appendix 9.3. In order to reach a destination host h2 in rack R2 from a source host h1 in rack R1, a flow needs to reach (VRF k, R2) from (VRF k, R1). If the shortest path between R1 and R2 is $<$ k, then this design ensures that all paths of length $\leq K$ in the physical topology can be used since they all have cost K in the VRF graph.

Theorem 2. *For two routers in the topology R1 and R2 separated by distance L, the shortest path in VRF graph from (VRF K, R1) to (VRF K, R2) has length = max(L,K).*

In addition, we note that a simple change to BGP's path selection process would further simplify the above routing design and remove the need to configure VRFs. Specifically, current vendor implementations typically do not support multipath route selection across routes with different AS-path lengths. Supporting this behavior can be done easily by allowing the two commands "bgp ignore-as-path" and "bgp maximum-paths" to be configured simultaneously, a combination currently disallowed in common vendor implementations. We also note that the routing configurations at each router can be generated by a simple script to avoid errors.

5.2 Weight Assignment

Different traffic patterns induce different bottlenecks in the network, and thus work better with different paths. Uniform traffic tends to induce fabric congestion, where the total traffic volume is larger than the network capacity, and prefers to use shorter paths to minimize the bandwidth tax. By contrast, skewed traffic (*e.g.,* incast, bursts) tends to induce rack congestion, where a large volume of traffic needs to enter or leave the rack while most of the fabric remains underutilized, prefers to use more diverse paths–regardless of how long they are–to maximize traffic out of (or into) the rack. The routing scheme (*i.e.,* path selection together with weight assignment) needs to accommodate different spatial traffic patterns. Furthermore, in data centers, traffic changes over time [12, 60, 61, 74], and the routing scheme needs to accommodate temporal traffic variability.

Traffic-oblivious weight assignment. When path weights must be pre-installed on switches without any visibility into the traffic matrices, both equal-split and max-min-fair path weights (over SU-K-based path sets) are robust defaults: they avoid biasing towards either shorter or diverse paths and instead achieve a good balance between path length and path diversity. In our small-scale setting, however, many paths are two hops and thus largely disjoint for each rack pair, so max-min fairness provides little advantage over simple equal splitting. Therefore, Starfish defaults to equal-split path weights when traffic-oblivious routing is required (*e.g.,* with controller malfunction or extreme traffic volatility).

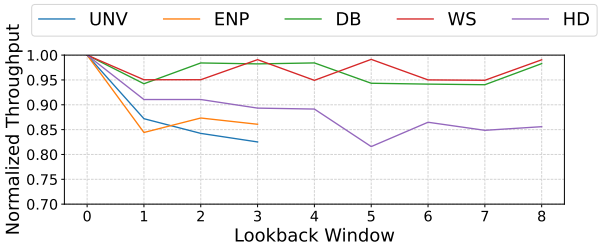


Figure 11: Real-world traces demonstrate temporal correlation, approximated by how close the throughput is to the optimal if a routing strategy is generated based on a past TM. The lookback window indicates the number of 30-minute intervals in the past. There is a clear correlation across TMs over time and the strength of the correlation varies for different traces.

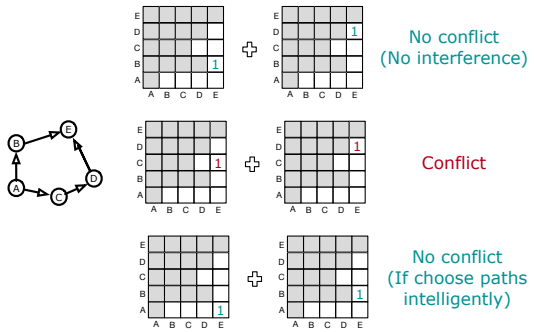


Figure 12: A toy example of topology with link capacity 1, illustrating the concept of non-conflicting TMs. The gray cells indicate unviable communication pairs. Cells with 1 indicate 1 unit of traffic and empty white cells indicate no traffic.

Traffic-aware weight assignment. While traffic is dynamic, some traffic characteristics persist over time [16, 58, 66]. Figure 11 confirms that there is indeed temporal correlation across traffic matrices (TMs), based on real-world traces (trace details in §6.1). This temporal correlation offers opportunities to optimize path weights based on historical TMs and further extract the performance benefits offered by DRing. Starfish computes path weights periodically at a coarse granularity (every 30 minutes in our evaluation). The controller formulates weight assignment as a multi-commodity flow linear program (LP), taking TM from the previous interval as input and producing weights for the current interval. We present the LP in Appendix §9.4 and highlight two key insights below.

First, a good set of pre-selected paths facilitates further path weight optimization. To illustrate, we first define a concept of *non-conflicting traffic matrices*: two non-conflicting TMs share some common optimal routing solution that does not compromise their individual optimality. Consider the topology and TMs in Figure 12. It is trivial that first-row TMs do not conflict while second-row TMs conflict with each other.

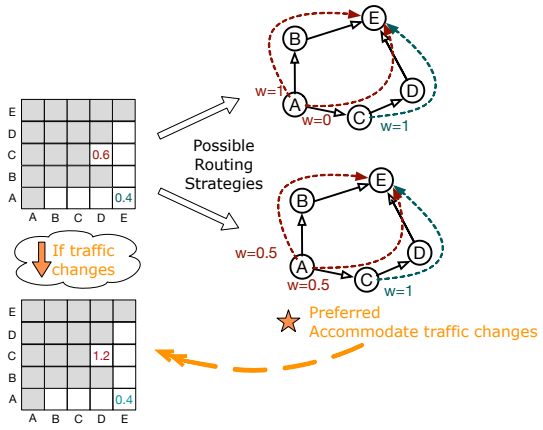


Figure 13: Spreading out traffic on more paths helps accommodate future traffic variation by leaving link capacity headroom and reducing the chances of future TMs conflicting.

Interestingly, for last-row TMs, they may seem conflicting at first glance but in fact, they do not conflict with each other, if we can choose paths more intelligently and allow traffic from A to E to traverse [A,C,D,E] instead of [A,B,E]. This example highlights the importance of a good path selection scheme, one where ample near-shortest paths like [A,C,D,E] are included and traffic from A to E can traverse [A,C,D,E] when needed, avoiding conflicts with other TMs. Moreover, we find that instead of handpicking only a few good paths in the path selection phase, keeping a larger set of paths (*e.g.,* the ones selected by SU-K) offers more flexibility for the central controller to shift traffic around and avoid conflicts across TMs.

Second, distributing traffic on more paths helps with accommodating traffic variations. The intuition is illustrated in Figure 13. If given only the top TM, both routing strategies are valid and equally good. However, if the traffic later changes to the bottom TM, the strategy below where traffic is more spread out on paths [A,B,E] and [A,C,D,E] still achieves full throughput, while the throughput of the strategy above drops to only 83%.

6 Evaluation

We evaluate Starfish in htsim simulator [69] and compare it with the status-quo (leaf-spine), related topologies (Jellyfish [78], Dragonfly [46]) and related routing schemes (shortest-path routing, WCMP [95], FatPaths [19], VLB [48, 92], SMORE [35, 50]). We show the following key results: **(1)** Starfish outperforms baselines on real-world traces. **(2)** Starfish is resilient to random link and ToR failures. Starfish's performance benefits are more pronounced **(3)** at smaller scales and **(4)** with larger oversubscription.

6.1 Setup

Traffic generation. We use real-world traffic traces from a university data center (UNV) and a private enterprise data center (ENP) from [15], and three production clusters, for database (DB), web servers (WS), and Hadoop servers (HD) respectively, from Meta's Altoona Data Center [70]. Table 1 in Appendix §9.5 presents a summary of statistics for how we parse the traces. We use a 30-minute interval, which provides sufficient samples for statistical significance for each trace. For each interval and each rack pair, we aggregate the traffic volume and sample from a flow-size distribution (described below) until their total matches the aggregate volume. To ensure fair comparison, flows are generated at the server level and the offered loads are normalized by the topology capacity to account for small differences in topology configurations. Flow sizes are picked from a standard Pareto distribution with mean 100KB and scale=1.05 to mimic irregular flow sizes in a typical data center [13]. Flow start times are chosen uniformly at random across the simulation window.

Topology baselines. We experiment with four topologies using 64-port switches. We try our best to construct all topologies using the same set of equipment to equalize hardware cost, including servers, switches and links, while omitting considerations *e.g.,* copper vs fiber cabling, power and cooling, *etc.* Some topologies impose parameter constraints, so exact equipment parity is not always feasible; in those cases, we find the closest possible topology configurations. To further ensure fair comparison, we ensure at most 3% difference in the number of servers and switches (following prior work *e.g.,* [18]). (1) Leaf-spine: 64 leaf switches, 16 spine switches, 3072 servers. We choose an oversubscription ratio of 3, matching an industry-recommended configuration [8]. Leaf-spine is routed using the standard ECMP, where traffic is split equally on all shortest paths. *Note that in leaf–spine, both traffic-oblivious and traffic-aware routing reduce to ECMP with equal-split path weights.* (2) DRing: 80 switches, 2988 servers (2.8% fewer than leaf-spine), 12 supernodes. (3) Jellyfish: 80 switches, 3072 servers. We use a regular random graph (a high-end expander [82]) for construction. Jellyfish is routed using Starfish's traffic-aware routing. We include Jellyfish as a representative of expander-based topologies. (4) Dragonfly: 78 switches (2.5% fewer), 3120 servers (1.6% more), 39 groups. $p = 40, a = 2, h = 19$. This configuration leaves 4 ports empty per switch. We intentionally adopt a favorable Dragonfly configuration, maximizing the number of ports used and allocating more inter-group links for alleviating known hot spots on inter-group links [18]. We approximate Universal Globally-Adaptive Load-Balanced (UGAL) [46] with a path selection scheme combining shortest paths and VLB paths with traffic-aware weight assignment.

Routing baselines. We experiment with seven routing schemes. (1) Starfish: Select paths using SU-2. Assign weights adaptively to traffic using a central controller, where we solve the linear program with a commercial solver Gurobi [4], setting Method [5] to 2 and Crossover [4] to 0. Following [50], we set path weight quantum to $\frac{1}{64}$. (2) Starfish (Oblivious): Select paths using SU-2. Assign equal weights to the selected paths. (3) Shortest Path routing (SRT): Select the shortest paths. Assign weights adaptively. (4) Weighted-Cost Multi-

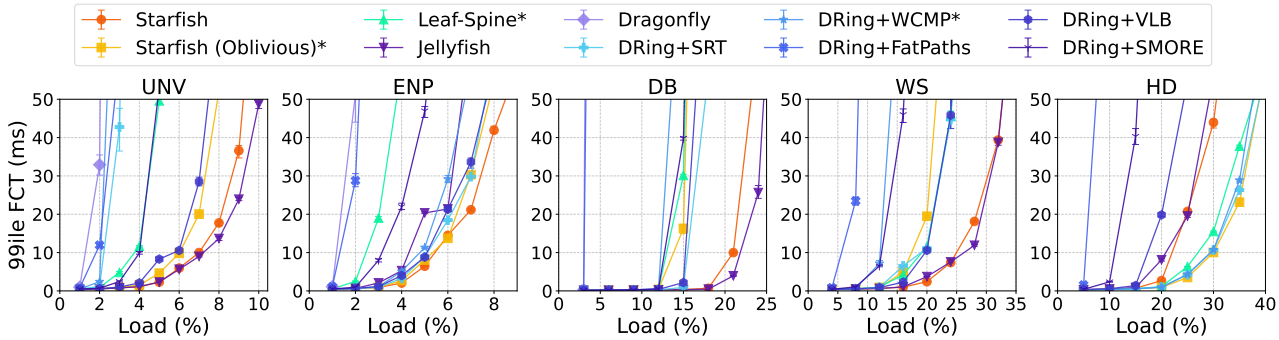


Figure 14: Starfish delivers lower tail latency at comparable load (and higher load at comparable tail latency), outperforming baselines in the network load v.s. tail latency tradeoff across real-world traffic traces. Schemes marked with * are traffic-oblivious, and the remaining schemes are traffic-aware. Curves further to the right are better.

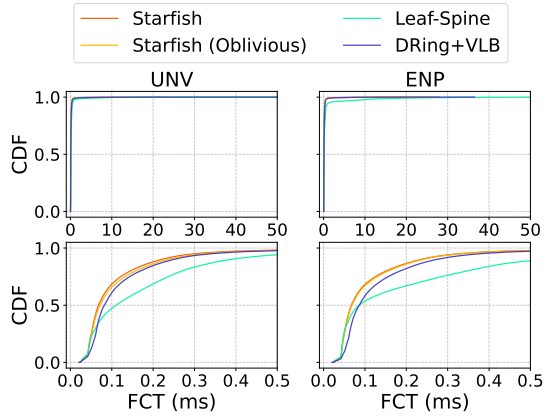


Figure 15: Starfish outperforms baselines across all percentiles.

Path (WCMP): Select the shortest paths. We approximate WCMP by assigning max-min fair weights to paths selected (see discussion in §7.2). (5) FatPaths: Select paths using its layered routing approach. Assign weights adaptively. We route per-flow instead of per-flowlet as suggested in the paper, to ensure fair comparison. (6) Valiant Load Balancing (VLB): Select a random first hop, and use the shortest path from the second hop onwards. Assign weights adaptively. (7) SMORE: Select paths according to racke's algorithm [67,68]; done in the VM provided by the SMORE paper. Assign weights adaptively.

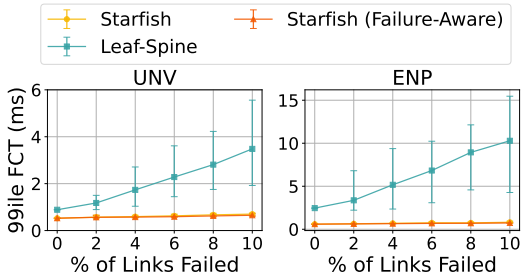
Simulation. We use the htsim-based packet-level simulator [69], configured with TCP and 10Gbps links of latency 2us. Packet size is 1500B. We simulate with source-based routing. The simulation runs for 144ms (in simulation time). We collect measurements after the warmup phase when the network condition has stabilized and after all flows finish.

6.2 High Performance on Real-world Traces

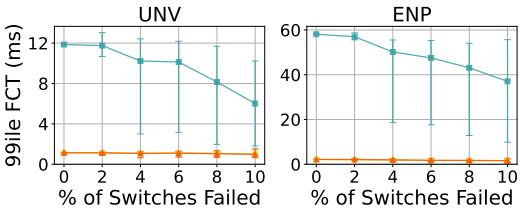
Starfish delivers lower tail latency at comparable load (and higher load at comparable tail latency), outperforming baselines in the network load vs. tail latency tradeoff across real-world traffic traces. Figure 14 reports the 99th percentile flow completion time (FCT) of Starfish and baselines against increasing offered network loads under five real-world traces across five random trials. We truncate the plots at 50ms tail latency; results beyond this point do not affect the conclusion. Across five traces, Starfish supports 56.38% more traffic on average than leaf-spine while ensuring the tail latency is below 50ms. Even when used with traffic-oblivious weight assignment, Starfish (Oblivious) supports 32.91% more traffic than leaf-spine. This implies that small-scale data centers can obtain significantly more network capacity simply by re-arranging the switching hardware they currently have. In addition, Jellyfish, when used with Starfish's traffic-adaptive routing, supports 51.63% more traffic on average than leaf-spine, highlighting the effectiveness and generality of Starfish's routing when applied to expander-based topologies.

Starfish outperforms all non-Jellyfish baselines on every trace except HD. HD exhibits high temporal variability and more uniform spatial distribution, thus favoring shortest-path-dominant schemes, *e.g.,* leaf-spine, DRing with ECMP or WCMP. Starfish's adaptive routing underperforms when traffic is highly variable and the temporal correlation is not strong enough for further weight optimizations. As a result, Starfish (Oblivious) supports 22.05% more traffic than Starfish under HD. Jellyfish, an expander-based, high-expansion topology, is expected to deliver strong performance when used with Starfish's routing scheme. Leaf-spine underperforms due to higher oversubscription and the resulting higher rack congestion. Dragonfly performs the worst, largely because paths are long and path choices are limited; on all traces except UNV and ENP, its tail latency exceeds 50 ms even at the lowest tested load. On DRing, Starfish 's routing outperforms all routing baselines. It supports 26.41% more traffic on average than DRing + VLB, the strongest routing baseline, while keeping the tail latency below 50ms. SRT and WCMP perform poorly because they restrict to shortest paths, which is too restrictive on a direct topology like DRing. SMORE and FatPaths use non-shortest paths but underperform because they hand-select only 4 and 3 paths per rack pair, respectively, which is insufficient.

In fact, Starfish performs well not only at the tail but across all percentiles, with traffic-aware or traffic-oblivious weights.



(a) Random link failures



(b) Random ToR failures

Figure 16: Starfish is resilient to random link and ToR failures, outperforming leaf-spine and approaching failure-aware.

Figure 19 in Appendix 9.6 shows that the performance gains in Starfish also hold for the median FCT. Moreover, Figure 15 reports FCTs at all percentiles of Starfish, Starfish (Oblivious), leaf-spine and DRing+VLB (the strongest routing baseline), under UNV and ENP for one representative run. The top-row figures show FCTs up to 50ms (consistent with Fig. 14) and the bottom-row figures are the zoomed-in views. At lower percentiles, Starfish, Starfish (Oblivious) and leaf-spine are comparable, while DRing+VLB is worse due to longer paths from the random first hops. Starting around the 30th percentile, leaf-spine degrades sharply, reflecting its higher rack congestion.

6.3 Failure Tolerance

Starfish is resilient to random link and ToR failures. Figure 16 reports the 99th percentile FCT of Starfish, leaf-spine, and Starfish (Failure-Aware) under UNV and ENP. Failure-Aware is an oracle baseline that optimizes routing with full visibility of the failed links/switches. In Figure 16a, we evaluate with increasing percentages of failed links with ten random trials. Following CONGA [12], we model a failed link as retaining only 50% of its bandwidth. We normalized the number of failed links by the topology capacity to account for minor configuration differences. Starfish degrades gracefully under random link failures, outperforming leaf-spine. Moreover, Starfish's performance approaches the failure-aware performance, and exhibits minimal variance across trials. With 10% failed links, the tail latency in DRing increases by only 33.42% and 37.11% for UNV and ENP, respectively, whereas the increases are more than 100% for leaf-spine due to the rapid loss of path diversity and its smaller network capacity. Moreover, at 10% link failure, Starfish's tail latency is only 7.61% and 11.31% higher than the failure-aware optimal. In Figure 16b, we evaluate with increasing percentages of failed ToRs with ten random trials (with double the injected load from Fig. 16a). When a ToR switch fails, all servers attached to that ToR stop sending and receiving traffic. For Starfish, all servers attached to a failed ToR also stop transiting traffic. As each ToR is attached to fewer servers in DRing than leaf-spine, DRing disconnects fewer servers than leaf-spine with the same percentage of ToR failures. Similarly, Starfish's performance degrades slowly under random ToR failures, where its tail latency is 88.19% and 96.24% lower for UNV and ENP respectively, than leaf-spine on average.

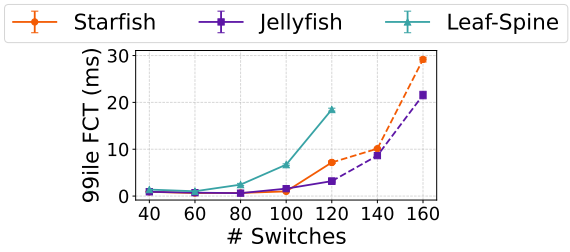


Figure 17: Starfish performs particularly well at small scale, but its performance degrades at larger scale when the supergraph size has to increase to accommodate more switches. Dashed lines indicate a modified topology setup (details in §6.4).

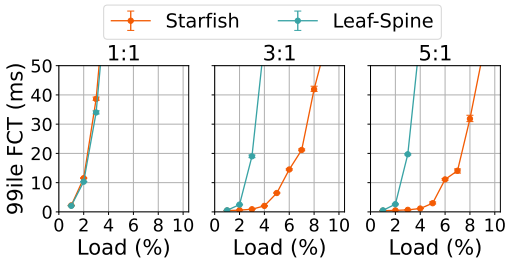


Figure 18: Starfish's performance gains are more pronounced with higher rack oversubscription *i.e.,* higher rack congestion.

6.4 Impact of Scale

Starfish performs particularly well at small scale, when the supergraph size is kept small. Figure 17 reports the 99th percentile FCT of Starfish, leaf-spine and Jellyfish under ENP with increasing numbers of switches. For the first five data points, we set the number of supernodes to 12, set oversubscription ratio to 3 and construct the closest possible DRing, leaf-spine and Jellyfish topologies. It simulates the scenario of scaling DRing by adding more switches to each supernode. Starfish excels at small scales, yielding an average tail latency 58.20% smaller than leaf-spine across the five sizes. Starfish's tail latency is relatively stable because we add switches without increasing the supergraph size (*i.e.,* no new supernode) and thus the average path lengths do not increase. However, as discussed in §4.4, we are restricted by port count on a switch and unable to always scale a DRing by adding more switches to supernodes. The last two data points in Figure 17 simulate the scenario of scaling DRing by adding supernodes. We use 96-port switches and set the number of switches per

supernode to 10. Leaf-spine stops earlier because it does not scale to those sizes under existing choice of configuration parameters. Starfish's performance deteriorates at larger scale, yielding an average tail latency 26.19% higher than Jellyfish.

6.5 Impact of Oversubscription

One key contributor to Starfish 's performance gains is the reduced rack oversubscription. Figure 18 reports the 99th percentile FCT of Starfish and leaf-spine when running ENP with different ToR oversubscription ratios across five random trials. The performance advantage of Starfish over leaf-spine grows with oversubscription. With no oversubscription (1:1), the performances of Starfish and leaf-spine are on par. With 3:1 and 5:1 oversubscription, Starfish supports roughly twice the traffic than leaf-spine when keeping tail latency below 50ms. This follows from Starfish's direct topology, which lowers rack oversubscription and hence rack congestion.

7 Related Work

7.1 Topology Literature

A recent workshop paper [37] investigated topology design for small-scale data centers. This paper differs in incorporating insights on the topology construction, wiring and deployment discussions, analysis of path diversity and fault tolerance, a routing design including traffic-aware weight assignment, and comprehensive evaluation.

Clos-like topologies. Fat tree [10, 55] is dominant in hyper-scale data centers, and Portland [63], VL2 [31], Jupiter [75], fbfabric [6], F10 [57] improve upon fat tree and build fat-tree-like topologies. These topologies are indirect.

Topologies from HPC. Butterfly [26] can be indirect and lacks path diversity. FlattenedButterfly [47] and HyperX [9] improve upon that but use many global, or long-distance, links between routers, causing it to be especially brittle when those links fail. Dragonfly [46], Dragonfly+ [73] and Galaxyfly [54] improves upon that but are still sensitive to congestion and failures on global links. Dragonfly+ is also indirect.

Expander-based topologies. Jellyfish [78] and Xpander [82] are complex with wiring. Bundlefly [53] and PolarStar [52] impose parameter constraints and have limitations in supporting topologies at small scales. SpectralFly [89] is difficult with wiring, while Slimfly [18, 20] and Bundlefly have no clear incremental expansion plan, as adding more routers requires rewiring a lot of links. Moreover, Slimfly imposes very strict conditions on its size (*i.e.,* $d \geq \Omega(\sqrt{n})$, d for port count, n for total number of switches) [82]. In addition, Polarfly [51] suffers from switch and link asymmetry and thus the links connecting quadric and non-quadric nodes are more brittle to failures.

Other topologies. Small-world data centers [72] is difficult with incremental expansion with its regular lattice underlying the topology, Random Shortcut Topologies [49] is difficult with wiring with its random links added, and FatClique [91] has long paths. **Server-centric topologies**, *e.g.,* DCell [34], BCube [33], Mdcube [87], rely on multi-port servers with a forwarding engine. **Reconfigurable topologies**, *e.g.,* Helios [28], c-through [85], Rotornet [61], Projector [29], Firefly [36], Flat-tree [88], Lightwave [56], Proteus [79] and [96], require special hardware to support optical switching. In addition, orthogonal to our work, REWIRE [23] and LEGUP [24] look into optimization frameworks that search for Clos network constructions. Condor [71] generates topology by specifying constraints instead of describing the topological structure. Minimal Rewiring [94] tackles expanding a Clos DC. [42] analyzes topology capacity theoretically.

7.2 Routing Literature

Traffic-oblivious schemes. ECMP is widely adopted in leaf-spine and Clos networks for its simplicity, but using only shortest paths is insufficient for direct topologies. WCMP [95] proposes weight assignment based on path capacity, but does not specify how to calculate path capacity when paths are non-disjoint. Its weight reduction algorithm and stripping alternative are orthogonal to our work. We compare to VLB [48,92] and Layered Routing in FatPaths [19] in our evaluation.

Traffic-aware schemes. Data center traffic engineering, flow scheduling and load balancing schemes *e.g.,* HULA [45], CONGA [12], Drill [30], Mahout [25], Baatdaat [80] require custom ASICs for switches, while Opera [60] requires optical switch circuits and DeTail [90] requires PFC support. Others *e.g.,* Presto [38], VL2 [31], microTE [16], CLOVE [44], MPTCP [86] require modifications to the host stack. We consider these approaches requiring special hardware and/or protocols not easily deployable. Moreover, CONGA [12] and Hedera [11] also assume multi-rooted tree topologies and are thus inapplicable. Some **WAN traffic engineering schemes** have different primary objectives. B4 and after [40] aims at improving fairness while Pretium [41], COIN [93], Cascara [76], Shoofly [77] aim at reducing cost. We complement schemes *e.g.,* DOTE [65] where they only focus on weight assignment and assume a given path set. Our weight assignment scheme is similar to those used in SWAN [39], SMORE [50]. We compare with SMORE in our evaluation. We complement **fine-grained load balancing** schemes *e.g.,* LAPS [84], FlowBender [43], LetFlow [83], as we answer which paths to use and in what ratio, while they answer at what granularity traffic is split.

8 Conclusion

This paper presents Starfish, a topology-routing co-design that achieves high performance, fault tolerance and deployability at small scale. We highlight the distinct design space at small scale and a practical routing for non-Clos-like topologies.

Acknowledgements

We sincerely thank our shepherd, Boris Pismenny, and the anonymous reviewers for their valuable feedback. This work was supported by the National Science Foundation (NSF) through Grant CNS-2442625.

References

[1] Arista cloudvision. https://www.arista.com/en/products/eos/eos-cloudvision.

[2] Cisco application centric infrastructure (cisco aci). https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/solution-overview-c22-741487.html.

[3] Gns3. https://gns3.com/.

[4] Gurobi optimization. https://www.gurobi.com/.

[5] Gurobi parameter reference. https://docs.gurobi.com/projects/optimizer/en/current/reference/parameters.html#method.

[6] Introducing data center fabric, the next-generation facebook data center network. https://code.facebook.com/posts/360346274145943.

[7] Juniper apstra. https://www.juniper.net/us/en/products/network-automation/apstra.html.

[8] Cloud networking scale out - arista. https://www.arista.com/assets/data/pdf/Whitepapers/Cloud_Networking__Scaling_Out_Data_Center_Networks.pdf, 2016.

[9] Jung Ho Ahn, Nathan Binkert, Al Davis, Moray McLaren, and Robert S Schreiber. Hyperx: topology, routing, and packaging of efficient large-scale networks. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, 2009.

[10] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[11] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92. San Jose, USA, 2010.

[12] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 503–514, 2014.

[13] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 253–266, 2012.

[14] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(1):1–29, 2020.

[15] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280, 2010.

[16] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the seventh conference on emerging networking experiments and technologies*, pages 1–12, 2011.

[17] Maciej Besta, Jens Domke, Marcel Schneider, Marek Konieczny, Salvatore Di Girolamo, Timo Schneider, Ankit Singla, and Torsten Hoefler. High-performance routing with multipathing and path diversity in ethernet and hpc networks. *IEEE Transactions on Parallel and Distributed Systems*, 32(4):943–959, 2020.

[18] Maciej Besta and Torsten Hoefler. Slim fly: A cost effective low-diameter network topology. In *SC'14: proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 348–359. IEEE, 2014.

[19] Maciej Besta, Marcel Schneider, Marek Konieczny, Karolina Cynk, Erik Henriksson, Salvatore Di Girolamo, Ankit Singla, and Torsten Hoefler. Fatpaths: Routing in supercomputers and data centers when shortest paths fall short. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–18. IEEE, 2020.

[20] Nils Blach, Maciej Besta, Daniele De Sensi, Jens Domke, Hussein Harake, Shigang Li, Patrick Iff, Marek Konieczny, Kartik Lakhotia, Ales Kubicek, et al. A {High-Performance} design, implementation, deployment, and evaluation of the slim fly network. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1025–1044, 2024.

[21] Broadcom. Bcm78910 series (tomahawk 6). https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78910-series. Accessed: 2026-02-09.

[22] Sudheer Chunduri, Taylor Groves, Peter Mendygral, Brian Austin, Jacob Balma, Krishna Kandalla, Kalyan Kumaran, Glenn Lockwood, Scott Parker, Steven

Warren, et al. Gpcnet: Designing a benchmark suite for inducing and measuring contention in hpc networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–33, 2019.

[23] Andrew R Curtis, Tommy Carpenter, Mustafa Elsheikh, Alejandro López-Ortiz, and Srinivasan Keshav. Rewire: An optimization-based framework for unstructured data center network design. In *2012 Proceedings IEEE INFOCOM*, pages 1116–1124. IEEE, 2012.

[24] Andrew R Curtis, Srinivasan Keshav, and Alejandro Lopez-Ortiz. Legup: Using heterogeneity to reduce the cost of data center network upgrades. In *Proceedings of the 6th International COnference*, pages 1–12, 2010.

[25] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*, pages 1629–1637. IEEE, 2011.

[26] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[27] Daniele De Sensi, Salvatore Di Girolamo, Kim H McMahon, Duncan Roweth, and Torsten Hoefler. An in-depth analysis of the slingshot interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2020.

[28] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 339–350, 2010.

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

[30] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 225–238, 2017.

[31] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[32] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. Cerberus: The power of choices in datacenter topology design-a throughput perspective. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–33, 2021.

[33] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 63–74, 2009.

[34] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 75–86, 2008.

[35] Mohammad Taghi Hajiaghayi, Robert Kleinberg, and Tom Leighton. Semi-oblivious routing: lower bounds. In *SODA*, volume 7, pages 929–938, 2007.

[36] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R Das, Jon P Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 319–330, 2014.

[37] Vipul Harsh, Sangeetha Abdu Jyothi, and P Brighten Godfrey. Spineless data centers. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 67–73, 2020.

[38] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 2015.

[39] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pages 15–26, 2013.

[40] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, et al. B4 and after: managing hierarchy, partitioning,

and asymmetry for availability and scale in google's software-defined wan. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 74–87, 2018.

[41] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 73–86, 2016.

[42] Sangeetha Abdu Jyothi, Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. Measuring and understanding throughput of network topologies. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 761–772. IEEE, 2016.

[43] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 149–160, 2014.

[44] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. Clove: Congestion-aware load balancing at the virtual edge. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, page 323–335, New York, NY, USA, 2017. Association for Computing Machinery.

[45] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, pages 1–12, 2016.

[46] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. *ACM SIGARCH Computer Architecture News*, 36(3):77–88, 2008.

[47] John Kim, William J Dally, and Dennis Abts. Flattened butterfly: a cost-efficient topology for high-radix networks. In *Proceedings of the 34th annual international symposium on Computer architecture*, pages 126–137, 2007.

[48] Murali Kodialam. Efficient and robust routing of highly variable traffic. *Proceedings of HotNets, Nov. 2004*, 2004.

[49] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, D Frank Hsu, and Henri Casanova. A case for random shortcut topologies for hpc interconnects. *ACM Sigarch Computer Architecture News*, 40(3):177–188, 2012.

[50] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 157–170, 2018.

[51] Kartik Lakhotia, Maciej Besta, Laura Monroe, Kelly Isham, Patrick Iff, Torsten Hoefler, and Fabrizio Petrini. Polarfly: a cost-effective and flexible low-diameter topology. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022.

[52] Kartik Lakhotia, Laura Monroe, Kelly Isham, Maciej Besta, Nils Blach, Torsten Hoefler, and Fabrizio Petrini. Polarstar: Expanding the horizon of diameter-3 networks. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 345–357, 2024.

[53] Fei Lei, Dezun Dong, Xiangke Liao, and José Duato. Bundlefly: a low-diameter topology for multicore fiber. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–11, 2020.

[54] Fei Lei, Dezun Dong, Xiangke Liao, Xing Su, and Cunlu Li. Galaxyfly: A novel family of flexible-radix low-diameter topologies for large-scales interconnection networks. In *Proceedings of the 2016 International Conference on Supercomputing*, pages 1–12, 2016.

[55] Charles E Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers*, 100(10):892–901, 2012.

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

[57] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A fault-tolerant engineered network. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 399–412, 2013.

[58] Ximeng Liu, Shizhen Zhao, Yong Cui, and Xinbing Wang. Figret: Fine-grained robustness-enhanced traffic engineering. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 117–135, 2024.

[59] Edge Data Center Market. Edge data center market size and share - comprehensive analysis. https://www.marketsandmarkets.com/ResearchInsight/size-

and-share-of-edge-data-center-market.asp, 2023.

[60] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1–18, 2020.

[61] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 267–280, 2017.

[62] Jeffrey C Mogul and John Wilkes. Physical deployability matters. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pages 9–17, 2023.

[63] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 39–50, 2009.

[64] NVIDIA. Spectrum-x networking platform. https://www.nvidia.com/en-us/networking/spectrumx/. Accessed: 2026-02-09.

[65] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. {DOTE}: Rethinking (predictive){WAN} traffic engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1557–1581, 2023.

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

[67] Harald Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52. IEEE, 2002.

[68] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 255–264, 2008.

[69] Costin Raiciu, Damon Wischik, and Mark Handley. Practical congestion control for multipath transport protocols. *University College London, London/United Kingdom, Tech. Rep*, 2009.

[70] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, 2015.

[71] Brandon Schlinker, Radhika Niranjan Mysore, Sean Smith, Jeffrey C Mogul, Amin Vahdat, Minlan Yu, Ethan Katz-Bassett, and Michael Rubin. Condor: Better topologies through declarative design. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 449–463, 2015.

[72] Ji-Yong Shin, Bernard Wong, and Emin Gün Sirer. Small-world datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–13, 2011.

[73] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi. Dragonfly+: Low cost topology for scaling datacenters. In *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pages 1–8. IEEE, 2017.

[74] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A network architecture for disaggregated racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 255–270, 2019.

[75] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM computer communication review*, 45(4):183–197, 2015.

[76] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. Cost-effective cloud edge traffic engineering with cascara. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 201–216, 2021.

[77] Rachee Singh, Nikolaj Bjorner, Sharon Shoham, Yawei Yin, John Arnold, and Jamie Gaudette. Cost-effective capacity provisioning in wide area networks with shoofly. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 534–546, 2021.

[78] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. Jellyfish: Networking data

centers randomly. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 225–238, 2012.

[79] Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, and Yueping Zhang. Proteus: a topology malleable data center network. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[80] Fung Po Tso and Dimitrios P Pezaros. Baatdaat: Measurement-based flow scheduling for cloud data centers. In *2013 IEEE Symposium on Computers and Communications (ISCC)*, pages 000765–000770. IEEE, 2013.

[81] Uptime Institute. Uptime institute global data center survey 2024. https://datacenter.uptimeinstitute.com/rs/711-RIA-145/images/2024.GlobalDataCenterSurvey.Report.pdf?version=0, 2024.

[82] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 205–219, 2016.

[83] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 407–420, 2017.

[84] Ying Wan, Haoyu Song, Yu Jia, Yunhui Yang, Tao Huang, and Zhikang Chen. Laps: Joint load balancing and congestion control on unequal-cost multi-path data center networks. In *Proceedings of the 2nd Workshop on Networks for AI Computing*, pages 11–18, 2025.

[85] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Eugene Ng, Michael Kozuch, and Michael Ryan. c-through: Part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 327–338, 2010.

[86] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *NSDI*, volume 11, pages 8–8, 2011.

[87] Haitao Wu, Guohan Lu, Dan Li, Chuanxiong Guo, and Yongguang Zhang. Mdcube: a high performance network structure for modular data center interconnection. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 25–36, 2009.

[88] Yiting Xia, Xiaoye Steven Sun, Simbarashe Dzinamarira, Dingming Wu, Xin Sunny Huang, and TS Eugene Ng. A tale of two topologies: Exploring convertible data center network architectures with flat-tree. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 295–308, 2017.

[89] Stephen Young, Sinan Aksoy, Jesun Firoz, Roberto Gioiosa, Tobias Hagge, Mark Kempton, Juan Escobedo, and Mark Raugas. Spectralfly: Ramanujan graphs as flexible and efficient interconnection networks. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1040–1050. IEEE, 2022.

[90] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. Detail: Reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 139–150, 2012.

[91] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. Understanding lifecycle management complexity of datacenter topologies. In *NSDI*, volume 19, pages 235–254, 2019.

[92] Rui Zhang-Shen and Nick McKeown. Designing a predictable internet backbone network. HotNets, 2004.

[93] Gongming Zhao, Jingzhou Wang, Hongli Xu, Zhuolong Yu, and Chunming Qiao. Coin: Cost-efficient traffic engineering with various pricing schemes in clouds. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[94] Shizhen Zhao, Rui Wang, Junlan Zhou, Joon Ong, Jeffrey C Mogul, and Amin Vahdat. Minimal rewiring: Efficient live expansion for clos data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 221–234, 2019.

[95] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. Wcmp: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, pages 1–14, 2014.

[96] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y Zhao, and Haitao Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):443–454, 2012.

9 Appendix

9.1 OSR & α for Fat Trees

We can easily compute the α for a fat tree [10] with switch degree k as defined below:

$$\begin{aligned}\alpha\big(T=FatTree(k)\big) &= \frac{OSR(F(T))}{OSR(T)} \\ &= \frac{\frac{k}{2}/\frac{k}{2}}{\Big(k-\big(\frac{k^3}{4}/\frac{5k^2}{4}\big)\Big)\Big/\big(\frac{k^3}{4}/\frac{5k^2}{4}\big)} \\ &= \frac{1}{4}\end{aligned}$$

As earlier, $F(T)$ is a direct topology built with the same hardware as T. Note that number of server ports in one rack of a random graph built with the same equipment as a fat tree with switch degree = k is $\big(\frac{k^3}{4}/\frac{5k^2}{4}\big)$ and the number of network ports in one rack is equal to $(k-$ server ports$)$. The above calculation is for an oversubscription ratio $r=1$, but the result holds for any r.

9.2 Oversubscription Reduction with Heterogeneous Switches

Define $LeafSpine'(x,y,a)$, for arbitrary (positive integer) parameters x, y and a, as the following: y spine switches of $(x+y+a)$ ports, each connected to all leafs; and $(x+y+a)$ leaf switches of $(x+y)$ ports, each connected to all spines and x servers (assume all ports are fully utilized). First, we have $OSR(T=LeafSpine'(x,y,a))=\frac{x}{y}$. For the corresponding direct network $F(T)$ built with the same equipment with servers distributed across all switches, the $(x+y+a)$-port switches get $\frac{x(x+y+a)}{x+2y}$ servers, and the $(x+y)$-port switches get $\frac{x(x+y)}{x+2y}$ servers.

$$\begin{aligned}OSR(F(T)) &= max\Big(\frac{\text{Server ports per switch}}{(x+y)-\text{Server ports per switch}}\Big) \\ &= max\Big(\frac{x(x+y+a)/(x+2y)}{(x+y+a)-\big(x(x+y+a)/(x+2y)\big)}, \\ &\quad \frac{x(x+y)/(x+2y)}{(x+y)-\big(x(x+y)/(x+2y)\big)}\Big) \\ &= \frac{x}{2y}\end{aligned}$$

Thus, $\alpha\big(T=LeafSpine'(x,y,a)\big)=\frac{OSR(F(T))}{OSR(T)}=\frac{1}{2}$.

9.3 Correctness Proof of Shortest-Union(K)'s VRF Implementation

Proof. Let us say the shortest path between R1 and R2 in the topology is (R1, A_1,A_2....A_{L-1}, R2).

Case 1: $L\geq K$: consider the path $\big($(VRF K, R1), (VRF 1, A_1), (VRF 1, A_2) ... (VRF 1, A_{L-k+1}), (VRF 2, A_{L-K+2}) ... (VRF K-1, A_{L-1}) (VRF K, R2)$\big)$, which has cost L. Since all links have cost ≥ 1, any other shortest path between (VRF K, R1) and (VRF K, R2), which will have at least L hops, will also have cost at least L. Hence, the shortest path length in the VRF graph is L.

Case 2: $L<K$: The path $\big($(VRF K, R1), (VRF $L-1$, A_1), (VRF $L-2$, A_2) ... (VRF K-1, A_{L-1}), (VRF K, R2)$\big)$ has cost K (the first link has cost $K-L$, all other links have cost 1). Hence, the distance between (VRF K, R1) and (VRF K, R2) is at most K. Next, we show that any other path between (VRF K, R1) and (VRF K, R2) length at least K. If the second hop in the path belongs to VRF i of an adjacent node of R1, then the length of the path is $\geq i+(K-i)=K$ since at least $K-i$ hops are needed to reach VRF K of any node from VRF i of any node. □

9.4 Linear Program for Traffic-Aware Weight Assignment

Let $WeightOptimizer()$ be a function that takes in a topology, a traffic demand matrix and a set of paths for each rack pair and produces a set of path weights for each communicating rack pair, maximizing throughput under a fluid flow model. $WeightOptimizer()$ is implemented with a linear program with a multi-commodity flow formulation. Given a network $G(V,E)$ where edge $(i,j)\in E$ has capacity $c(i,j)$ and a pair of sending and receiving racks $(u,v),u,v\in V$ has traffic demand $d(u,v)$. $P(u,v)$ denotes the set of paths the rack pair (u,v) can use. $Q(i,j)$ denotes the set of paths that traverse through edge (i,j) and includes a list of tuples (u,v,p) which denotes path p for the flow from rack u to rack v. $t(u,v,p)$ denotes the amount of traffic on path p from rack u to rack v. T denotes *throughput*, defined as a maximum t such that for each pair of $(u,v),d(u,v)>0$, it is feasible to route $t*d(u,v)$ traffic through the network [42].

$$\begin{aligned}&\text{Maximize } T \\ &\text{Subject to} \\ &\forall(u,v) \qquad \sum_{p\in P(u,v)} t(u,v,p)\geq T*d(u,v) \\ &\forall(i,j)\in E \qquad \sum_{\forall(u,v,p)\in Q(i,j)} t(u,v,p)\leq c(i,j)\end{aligned}$$

The objective is to maximize throughput T, subject to two constraints: **(i)** Traffic for any pair of racks (u,v) is transmitted. **(ii)** All link capacities are respected.

9.5 Summary of Statistics for Trace Parsing

Table 1 summarizes the statistics for how we parse the five real-world traces, in a way that preserves high confidence levels and statistical significance.

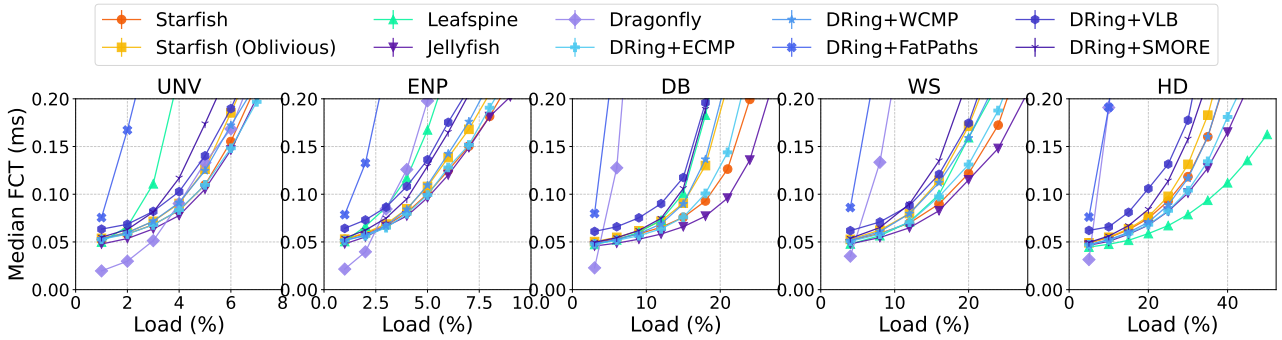


Figure 19: Starfish delivers strong performance across real-world traces when looking at the median FCT. Schemes tested are identical to those in Figure 14.

Table 1: Statistics for trace parsing.

	UNV	ENP	DB	WS	HD
Total length (h)	384	1920	24	24	24
Per-interval length (h)	48	240	0.5	0.5	0.5
#intervals	8	8	48	48	48
Confidence level	0.9	0.9	0.99	0.99	0.99
Significance level	0.1	0.1	0.01	0.01	0.01
Relative error (%)	14.41	22.87	3.07	3.07	2.01

9.6 Starfish's Performance at Median FCT

Figure 19 shows that Starfish delivers strong performance across real-world traces when looking at the median FCT. Plots are truncated at 0.20ms.