

Beyond fat-trees without antennae, mirrors, and disco-balls

Simon Kassing
ETH Zürich
simon.kassing@inf.ethz.ch

Asaf Valadarsky
Hebrew University of Jerusalem
asaf.valadarsky@mail.huji.ac.il

Gal Shahaf
Hebrew University of Jerusalem
gal.shahaf@mail.huji.ac.il

Michael Schapira
Hebrew University of Jerusalem
schapiram@huji.ac.il

Ankit Singla
ETH Zürich
ankit.singla@inf.ethz.ch

Abstract

Recent studies have observed that large data center networks often have a few hotspots while most of the network is underutilized. Consequently, numerous data center network designs have explored the approach of identifying these communication hotspots in real-time and eliminating them by leveraging flexible optical or wireless connections to dynamically alter the network topology. These proposals are based on the premise that statically wired network topologies, which lack the opportunity for such online optimization, are fundamentally inefficient, and must be built at uniform full capacity to handle unpredictably skewed traffic.

We show this assumption to be false. Our results establish that state-of-the-art static networks can also achieve the performance benefits claimed by dynamic, reconfigurable designs of the same cost: for the skewed traffic workloads used to make the case for dynamic networks, the evaluated static networks can achieve performance matching full-bandwidth fat-trees at two-thirds of the cost. Surprisingly, this can be accomplished even without relying on *any* form of online optimization, including the optimization of routing configuration in response to the traffic demands.

Our results substantially lower the barriers for improving upon today's data centers by showing that a static, cabling-friendly topology built using commodity equipment yields superior performance when combined with well-understood routing methods.

CCS Concepts

• **Networks** → **Data center networks**;

Keywords

Data center; Topology; Routing

ACM Reference format:

Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. 2017. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proceedings of SIGCOMM '17, Los Angeles, CA, USA, August 21–25, 2017*, 14 pages.
<https://doi.org/10.1145/3098822.3098836>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGCOMM '17, August 21–25, 2017, Los Angeles, CA, USA

© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.

ACM ISBN 978-1-4503-4653-5/17/08...\$15.00

<https://doi.org/10.1145/3098822.3098836>

1 Introduction

Virtually every popular Web service today is backed by data center infrastructure. With the growth of these services, the supporting infrastructure has also acquired massive scale, with data centers being designed with as many as 100,000 servers [15]. For such large facilities, engineering full-bandwidth connectivity between all pairs of servers entails significant effort and expense. Further, for many facilities, at any given time, only a fraction of servers may require high-bandwidth connectivity [13, 17], making such a design seem wasteful. However, traditional topologies like the fat-tree [3] present network designers with only two choices: (a) either build an expensive, rearrangeably non-blocking network; or (b) build a cheaper, oversubscribed network that does not provide high-bandwidth connectivity even to small (albeit arbitrary) subsets of the server pool, thereby needing aggressive workload management.

Over the past few years, numerous data center network designs have tackled this problem based on the idea of adapting the topology itself to the traffic demands [10, 12–14, 17, 18, 24, 27, 35, 40]. The key insight in this literature is that if only a few network hotspots exist at any time, perhaps through online measurement, these hotspots can be identified, and the network topology can be optimized dynamically to alleviate them. Reconfigurable wireless and optical elements enable such online adjustments of connectivity. For many workloads, such a network can match the performance of a much more expensive interconnect that provides full bandwidth between all pairs of servers at all times. The ingenuity of the many proposals along these lines lies in the varied capabilities of the reconfigurable elements, how they are connected together (including movable wireless antennae [17], ceiling mirrors [18, 40], and disco-balls [13]), and the algorithms for online management of connectivity.

This broad approach has obvious, intuitive appeal, and indeed, some of the evaluations show performance similar to a full-bandwidth fat-tree at 25–40% lower cost for some workloads [13, 18]. Not only are such results impressive, it is also unclear whether there are any other viable options besides full-bandwidth topologies and such dynamic topologies. Before fleshing out their reconfigurable optics-based architecture, Helios [12], its authors summarized the (2010) state of data center network topology design as follows:

“Unfortunately, given current data center network architectures, the only way to provision required bandwidth between dynamically changing sets of nodes is to build a non-blocking switch fabric at the scale of an entire data center, with potentially hundreds of thousands of ports.”

More recently, the 3D-beamforming proposal [40] (2012) made a similar assessment, and FireFly [18] (2014) explicitly considered only two design possibilities: a full bisection-bandwidth network and topology dynamism. This literature has thus implied that static networks, which lack any opportunity for online topology optimization, are inherently inefficient for unpredictably skewed network traffic, and consequently, that dynamic topologies are the only alternative to expensive full-bandwidth fabrics.

The goals of our work are to examine this presumption critically, explore the utility and limits of topology dynamism, develop an understanding of the relative strength of static and dynamic networks, and outline directions for improving upon today's data centers. Towards these goals we make the following contributions:

(1) A metric for network flexibility: So far, there has been no established metric for the flexibility of topologies towards accommodating skewed traffic, with practitioners relying on specific workload instances for evaluations. We thus propose an intuitive metric capturing a topology's performance under traffic skew.

(2) A comparison of static and dynamic topologies: We compare static and dynamic networks under the assumptions of optimal topology dynamics and traffic engineering. This comparison uses a linear program optimization of a fluid-flow model, thus abstracting out possible inefficiencies of routing, congestion control, and the traffic-estimation and optimization needed for dynamic networks. Our results show that at equal-cost, recently proposed static networks outperform dynamic networks in this setting, with both achieving substantial performance benefits over fat-trees.

(3) Routing on static networks: Translating performance in fluid-flow models to that at packet-level can be challenging, particularly with dynamic, unpredictable traffic. Surprisingly, we find that even simple, *oblivious* routing achieves high performance for the static networks we consider. Thus, even online optimization of routes is not essential for achieving efficiency substantially higher than fat-trees and comparable with dynamic networks. Over the same skewed workloads recently used to make the case for dynamic networks and heretofore considered challenging for static networks, static networks achieve performance gains over fat-trees similar to those claimed by dynamic networks at the same price point.

(4) Concrete, deployable alternatives to today's data centers: Our results suggest that the least-resistance path beyond fat-trees may lie in transitioning to superior, cabling-friendly, static networks, such as Xpander [33], and that dynamic networks have not yet demonstrated an advantage over such static networks. We discuss the implications for future research on data center network design, especially for dynamic topologies, in §7.

To aid reproducibility of our results, help researchers and practitioners run further experiments, and provide an easy-to-use baseline for future research on dynamic networks to compare against, our simulation framework is available online [1].

2 Network flexibility

This section examines the instructive example of the inflexibility of oversubscribed fat-trees towards skewed traffic matrices (§2.1) and introduces a quantitative notion of the desired flexibility (§2.2).

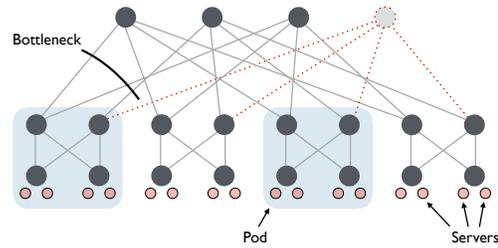


Figure 1: A $k = 4$ fat-tree. With more than 75% of the network's capacity intact, if all 4 servers in one pod sent traffic to servers in another pod (thus involving 50% of all servers), each would get only 75% of the bandwidth.

2.1 Fat-trees are inflexible

Oversubscribed fat-trees are fundamentally limited in their ability to support skewed traffic matrices. In the example shown in Fig. 1, the fat-tree is oversubscribed by removing one root switch. If each server in one pod is communicating with a (different) server in another pod, not all such connections can get full bandwidth. In this example, the network still has more than 75% of its original capacity, and yet cannot provide full bandwidth connectivity for a traffic matrix involving only 50% of the servers. This observation can be formalized quite simply, as follows:

OBSERVATION 1. *If a fat-tree built with k -port switches is oversubscribed to x fraction of its full capacity, then there exists a traffic matrix involving $2/k$ -fraction of the servers such that no more than x fraction of throughput per server is achievable.*

PROOF. The proof is constructive, *i.e.*, we provide the specific traffic matrix which is bottlenecked to no more than x per-server throughput. If the oversubscription is at the top-of-rack (ToR) switch, with each ToR supporting s servers with only sx network ports, then trivially, any traffic matrix (with $2s$ servers) where each server under one ToR sends traffic to a unique server under another ToR, is bottlenecked at the sender's ToR to x throughput. Note that nonuniform oversubscription only pushes throughput lower — in that case, some ToR uses even fewer connections to the network and achieves throughput lower than x .

Next, suppose that the aggregation layer (*i.e.*, the middle layer of switches) is oversubscribed to an x fraction, *i.e.*, each switch in this layer has a links connecting it to ToRs and ax links connecting it to the core layer (the upper layer of switches). Then, any traffic matrix where each server in a pod sends traffic to a unique server not within the same pod, is bottlenecked at the aggregation layer to x fraction of throughput. The same argument applies when the core layer is oversubscribed. In these cases, the number of servers in the difficult traffic matrices is two times the number of servers in a pod, *i.e.*, $2/k$ of the servers in the network. \square

For a fat-tree built using 64-port switches, a $2/k$ fraction of the servers would be a mere 3%. If this fat-tree were built at, say 50% oversubscription, a pair of pods comprising only 3% of the network's servers could still not achieve full throughput, even while the rest of the network idles. Only if 50% or less of the servers in *each* of the pods were involved, could such an oversubscribed fat-tree achieve full throughput for these servers. Of course, if it were known *a priori* which pods might need higher bandwidth, different pods may be

oversubscribed differently, but this would be a strong assumption on the predictability and stability of traffic, and pod-pod traffic would need aggressive management in such scenarios. Other Clos-network-based designs suffer from similar problems.

Thus, as literature on topology adaptation rightly points out, oversubscribed fat-trees score low on any metric of network flexibility. But what precisely *is* the metric? So far, there has not been a clearly specified and easily replicable standard for evaluating the flexibility of a topology towards accommodating skewed traffic matrices¹. In the following, we offer a simple starting point.

2.2 Throughput proportionality

We would like to build static networks that can move around their limited capacity to meet traffic demands in a targeted fashion. A network’s total capacity is fixed, and defined by the total link capacity of its edges, and for each server, the network expends this capacity on routing its flows. We may hope that as the number of servers participating in the traffic matrix (TM) decreases, we see a proportional increase in throughput. In the following, we describe a benchmark for network flexibility capturing this intuition.

Modeling throughput per server: Consider an arbitrary static network G with N servers. A traffic matrix M captures traffic demands between the N servers, with m_{ij} specifying the demand from server i to server j . We restrict traffic matrices to the hose model, *i.e.*, the sum of demands from (to) each server is limited by the outgoing (incoming) capacity of its network interfaces.

G is said to support a TM M with the *satisfaction matrix* T_M if each flow $i \rightarrow j$ in M achieves throughput $T_M(i, j)$ without violating link-capacity constraints. Throughput for server i is then $t_i = \sum_j T_M(i, j)$. G is said to support M with throughput t if there is a satisfaction matrix where each server simultaneously achieves throughput at least t , *i.e.*, $\exists T_M: \forall i \in [N] t_i \geq t$. Defining throughput in this way, at a per-server granularity (rather than flow or server-pair granularity) allows a simpler description of TMs with active servers, without constraints on how each active server spreads its traffic across destinations, and more directly captures the only constraints in the hose model: the servers themselves. Accordingly, throughout the following discussion, we shall refer to throughput in terms of a fraction of the line-rate $\in [0, 1]$.

A network shall be called *throughput-proportional* (TP), **if** when built such that it achieves throughput α per server for the worst-case TM, **then** it achieves throughput $\min\{\alpha/x, 1\}$ per server for *any* TM with only an x fraction of the servers involved. This definition captures the intuition described above: as the number of servers involved in communication decreases, a TP network can increase per-server throughput proportionally². Note that one can easily map a fuzzier classification of servers as “hot” or not, to the strict binary of “involved in the TM” or not.

¹FireFly [18] used an abstract model to evaluate the benefit of dynamic network links compared to a fat-tree based on performance improvements over random traffic matrices. Our objective is very different: quantify how well a given static network accommodates *arbitrarily skewed* traffic matrices. Firefly’s “Metric of Goodness” also falls short of this: as prior work shows [20], bisection bandwidth can be a logarithmic factor away from throughput, and this factor varies for topologies.

²We acknowledge the parallels with energy proportional networking [2, 19], but note that EPN addresses a different question along the lines of which links to turn off, or change the data rate on, etc. to reduce the network’s energy usage.

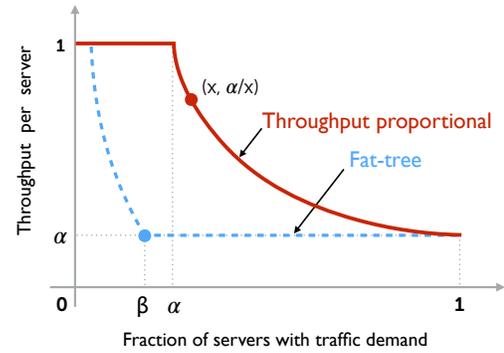


Figure 2: A throughput-proportional network would be able to distribute its capacity evenly across only the set of servers with traffic demands.

This notion of a throughput-proportional network is illustrated in Fig. 2, which also contrasts it with the fat-tree’s behavior. As discussed in §2.1, for the fat-tree with a simple pod-to-pod TM, if a fraction greater than $\beta = 2/k$ of servers are involved, throughput will be limited to α . As the fraction of servers involved drops below β , throughput per server increases proportionally, hitting 1 only when α fraction of the pod itself is involved.

As the fat-tree example illustrates, network bottlenecks may prevent a network from achieving throughput proportionality. But is such proportionality unattainable for other statically wired networks as well? Below, we prove that for a generic statically wired network, per-server-throughput *cannot* improve more than proportionally, at least for certain classes of traffic matrices.

Permutation TMs. A permutation traffic matrix over k servers involves each of these servers communicating with exactly one other unique server. We prove the below result.

THEOREM 2.1. *Over the class of permutation TMs, throughput in a static network cannot increase more than proportionally as the fraction of servers involved in the TM decreases.*

Theorem 2.1 follows from the following lemma.

LEMMA 2.2. *If, for some $x \in (0, 1]$, G supports throughput t for every permutation matrix over x fraction of the servers, then it supports throughput xt for any permutation matrix over all servers.*

Before diving into the proof of Lemma 2.2, we explain why it indeed implies Theorem 2.1. Assume, for the point of contradiction, that the statement of Theorem 2.1 is false and so throughput does increase *more* than proportionally in α for some static network G . Then, by the definition of throughput proportionality for some fraction of servers x , the supported throughput for all permutation TMs involving only an x -fraction of the servers is some β greater than $\frac{\alpha}{x}$. By Lemma 2.2, G supports all permutation TMs with throughput $x\beta > x\frac{\alpha}{x} = \alpha$, contradicting the definition of α as the worst-case throughput across all permutation TMs for G . Now, let us prove Lemma 2.2.

PROOF. (of Lemma 2.2) Consider a permutation TM M over all n servers in G . Since in any permutation TM, each server communicates with exactly one other unique server, M comprises $\frac{n}{2}$ distinct communicating pairs of servers. Selecting $\frac{xn}{2}$ of these pairs and only considering the communication between these pairs,

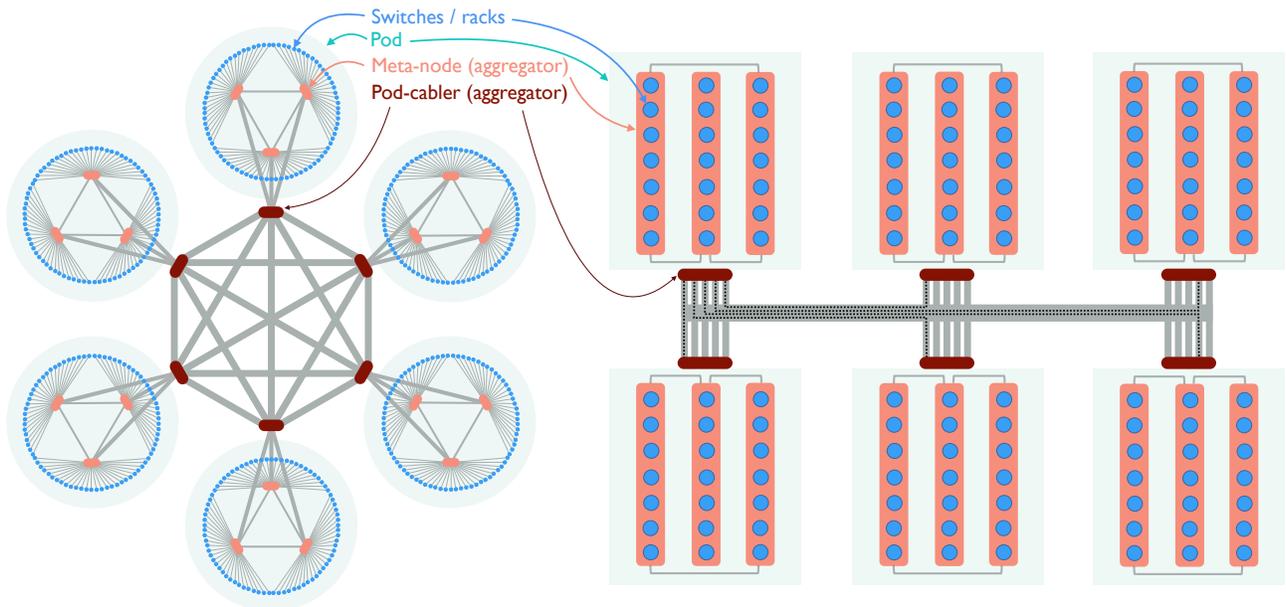


Figure 3: An Xpander network with 486 24-port switches, supporting 3402 servers broken into 6 pods, each with 3 meta-nodes. Left: all switches are shown at the circles’ circumferences; other nodes are cable-aggregators. Right: Floor plan for the same 6 pods. Each pod has 3 rows of racks, each being a meta-node. Each meta-node’s 27 switches and their connected servers fit in 7 racks (circles) of 48 rack units each, after accounting for cooling and power.

gives rise to a permutation TM on an x -fraction of the servers, and is thus supported (by the statement of the lemma) at throughput t . There are precisely $K = \binom{n/2}{xn/2}$ possible selections of such a set of communicating pairs. If we scale the throughput of each of the flows in such a TM to $\frac{1}{K}$, the utilization of any link in G over this scaled-down TM is now at most $\frac{1}{K}$. The scaled-down throughput of all TMs induced by choosing $\frac{xM}{2}$ of the communicating pairs in M can thus be supported *simultaneously*. Observe, however, that every communicating pair in M appears in precisely $\binom{n/2-1}{xn/2-1}$ such selections, and so its total achieved throughput should be scaled up by this factor. As the number of servers in the network, n , increases, the resulting throughput converges to xt . \square

Following the strategy for the above proof for the family of permutation TMs, we have also been able to prove analogous results for several other TM families of interest, including all-to-all, many-to-one, and one-to-many TMs. However, we can only conjecture on a more general result over the larger class of hose TMs as follows:

CONJECTURE 2.3. *Throughput in a static network cannot increase more than proportionally as the fraction of active servers decreases.*

One strategy for proving this stronger result would be to prove, as conjectured below, that permutations are worst case TMs. Such a result, combined with Lemma 2.2, would then prove Conjecture 2.3. Proving that permutation traffic matrices are a corner case would also be of independent interest – as prior work

notes, the complexity of finding worst-case TMs for arbitrary networks is unknown [20] and such a result could be significant step in that direction.

CONJECTURE 2.4. *Given a static network G with N servers and an arbitrary TM M for which G achieves throughput t per server, there exists a permutation TM P for which G achieves throughput $\leq t$.*

The impossibility of *exceeding* throughput proportionality in static networks (at least for certain families of TMs) makes it an idealized benchmark for network flexibility towards skewed traffic: as traffic consists of a smaller and smaller fraction of servers becoming hotspots, how well does throughput scale compared to TP? §5 addresses this question experimentally.

3 State-of-the-art static network topologies

The Helios authors’ assessment of the inflexibility of statically-wired networks (noted in §1) was likely accurate at the time, but in the intervening years, a large number of new static topologies have been proposed for data centers, including Jellyfish [31], Longhop [32], Slimfly [9], and Xpander [33]. All of these have a common feature: unlike the Fat-tree [3], they do not break the network into layers, instead retaining a flat structure, where top-of-rack switches are directly wired to each other. However, there are sizable differences in performance even across flat topologies [20]. In particular, Jellyfish and Xpander achieve near-identical, high performance (modulo a small variance for Jellyfish due to its randomness), due to their being good expander graphs [33]. We shall refer to these topologies as **expander-based** networks as we expect results to be similar for other network designs based on near-optimal expander graphs, such as LPS [25, 33].

For readers concerned by Jellyfish’s randomness, we note that **Xpander** is deterministic, and provides symmetry and an intuitive notion of inter-connected clusters of racks (“meta-nodes” and pods), enabling clean aggregation of cables into a small number of bundles. As noted in [29], such bundling can “reduce fiber cost (capex + opex) by nearly 40%”. We illustrate these desirable properties in Fig. 3 for an Xpander configured to cost 33% less than a full-bandwidth fat-tree with $k=24$. For a more detailed exposition including cable counts and lengths, we refer readers to the Xpander paper [33].

The throughput of expander-based topologies on largely uniform workloads or/and fluid-flow models has been evaluated before, but it remains unclear whether (a) these claims extend to the skewed workloads that dynamic networks target; and (b) the claimed throughput advantage can be translated into low flow completion times. Sections §5 and §6 address these questions.

4 Dynamic network topologies

Instead of delving into the details of any one of the myriad dynamic network designs [10, 12–14, 17, 18, 24, 27, 35, 40], we endeavor to tackle an abstract model that covers, with reasonable fidelity, the existing proposals, as well as similar future extensions.

In a generic dynamic network, each ToR switch may have a certain number of flexible ports, say k , which can be connected to available flexible ports on other ToRs. The various proposed realizations of this approach differ in the connectivity options they

	Static	FireFly	ProjecToR
SR transceiver	\$80	\$80	-
Optical cable (\$0.3 / m)	\$45	-	-
ToR port	\$90	\$90	\$90
ProjecToR Tx+Rx	-	-	\$80 to \$180
DMD	-	-	\$100
Mirror assembly, lens	-	-	\$50
Galvo mirror	-	\$200	-
Total	\$215	\$370	\$320 to 420

Table 1: Cost per network port for static and recent dynamic networks. Component costs are from ProjecToR [13]. Each cable in a static network is accounted for with 300 meter length, with its cost shared over its two ports.

make available, the reconfiguration time needed to change the connectivity of flexible ports, the per-port cost, and the algorithms used to configure the interconnect in real-time. For greatest flexibility, we disregard constraints that limit connectivity, and allow any ToR to connect to any other ToR. Further, our focus is on the *potential* performance of dynamic topologies, and thus we ignore algorithmic inefficiencies to the largest extent possible. We nevertheless mention two factors that can have a large impact on the performance of dynamic networks:

Direct-connection heuristics: The numerous dynamic designs referenced above, all prioritize direct connections between ToR-pairs with traffic demand. While FireFly [18] in general uses multi-hop relaying, direct connections between pairs of communicating racks are prioritized by its heuristics. As we shall see, such heuristics can impair the potential of dynamic topologies.

Buffering: The limited number of flexible ports, together with the reconfiguration time needed to move their connectivity, implies that dynamic topology proposals need to buffer packets until a suitable connection is available, thus adding latency. If such buffering is not feasible, *e.g.*, due to most flows being short and latency-sensitive, it becomes necessary to carry traffic over multiple hops.

We do not believe that any past proposal addresses these issues in entirety, but perhaps these problems can be fully addressed by future proposals, so we model two alternatives: (a) an “unrestricted” optimal design unaffected by both factors; and (b) a “restricted” design that suffers from both, *i.e.*, picks the ToR-level topology prioritizing direct connections between *communicating* ToRs, and requires multi-hop connectivity in cases where the TM is such that not all communicating pairs can be concurrently connected by direct connections.

A more realistic abstraction of dynamic networks would lie somewhere between these two extremes, but would need greater machinery to capture the waiting time involved in dynamic connectivity. The utility of the restricted model is in developing intuition about the role of buffering and the time spent waiting for dynamic connectivity in dynamic networks.

Lastly, it is worth noting that the flexibility of dynamic topologies comes at a substantially higher per-port cost for the flexible ports. The cost advantage of dynamic designs over fat-trees stems from using *fewer* ports. However, this is also true of expander-based networks. For equal-cost comparisons, networks should thus be configured with the **same total expense on ports**. We shall denote

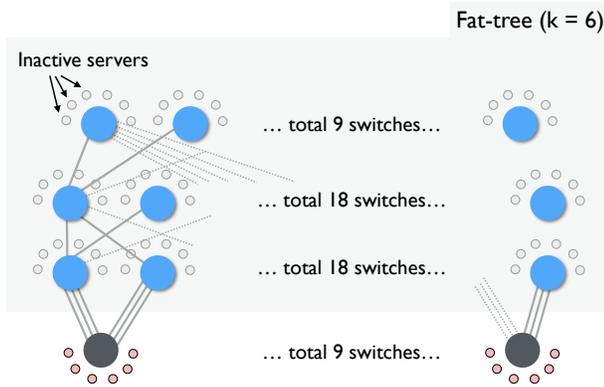


Figure 4: This topology provides full throughput to all active servers, where any scheme in the restricted dynamic model could not.

as δ , the cost of a flexible port normalized to that of a static port, with the cost of a static port including half of a 300 meter cable's cost. Based on component costs for ProjecToR and FireFly (Table 1), the lowest estimates imply $\delta = 1.5$. Therefore, for supporting the same number of servers, a dynamic network can only buy at most $0.67\times$ the network ports used by an equal-cost static network. Of course, if there were no additional cost for flexibility, *i.e.*, $\delta = 1$, unrestricted dynamic networks would have an advantage: trivially, they can *at least* do anything a static network can.

4.1 Static v. un/restricted dynamic nets: a toy example

Consider a network with 54 switches, each with 12 ports, 6 of which are attached to servers. The devices and links necessary for this discussion are shown in Fig. 4. Now assume that the TM involves servers on only the 9 racks at the bottom. Thus, servers on each of the other 45 switches are inactive and irrelevant, and one can think of these switches as just 6-port devices. These 45 switches can then be connected in a standard fat-tree topology with $k = 6$, while exposing 54 of their ports to traffic sources and sinks between which they provide full bandwidth. These 54 ports can be connected in any convenient manner to the 9 switches with active servers, thus providing full bandwidth between all active servers.

Using the topology in Fig. 4, the *unrestricted* model can, of course, achieve full throughput. Even if the limitation of direct-connection heuristics was imposed, but unlimited buffering was allowed, full throughput could still be achieved by moving the dynamic connections between the 9 racks in a round-robin fashion – at any moment, the network can deliver the same capacity as the servers need. (The number of servers on active racks equals their outgoing links, and each packet consumes one unit capacity in the network via the direct connection.) However, in practice, buffering cannot be unlimited, and at relatively small time intervals, the reconfiguration overhead must be incurred. ProjecToR's recommended duty cycle, for instance, could achieve 90% of full throughput.

Notice that the topology in Fig. 4 does not connect any of the 9 switches with traffic demands directly, unlike a *restricted* dynamic network. Additionally, the absence of buffering will require that all flows be *concurrently* serviced, implying that for all-to-all traffic between these switches, there is no advantage to moving links around. This makes the restricted dynamic model no better than

the best possible static topology connecting the 9 racks using their direct links. The performance of any such a topology is upper bounded (computed as in [30]) at 80% of the full throughput.

Interestingly, the best known static networks of the same cost achieve full throughput for near-worst-case traffic patterns across the same number of communicating servers, without being designed with any awareness of which servers will be active. We verified this with experiments over Jellyfish in two configurations supporting the same number of servers: (a) with 9 network ports at each of the 54 switches, instead of a dynamic network's 6 (*i.e.*, $\delta = 1.5$); and (b) with the same port count of 12 for all switches as in the above discussion, but with 81 such switches (again, $\delta = 1.5$). Other expander-based data centers would achieve the same result.

This toy example illustrates the question at the heart of this work: do a dynamic network's fewer / more expensive, but flexible connections compare favorably to a larger number of cheaper static connections? We shall later explore this question in greater depth quantitatively, but it is also useful to point out the several qualitative factors that may put dynamic networks at a disadvantage.

4.2 Barriers to the deployment of dynamic networks

Dynamic topologies are an intuitive and exciting idea, but bear little resemblance to present practice in data centers, thus posing unique challenges for deployment:

- Unfamiliar problems in device packaging, spatial planning.
- Monitoring and debugging the highly ephemeral networks.
- The impact of environmental factors like dust, vibration, and temperature on device alignment and functioning.
- Lack of clarity on the reliability and lifetime of the used devices in environments they are not intended for.
- Lack of operator experience with the devices involved.

Certainly, not all the above criticisms apply across all proposals in this direction, but some are fundamental, *e.g.*, monitoring and debugging networks that are themselves changing. Deploying a different static topology like Xpander may also require changes, *e.g.*, to any automation and operator training specific to Clos networks, but these barriers are substantially lower, as evidenced by the recent deployment of the DragonFly [23] topology in the high-performance computing space.

5 Static \neq Inflexible

This section presents a head-to-head comparison of static and dynamic networks, focused on the topology *models* themselves, neglecting any inefficiencies from routing and congestion control, and additionally, in the case of dynamic networks, the dynamic topology optimization. §6 will address, for static networks, the problem of translating results from this idealized setting to low flow-completion times under dynamic, skewed network traffic.

We verified that Xpander and Jellyfish achieve identical performance. Experiments in this section use Jellyfish as its ease of construction with arbitrary switch and port counts allows us to include two other recent static networks for comparison. Results in §6 use Xpander to side-step concerns about Jellyfish's randomness.

Both dynamic and static networks are evaluated here under skewed but difficult (ideally, worst-case) TMs – in line with the

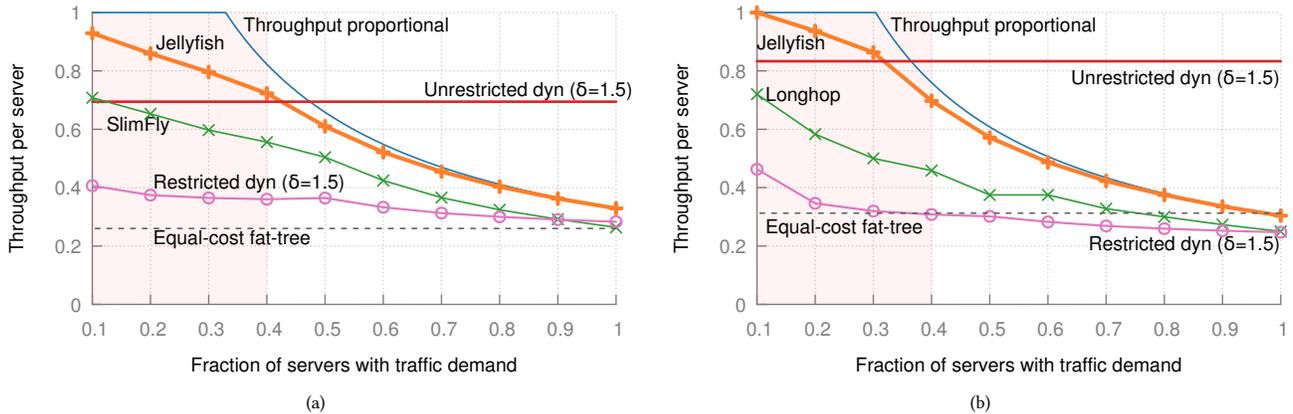


Figure 5: Throughput proportionality and dynamic networks compared with (a) SlimFly and a same-equipment Jellyfish; and (b) Longhop and a same-equipment Jellyfish. According to measurements used to make the case for dynamic topologies, the shaded region is the regime of interest. For both cases, the unrestricted model would achieve full throughput for $\delta = 1$, i.e., if there were no additional cost for flexibility.

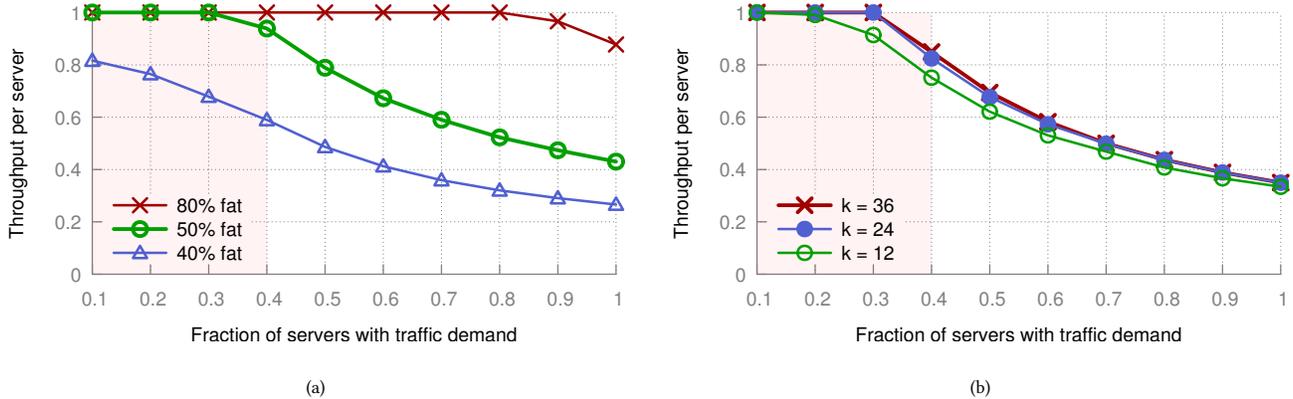


Figure 6: A direct comparison between a full bandwidth fat-tree and an oversubscribed Jellyfish network: (a) Jellyfish with fewer switches — 80%, 50%, and 40% — as a $k = 20$ fat-tree, while supporting the same number of servers. With 50% fewer switches, it still provides nearly full bandwidth connectivity between any 40%-subset of servers. (b) This advantage is consistent or improves with larger k (12, 24, 36). Jellyfish supports $2\times$ the fat-tree’s servers in each case.

definition of TP (§2.2), we want oversubscribed networks to provide high throughput for *any* TM involving small subsets of servers.

We borrow heavily from recent work on comparing topologies [20], using the associated throughput evaluation tool [21]. We use a series of skewed TMs, increasing the fraction of server racks participating in the TM, with no flows between non-participating racks. For static networks, we use longest matching TMs [20], whereby each participating rack sends all its traffic to one other rack, and the rack pairings maximize distance between communicating racks using a heuristic: maximum-weight matching, with the weights being the distances between racks. Intuitively, flows along long paths consume resources on many edges, and the large rack-to-rack flows reduce opportunities for load balancing traffic. These TMs have been shown (empirically) to be harder than TMs such as all-to-all [20]. Thus, while finding the worst-case TM is a computationally non-trivial problem, we made our best efforts to evaluate static networks under difficult TMs.

In the context of dynamic networks, longest matching TMs are meaningless: by changing the topology, distances between racks can be changed. The unrestricted model is simple to dispose of, regardless of TM. As long as the bottlenecks are not at the servers, independent of the number of ToRs networked, it can achieve per-server throughput $\min\{1, \frac{r}{s}\}$, if every ToR has r network ports and s server ports — at any moment, a ToR can be delivering r units of traffic directly to a destination, and at most producing s units. The independence from the number of racks comes from ignoring the reconfiguration time and buffering, which would be important concerns in any evaluation of latency or flow completion time.

For the restricted model, instead of evaluating specific mechanisms for computing topologies, we compute an upper-bound on the performance of *any* topology which could be built using the fixed network degree r at each ToR, as explained in §4.1.

Results: All the non-fat-tree networks achieve much higher performance than a same-cost oversubscribed fat-tree, particularly

as fewer servers participate in the TM (leftward along the x -axis). Fig. 5(a) shows results for SlimFly (578 ToRs, 25 network- and 24 server-ports per ToR) [9], Jellyfish built with exactly the same equipment, and the throughput proportionality (TP) curve using the throughput achieved for Jellyfish at $x=1.0$ as the base³ (*i.e.*, α in Fig. 2). For a hypothetical TP-network built at the same oversubscription as Jellyfish, in this case, when fewer than 35% of servers are involved, each would obtain full throughput. The restricted dynamic topology model (with two-thirds the network ports used by the static networks, *i.e.*, $\delta = 1.5$) performs poorly. The unrestricted dynamic model ($\delta = 1.5$) achieves lower throughput than Jellyfish when a smaller fraction of servers is involved. It is noteworthy, that this is the operating regime for many deployments – recent measurements across 4 large production clusters showed 46-99% of rack-pairs exchanging *no* traffic [13] in a representative 5-minute window. Fig. 5(b) shows broadly similar results in a configuration based on the Longhop topology [32] (512 ToRs, 10 network- and 8 server-ports per ToR).

Clearly, reducing the number of network ports hurts the dynamic topologies due to ToR-level bottlenecks. Thus, in an alternative approach to equal-cost comparisons, instead of reducing the network ports for the dynamic networks, we also evaluated Jellyfish with $\delta \times$ the network ports. As in the example in §4, we evaluated two possibilities: (a) giving Jellyfish $\delta \times$ switches of the same port-count, and (b) giving Jellyfish the same number of switches, but each with $\delta \times$ network ports. In both settings, even with $\delta = 1.5$, Jellyfish achieved full throughput in the regime of interest.

Beyond comparisons with dynamic networks, we also attempt to quantify how much more efficient expander-based networks can be than fat-trees for such skewed traffic. Fig. 6(a) shows the results for Jellyfish built using the same number of servers, and 80%, 50%, and 40% of the switches available to a full fat-tree with $k=20$ (*i.e.*, 500 switches, with 20 ports each, and 2000 servers). With 50% of the fat-tree’s switches (and 37.5% its network cables; the server-switch cables are the same number, of course), Jellyfish can provide nearly full bandwidth as long as <40% of servers participate in the TM.

Fig. 6(b) shows that Jellyfish’s advantage is consistent, or improves with scale, as it is built using the same set of switches as full fat-trees built with $k=12, 24$, and 36, but with *twice* the servers in each case. Note that Jellyfish topologies are being impaired more severely than may be evident: adding more servers at a switch with the same port-count also reduces the number of network ports available to connect to other switches.

Results in Fig. 5 and 6 cover oversubscription ($1:\frac{1}{\alpha}$) ranging from 1:4 through nearly 1:1. Throughout the regime of interest, for large α , both unrestricted dynamic networks (with $\delta = 1.5$) and Jellyfish achieve full throughput, and for smaller values of α , both fall short, but Jellyfish compares well with dynamic networks.

³Our interest is in assessing how well the best-performing static networks compare to TP, and thus we only show the TP curve for Jellyfish. Given that Jellyfish is not far from TP, combining this experimental result with the analysis in §2.2 implies that static networks cannot exceed Jellyfish’s scaling characteristic, and more generally, that of expander-based networks, by a large margin.

To summarize, in a fluid-flow model ignoring inefficiencies in routing, congestion control, and dynamic optimization, known static topologies provide substantial efficiency gains over fat-trees. Further, particularly under the skewed workloads used to make the case for dynamic topologies, they fail to provide an advantage over these static topologies.

6 Simple, effective routing on static networks

Routing on expander-based networks is nontrivial [31, 33], and so far, solutions have depended on MPTCP [36] over k -shortest paths [38]. While this approach has been shown to achieve throughput approximating the linear program solution for certain traffic matrices, the dependence on MPTCP poses deployment challenges, and also requires a long convergence period, thus only reaching optimality for long-running flows. In addition, the use of k -shortest path routing requires significant architectural changes. Can we achieve low flow completion times on such networks, especially under changing, skewed traffic, with simple and easy-to-deploy routing and congestion control schemes?

We begin by examining two well-understood routing schemes: ECMP and VLB. We investigate the performance of these routing schemes with DCTCP [5], which is already deployed in data centers. Xpander [33] is used throughout this section as a representative expander-based network. In the following, we show that both ECMP and VLB provide poor performance in certain scenarios. We then show that a hybrid of these two simple routing schemes suffices for attaining high performance across diverse workloads.

6.1 ECMP does not always suffice

Expander-based static networks connect ToRs directly to each other. Consider any pair of ToRs which are direct neighbors (Fig. 7(a)), and a traffic matrix consisting of only traffic between these two racks. For this traffic, ECMP enables the use of only the direct link between these racks, even though the rest of the network is unused. This inability to use multiple paths creates a bottleneck, degrading throughput. The result of packet-level simulations for this scenario is shown in Fig. 7(b), where just 10 servers on two adjacent racks send each other traffic. (Details of the experiment are unimportant for now, but curious readers can refer to §6.4.) As soon as the load is enough to saturate the bottleneck, the average flow completion time (FCT) becomes much higher in Xpander-ECMP.

Using Valiant Load Balancing [39] in Xpander on the other hand, achieves much better results; using random via points to bounce traffic through enables the use of path diversity that ECMP prohibits. Perhaps VLB is the right answer, then?

6.2 VLB does not always suffice

Unfortunately, VLB has not one, but two shortcomings: first, as is already clear from Fig. 7(b), the use of longer paths inflates network RTTs, and thus FCT for short flows which are RTT bound, even though high throughput can be achieved. Second, when there is in fact high traffic demand throughout the network, VLB’s use of random via points makes inefficient use of network bandwidth. FCT results for such a scenario, with Xpander with an all-to-all traffic matrix, are shown in Fig. 7(c). As the load increases, VLB’s performance deteriorates, and ECMP achieves much better

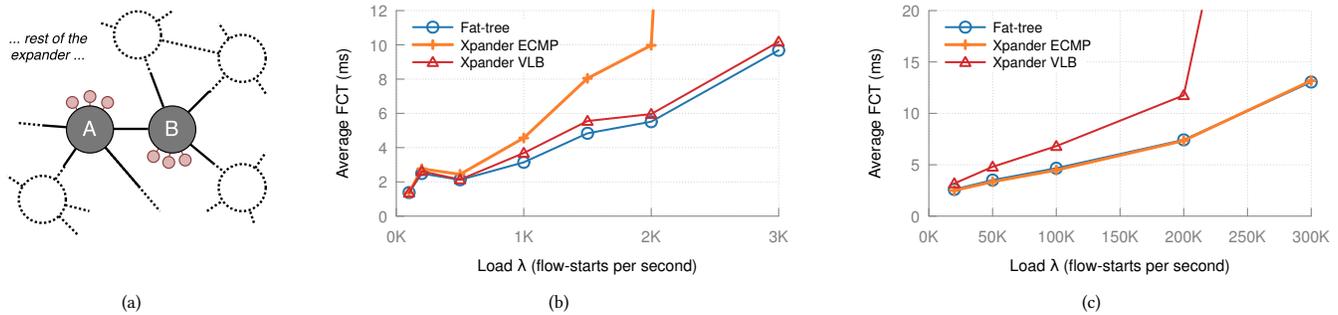


Figure 7: Failure scenarios for ECMP and VLB: (a) ECMP fails to use path diversity between directly connected ToRs. (b) Average FCT, in a scenario where only 10 servers on two adjacent racks in Xpander are active. For the fat-tree, servers on two racks in the same pod are active. (c) Average FCT for all-to-all traffic.

results, matching the full-bandwidth fat-tree because the workload is uniformly spread, and the optimal choice is indeed to use shortest paths for all traffic.

6.3 A robust ECMP-VLB hybrid

These corner-cases already yield useful information: for workloads like the one described by Facebook [28], ECMP on Xpander would indeed perform well, matching the fat-tree’s performance at much lower cost, as we shall see later. Further, for skewed workloads like those considered by dynamic networks such as ProjecToR [13], VLB suffices, modulo the increase in FCT for short flows, which is determined more by RTT than bandwidth. These observations provide the intuition for a simple hybrid scheme that achieves high performance across a diversity of workloads.

We begin with the following hybrid: packets for a flow are forwarded along ECMP paths until this flow encounters a certain congestion threshold (e.g., a number of ECN marks), following which, packets for this flow are forwarded using VLB. While we found over our experiments that this scheme does in fact match a full-bandwidth fat-tree’s performance across the workloads we consider, it requires looking at congestion behavior to adaptively decide on forwarding behavior.

We have found that a simpler design achieves the same results for the workloads of interest: a flow is forwarded along ECMP paths until this flow has sent a certain threshold number of bytes, Q , following which, the flow is forwarded using VLB. Further, instead of switching routes at packet granularity, we do so for flowlets [22, 34], i.e., for each new flow’s flowlets, ECMP paths are chosen; for flowlets after the Q -threshold, VLB is used. We refer to this scheme as **HYB**. HYB is an oblivious routing scheme in the sense that it does not decide on routing table configurations based on traffic. It is minimally non-oblivious in the sense that it uses flow size (packets sent so far) to decide whether to switch from ECMP to VLB. However, this is easy to accomplish, e.g., at the hypervisor: once the Q -threshold is reached, packets can be encapsulated to be delivered to an intermediate switch (over ECMP routes to this intermediate), which decapsulates and delivers packets to the destination (again, over ECMP routes from the intermediate to the destination). Similar encaps-decap to achieve VLB has been used in designs which were deployed, like Microsoft’s VL2 architecture [16].

How does one set Q ? It is enough to set Q based on an operator’s notion of “short flow” size, as this ensures short flows (which send

less than Q bytes) use shortest paths, but are also insulated from long running flows, which are load balanced through the entire fabric using VLB. In our experiments, we use $Q=100$ KB.

Our evaluation below shows that this scheme works well across the workloads that we test, including the skewed workloads used in recent data center design papers. Nevertheless, it is useful to acknowledge the **limitations** of this routing scheme: its performance deteriorates if large flows can saturate the network, as VLB uses $2\times$ the capacity per byte compared to ECMP, and for such a high-utilization workload, this bandwidth would be needed elsewhere. Performance also deteriorates if “short flows” are voluminous enough to saturate ECMP bottlenecks. For the scenario of two neighboring racks with only one ECMP path, e.g., at 100 Gbps with 100 KB flows, this would require a concurrent flow rate exceeding 125,000 per second just between these two racks (with all flows hitting this size). This is an order of magnitude larger than reported measurements [8] at any switch, which we can reasonably expect to be much larger than between a specific switch pair. Also note that this is not fundamental, but a consequence of the pragmatic choice of using the simplified Q -threshold in HYB, instead of the congestion-aware method described above, which would sidestep this issue.

There is significant potential for future work to design superior routing (see §7), but our present objective is less ambitious: demonstrating that for workloads published in the literature, even this simple design suffices to match a full-bandwidth fat-tree’s performance at much lower cost.

6.4 Experimental setup

We use a custom packet simulator [1] to evaluate expander-based and fat-tree data center networks under a variety of workloads, and measure flow completion time and throughput.

Topologies: A full-bandwidth fat-tree ($k=16$, 1024 servers, 320 switches, each with 16 10 Gbps ports) serves as the baseline. In line with work on dynamic networks, which claim a 25-40% cost advantage to match the fat-tree’s performance [13, 18], we use Xpander built at 33% lower cost, i.e., a total of 216 switches, each still with 16 ports⁴. This network supports a total of 1080 servers.

Routing and congestion control: We evaluate ECMP and HYB on Xpander, and ECMP on the fat-tree. Both networks use flowlet

⁴An alternative is to fix the number of switches to match the fat-tree’s 128 ToRs, but give each 27 ports instead of 16. This would only favor Xpander.

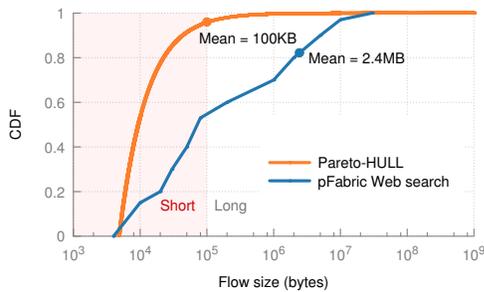


Figure 8: The flow size distributions used in our experiments.

switching. The congestion control mechanism is DCTCP [5]. DCTCP’s ECN marking threshold is set to 20 full-sized packets. The flowlet timeout gap is set to 50 μ s.

Workload: The workload is defined by three factors: a probability distribution capturing flow sizes; another capturing the sources and destinations of flows; and a third for flow arrivals. Throughout, flow arrivals are Poisson distributed, and we present results across a range of arrival rates aggregated across the entire network. The two flow size distributions we use are from past work [6, 7], and are shown in Fig. 8. We experimented with a variety of distributions for communication pairs:

- ProjecToR’s rack-to-rack communication probabilities from a Microsoft data center [13] are used as is, with a particular server within a rack chosen uniformly at random from the rack’s servers. This is a highly skewed workload, with 77% of bytes being transferred between 4% of the rack-pairs.
- A2A(x): A distribution capturing all-to-all traffic restricted to x fraction of racks, with the rest being inactive. For the fat-tree, the first x fraction are used, and for Xpander, a random x fraction. The probability of a flow to start between any pair of servers at active racks is uniform. Facebook’s workload, where “demand is wide-spread, uniform” [28], but not all-to-all, could be approximated with A2A between some fraction of servers.
- Permute(x): A distribution capturing random permutation traffic between x fraction of the racks, with others inactive. The probability of a flow to start between any matched pair of racks is uniform, and zero for pairs that are not matched. This is a challenging workload, because the rack-to-rack consolidation of flows limits opportunities for load balancing.

Experiment framework: We set a total number of flows, F , and a flow arrival rate, λ . At each (Poisson) flow arrival, a source and destination for the flow are picked from a chosen communication pair distribution, and flow size is picked from a chosen flow size distribution. The number of active servers is always the same in any comparisons, and an identical set of flows is run between the active servers by fixing the seed for the random number generator. The statistics are calculated over the flows started in the [0.5s, 1.5s) time interval, and the experiment runs until all flows in this interval finish. The number of flows, F , is chosen such that the simulated time is at least 2 seconds. We calculate average FCT for all flows, 99th percentile FCT for short flows (<100 KB), and average throughput for the rest of the flows. For experiments where λ is

varied, we increase λ until the full-bandwidth fat-tree is persistently overloaded, *i.e.*, more flows arrive per unit time than finish.

6.5 Results: HYB performs well across workloads

Results for the **pFabric flow size distribution** over the A2A and Permute traffic matrices are shown in Fig. 9, 10, and 11. Across these scenarios, Xpander with HYB matches the full-bandwidth fat-tree’s performance for the skewed workloads, *i.e.*, when the fraction of active racks is not large. For the 99th %-ile FCT for short flows, the performance of Xpander matches the fat-tree across nearly the entire range of traffic matrices (Fig.9(b) and 10(b)). As expected, when the fraction of racks involved is large, throughput and overall average FCT with HYB deteriorate. We remind readers that for data centers with skewed traffic, a small fraction of servers are hot.

Fig. 11 also shows the results for the permutation workload (over 31% of racks) with increasing flow arrival rates. Xpander with HYB matches the fat-tree’s performance closely. The analogous set of results for A2A are similar and are omitted.

Included in Fig. 11 is also a “77%-fat-tree”, which is an over-subscribed fat-tree built at 23% lower cost than the full fat-tree, and performance for which begins to deteriorate much earlier. Configuring an over-subscribed fat-tree of 33% lower cost for a more precise comparison was, unfortunately, difficult due to constraints like needing the same number of servers per leaf switch.

Also worth noting is that while ECMP over Xpander performs extremely poorly for Permute (Fig. 10(b)) as expected, it achieves high performance for A2A (Fig. 9(b)). Thus, for uniform-like workloads, even ECMP over Xpander would be sufficient.

Next, we evaluate Xpander using a different flow size distribution: the **Pareto flow-size distribution** from HULL [6]. Most flows in this distribution are small, with the 90th percentile being smaller than 100 KB. Thus, a much larger flow arrival rate is used in our experiments to generate load comparable to our experiments for the pFabric distribution above. For this flow size distribution, all results for Xpander are better than those with the pFabric distribution, so we only include one: the 99th percentile FCT for short flows for the A2A(0.31) in Fig. 12. With small flow sizes, network RTT bounds flow completion time more than bandwidth, and Xpander’s shorter paths result in lower FCT than the full-bandwidth fat-tree.

6.6 Results: Matching the gains of dynamic networks

ProjecToR’s authors compared their design to a full-bandwidth fat-tree [13], configuring ProjecToR with 128 ToRs (same as the fat-tree), each with 16 dynamic network connections (as opposed to 8 network ports per ToR in the fat-tree), and 8 servers (and no other intermediate switches, as opposed to 192 more in the fat-tree). We compare Xpander to the fat-tree in precisely the same configuration with a total of 128 switches, but using 16 *static* network ports per switch, instead of dynamic. Neither topology in this configuration has any other intermediate switches, only the same ToRs with the same port counts. Note that this Xpander is *lower* cost than ProjecToR, using the same number of *static* network ports as ProjecToR uses dynamic ones, without any adjustment for the cost factor, δ . We present results in two settings: one in which server-switch links have the same 10 Gbps capacity as all other links, and another with unconstrained capacity for server-switch

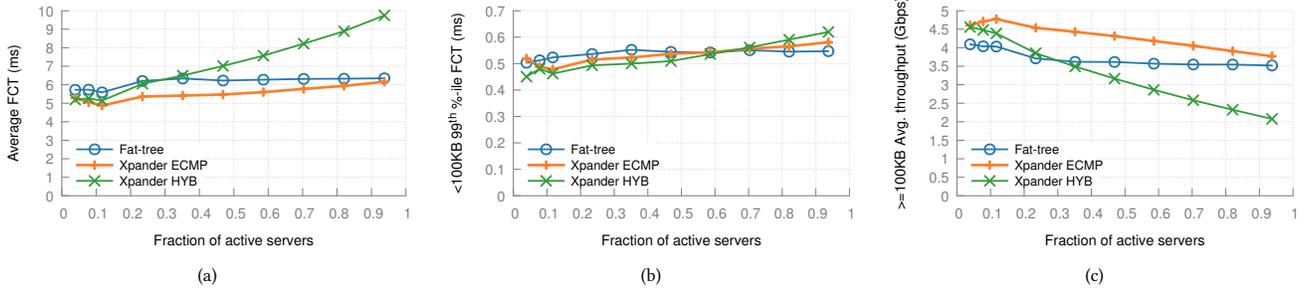


Figure 9: $A2A(x)$ with x increasing on the x axis, with $pFabric$'s flow size distribution and 167 flow arrivals per second per server: (a) average FCT, (b) 99th %-tile FCT for short flows, and (c) average throughput for long flows.

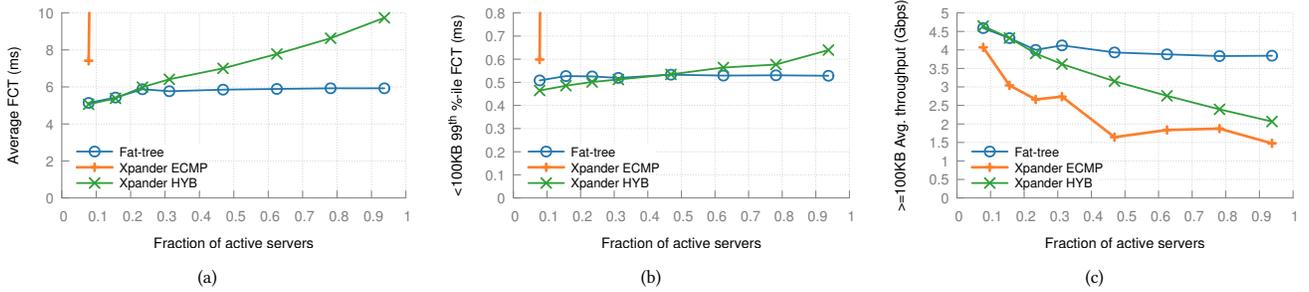


Figure 10: $Permute(x)$ with x increasing on the x -axis, with $pFabric$'s flow size distribution and 167 flow arrivals per second per server: (a) average FCT, (b) 99th %-tile FCT for short flows, and (c) average throughput for long flows.

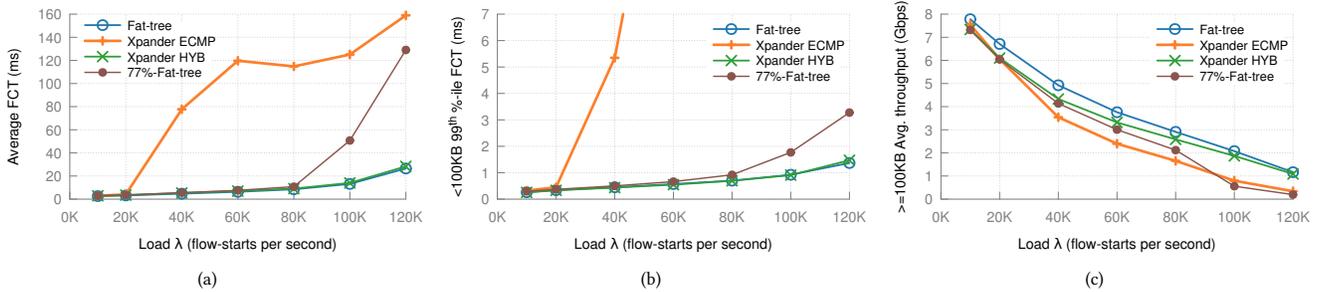


Figure 11: $Permute(0.31)$ with $pFabric$'s flow size distribution and increasing aggregate flow arrival rate on the x -axis: (a) average FCT, (b) 99th %-tile FCT for short flows, and (c) average throughput for long flows. (320 servers out of 1024 form a 0.31 fraction, and this translates to an Integer number of racks.)

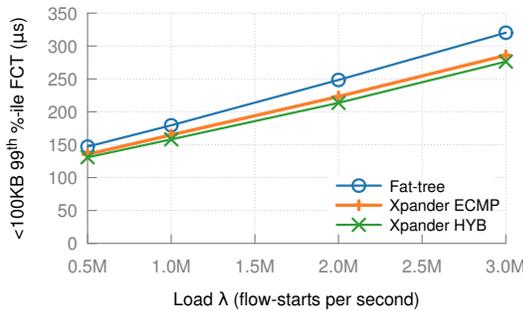


Figure 12: $A2A(0.31)$ with the Pareto-HULL flow size distribution: 99th percentile FCT for short flows.

links. The latter model is used only with reference to ProjecToR (i.e., in this sub-section and in §6.7), as this is how ProjecToR was

evaluated, to effectively model an oversubscribed fat-tree with additional servers underneath each rack. All other experiments in the paper are conducted with server-switch links obeying the same capacity constraints as switch-switch links.

Fig. 13(a) and Fig. 13(b) show the average FCT (over all flows) and 99th percentile FCT for short flows in the same setting as ProjecToR (i.e., ignoring server-level bottlenecks). As flow arrival rate increases, Xpander achieves up to 90% lower average as well as 99th percentile FCT, when using HYB. These gains over the fat-tree are similar to those described for ProjecToR [13]. Thus, compared to an oversubscribed fat-tree, both ProjecToR and Xpander can achieve significant performance improvement. In this comparison, Xpander is built at 30% lower cost than ProjecToR (assuming $\delta = 1.5$, based on the low-end cost estimates from ProjecToR).

When the server-switch link capacity constraint is modeled, Xpander matches the fat-tree's performance, as shown in Fig. 13(c).

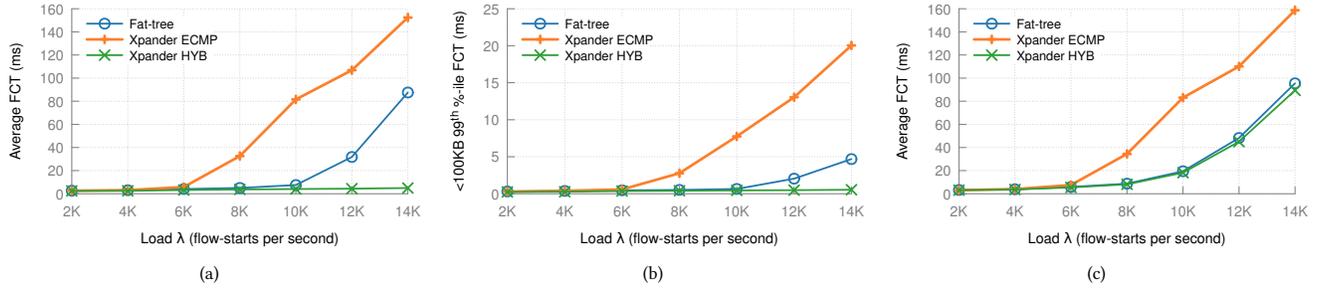


Figure 13: A ProjecToR-like comparison: (a) average and (b) 99th percentile FCT for short flows for Xpander and fat-tree networks, when server-level bottlenecks are ignored; and (c) average FCT when server-level bottlenecks are also modeled.

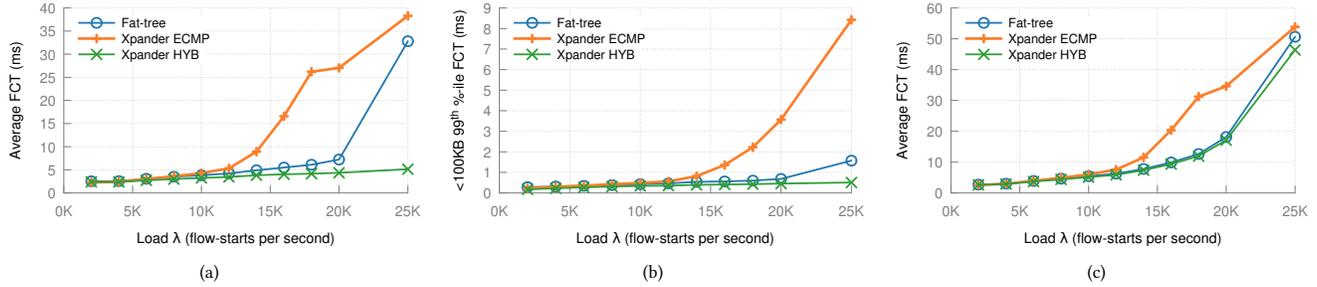


Figure 14: Skew(0.04,0.77) with pFabric’s flow size distribution: (a) average and (b) 99th percentile FCT for short flows for Xpander and fat-tree networks, when server-level bottlenecks are ignored; and (c) average FCT when server-level bottlenecks are also modeled.

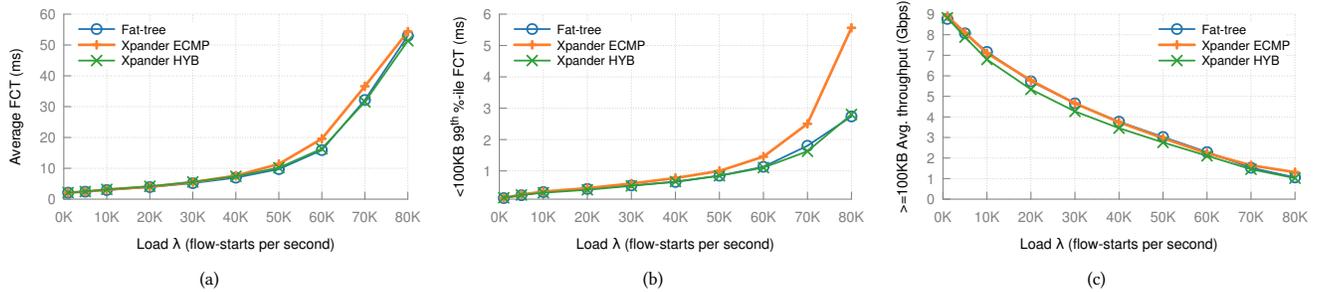


Figure 15: A $k = 24$ fat-tree compared to an Xpander built at only 45% of its cost under the same traffic model as Fig. 14. The range of aggregate flow arrivals per second explored in this experiment is scaled by the same factor as the total number of servers increases compared to the smaller fat-tree in Fig. 14.

In this setting, the full-bandwidth fat-tree is indeed able to provide full bandwidth, and leaves little room for improvement. ProjecToR’s analysis method is thus modeling scenarios where the fat-tree is oversubscribed by adding more servers at the ToR, thus hitting ToR-outlink bottlenecks earlier.

6.7 Results: more on skewed traffic

In the following, we suggest a model for skewed traffic, which allows experimentation with different degrees of skew and at different scales: $Skew(\theta, \phi)$, where θ is the fraction of “hot” racks, and ϕ the fraction of traffic concentrated among these racks. The hot racks, R_{hot} , comprise θ fraction of racks (randomly chosen), the rest being cold racks, R_{cold} . Each rack in R_{hot} has probability of participating in a communication proportional to $\frac{\phi}{|R_{hot}|}$, and each in R_{cold} proportional to $\frac{(1-\phi)}{|R_{cold}|}$. For any rack-pair, the product of these probabilities followed by normalization yields

the probability of communication between them. $Skew(0.04,0.77)$ models a simplification of the ProjecToR traffic matrix. Fig. 14 shows results analogous to the ProjecToR comparison in Fig. 13, but using this simplified ToR-communication probability distribution; these are largely similar to those in Fig. 13.

We also test $Skew(0.04,0.77)$ at larger scale, with a $k=24$ fat-tree, and an Xpander built at only 45% of its cost, using 322 switches of 24 ports each, instead of the fat-tree’s 720 switches. The results are shown in Fig. 15. (These experiments do incorporate the server-switch link capacity constraints.) ECMP over Xpander performs better for this larger topology, although its performance still deteriorates at high flow rates, when short flows can saturate the shortest-path bottlenecks. Xpander with HYB matches the fat-tree’s performance. In line with scaling results from the fluid-flow model (§5), Xpander’s cost-efficiency improves with scale.

Thus, across the workloads tested, when the fraction of hot servers is not large, an expander-based network built at 33% lower cost can match a full-bandwidth fat-tree’s performance using simple routing. For uniform-like traffic, ECMP on such networks is enough to match a full-bandwidth fat-tree’s performance at lower cost.

7 Lessons and future directions

Our results show that not only is topology dynamism not essential to matching the performance of full-bandwidth fat-trees at lower cost, this can be achieved even without dynamic optimization of routing. This surprising result nevertheless leaves many intriguing questions wide open for both static and dynamic networks:

7.1 Static networks

While our results already show substantial gains over fat-trees, we have only scratched the surface of routing possibilities on expander-based networks. Results in the fluid-flow model reveal potential benefits much larger than what our simple-by-design routing scheme achieves — throughput gains of more than 3× over equal-cost oversubscribed fat-trees (Fig. 5), and matching the performance of full-bandwidth fat-trees at 50% lower cost (Fig. 6).

Several important questions thus remain unanswered:

- What is the best oblivious routing scheme for expander-based networks?
- Does exploiting *only* the knowledge that the TM is skewed (e.g., only α fraction of servers are “hot”) help an otherwise oblivious scheme?
- How much can performance be further improved by adaptive routing? For fat-trees, schemes like CONGA [4] provide significant improvements. Do the same methods work well for expander-based networks?

A better understanding of the throughput proportionality metric would also be valuable, including resolving the conjectures in §2.

7.2 Dynamic networks

Prior research on dynamic data center networks indeed achieves significant improvements over fat-trees, but as argued above, these improvements do not reflect an inherent advantage of incorporating dynamic connectivity in the topology.

We argue that demonstrating such an advantage requires:

- ... comparing to expander-based static networks
- ... at equal cost
- ... using more expressive routing than ECMP, and
- ... accounting for added latency from topology dynamics.

As we have shown in §2.1, fat-trees are nearly ideally *inflexible* towards skewed traffic, and thus an easy baseline, especially compared to expander-based networks such as Xpander [33] and Jellyfish [31]. Further, even expander-based networks require the use of more expressive routing than just ECMP to provide their efficiency improvements (§6). Using ECMP handicaps static networks in comparisons with dynamic designs where sophisticated routing and topology optimizations are used. Also, comparing networks without equalizing cost simply does not yield any

actionable information⁵. Specifically, if a dynamic network design under consideration uses x ports that cost δ times the cost of a “static port” (δ varies across technologies), the point of comparison should be an expander-based design with δx ports.

In many dynamic designs, end-hosts often need to wait until connectivity between them is available, leading to additional latency and buffering. Accounting for these delays and their interplay with congestion control is essential for meaningful comparisons.

Our simulation framework is available [1] as an easy-to-use baseline for future work on dynamic networks to compare against.

8 Recent developments in dynamic networks

The past decade has produced a large number of novel and interesting proposals for data center network topologies, both static (§3) and dynamic (§4). At least three new proposals have been made concurrently with this work [11, 26, 37], which we discuss below.

Flat-tree [37] proposes the use of small additional switches in the topology which function as “converters”, making some connections more “local” when needed, in an effort to match Jellyfish’s performance while preserving structure. However, Xpander can achieve these objectives without additional converter switches.

MegaSwitch [11] proposes the use of wavelength division multiplexing over a set of racks arranged in an optical ring, which limits its present design to 33 racks. MegaSwitch does not include a crisp cost comparison as the optical components needed (e.g., transceivers) are not standard, commodity equipment.

RotorNet [26] is a novel dynamic topology proposal that differs from the prevalent approach to the design of dynamic data centers by not relying on any dynamic optimization in response to traffic estimation. Rather, RotorNet cycles through a series of pre-determined optical port matchings in a traffic agnostic manner. Investigating rigorously, following §7, whether RotorNet outperforms state-of-the-art static networks is deferred to future research. While RotorNet alleviates some of the problems past approaches faced, it still involves nontrivial challenges, such as accommodating latency-sensitive traffic.

Thus, even the most recent dynamic designs have not yet demonstrated an advantage over expander-based static networks.

9 Conclusion

Our results show that state-of-the-art static networks designed using commodity data center network equipment and employing simple routing protocols provide the same cost-efficiency advantages over full-bandwidth fat-trees as claimed by recent proposals on dynamic networks. We believe that investigating and advancing the deployability of these static network designs in practice is a promising approach for moving beyond today’s prevalent data center architectures. We argue that for dynamic networks to be compelling, they should demonstrate substantial improvements over these static networks when compared at equal cost.

Acknowledgments

We would like to thank Ratul Mahajan for his insights on the limitations of the restricted topology adaptation model. We are

⁵Equivalently, one could match performance and quantify the cost difference.

also grateful to our colleagues who provided helpful feedback on this work, including Monia Ghobadi, Torsten Hoefler, George Porter, Marcel Schneider, Laurent Vanbever, and the anonymous SIGCOMM reviewers and shepherd. Asaf Valadarsky is supported by a Microsoft Research Ph.D. Scholarship. Michael Schapira is supported by the PetaCloud Consortium.

References

- [1] Netbench. (2017). <https://github.com/ndal-eth/netbench>.
- [2] Dennis Abts, Michael R Marty, Philip M Wells, Peter Klausler, and Hong Liu. Energy Proportional Datacenter Networks. *ACM/IEEE ISCA* (2010).
- [3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture. *ACM SIGCOMM* (2008).
- [4] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. *ACM SIGCOMM* (2014).
- [5] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). *ACM SIGCOMM* (2010).
- [6] 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. *USENIX NSDI* (2012).
- [7] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pFabric: Minimal Near-optimal Datacenter Transport. *ACM SIGCOMM* (2013).
- [8] Theophilus Benson, Aditya Akella, and David A Maltz. Network Traffic Characteristics of Data Centers in the Wild. *ACM SIGCOMM* (2010).
- [9] M. Besta and T. Hoefler. Slim Fly: A Cost Effective Low-Diameter Network Topology. *IEEE/ACM SC* (2014).
- [10] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. *USENIX NSDI* (2012).
- [11] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. Enabling Wide-Spread Communications on Optical Fabric with MegaSwitch. *USENIX NSDI* (2017).
- [12] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiah Fainman, George Papan, and Amin Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. *ACM SIGCOMM* (2010).
- [13] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Houman Rastegarfar, Pierre-Alexandre Blanche, Madeleine Glick, Daniel Kilper, Janardhan Kulkarni, Gireeja Ranade, and Nikhil Devanur. ProjecToR: Agile Reconfigurable Datacenter Interconnect. *ACM SIGCOMM* (2016).
- [14] Madeleine Glick, David G Andersen, Michael Kaminsky, and Lily Mummert. Dynamically Reconfigurable Optical Links for High-bandwidth Data Center Networks. *Optical Fiber Communication Conference* (2009).
- [15] Google. Pulling Back the Curtain on Google's Network Infrastructure. (2015). <https://goo.gl/hx0vz3>.
- [16] 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. *ACM SIGCOMM* (2009).
- [17] Daniel Halperin, Srikanth Kandula, Jitendra Padhye, Paramvir Bahl, and David Wetherall. Augmenting Data Center Networks with Multi-Gigabit Wireless Links. *ACM SIGCOMM* (2011).
- [18] 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. *ACM SIGCOMM* (2014).
- [19] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. ElasticTree: Saving Energy in Data Center Networks. *USENIX NSDI* (2010).
- [20] Sangeetha Abdu Jyothi, Ankit Singla, Brighten Godfrey, and Alexandra Kolla. Measuring and Understanding Throughput of Network Topologies. *IEEE SC* (2016).
- [21] Sangeetha Abdu Jyothi, Ankit Singla, Chi-Yao Hong, Lucian Popa, Brighten Godfrey, and Alexandra Kolla. Topobench. (2016). <https://github.com/netarch/topobench/>.
- [22] Srikanth Kandula, Dina Katabi, Shan Sinha, and Arthur Berger. Flare: Responsive Load Balancing Without Packet Reordering. *ACM CCR* (2007).
- [23] John Kim, William J. Dally, Steve Scott, and Dennis Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. *ACM SIGARCH* (2008).
- [24] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. Quartz: A New Design Element for Low-Latency DCNs. *ACM SIGCOMM* (2014).
- [25] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan Graphs. *Combinatorica* (1988).
- [26] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papan, Alex C. Snoeren, and George Porter. RotorNet: A Scalable, Low-complexity, Optical Datacenter Network. *ACM SIGCOMM* (2017).
- [27] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiah Fainman, George Papan, and Amin Vahdat. Integrating Microsecond Circuit Switching into the Data Center. *ACM SIGCOMM* (2013).
- [28] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the Social Network's (Datacenter) Network. *ACM SIGCOMM* (2015).
- [29] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannan, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. *ACM SIGCOMM* (2015).
- [30] Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. High Throughput Data Center Topology Design. *USENIX NSDI* (2014).
- [31] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking Data Centers Randomly. *USENIX NSDI* (2012).
- [32] Ratko V. Tomic. Optimal Networks from Error Correcting Codes. *ACM/IEEE ANCS* (2013).
- [33] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards Optimal-Performance Datacenters. *ACM CoNEXT* (2016).
- [34] Erico Vanini, Rong Pan, Mohammad Alizadeh, Tom Edsall, and Parvin Taheri. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. *USENIX NSDI* (2017).
- [35] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T. S. Eugene Ng, Michael Kozuch, and Michael Ryan. c-Through: Part-time Optics in Data Centers. *ACM SIGCOMM* (2010).
- [36] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. *USENIX NSDI* (2011).
- [37] Yiting Xia and TS Eugene Ng. Flat-tree: A Convertible Data Center Network Architecture from Clos to Random Graph. *ACM HotNets* (2016).
- [38] J.Y. Yen. Finding the K Shortest Loopless Paths in a Network. *Management Science* (1971).
- [39] Rui Zhang-Shen and Nick McKeown. Designing a Predictable Internet Backbone with Valiant Load-balancing. *IEEE IWQoS* (2005).
- [40] 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* (2012).