Xpander: Towards Optimal-Performance Next-Generation Datacenters

12 pages

Asaf Valadarsky*
asaf.valadarsky@mail.huji.ac.il

Gal Shahaf*
gal.shahaf@mail.huji.ac.il

Michael Dinitz†
mdinitz@cs.jhu.edu

Michael Schapira*
schapiram@huji.ac.il

ABSTRACT

Despite extensive efforts to meet ever-growing demands, today's datacenters often exhibit far-from-optimal performance in terms of network throughput and utilization, resiliency to failures, cost efficiency, incremental expandability, and more. Consequently, many novel architectures for high performance datacenters have been proposed. We first observe that the benefits of state-of-the-art proposals are, in fact, derived from the fact that these are (implicitly) utilizing "expander graphs" (aka expanders) as their network topologies, thus unveiling a unifying theme of these proposals. We show, however, that these proposals are not optimal with respect to performance, do not scale, or suffer from seemingly insurmountable deployment challenges. We leverage these insights to present Xpander, a novel datacenter architecture that achieves near-optimal performance and provides a tangible alternative to existing datacenter designs. Xpander's design turns ideas from the rich graph-theoretic literature on constructing optimal expanders into an operational reality. We evaluate Xpander via theoretical analyses, extensive simulations, experiments with a network emulator, and an implementation on an SDN-capable network testbed. Our results demonstrate that Xpander significantly outperforms both traditional and proposed datacenter designs. We discuss challenges to real-world deployment and explain how these can be resolved.

1. INTRODUCTION

The rapid growth of Internet services is placing tremendous demands on datacenters. Yet, as evidenced by the extensive research on improving datacenter performance [22, 23, 5, 45, 49, 21, 43], today's datacenters often exhibit far-from-optimal performance in terms of network utilization, resiliency to failures, cost efficiency, amenability to incremental growth, and more.

1.1 Unveiling the Secret to High Performance

We first observe that state-of-the-art proposals for next-generation datacenters, e.g., designing low-diameter networks (a la Slim Fly [8]) and randomly networked datacenters (a la Jellyfish [45]), have an implicit unifying theme: utilizing an "expander graph" (aka expander [24]) as the network topology and exploiting the diversity of short paths afforded by expanders for efficient delivery of data traffic. Hence, such proposals can be viewed as points in a much larger space of "expander datacenters". We show, however, that these points are either not sufficiently close to optimal performance-wise, are inherently not scalable, or face significant deployment and maintenance challenges (e.g., in terms of unpredictability and wiring complexity).

We thus argue that the quest for optimal-performance datacenter designs is inextricably intertwined with the rich body of research in mathematics and computer science on building good expanders. We aim to identify a point in this design space that offers *near-optimal performance* guarantees while providing a *tangible* alternative for today's datacenters (e.g., in terms of cabling costs and complexity, physical layout, backwards compatibility, and more). We present Xpander, a novel expander-datacenter architecture carefully engineered to accomplish this. We next elaborate on expanders and Xpander.

1.2 Why Expanders?

Intuitively, in an expander graph the total capacity from any set of nodes S to the rest of the network is large with respect to the size of S. We present a formal definition of expanders in Section 2. Since this implies that in an expander every cut in the network is traversed by many links, traffic between nodes is (intuitively) never bottlenecked at a small set of links, leading to good throughput guarantees. Similarly, as every cut is large, every two nodes are (intuitively) connected by many edge-disjoint paths, leading to high resiliency to failures. Our first contribution is formalizing these intuitions in the *datacenter-specific* context and validating them through a combination of theoretical analyses, simulations, experiments with a network emulator, and experiments on a network of SDN switches.

Constructing expanders is a prominent research thread in both mathematics and computer science. While random graphs are provably good expanders (in expectation [10, 18]), building well-structured, deterministic expanders is far more

*School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel
†Department of Computer Science, Johns Hopkins University, USA

challenging and is the subject of extensive attention (see, e.g., Margulis's construction [32], algebraic constructions [30], and constructions that utilize the so-called "zig-zag product" [41]). In our quest for the "right" point in this design space, our aim is to first identify a specific construction of expanders that both generates near-optimal expanders and is feasible from a practical perspective (scalability, complexity, etc.). We must also grapple with the challenge of identifying backwards-compatible routing and congestion control schemes that take advantage of the path diversity inherent to such a construction.

1.3 Why Xpander?

Our construction of Xpander's datacenter topology utilizes the graph-theoretic notion of "lifting" a graph [9, 14]. We evaluate Xpander through formal analyses, extensive and highly optimized flow-level and packet-level simulations, experiments with the mininet network emulator, and an implementation on an SDN-capable network testbed (OCEAN [3]).

Xpander robustly achieves high performance. Our results reveal that Xpander achieves near-optimal bisection bandwidth and all-to-all throughput (used to quantify the performance of datacenter topologies in [45, 44, 25]). Xpander significantly outperforms traditional datacenters (fat trees [5]). Our results, in fact, indicate that Xpander matches the performance of today's datacenters with roughly $80 - 85\%$ of the switches. We show, moreover, that Xpander achieves the same performance as random networks, the current state-of-the-art with respect to performance [45, 44, 25] (but which suffer from severe impediments to deployment, as discussed later), and outperforms Slim-Fly [8].

Beyond the above improvements in performance, our results establish that Xpander is significantly more robust than today's datacenters to network changes. Studies of datacenter traffic patterns reveal tremendous variation in traffic over time [6]. Unfortunately, a network topology that fares well with respect to throughput in one traffic scenario (e.g., all-to-all) might fail miserably in other scenarios. We show that Xpander is robust to variations in traffic. Specifically, Xpander provides close-to-optimal guarantees with respect to *any* (even adversarially chosen!) traffic pattern. We show, moreover, that *no* other network topology can do better. Xpander's performance also degrades much more gracefully than fat trees under network changes due to failed equipment.

Alongside the above merits, Xpander, unlike today's rigid datacenter networks, can be incrementally expanded to any size while preserving its high performance, thus meeting the need of companies (such as Google, Facebook and Amazon) to constantly and incrementally grow existing datacenters. Our experiments also show that Xpander exhibits lower average (shortest) path lengths and diameters than today's datacenters.

Overcoming challenges to deployment. We analyze the challenges facing the deployment of Xpander in practice through detailed investigations of various scenarios (from "container datacenters" to large-scale datacenters). Our analyses provide evidence that Xpander is realizable with monetary and power consumption costs that are comparable or lower than those of today's prevalent datacenter architectures and, moreoever, that its inherent well-structuredness and order render wiring Xpanders manageable (avoiding, e.g., the inherent unstructuredness and complexity of random network designs a la Jellyfish [45]).

1.4 Organization

We provide a formal exposition of expanders and expander datacenters in Section 2, and of Xpander in Section 3. We show that Xpander indeed attains near-optimal performance in Section 4.2, matching the performance of randomly networked datacenters (the current state-of-the-art with respect to performance). We compare Xpander to fat trees [5] and Slim Fly [8] in Sections 5 and 6, respectively. We discuss deployment challenges and solutions in Section 7, and related work in Section 8. We conclude in Section 9.

2. EXPANDER DATACENTERS

We provide below a formal exposition of expanders. We discuss past proposals for novel high-performance datacenter architectures and explain why they are, in fact, "expander datacenters".

2.1 Expanders

Consider an (undirected) graph $G = (V, E)$ (where V and E are the vertex set and edge set, respectively). For any subset of vertices S, let $|S|$ denote the size of S, let $\partial(S)$ denote the set of edges leaving S, and let $|\partial(S)|$ denote the size of $\partial(S)$. Let n denote the number of vertices in V (that is, $|V|$). The *edge expansion* $EE(G)$ of a graph G on n is $EE(G) = \min_{|S| \leq \frac{n}{2}} \frac{|\partial(S)|}{|S|}$. We say that a graph G is *d-regular* if the degree of each vertex is exactly d. We call a d-regular graph G an *expander* if $EE(G) = c \cdot d$ for some constant $c > 0$.

2.2 Expander-Based Datacenters

We refer to datacenters that rely on an expander graph as their network topology as "expander datacenters". We consider two recent approaches to datacenter design that have been shown to yield significantly better performance than both today's datacenters and many other proposals: low-diameter networks, e.g., Slim Fly [8], and randomly networked datacenters, e.g., Jellyfish [45]. We observe that these two designs are, in fact, expander datacenters. We show in the subsequent sections that this is indeed what accounts for their good performance.

Jellyfish. Classical results in graph theory establish that random graphs are remarkably good expanders [10, 18]. It was shown recently in [45, 44] that randomly networked datacenters achieve close-to-optimal performance guarantees. In-

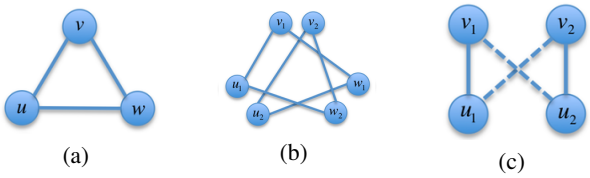


Figure 1: Illustration of 2-Lift

deed, to date, randomly networked datacenters are the state-of-the-art, in terms of performance.

Unfortunately, the inherent unstructuredness of random graphs makes them hard to reason about (predict, diagnose, etc.) and to build (e.g., in terms of wiring complexity), and thus poses serious, arguably insurmountable, obstacles to their adoption in practice. Worse yet, as with any probabilistic construct, the good traits of random graphs are guaranteed only with *some* probability. Hence, while utilizing random network topologies in datacenters is an important thought experiment, the question arises: Can a *well-structured* and *deterministic* construction achieve the same benefits *always* (and not probabilistically)? We show that Xpander indeed accomplishes this.

Slim Fly. Slim Fly [8] (SF) leverages a graph-theoretic construction of low-diameter graphs [33] (of diameter 2 or 3). Intuitively, by decreasing the diameter, less capacity is wasted when sending data, and hence overall performance is higher. [8] shows that SF outperforms existing and proposed datacenter architectures, but performs worse than random topologies.

We prove, in Section 6, that SF is a good expander. Our results for expander datacenters thus suggest that SF's good edge expansion is, in fact, the explanation for its good performance, and *not* its low diameter. Indeed, our findings (see Section 6) suggest that by optimizing the diameter-size tradeoff, Slim Fly sacrifices a small amount of expansion, which leads to worse performance than random networks (and Xpander) as the network gets larger. Worse yet, as explained in Section 6, the low diameter of SF imposes an extremely strict condition on its size (as a function of port count of each switch), imposing, in turn, a strict limit on the scalability of Slim Fly. Xpander, in contrast, can be constructed for virtually any given swith port-count and network size.

3. XPANDER: OVERVIEW AND DESIGN

In light of the above limitations of past proposals, our aim is to identify a datacenter architecture that achieves near-optimal performance yet overcomes deployment challenges. We next present the Xpander datacenter design. We show in the following sections that Xpander indeed accomplishes these desiderata.

3.1 Network Topology

Lifting a graph. Consider the graph G depicted in Figure 1(a). Our construction of Xpander leverages the idea of "lifting" a graph [9, 14]. We start by explaining *2-lifts*. A 2-lift of G is a graph obtained from G by (1) creating two vertices v_1 and v_2 for every vertex v in G; and (2) for every edge $e = \{u, v\}$ in G, inserting two edges (a matching) between the two copies of u (namely, u_1 and u_2) and the two copies of v (namely, v_1 and v_2). Figure 1(b) is an example of a 2-lift of G. Observe that the pair of vertices v_1 and v_2 can be connected to the pair u_1 and u_2 in two possible ways, described by the solid and dashed lines in Figure 1(c). [9] proves that if the original graph G is an expander, the 2-lift of G obtained by choosing between every two such options at random is also an expander. [9] also shows how these simple random choices can be *derandomized*, i.e., how to achieve the same guarantee in a deterministic manner.

The idea of 2-lifting can be generalized to k-lifting for an arbitrary choice of k in a straightforward manner: create, for every vertex v in G, k vertices, and for every edge $e = \{u, v\}$ in G, insert a matching between the k copies of u and the k copies of v. As with 2-lifting, k-lifting an expander graph via random matchings results in a good expander [14, 17]. We show (empirically) that this, too, can be derandomizedsee Appendix A

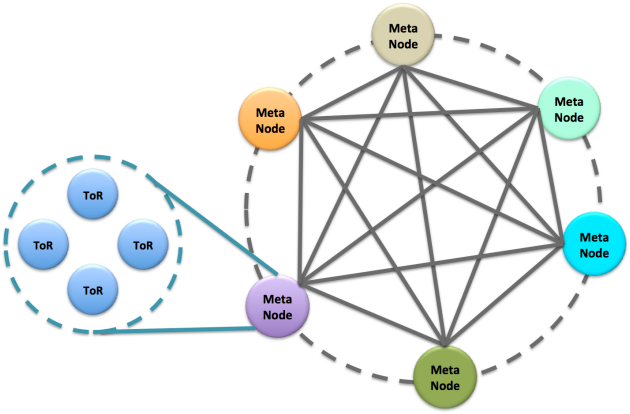


Figure 2: An Xpander topology sketch

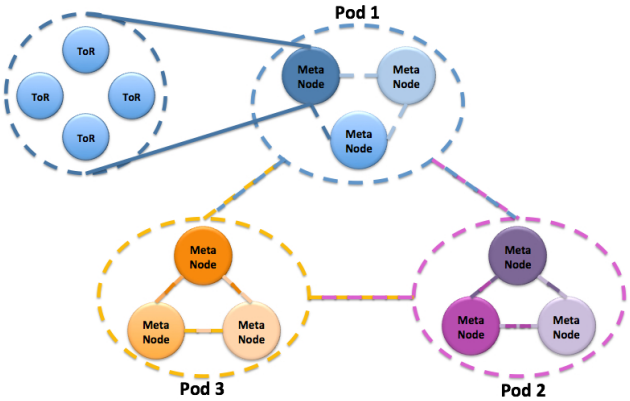


Figure 3: Division of an Xpander into Xpander-pods.

Xpander's network topology. To construct a d-regular Xpander network, where each node (vertex) represents a top-of-rack (ToR) switch, and d represents the number of ports per switch used to connect to other switches (all other ports are connected to servers within the rack), we simply do the following: we start with the complete d-regular graph on $d+1$ vertices, and then repeatedly lift this graph in a manner that preserves expansion (as explained above). We point out that almost all of our constructions of Xpander topologies below involve only 2-lifts.

Although lifting a graph (at least) doubles the number of nodes, we show in Section 4.4 how Xpander topologies can be incrementally grown (a single node at a time) to any desired number of nodes while retaining good expansion.

3.2 Xpander's Logical Organization

Observe that, as described in Figure 2, an Xpander network can be regarded as composed of multiple "meta nodes" such that (1) each meta-node consists of the same number of ToRs (2) every two meta-nodes are connected via the same number of links, and (3) no two ToRs within the same meta-node are directly connected. Each meta-node is, essentially, the group of nodes which correspond to one of the original $d+1$ nodes. Also, an Xpander can naturally be divided into smaller Xpanders ("Xpander-pods"), each of the form depicted in Figure 2, such that each pod is simply a collection of meta-nodes. See Figure 3 for an illustration. Observe that division of an Xpander into Xpander-pods need not be into pods of the same size.

We show in Section 7 how this clean logical structure of Xpander networks can be leveraged to tame cabling complexity.

3.3 Routing and Congestion Control

To exploit Xpander's rich path diversity, traditional routing with ECMP and TCP congestion control are insufficent. Xpander thus, similarly to [45], employs multipath routing via K-shortest paths [50, 16] and MPTCP congestion control [48]. K-shortest paths can be implemented in several ways, including OpenFlow rule matching [34], SPAIN [35], and MPLS tunneling [42].

4. NEAR-OPTIMAL PERFORMANCE

We show that Xpander achieves near-optimal performance in terms of throughput and bisection bandwidth guarantees, robustness to traffic variations, resiliency to failures, incremental expandability, and path lengths.

Importantly, both our simulations and theoretical results benchmark Xpander against a (possibly unattainable) upper bound on performance achievable by *any* datacenter network. To benchmark also against the state-of-the-art, in terms of performance, we also show that Xpander matches the outstanding performance of random datacenter architectures, demonstrated in [45, 44]. We will later (Section 7) explain how Xpander's design mitigates the severe deployment challenges facing randomly networked datacenters.

We point out that while our experimental results are Xpander-specific, our theoretical results can be extended to *any* expander datacenter, and so also shed light on the good performance of past proposals that fall into this category (e.g., [45, 8]). We show in Section 6, however, that Xpander also outperforms recently proposed expander datacenters, namely, Slim Fly [8].

4.1 Near-Optimal Bisection Bandwidth

Recall that bisection bandwidth is the minimum number of edges (total capacity) traversing a cut whose sides are of *equal* size [51], or formally $\min_{S:|S|=\frac{n}{2}} |\partial(S)|$ (see relevant notation in Section 2). As expanders intuitively guarantee that *all* cuts are large, not surprisingly, good expanders (such as Xpander and random graphs) are guaranteed to have good bisection bandwidth. We prove that Xpander indeed achieves near-optimal bisection bandwidth. Note that the bisection bandwidth of any d-regular graph is at most $\frac{n}{2} \cdot \frac{d}{2} = \frac{nd}{4}$.[1]

THEOREM 4.1. *An Xpander graph on n vertices has bisection bandwidth at least $\frac{n}{2}\left(\frac{d}{2} - O(d^{3/4})\right)$.*

PROOF. We know from [17] that any Xpander graph has edge expansion at least $\frac{d}{2} - O(d^{3/4})$. Thus, as bisection bandwidth measures the number of edges across cuts that divide the set of n nodes into two sets of size $\frac{n}{2}$, it is at least $\frac{n}{2}\left(\frac{d}{2} - O(d^{3/4})\right)$. □

We next turn out attention to analyzing throughput directly.

4.2 Near-Optimal Throughput

4.2.1 Simulation Framework

Simulated networks. We ran simulations on Xpander networks for many choices of number of nodes n and node-degree d. We point out that the node degree d in our simulations refers only to the number of ports at a switch used to connect to other switches and not to switch-to-server ports. We shall henceforth use k to refer to the total number of ports at each switch (used to connect to either other switches or server). We tested every even degree in the range 6-30 and up to 600 switches (and so thousands of servers) using a flow-level simulator (see below). As our simulations show the exact same trends for all choices of parameters, we display figures for selected few choices of n and d. To validate that Xpanders indeed achieve near-optimal performance, we benchmarked Xpanders also against Jellyfish's performance (averaged over 10 runs), shown to be near-optimal in [45, 44]. We also simulate large Xpander networks, the largest supporting 27K servers, using the MPTCP packet simulator [2].

Computing throughputs. We compute the following values for every network topology considered: (1) the maximum all-to-all throughput, that is, the maximum amount

[1]That is what is achieved by a random bisection and thus the worst bisection is no better.

of flow-level traffic α that can be concurrently routed between every two switches in the network without exceeding link capacities (see formal definition in Section 4.2.3); (2) the flow-level throughput under skewed traffic matrices (elephants and mice); and (3) the throughput under K-shortest paths [50] and combined with MPTCP [48].

Intuitively, (1)+(2) capture the maximum flow achievable when the full path diversity of Xpanders (and Jellyfish) can be exploited, whereas (3) captures the packet-level performance under Xpander's specific routing and congestion control schemes. Thus, our simulations quantify both the best (flow-level) throughput achievable *and* how closely Xpander's routing and congestion control protocols approach this optimum.

To compute (1)+(2), our simulations ran the CPLEX optimizer [1] on a 64GB, 16-cores server. Our simulation framework is highly-optimized, allowing us to investigate datacenter topologies with significantly higher parameter values (numbers of switches n, servers, and per switch port counts d) than past studies. To compute the throughput under K-shortest paths and MPTCP we use the MPTCP packet-level simulator [2]. We later (Section 5.3) validate these results using the mininet network emulator [27].

4.2.2 Simulations

Figure 4 describes our representative results for all-to-all throughput (the y-axis) as a function of the number of servers in the network (the x-axis) when the switch's inter-switch degree (ports used to connect to other switches) is $d = 6$ (left) and $d = 10$ (right). The results are normalized by the theoretical (possibly unattainable) upper bound on the throughput of *any* network [44]. Clearly the achieved performance is close to the (possibly unattainable) theoretical optimum. Note that there is a dip in the performance, but as explained in [44], this is a function of how the upper bound, which at this point becomes unattainable, is calculated. Even here, however, both Xpander and Jellyfish remain fairly close to the (unattainable) upper bound. We show in the subsequent sections that this level of performance is above that of both today's datacenters and Slim Fly [8].

We also measured, using the MPTCP packet-level simulator [2], the average per-server throughput (as a percentage of the servers' NIC rate) for different choices of parameter K in K-shortest paths, (1) the throughput when the number of MPTCP subflows is 8 (the recommended value [45, 48]), and (2) the throughput when the number of subflows equals K. Our results show that when $K \geq 6$ and the number of MPTCP subflows equals K, the server average throughput is very close to its maximum outgoing capacity. We present our representative results in Figure 5, where we use $d = 30$ and up to 496 switches (or nearly 3K servers). The results for other choices of d and n exhibit the same trends.

We also evaluated the throughput of Xpander for "skewed traffic matrices", where each of T randomly-chosen pairs of nodes wishes to transmit a large amount of traffic, namely β times units of flow, and all other pairs only wish to exchange a single unit of flow (as in the all-to-all scenario). We simulated this scenario for network size $n = 250$, every even $d = 2, 4, \ldots, 24$, and all combinations of $T \in \{1, 6, 11, \ldots, 46\}$ and $\beta \in \{4, 40, 400\}$.

We computed, for each choice of parameters, the network throughput α, that is, the maximum fraction of each traffic demand that can be sent through the network concurrently without exceeding link capacities. The results are again normalized by a simple theoretical (and unattainable) upper bound on *any* network's throughput for these traffic demands (calculation omitted). Our simulation results for skewed traffic matrices show that the throughput achieved by Xpander is almost always (over 96% of results) within 15% of the (unattainable) upper bound on the optimum throughput, and typically within $5 - 10\%$ from the theoretical optimum. See Table 1 for a breakdown of the results. We use the MPTCP simulator to compare Xpander vis a vis fat tree in Section 5, showing that Xpander provides the same level of performance with much fewer switches.

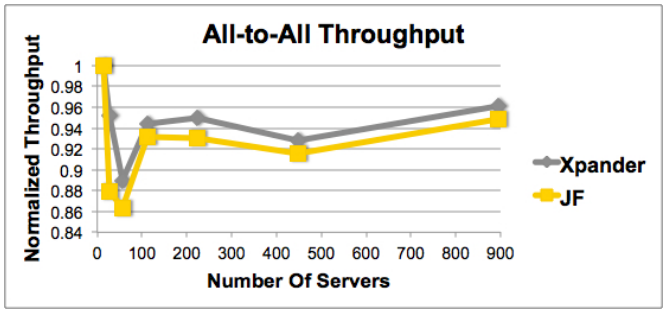
4.2.3 Theory

We next consider the following simple fluid-flow model of network throughput [25]: A network is represented as a capacitated graph $G = (V, E)$, where vertex set V represents (top-of-rack) switches, and edge set E represents switch-to-switch links. All edges have a capacity of 1. A *traffic matrix* T specifies, for every two vertices (switches) $u, v \in V$, the total amount of requested flow $T_{u,v}$ from servers connected to u to servers connected to v. The *network throughput* under traffic matrix T is defined as the maximum value α such that $\alpha \cdot T_{u,v}$ flow can be routed *concurrently* from each vertex u to each vertex v without exceeding the link capacities. For a graph G and traffic matrix T, let $\alpha(G, T)$ denote the throughput of G under T. We refer to the scenario in which $T_{u,v} = 1$ for every $u, v \in V$ (i.e. every node aims to send 1 unit of flow to every other node) as the "*all-to-all setting*". We will slightly abuse notation and let $\alpha(G)$ denote the throughput of G in the all-to-all setting.

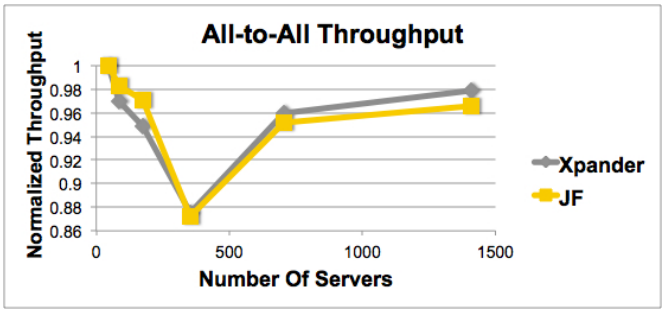
We present several simple-to-prove results on Xpander's throughput guarantees. We point out that all these results can be extended to all "good enough" expanders and thus also account for the good performance of other expander datacenters (a la Jellyfish and Slim Fly). While possibly folklore, these results do not appear to have been stated or proven previously (to the best of our knowledge). We thus include them here for completeness.

THEOREM 4.2. *In the all-to-all setting, the throughput of a d-regular Xpander G on n vertices is within a factor of $O(\log d)$ of that of the throughput-optimal d-regular graph on n vertices.*

PROOF. Let G^* be an arbitrary d-regular graph on n vertices. A simple argument (omitted) shows that in any d-regular graph, from any node u there are $\Omega(n)$ other nodes at distance at least $\Omega(\log_d n)$ from u. Call these nodes $F(u)$.

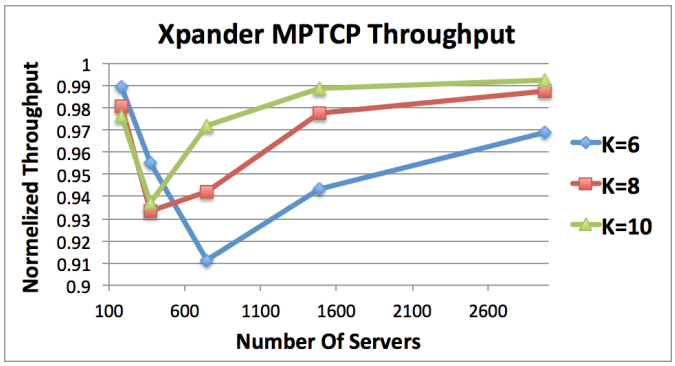


(a) $k = 8$. There are 2 servers placed under each switch

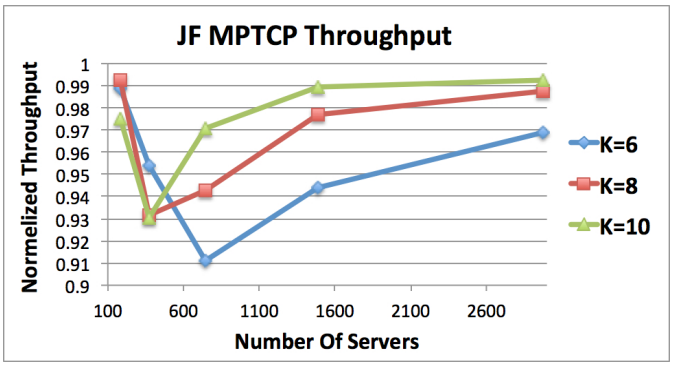


(b) $k = 14$. There are 4 servers placed under each switch.

Figure 4: Results for all-to-all throughput



(a) Xpander



(b) Jellyfish

Figure 5: Results for K-Shortest & MPTCP with $K = \#Subflows$. $k = 36$, there are 6 servers placed under each switch.

Distance from Optimum	Xpander	JellyFish
throughput<80%	<1%	<1%
80% ≤ throughput <85%	2.3%	2.3%
85% ≤ throughput <90%	16.14%	16.14%
90% ≤ throughput <95%	44.48%	48.03%
95% ≤ throughput	36.61%	32.67%

Table 1: Distance of throughput from the (unattainable) optimum for various combinations of β, T, d.

If we send $\alpha(G^*)$ flow between each pair of nodes, the total capacity used (i.e., the sum over all edges of the flow traversing the edge) is at least $\sum_{u\in V}\sum_{v\in F(u)}\alpha(G^*)\cdot\Omega(\log_d n) = \Omega(\alpha(G^*)\cdot n^2\log_d n)$. Since the total capacity is at most $\frac{nd}{2}$, this implies that $\alpha(G^*) \le O\left(\frac{d}{n\log_d n}\right) = O\left(\frac{d\log d}{n\log n}\right)$.

On the other hand, we can lower bound $\alpha(G)$ using the *flow-cut gap* from [28, 7]. This theorem says that the throughput is at least $\frac{1}{O(\log n)}$ times the minimum (over all cuts) of the capacity across the cut divided by the demand across the cut. When applied to the all-to-all setting, this implies that

$$\alpha(G) \ge \frac{1}{O(\log n)} \min_{S\subseteq V:|S|\le n/2} \frac{|\partial(S)|}{|S|(n-|S|)}$$

By the definition of edge expansion, we know that $|\partial(S)| \ge EE(G)|S|$, and since G is an expander we know that $EE(G) \ge \Omega(d)$. Combining these, we get that $\alpha(G) \ge \frac{1}{O(\log n)}\frac{d}{n}$, and thus $\alpha(G^*) \le O(\log d)\alpha(G)$. □

The next two results, when put together, show that Xpanders are, in a sense, the network topology most resilient to adversarial traffic scenarios.

THEOREM 4.3. *For any traffic matrix T and d-regular Xpander G on n vertices, $\alpha(G,T)$ is within a factor of $O(\log n)$ of that of the throughput optimal d-regular graph on n vertices with respect to T.*

PROOF. We can again use the flow-cut gap of [28, 7]. For a subset $S \subseteq V$ with $|S| \le n/2$, let $T(S) = \sum_{u\in S}\sum_{v\notin S} T_{u,v}$ be the total demand across S. Let G^* be an arbitrary d-regular graph on n vertices. Then the flow-cut gap guarantees that for any graph G',

$$\frac{1}{O(\log n)} \min_{|S|\le n/2} \frac{|\partial(S)|}{T(S)} \le \alpha(G',T) \le \min_{|S|\le n/2} \frac{|\partial(S)|}{T(S)}$$

Now we can upper bound the throughput of G^* by noting that $|\partial(S)| \le d|S|$, so $\alpha(G^*,T) \le \min_{|S|\le n/2}\frac{d|S|}{T(S)}$. On the other hand, we can lower bound the throughput of G by noting that $|\partial(S)| \ge |S| \times EE(G) \ge \Omega(d|S|)$ (since G is

an expander), and thus

$$\alpha(G,T) \geq \frac{1}{O(\log n)} \min_{|S|\leq n/2} \frac{\Omega(d|S|)}{T(S)}$$

Thus $\alpha(G^*,T) \leq O(\log n)\alpha(G,T)$, as claimed. □

THEOREM 4.4. *For any d-regular graph G on n vertices, there exists a traffic matrix T and a d-regular graph G^* on n vertices such that $\alpha(G^*,T) \geq \Omega(\log_d n) \cdot \alpha(G,T)$.*

PROOF. Let $\pi : V \to V$ be a random permutation. Let G^* be the graph obtained by including an edge $\{u,v\}$ if and only if $\{\pi^{-1}(u), \pi^{-1}(v)\}$ is an edge in G (i.e. G^* is just a random isomorphism of G). Let T be the traffic matrix such that $T_{u,v} = 1$ if $\{u,v\} \in E(G^*)$ and $T_{u,v} = 0$ otherwise. Then clearly $\alpha(G^*,T) = 1$, since the traffic demands are exactly the edges of G^*.

To analyze what happens in G, note that since G has maximum degree d, the average distance between two nodes is $\Omega(\log_d n)$. So since π is a random permutation, the expected distance in G between two nodes that are adjacent in G^* is also $\Omega(\log_d n)$. So by linearity of expectations, $\mathbf{E}\left[\sum_{\{u,v\}:T_{u,v}=1} dist_G(u,v)\right] \geq \Omega(dn \log_d n)$. Thus we can fix some π where this is true, i.e. where $\sum_{\{u,v\}:T_{u,v}=1} dist_G(u,v) \geq \Omega(dn \log_d n)$. Then if we send $\alpha(G,T)$ units of flow for every traffic demand, the total capacity used up is at least $\Omega(\alpha(G,T) \cdot dn \log_d n)$. On the other hand, the total capacity in G is only $|E| = dn/2$. Thus $\alpha(G,T) \leq O(1/\log_d n)$. Putting this together, we get that $\alpha(G,T) \leq O\left(\frac{1}{\log_d n}\right) \cdot \alpha(G^*,T)$. □

4.3 Near-Optimal Resiliency to Failures

4.3.1 Connectivity Guarantees

It is easy to prove that in any d-regular Xpander the number of edge-disjoint paths between any two vertices is exactly d. Since this is the maximum possible number of such paths in a d-regular graph, Xpander's network topology provides optimal connectivity between any two communicating end-points and can thus withstand the maximum number of link-failures (specifically, $d-1$) without disconnecting two switches.

THEOREM 4.5. *In any d-regular Xpander, any two vertices are connected by exactly d edge-disjoint paths.*

PROOF. Let $u, v \in V$. By Menger's Theorem, the number of edge-disjoint paths from u to v is equal to the minimum cut separating u and v. Consider any cut $(S, \bar{S})$ with $u \in S$ and $v \in \bar{S}$. Without loss of generality, suppose $|S| \leq \frac{n}{2}$. Then, if $|S| \geq d$, since the edge expansion is at least 1 the number of edges across the cut is at least $EE(G) \cdot |S| \geq d$. If $|S| < d$, then since each node in S has degree d the number of edges crossing the cut must be at least $d|S| - |S|(|S|-1)$, which for $|S|$ between 1 and $d-1$ is at least d. Thus, every cut separating u and v has at least d edges, and so there are d edge-disjoint paths between u and v. □

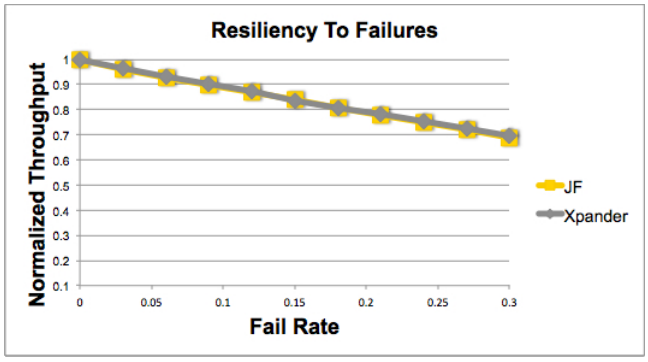


Figure 6: Throughput under link failures.

This result can also be extended to any "good enough" expander.

4.3.2 Graceful Performance Degradation

We compute the all-to-all server-to-server throughput in Xpander and Jellyfish after failing X links uniformly at random, where X ranges 0% to 30% in increments of 3%. We repeated this for Xpander networks of many sizes and node-degrees. Figure 6 describes our (representative) results for Xpander and Jellyfish of 708 servers and 236 14-ports switches (with one switch port left unused). As shown in the figure, the throughput of Xpander degrades linearly with the failure rate. We show in Section 5 through both flow-level and packet-level simulations that the throughput of Xpanders indeed dominates that of fat trees.

Intuitively, this linear dependence on the failure rate is natural. If the probability of link failure is p, then after failures the graph is similar to a $((1-p)d)$-regular Xpander. This is because for each cut S the number of edges across it ($|\partial(S)|$) before failure was large (since G was an expander), and so after failure the number of edges across the cut is tightly concentrated around its expectation, which is $(1-p)|\partial(S)|$.

4.4 Incremental Expandability

Companies such as Google, Facebook and Amazon constantly expand existing datacenters to meet ever-growing demands. A significant advantage of random graphs (i.e., Jellyfish) over traditional datacenter topologies (e.g., fat trees) is the ability to incrementally expand the network (without resorting to leaving many ports unused etc.) [45].

We present a deterministic heuristic for incrementally expanding a d-regular Xpander with few wiring changes when adding a new node (ToR): To add a new node to the datacenter, disconnect $\frac{d}{2}$ links and connect the d incident nodes to the newly added node (recall that d is the number of ports used to connect to other switches). Observe that this is indeed the minimal rewiring needed as at least $\frac{d}{2}$ links must be removed to "make room" for the new switch. The key challenge is selecting the links to remove. Intuitively, our heuristic goes over all links and quantifies the loss in edge expansion from removing each link, and then removes the $\frac{d}{2}$ links

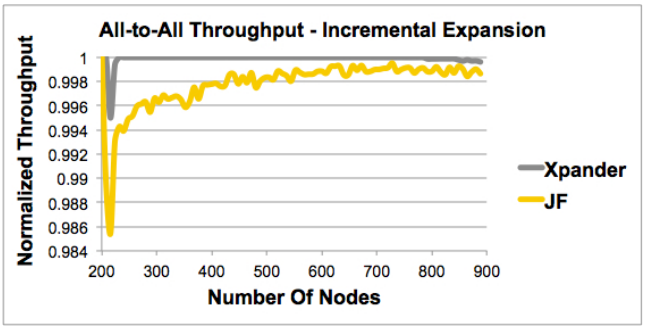


Figure 7: Throughput under incremental expansion for $k = 32$, incrementally adding 1 switch and 8 servers at each step.

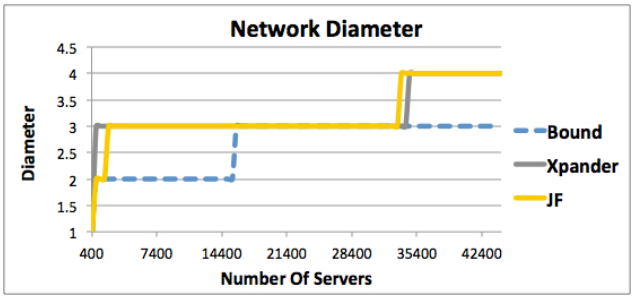


Figure 8: Results for diameter. We use $k = 48$ switches with 12 servers placed under each switch.

whose removal is the least harmful in this regard. Importantly, since computing edge expansion is, in general, computationally hard [28], our heuristic relies on the tractable notion of spectral gap [24], which approximates the edge expansion. We refer the reader toAppendix B for a technical exposition of this heuristic.

We compute the all-to-all throughput of topologies that are gradually expanded from the complete d-regular graph using our deterministic incremental growth algorithm to the theoretical (possibly unattainable) upper bound on the throughput of *any* d-regular datacenter network.

Figure 7 shows the all-to-all throughput results for an Xpander with 32-port switches. At each size-increment-step, one switch (and 8 servers) is added, thus gradually growing the datacenter network from 200 servers to 900 servers. As shown in the figure 7 all Xpander datacenter networks obtained in this manner achieve near-optimal throughput. We benchmark Xpander against the incremental growth of Jellyfish, as presented in [45].

4.5 Short Path-Lengths and Diameter

As shown in Figure 9, all evaluated Xpander (and Jellyfish) networks exhibit the same average (shortest) path lengths and are, in fact, usually within 5% of the lower bound on average path length in [12]. (Results for many other choices of n and d lead to the same conclusion.) Thus, Xpander effectively minimizes the average path length between switches.

A straightforward argument (omitted) shows that $\lceil \log_d n \rceil$ is a lower bound on the diameter of a d-regular graph. Figure 8 shows that all Xpanders evaluated are within just 1 hop from this theoretical lower bound.

fat tree Degree	#Switches	#Servers	Throughput
8	80%	100%	121%
10	100%	100%	157%
12	80.5%	100%	103%
14	96%	103%	122%
16	80%	100%	120%
18	90%	100%	137%
20	80%	100%	118%
22	89%	102%	121%
24	80%	100%	111%

Table 2: Xpanders vs. fat trees (FT), flow-level simulations. Percentages are Xpander/FT.

5. XPANDER *VS.* FAT TREE

We next detail the significant performance benefits of Xpander datacenters over fat trees. We show that Xpander can support the same number of servers as a fat tree at the same (or better) level of performance with only about $80 - 85\%$ of the switches. We also show that Xpander is more resilient to failures.

5.1 Better Performance, Less Equipment

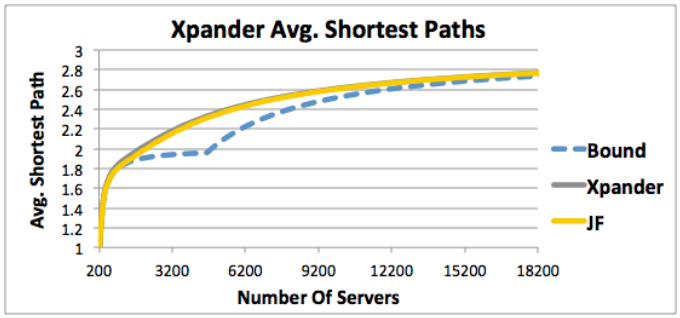
We examine uniform fat tree topologies for every even degree (number of ports per switch) between 8 and 24. We identify, for each fat tree in this range, an Xpander with much fewer (roughly 80-85%) switches that supports the same number of servers with at least the same level of server-to-server throughput.[2] See details in Table 2.

To construct the relevant Xpander for each fat tree, an Xpander is generated as explained in Section 3.1 and incremented to the appropriate size using our deterministic incremental growth algorithm presented in Section 4.4.

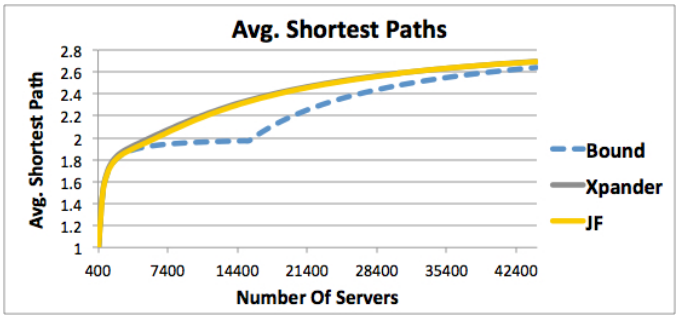
5.2 Simulation Results

Throughput. We evaluate the all-to-all throughput of Xpander and fat trees. We show in Table 2 the all-to-all server-to-server throughput in fat trees and Xpanders without any link failures. As can be seen in Table 2, Xpander is able to achieve comparable, if not better, server-to-server throughput than fat trees *even* with as few as 80% of the switches. Table 3 shows the results of extensive packet based-simulations using the MPTCP network simulator for Xpander and fat trees with $k = 32$ and $k = 48$, containing 8K and 27K servers, respectively. Again, Xpander networks achieve

[2]We now consider server-to-server all-to-all throughput and not switch-to-switch all-to-all throughput as in a fat tree not all switches originate traffic.

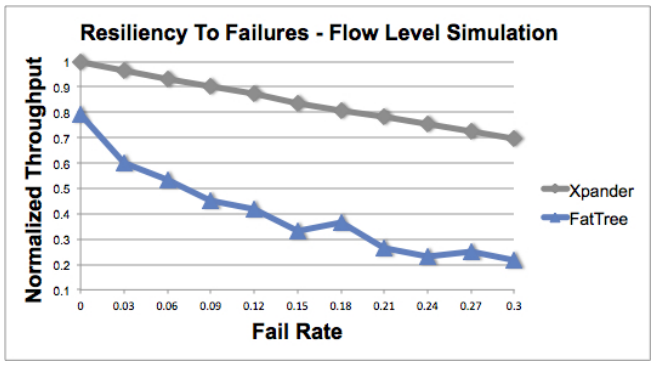


(a) $k = 32$. There are 8 servers placed under each switch.

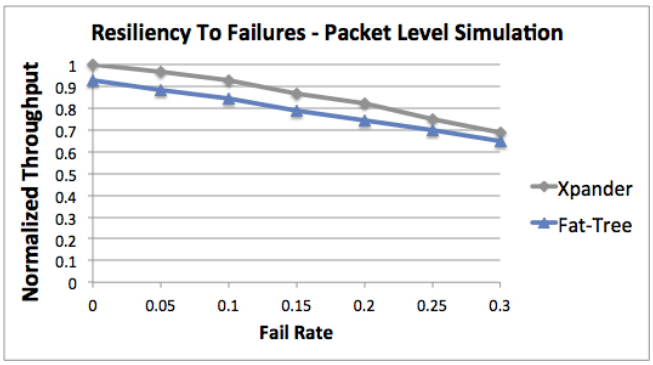


(b) $k = 48$. There are 12 servers placed under each switch.

Figure 9: Results for avg. path length



(a) d=14



(b) d=32

Figure 10: Server-to-server throughput degradation with failures. Flow level (on the left) and Packet level (on the right)

Fat Tree Degree	#Switches	#Servers	Packet Simulation Throughput
32	90%	98.5%	110%
48	88%	100%	102%

Table 3: Xpanders vs. fat trees, packet-level simulation results. Percentages are Xpander/FT.

similar or better performance to that of fat trees, with significantly fewer switches.

To explore how Xpander and fat tree compare for other traffic patterns, we also simulate a fat tree of 8K servers (i.e., $k = 32$) and its matching Xpander under the following "Many-to-One" scenario using the MPTCP packet level simulator. We randomly select 10% of the servers as our destinations and for each such server we select at random x% of the servers to generate traffic destined for that server, where x=1%, 1.5%, 2%, 2.5% and 3% (1% is roughly 80 servers). Table 5 presents the averaged results of 4 such simulations. We conclude that, once again, Xpander provides the same level of performance with less network equipment.

Robustness to failures. We compute the all-to-all server-to-server throughput in fat trees and Xpanders after failing X links uniformly at random, where X ranges 0% to 30% in increments of 3%. We repeated this for fat trees of all even degrees in the range 8-24 and the corresponding Xpanders from Table 2. Figure 10(a) describes our (representative) results for fat trees of degree $k = 14$ (vs. Xpander). We further simulate a fat tree with $k = 32$, containing 1280 switches and 8192 servers, against an Xpander containing 90% of the switches and 98.5% of the servers, under a random-shuffle permutation matrix after failing S links uniformally at random, where S ranges from 0% to 30% in increments of 5%. The results of this simulation are described in Figure 10(b). Both results (flow-level and packet-level alike) show that

Percentage Of Servers Routing To Each Destination	Packet Simulation Throughput
1%	99.6%
1.5%	99.3%
2%	101%
2.5%	103%
3%	103%

Table 5: Xpanders vs. fat trees, 10% of the servers are selected as destinations. Percentages for throughput are Xpander/FT.

Tested Topology	Random Shuffle		One-to-Many		Many-to-One		Big-and-Small	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max
Xpander	19.66	58.86	79.52	104.03	70.09	90.88	28.66	120.21
FatTree (TCP+ECMP)	26.7	102.86	80.72	89.94	80.79	91.51	42.72	220.1
FatTree (MPTCP+ECMP)	17.94	105.71	78.18	138.5	69.56	91.31	31.75	180.64

Table 4: Xpander and fat trees under various traffic matrices. Values are given in seconds and indicate the average finishing time for transmission.

Xpander indeed exhibits better resiliency to failures than fat tree. We note that the smaller gap between Xpander and fat tree in the MPTCP simulations can be explained by the fact that the per switch degree is higher and so naturally routing is less affected by failures.

5.3 Experiments with a Network Emulator

To show that an Xpander with significantly less switches can achieve comparable performance to fat trees, we used mininet [27] and the RipL-POX [4] controller to simulate fat tree [5] networks under various workloads, and for two routing & congestion control schemes: (1) ECMP with TCP and (2) K-shortest-paths with $K = 8$ and MPTCP with 8 subflows. We also simulated Xpanders for the same workloads under K-shortest-paths with $K = 8$ and MPTCP with 8 subflows. These simulations were performed on a VM running Ubuntu 14.04 with MPTCP kernel version 3.5.0-89 [11]. We chose, for compute and scalability constraints, to evaluate a fat tree network of degree 8, which contains 80 switches and 128 servers. We tested against this fat tree topology the corresponding Xpander datacenter from Table 2, which contains only 64 switches and the same number of servers. All links between switches in both networks are of capacity 1Mbps.

The workloads considered are: (1) Random Shuffle, where the 128 servers are divided into two halves, and each member of the first half sends a 1Mb file to a (unique) randomly chosen member of the other half, (2) One-to-Many, where 4 servers are randomly selected and these servers send a 1Mb file to 10 other randomly chosen (unique) hosts, (3) Many-to-One, the reversed scenario, where 40 different servers send 1Mb file each to 4 servers (10 sending servers per each receiving server), and (4) Big-And-Small, which is similar to Random Shuffle, only in addition each of 8 randomly chosen servers sends a 10Mb file to a unique other server. Our results are summarized in Table 4. Observe that even with 80% of the switches, Xpander provides comparable or better performance.

5.4 Implementation on a Network Testbed

To validate the above findings, we use the "Ocean Cluster for Experimental Architectures in Networks" (OCEAN [3]), an SDN-capable network testbed composed of 13 Pica8 Pronto 3290 switches each with 48 ports. We used the OCEAN platform to experimentally evaluate two datacenter networks: (1) a fat tree composed of 20 4-port switches, of which 8 are top-of-rack switches, each connected to 2 servers (16 servers overall), and (2) an Xpander consisting of 16 4-port switches, each connected to a single server (again, 16 servers overall). Observe that while both networks support the same number of servers, the number of switches in the Xpander is 80% of the number of switches in the fat tree (16 vs. 20). Similarly, the number of links connecting switches to other switches in the Xpander is 75% that of the fat tree (24 vs. 32). Observe, however, that the network capacity *per server* in Xpander is much higher as the number of ToRs is higher (16 vs. 8), and each ToR is connected to less servers (1 vs. 2) and to more other switches (3 vs. 1). Thus, intuitively, the Xpander can provide comparable or better performance with less network equipment. Our experiments on OCEAN confirm this intuition.

Scenario	Min	Max	Average
Random Shuffle	234%	83%	135%
One-to-Many	94%	138%	123%
Many-to-One	115%	86%	92%

Table 6: Results for the physical simulations, all values are the averaged value of Xpander/FatTree.

ECMP routing and TCP are used to flow traffic in the fat tree, whereas for the Xpander we use k-Shortest Paths, with $k = 3$, and MPTCP with 3 subflows (and so on each of the 3 distinct paths between a source and a destination there is a separate MPTCP subflow). We evaluate the (min, max, and average) throughput under three different traffic scenarios: (1) random shuffle, in which every server routes to a single random destination, (2) many-to-one, in which a randomly chosen server receives traffic from the other 15 servers, and (3) one-to-many, in which a randomly chosen server sends traffic to all other servers. To compute the throughput for two communicating servers, an "endless" flow between these servers is generated through the creation of an iperf client and server and the averaged throughput is computed after 10 minutes. Our results for each simulation setting average over 5 independent runs.

Table 6 shows our results for the above three traffic scenarios (the rows) and for min, max, and average throughput (the columns), where % are Xpander/FT. Xpander achieves comparable or better results in each of the evaluated scenarios (again, using only 80% of the network equipment).

6. XPANDER *VS.* SLIM FLY

#Ports per Switch (Sw2Sw / Total)	#Switches (XPNDR / SF)	#Servers (XPNDR / SF)	Bisection Bandwidth	Cost per node	Power per node	Expansion
5 / 8	18 / 18 (100%)	54 / 54 (100%)	86%	123%	114%	77%
7 / 11	48 / 50 (96%)	192 / 200 (96%)	67%	79%	81%	72%
11 / 17	96 / 98 (98%)	576 / 588 (98%)	98%	106%	104%	84%
17 / 26	234 / 242 (96%)	2,106 / 2,178 (96%)	102%	102%	100%	92%
19 / 29	340 / 338 (104%)	3,400 / 3,380 (104%)	103%	95%	94%	96%
25 / 38	572 / 578 (99%)	7,436 / 7,541 (98%)	109%	95%	94%	102%
29 / 44	720 / 722 (99%)	10,800 / 10,830 (99%)	118%	104%	102%	103%
35 / 51	1080 / 1058 (102%)	17,280 / 16,928 (102%)	119%	102%	100%	101%
43 / 65	1672 / 1682 (99%)	36,784 / 37,004 (99%)	117%	99%	96%	107%
47 / 71	1920 / 1922 (99%)	46,080 / 46,128 (99%)	122%	103%	101%	108%
55 / 83	2688 / 2738 (96%)	75,264 / 76,664 (98%)	119%	91%	88%	112%

Table 7: Xpander vs. Slim-Fly. The first column specifies the number of ports per switch used for switch-to-switch and the total port count (for both Xpander and SF). Percentages are Xpander/SF.

We also contrast Xpander with Slim Fly, a recent proposal from the world of high-performance computing (HPC). SF leverages a graph-theoretic construction of low-diameter graphs [33] (either 2, the situation most explored in [8], or 3).

Intuitively, by decreasing the diameter, less capacity is wasted when sending data, and hence overall performance is higher. Indeed, [8] shows that SF outperforms existing and proposed datacenter architectures, but performs worse than random topologies, e.g., in terms in bisection bandwidth and resiliency to failures.

When comparing SF to Xpander, we first note that the low diameter of SF imposes an extremely strict condition on the relationship between the per node degree d and the number of nodes n: by requiring diameter 2, SF requires $d \geq \Omega(\sqrt{n})$. (Importantly, d here represents only the ports used to connect a switch to other switches, and so, to support servers, the actual port count must be even higher.) This imposes a strict limit on the scalability of Slim Fly. Xpander topologies, on the other hand, exist for essentially any combination of n and d and, in particular, can be used for arbitrarily large datacenters even with a fixed per-switch degree d.

Not only is Xpander more flexible than SF in supporting more nodes with smaller degrees, but it exhibits better performance than SF as the network grows (even in the degree/node regimes in which SF is well-defined). We used the simulation framework published in [8] to compare SF to Xpander in terms of performance and costs. The METIS partitioner [26] was used for approximating bisection bandwidth (as in [8]) and the code from [8] for cost and power consumption analysis (using the switch/cable values in [8]). We also computed the expansion for both categories of graphs (using spectral gap computation, which approximates edge expansion). See our results for Xpanders in Table 7 and for Jellyfish in Table E15 in the Appendix.

Our findings suggest that, in fact, the good performance of SF can be attributed to the fact that it is an expander datacenter. We back these empirical results with the following new theoretical result:

THEOREM 6.1. *A Slim-Fly network of degree d and diameter 2 has edge expansion at least $\frac{d}{3} - 1$.*

We argue, however, by optimizing the diameter-size tradeoff, Slim Fly sacrifices a small amount of expansion leading to worse performance than random networks and Xpander as the network gets larger. Our results reveal that for networks with less than 100 switches, SF is a better expander than both Xpander and Jellyfish and exhibits better bisection bandwidth. This advantage is reversed as the network size increases and in turn Xpander and Jellyfish become better expanders. Our results thus both validate and shed light on the results in [8], showing why random graphs (and Xpander) outperform SF for large networks.

We also show (Table 7) that Xpander's cost and power consumption are comparable to those of SF.

7. DEPLOYMENT

We grapple with important aspects of building Xpander datacenters: (1) equipment and cabling costs; (2) power consumption; and (3) physical layout and cabling complexity. We first present a few high-level points and then the results of a detailed analysis of Xpander deployment in the context of both small-scale (container-sized) datacenters and large-scale datcenters. We stress that our analyses are straightforward and, naturally, do not capture all the intricacies of building an actual datacenter. Our aim is merely to illustrate our main insights regarding the deployability of Xpanders (illustrating, for instance, its significant deployability advantages over random datacenter networks a la Jellyfish).

Physical layout and cabling complexity. As illustrated in Figures 2 and 3, an Xpander consists of several meta-nodes, each containing the same number of ToRs and connected to

each other meta-node via the same number of cables. No two ToRs within the same meta-node are connected. This "clean" structure of Xpanders has important implications: (1) placing all ToRs in a meta-node in close proximity (the same rack / row(s)) enables bundling cables between every two meta-nodes, and (2) a simple way to reason about and debug cabling is to color the rack(s) housing each meta-node in a unique color and color the bundle of cables interconnecting two meta-nodes in the respective two-color stripes. See illustration in Figure 2. Contrast this with the totally unstructured random topology of Jellyfish [45], in which no such color-coding is possible.

Equipment, cabling costs, and power consumption. As shown in Table 2, and validated in Sections 5.3 and 5.4, Xpanders can support the same number of servers at the same (or better) level of performance as traditional fat tree networks, with as low as 80% of the switches. This, of course, has important implications for equipment (switches/servers) costs and power consumption. We show, through analyzing cable numbers and lengths, that the reduced number of inter-ToR cables in Xpanders, compared to Clos networks/fat trees, translates to comparable or lower costs.

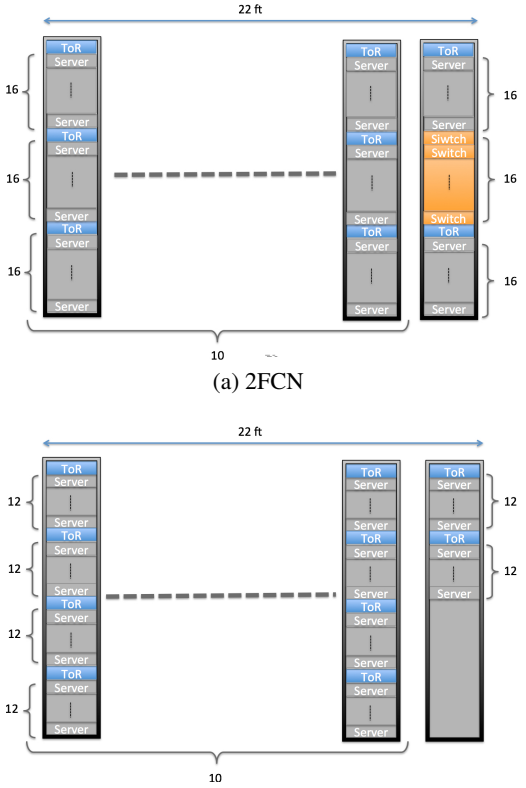
Analyzing deployment scenarios. We analyze below two case studies: (1) small clusters ("container datacenters"), and (2) large-scale datacenters.

7.1 Scenario I: Container Datacenters

As several ToR switches can sometimes be placed in the same physical rack (along with the associated servers), we distinguish between a Virtual Rack (VR), i.e., a ToR switch and the servers connected to it, and a Physical Rack (PR), which can contain several VRs. Our analyses assume that all racks are 52U (this choice is explained later) and are of standard dimensions (see Figure E15 in Appendix E), switches are interconnected via Active-Optical Cables (AOC), and servers are connected to ToR switches via standard copper cables.

We inspect 2-layered folded-clos network (FCN) of degrees 32 and 48 (see Figure C14 (a) in the Appendix). We select, for each of these two topologies, a matching Xpander with better performance. We consider 52U racks as these provide the best packing of VRs into PRs for Clos networks. Specifically, 3 VRs fit inside each physical rack for the $k = 32$ Clos network and 2 VRs fit into a PR for $k = 48$ 2-FCN network. The two matching Xpanders are created via a single 2-lift. As each VR in an Xpander contains less servers than that of the comparable 2-FCN network, more VRs can reside in each physical rack (for both degrees). See physical layouts of both the 2-FCNs and Xpander networks in Figures 11 and 12 and analysis in Table 8.

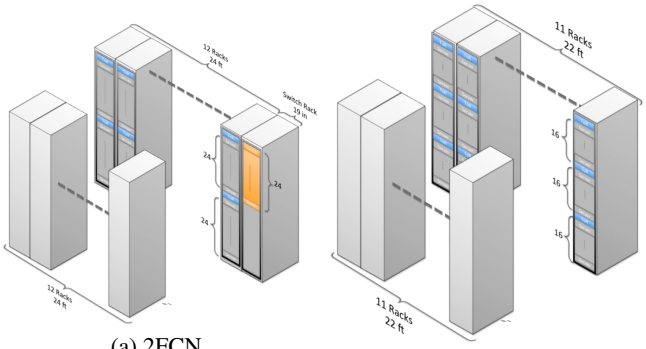
Clearly, the use of less switches in Xpanders immediately translates to a reduction in costs. An Xpander network of switch-to-switch degree d, where each meta-node contains x ToRs, requires $x \cdot \frac{d \cdot (d+1)}{2}$ AOC cables, whereas for a 2-FCN of switch degree (total port count) k there are $\frac{k^2}{2}$ cables. The



(a) 2FCN

(b) Xpander

Figure 11: A $k = 32$ 2FCN network topology and the matching Xpander



(a) 2FCN

(b) Xpander

Figure 12: A $k = 48$ 2FCN network topology and the matching Xpander

lower number of AOC cables in Xpanders, assuming 10m-long AOC cables, yields the cable lengths in Table 8. Importantly, the marginal cost of AOC cables greatly decreases with length and so the reduction in number of cables translates to potentially greater savings in costs.

A detailed analysis appears in Appendix E.1.

7.2 Scenario II: Large-Scale Datacenters

We now turn our attention to large-scale datacenters. Specif-

	#Switches	#Servers	#Physical Racks	#Cables	Cable Length	All-to-All Server to Server Throughput
k=32	42 vs. 48 (87.5%)	504 vs. 512 (98.44%)	11 vs. 11 (100%)	420 vs. 512 (82%)	4.2km vs. 5.12km (82%)	109%
k=48	66 vs. 72 (91.76%)	1,056 vs. 1,152 (91.67%)	22 vs. 25 (88%)	1056 vs. 1152 (91.6%)	10.56km vs. 11.52km (91.6%)	142%

Table 8: Xpander vs. 2-FCN. Percentages are Xpander/2-FCN

ically, we analyze the cost of building a uniform-degree fat tree with port-count $k = 32$ per switch (and so of size 1280 switches and 8192 servers) vs. a matching Xpander. We first present, for the purpose of illustration, the physical layout of each network in a single floor. We point out, however, that while deploying a fat tree (or the matching Xpander) of that scale in a single room might be physically possible, this might be avoided in the interest of power consumption and cooling considerations. We hence also discuss large-scale datacenters that are dispersed across multiple rooms/floors.

7.2.1 *Single Floor Plan*

A fat tree with total port count $k = 32$ per node contains 32 pods and $\frac{d^2}{4} = 256$ core switches, where each pod containing 16 ToRs, 512 servers and 16 aggregation switches, totaling in 8192 servers, 512 ToRs, and 512 aggregation switches. We present a straightfoward, hypothetical floor plan for deploying such a fat tree in Figure E17 in Appendix E.2. Again, we assume 52U physical racks as this is the most suitable for packing VRs in the fat tree, allowing us to fit 3 VRs in each PR and consequently an entire pod (including the aggregation switches) in a row of 6 PRs. We can place 2 such pods (rows) inside a hot/cold-aisle containment enclosure, resulting in 16 such enclosures for the entire datacenter. We end up with 12 rows, each containing exactly 18 physical racks (which, in turn, can house 3 pods), and core switches placed in 5 additional physical racks. We assume that, within a pod, 5m AOC cables are used to connect each ToR to its aggregation layer switches. Each of the 2-pod enclosures can be connected to the row of core switches using combination of 10m/15m/20m/25m AOC cables (depending of their proximity).

We compare this fat tree network with an Xpander network of degree 23 ($k = 30$-port switches instead of $k = 32$), constructed using four 2-lifts and another 3-lift. See side by side comparison of the two networks in Table 9. This specific Xpander houses 8064 servers under 1152 ToRs and consists of 24 meta-nodes, each containing 48 VR with 7 servers per VR. We present a possible floor plan for deploying this Xpander in Figure E16 in Appendix E.2. Using 52U racks, 6 VRs can be packed into a physical rack, resulting in a total of 8 racks per meta node. Again, each hot/cold-aisle containment enclosure houses 2 rows, resulting in 16 52U racks (8 in each row). We present the physical layout analysis for both networks in Table 9. See a more detailed analysis in Appendix E.2.

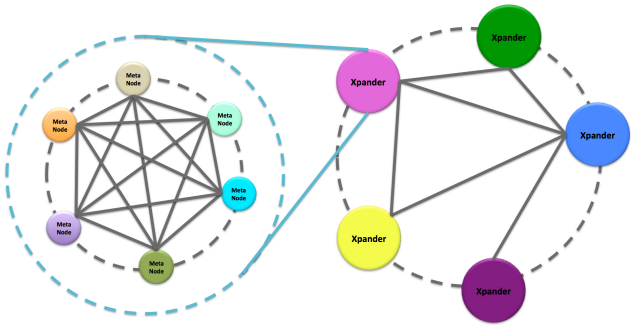


Figure 13: A sketch of an Xpander of Xpander graphs

7.2.2 *Xpander of Xpanders*

So far, our analysis of large-scale Xpanders assumed that the whole datacenter network fits in a single floor. This might not be feasible due to power supply and cooling constraints. To this end, the Xpander must be "broken" into several, dispersed, components. One approach to do so is by placing a single Xpander-pod (see Section 3.1), or several such pods, in a separate floor. Another possible approach is constructing an Xpander network interconnecting smaller Xpander networks, as illustrated in Figure 13: (1) each smaller Xpander will be housed in a single container/-room/floor and be constructed as illustrated above; (2) several (higher-degree) switches in each of these Xpanders will be designated as "core switches"; (3) these core switches will then be interconnected through an Xpander overlay network. Since, as evidenced by our results, an Xpander provides near-optimal performance guarantees (throughput, failure resiliency, average path length, etc.), this construction can yield good performance both with each smaller Xpander and between the Xpander networks.

8. RELATED WORK

We now elaborate on related research not covered in the prequel.

Datacenter networks. Datacenters have been extensively researched from many different angles, e.g., throughput optimization [44, 40, 25], failure resiliency [29, 20, 19], and expandability [45, 15]. In particular, many datacenter topolo-

Switch Degree	#Switches	#Servers	#Physical Racks	#Cables	Cable Length (m)	Ttl. Space (ft^2)
30 vs. 32 (93.75%)	1,152 vs. 1,280 (90%)	8,064 vs. 8,192 (98.44%)	192 vs. 221 (86.87%)	13,248 vs. 16,348 (80.85%)	220.8k vs. 174k (127%)	3.24k vs. 4k (81%)

Table 9: Xpander vs. fat tree. Percentages are Xpander/fat tree

gies have been proposed, including Clos networks [5, 21, 36], hypercubes [22, 49], small-world graphs [43], and random graphs [45, 44].

Expanders. Expanders play a key role in a host of applications, ranging from networking to complexity theory and coding. See the survey of Hoory, Linial, and Wigderson [24]. A rich body of literature in mathematics and computer science deals with constructions of expanders, e.g., Margulis's construction [32], algebraic constructions [30], and constructions that utilize the so-called "zig-zag product" [41]. Our construction of the Xpander datacenter topology utilizes the notion of 2-lifting a graph, introduced by Bilu and Linial [9, 31]. Utilizing expanders as network topologies has been proposed in the context of parallel computing and high-performance computing [46, 14, 13, 8], optical networks [39] and also for peer-to-peer networks and distributed computing [38, 37]. Our focus, in contrast, is on datacenter networking and on tackling the challenges that arise in this context (e.g., specific, throughput-related performance measures, specific routing and congestion control protocols, costs, incremental growth, etc.).

Relation to [47]. A recent workshop publication explores the applicability of expander constructions to datacenter design. Here, we provide a deeper and much more detailed evaluation of the merits of expander datacenters, in general, and of Xpander, in particular, including (1) many additional simulation results with the MPTCP simulator for Xpander and fat tree, (2) comparison of Xpander to Slim-Fly using the simulation framework in [8], (3) results for path-lengths, diameter, and incremental growth, (4) results for bisection bandwidth of Xpander, and (5) a detailed discussion of Xpander's deployment scenarios.

9. CONCLUSION

We showed that Xpander datacenters offer many valuable advantages over traditional datacenter designs and suggested practical approaches for building such datacenters. We believe that Xpander provides an appealing and tangible alternative to traditional datacenter designs.

10. REFERENCES

[1] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html.

[2] MPTCP Simulator v0.2. http://nets.cs.pub.ro/~costin/code.html.

[3] Ocean cluster for experimental architectures in networks (ocean). http://ocean.cs.illinois.edu/.

[4] RipL-POX, simple datacenter controller build on RipL. https://github.com/brandonheller/riplpox.

[5] AL-FARES, M., LOUKISSAS, A., AND VAHDAT, A. A scalable, commodity data center network architecture. In *SIGCOMM* (2008).

[6] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., AND VAHDAT, A. Hedera: Dynamic flow scheduling for data center networks. In *NSDI* (2010).

[7] AUMANN, Y., AND RABANI, Y. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.* (1998).

[8] BESTA, M., AND HOEFLER, T. Slim Fly: A cost effective low-diameter network topology. In *SC14* (2014).

[9] BILU, Y., AND LINIAL, N. Lifts, discrepancy and nearly optimal spectral gap. *Combinatorica* (2006).

[10] BOLLOBÁS, B. The isoperimetric number of random regular graphs. *Eur. J. Comb.* (1988).

[11] C. PAASCH, S. BARRE, ET AL. Multipath TCP in the Linux Kernel. http://www.multipath-tcp.org.

[12] CERF, V. G., COWAN, D. D., MULLIN, R. C., AND STANTON, R. G. A lower bound on the average shortest path length in regular graphs. *Networks* (1974).

[13] CHONG, F. T., BREWER, E. A., LEIGHTON, F. T., AND KNIGHT, T. F., J. Building a better butterfly: the multiplexed metabutterfly. In *ISPAN* (1994).

[14] CHONG, F. T., BREWER, E. A., LEIGHTON, F. T., AND KNIGHT, T. F., J. Scalable expanders: Exploiting hierarchical random wiring. In *Parallel Computer Routing and Communication*. 1994.

[15] CURTIS, A. R., KESHAV, S., AND LÓPEZ-ORTIZ, A. Legup: using heterogeneity to reduce the cost of data center network upgrades. In *CoNEXT* (2010).

[16] DE QUEIRÓS VIEIRA MARTINS, E., AND PASCOAL, M. M. B. A new implementation of yen's ranking loopless paths algorithm. *4OR* (2003).

[17] FRIEDMAN, J. Relative expanders or weakly relatively ramanujan graphs. *Duke Math. J. 118*, 1 (05 2003), 19–35.

[18] FRIEDMAN, J. *A Proof of Alon's Second Eigenvalue Conjecture and Related Problems*. Memoirs of the

American Mathematical Society. American Mathematical Soc., 2008.

[19] GEORGE B. ADAMS, I., AND SIEGEL, H. J. The extra stage cube: A fault-tolerant interconnection network for supersystems. *IEEE Trans. Computers* (1982).

[20] GILL, P., JAIN, N., AND NAGAPPAN, N. Understanding network failures in data centers: measurement, analysis, and implications. In *SIGCOMM* (2011).

[21] GREENBERG, A. G., HAMILTON, J. R., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D. A., PATEL, P., AND SENGUPTA, S. Vl2: a scalable and flexible data center network. In *SIGCOMM* (2009).

[22] GUO, C., LU, G., LI, D., WU, H., ZHANG, X., SHI, Y., TIAN, C., ZHANG, Y., AND LU, S. Bcube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM* (2009).

[23] GUO, C., WU, H., TAN, K., SHI, L., ZHANG, Y., AND LU, S. Dcell: a scalable and fault-tolerant network structure for data centers. In *SIGCOMM* (2008).

[24] HOORY, S., LINIAL, N., AND WIGDERSON, A. Expander graphs and their applications. *Bull. Amer. Math. Soc.* (2006).

[25] JYOTHI, S. A., SINGLA, A., GODFREY, P. B., AND KOLLA, A. Measuring throughput of data center network topologies. In *SIGMETRICS* (2014).

[26] KARYPIS, G., AND KUMAR, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* (1998).

[27] LANTZ, B., HELLER, B., AND MCKEOWN, N. A network in a laptop: Rapid prototyping for software-defined networks. In *HOTNETS* (2010).

[28] LINIAL, N., LONDON, E., AND RABINOVICH, Y. The geometry of graphs and some of its algorithmic applications. *Combinatorica* (1995).

[29] LIU, V., HALPERIN, D., KRISHNAMURTHY, A., AND ANDERSON, T. F10: A fault-tolerant engineered network. In *NSDI* (2013).

[30] LUBOTZKY, A., PHILLIPS, R., AND SARNAK, P. Ramanujan graphs. *Combinatorica* (1988).

[31] MARCUS, A., SPIELMAN, D. A., AND SRIVASTAVA, N. Interlacing families I: Bipartite ramanujan graphs of all degrees. In *FOCS* (2013).

[32] MARGULIS, G. A. Explicit constructions of expanders. *Problemy Peredači Informacii* (1973).

[33] MCKAY, B. D., MILLER, M., AND IR, J. A note on large graphs of diameter two and given maximum degree. *Journal of Combinatorial Theory, Series B* (1998).

[34] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: Enabling innovation in campus networks. In *SIGCOMM* (2008).

[35] MUDIGONDA, J., YALAGANDULA, P., AL-FARES, M., AND MOGUL, J. C. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *NSDI* (2010).

[36] MYSORE, R. N., PAMBORIS, A., FARRINGTON, N., HUANG, N., MIRI, P., RADHAKRISHNAN, S., SUBRAMANYA, V., AND VAHDAT, A. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM* (2009).

[37] NAOR, M., AND WIEDER, U. Novel architectures for p2p applications: The continuous-discrete approach. In *SPAA* (2007).

[38] PANDURANGAN, G., ROBINSON, P., AND TREHAN, A. DEX: Self healing expanders. In *SPAA* (2014).

[39] PATURI, R., LU, D.-T., FORD, J. E., ESENER, S. C., AND LEE, S. H. Parallel algorithms based on expander graphs for optical computing. *Appl. Opt.* (1991).

[40] POPA, L., RATNASAMY, S., IANNACCONE, G., KRISHNAMURTHY, A., AND STOICA, I. A cost comparison of datacenter network architectures. In *CoNEXT* (2010).

[41] REINGOLD, O., VADHAN, S., AND WIGDERSON, A. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *FOCS* (2000).

[42] ROSEN, E., VISWANATHAN, A., AND CALLON, R. Multiprotocol Label Switching Architecture. RFC 3031, 2001.

[43] SHIN, J.-Y., WONG, B., AND SIRER, E. G. Small-world datacenters. In *SoCC* (2011).

[44] SINGLA, A., GODFREY, P. B., AND KOLLA, A. High throughput data center topology design. In *NSDI* (2014).

[45] SINGLA, A., HONG, C.-Y., POPA, L., AND GODFREY, P. B. Jellyfish: Networking data centers randomly. In *NSDI* (2012).

[46] UPFAL, E. An o(log n) deterministic packet-routing scheme. *J. ACM* (1992).

[47] VALADARSKY, A., DINITZ, M., AND SCHAPIRA, M. Xpander: Unveiling the Secrets of High-Performance Datacenters. In *HOTNETS* (2015).

[48] WISCHIK, D., RAICIU, C., GREENHALGH, A., AND HANDLEY, M. Design, implementation and evaluation of congestion control for multipath TCP. In *NSDI* (2011).

[49] WU, H., LU, G., LI, D., GUO, C., AND ZHANG, Y. Mdcube: a high performance network structure for modular data center interconnection. In *CoNEXT* (2009).

[50] YEN, J. Y. Finding the k shortest loopless paths in a network. *Management Science* (1971).

[51] ZOLA, J., AND ALURU, S. *Encyclopedia of Parallel Computing*. Springer.

A. DERANDOMIZING LIFTS

Recall the notion of "2-lifting" a graph from [9] (from Section 2): A *2-lift* of G is a graph obtained from G by (1) creating two vertices v_1 and v_2 for every vertex v in G; and (2) for every edge $e = (u, v)$ in G, inserting two edges (a matching) between the two copies of u (namely, u_1 and u_2) and the two copies of v (namely, v_1 and v_2). The latter can be done in one of two ways: connecting each u_i to v_i, which we will refer to as a "|| matching", or connecting u_1 to v_2 and u_2 to v_1, which we will refer to as a "X matching".

Our heuristic creates a "criss-cross 2-lift" of G as follows: enumerate the edges in G arbitrarily and then 2-lift G such that all edges with odd numbers correspond to || matchings in the 2-lift and edges with even numbers correspond to X matchings. Let $\bar{G}$ denote the criss-cross 2-lift of G. Intuitively, the heuristic then repeatedly checks if substituting some || matching with a X matching, or vice versa, can improve the edge expansion of the graph, and changes the graph accordingly. This goes on until no such change can improve edge expansion.

As computing the exact edge expansion of a given graph is NP-hard [28], our heuristic instead computes the *spectral gap*, which closely approximates edge expansion (see [24] for an explanation of spectral gap and its relation to edge expansion).

Algorithm 1: Deterministic 2-Lift Heuristic

```
Input: Graph G = (V, E)
Ḡ = (Ē, V̄) ← a "criss-cross" 2-lift of G
Improved ← True
while Improved do
    Improved ← False
    G̃ = (Ẽ, Ṽ) ← Ḡ
    for e = (u, v) ∈ E do
        if (u_1, v_1) ∈ Ẽ then
            // an || matching
            replace the edges (u_1, v_1), (u_2, v_2) with
            (u_1, v_2), (u_2, v_1) in G̃
        else
            // an X connection
            replace the edges (u_1, v_2), (u_2, v_1) with
            (v, u), (v', u') in G̃
        GapOld ← SpectralGap(Ḡ)
        GapNew ← SpectralGap(G̃)
        if GapNew > GapOrg then
            Ḡ ← G̃
            Improved ← True
Output Ḡ
```

Observe that this heuristic can be extended to derandomizing k-lifts for any constant $k > 0$ in a straightforward manner.

B. INCREMENTAL EXPANSION

Our deterministic heuristic takes as input a graph G and number N and adds N nodes to G such that the addition of each node involves few wiring changes. Intuitively, with the addition of each node our heuristic selects the $\frac{d}{2}$ worst links in terms of contribution to edge expansion, removes them, and connects the new node to all nodes that are incident to the removed links. Unfortunately, as computing edge expansion is intractable, the spectral gap (see [24]) is used as a proxy for edge expansion.

Algorithm 2: Incremental Expansion Algorithm

```
Input: Graph G = (V, E) and number (desired size)
N ∈ ℕ d ← deg(G)
T ← |V|
GapMap ← {}
while |V| < T + N do
    Q ← {}
    for e = (v_1, v_2) ∈ E do
        E ← E\e
        LapMatrix ← LaplacianMatrix(G)
        eGap ← SmallestEigenValue(LapMatrix)
        E ← E ∪ {e}
        GapMap[e] = eGap
    Sort GapMap by value (highest to lowest)
    for e = (v_1, v_2) ∈ Keys(GapMap) do
        if |Q| = d then
            break
        else if v_1 ∉ Q and v_2 ∉ Q then
            Q ← Q ∪ {v_1, v_2}
            E ← E\e
    Add a new node v to V
    Connect v to the vertices in Q
Output G
```

C. OPTIMIZING LPS

We have used the standard maximum multi-commodity flow model for all of our simulations, with several optimizations, the model is as follows:

For a given graph $G = (V, E)$ with a capacity function $c : E \to \mathbb{R}^+$. There are K commodities each of which is defined by the triplet $K_i = (s_i, t_i, d_i)$, where s_i, t_i are the source and destination vertices and d_i is the desired amount of data we would like to pass from s_i to d_i. We define for each commodity K_i the variable f_e^i describing the amount of flow the commodity i sends on the edge e, and let the vector f^i hold these values. As there are multiple commodities present we would like to enforce some sort of fairness, we do that by demanding that each commodity could send an equal (maximal) fraction of it's demand - we denote that fraction as α. Let A be the node-arc incidence matrix of the graph G defined as follows:

$$A_{v,e} = \begin{cases} 1 & \exists u \in V \; s.t. \; e = (v,u) \\ -1 & \exists u \in V \; s.t. \; e = (u,v) \\ 0 & else \end{cases}$$

and define for every commodity k the following vector b^k:

$$b^k_e = \begin{cases} d_k & \exists u \in V \; s.t. \; e = (s_k,u) \\ -d_k & \exists u \in V \; s.t. \; e = (u,t_k) \\ 0 & else \end{cases}$$

Now using the above notations we can write the problem presented above as the following LP:

$$\begin{aligned} & \max \alpha \\ & s.t. \\ \forall i \in [K] \quad & \sum_{e \in E} A_{i,e} \cdot f^i_e - \alpha \cdot b^i_e = 0 \\ \forall e \in E \quad & \sum_{i \in [K]} f^i_e \leq c(e) \\ \forall e \in E, \forall i \in [K] \quad & 0 \leq f^i_e \end{aligned}$$

Even tough it is correct this form is far from being efficient as there could be many commodities sharing the same source or destination, which are treated as a separate entity - for example in an all-to-all scenario there are $|V|^2$ such commodities. We will now present a more compact and efficient form allowing us to reduce the size of the commodities to be linear in $|V|$.

Firstly we convert the LP to its dual form:

$$\begin{aligned} & \min \beta \\ & s.t. \\ \forall i \in [K] \quad & \sum_{e \in E} A_{i,e} \cdot f^i_e - b^i_e = 0 \\ \forall e \in E \quad & \sum_{i \in [K]} f^i_e - \beta \cdot c(e) \leq 0 \\ \forall e \in E, \forall i \in [K] \quad & 0 \leq f^i_e \end{aligned}$$

C.1 Optimizing All-to-All

Assuming there are K commodities of the form (s_i, t_i, d_i), we look at the set of source vertices $v_1, v_2, \ldots, v_K$ and we aggregate it by source in the set $AggComm = s^i | 1 \leq i \leq F$ and we note that $F \leq K$. We next define the following function $D : V \times V \to \mathbb{R}^+$:

$$D(s,t) = \begin{cases} d_i & (s,t,d_i) \; is \; a \; commodity \\ -d_i & (t,s,d_i) \; is \; a \; commodity \\ 0 & else \end{cases}$$

And so the previous form could be written like so:

$$\begin{aligned} & \min \gamma \\ & s.t. \\ \forall s \in AggComm \quad & \sum_{e \in E} A_{s,e} \cdot f^s_e = \sum_{\substack{j \\ (s,t_j,d_j) \in AggComm}} d_j \\ \forall s \in AggComm, \forall v \in V \backslash \{s\} \quad & \sum_{e \in E} A_{v,e} \cdot f^i_e = D(v,s) \\ \forall e \in E \quad & \sum_{i \in [K]} f^i_e - \gamma \cdot c(e) \leq 0 \\ \forall e \in E, \forall i \in [K] \quad & 0 \leq f^i_e \end{aligned}$$

We note that the actual flow fraction, α, is then given by $\alpha = \frac{1}{\gamma}$.

D. THEORETICAL GUARANTEES

In section 4.2.3 we presented the following theorem, without proof - the proof can be found below.

THEOREM D.1. *For any d-regular graph G on n vertices, there exists a traffic matrix T and a d-regular graph G^* on n vertices such that $\alpha(G^*, T) \geq \Omega(\log_d n) \cdot \alpha(G, T)$.*

PROOF. Let $\pi : V \to V$ be a random permutation. Let G^* be the graph obtained by including an edge $\{u, v\}$ if and only if $\{\pi^{-1}(u), \pi^{-1}(v)\}$ is an edge in G (i.e. G^* is just a random isomorphism of G). Let T be the traffic matrix such that $T_{u,v} = 1$ if $\{u, v\} \in E(G^*)$ and $T_{u,v} = 0$ otherwise. Then clearly $\alpha(G^*, T) = 1$, since the traffic demands are exactly the edges of G^*.

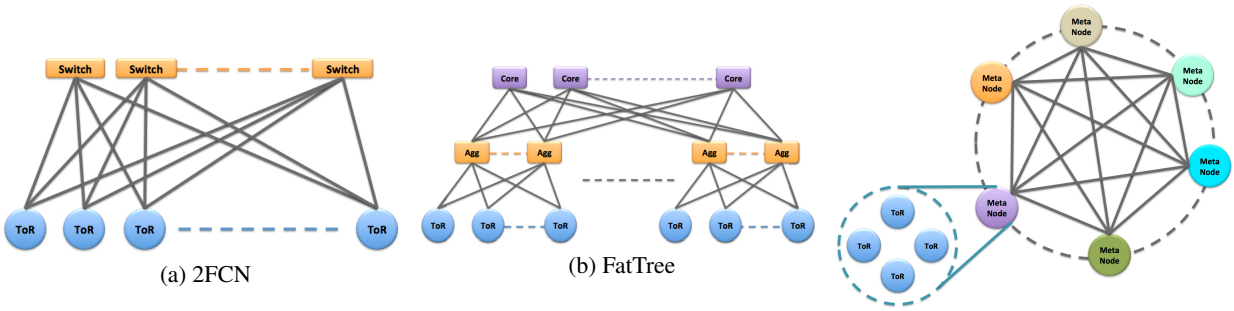
To analyze what happens in G, note that since G has maximum degree d, the average distance between two nodes is $\Omega(\log_d n)$. So since π is a random permutation, the expected distance in G between two nodes that are adjacent in G^* is also $\Omega(\log_d n)$. So by linearity of expectations, $\mathbf{E}\left[\sum_{\{u,v\}:T_{u,v}=1} dist_G(u,v)\right] \geq \Omega(dn \log_d n)$. Thus we can fix some π where this is true, i.e. where $\sum_{\{u,v\}:T_{u,v}=1} dist_G(u,v) \geq \Omega(dn \log_d n)$. Then if we send $\alpha(G,T)$ units of flow for every traffic demand, the total capacity used up is at least $\Omega(\alpha(G,T) \cdot dn \log_d n)$. On the other hand, the total capacity in G is only $|E| = dn/2$. Thus $\alpha(G,T) \leq O(1/\log_d n)$. Putting this together, we get that $\alpha(G,T) \leq O\left(\frac{1}{\log_d n}\right) \cdot \alpha(G^*, T)$. □

E. DEPLOYING XPANDERS

Throughout the physical analysis we assume a rack containing 52U, corresponding to the dimensions as in E15. A sketch of a Fat-Tree and 2-FCN networks can be found in Figure C14.

E.1 Container Datacenters

In section 7.1 we examined an Xpander network vs. 2-FCN with degrees 48 and 32. We gave a summarized results in Table 8, we provide additional results in Table E10 and Table E11



(a) 2FCN (b) FatTree (c) Xpander

Figure C14: A sketch of a 2-FCN, 3-FCN and Xpander network topologies

	#Switches	#Servers	Path Length Distribution in % (Between ToRs) [1,2,3]	Avg. Path Between ToRs	Path Length Distribution in % (Between Servers) [0,1,2,3]	#Edge Disjoint Paths Between ToRs	All-to-All Server to Server Throughput
k=32	42 vs. 48 (87.5%)	504 vs. 512 (98.44%)	[48.78,48.78, 2.44] vs. [0,100,0]	1.5366 vs. 2 (76.83%)	[2.38,47.62,47.62,2.38] vs. [3.12,0,96.88,0]	20 vs. 16 (125%)	109%
k=48	66 vs. 72 (91.76%)	1,056 vs. 1,152 (91.67%)	[49.23,49.23,1.54] vs. [0,100,0]	1.5231 vs. 2 (76.15%)	[1.52,48.48,48.48,1.52] vs. [2.08,0,97.92,0]	32 vs. 24 (133%)	142%

Table E10: Xpander vs. 2-FCN, theoretical analysis.

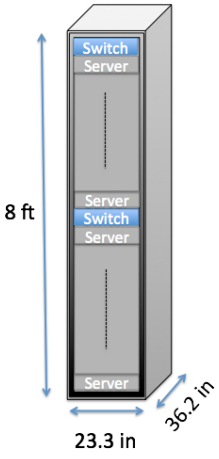


Figure E15: Server rack dimensions

E.2 Xpanders *vs.* Fat-Trees

In Section 5 we identified, for each uniform fat-tree topologies with even degree between 8 and 24, a more cost effective expander datacenter. Table E12 presents the details regarding these topologies.

Cost Analysis In Section 7.2.1 we suggested a possible deployment for Fat-Tree and Xpander topologies, Table E13 discusses the length of cables resulting from this analysis. We present a possible floor plan for both Xpander and Fat-Tree in figures Figure E16 and Figure E17 respectively.

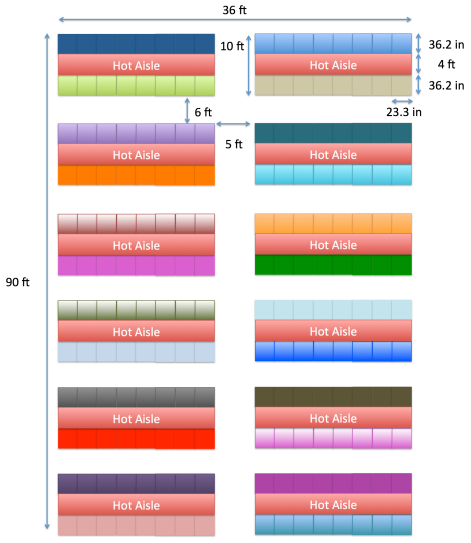


Figure E16: A $k = 30$ Xpander floor plan

	#VR	#SR in VR	#Physical Racks	#Cables	Cable Length
k=32	42 vs. 32	12 vs. 16 (75%)	11 vs. 11 (100%)	420 vs. 512 (82%)	4.2km vs. 5.12km
k=48	66 vs. 48	16 vs. 24 (66%)	22 vs. 25 (88%)	1056 vs. 1152 (91.6%)	10.56km vs. 11.52km

Table E11: Xpander vs. 2-FCN, physical analysis.

Fat-Tree degree	#Switches (#Servers) (Fat-Tree)	#Switches (#Servers) (Xpander)	#Ports used for servers (Xpander)	#Ports used for switches (Xpander)	#Ports used (Xpander)	Server to Server throughput (Xpander)	Server to Server throughput (Fat-Tree)
8	80 (128)	64 (128)	2	6	8/8	$9.786 \cdot 10^{-3}$	$8.065 \cdot 10^{-3}$
10	125 (250)	125 (250)	2	8	10/10	$6.435 \cdot 10^{-3}$	$4.082 \cdot 10^{-3}$
12	180 (432)	144 (432)	3	8	11/12	$2.428 \cdot 10^{-3}$	$2.347 \cdot 10^{-3}$
14	245 (686)	237 (711)	3	10	13/14	$1.805 \cdot 10^{-3}$	$1.473 \cdot 10^{-3}$
16	320 (1024)	256 (1024)	4	12	16/16	$1.177 \cdot 10^{-3}$	$9.84 \cdot 10^{-4}$
18	405 (1458)	365 (1460)	4	12	16/18	$9.51 \cdot 10^{-4}$	$6.9 \cdot 10^{-4}$
20	500 (2000)	400 (2000)	5	15	20/20	$5.98 \cdot 10^{-4}$	$5.03 \cdot 10^{-4}$
22	605 (2662)	544 (2720)	5	16	21/22	$4.58 \cdot 10^{-4}$	$3.77 \cdot 10^{-4}$
24	720 (3456)	576 (3456)	6	17	23/24	$3.22 \cdot 10^{-4}$	$2.9 \cdot 10^{-4}$

Table E12: Xpanders vs. Fat-Trees, all-to-all analysis

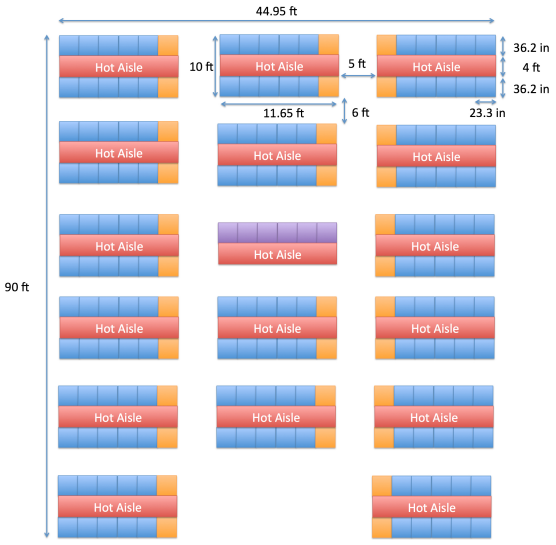


Figure E17: A $k = 32$ fat-tree floor plan

E.3 Xpander *vs.* Low-Diameter Networks

Below are the simulations for Jellyfish, as described in section 6.

#Ports per Switch (Sw2Sw / Total)	#SW	#SRV	Bisection Bandwidth	Expansion
5/8	100%	100%	81%	65%
7/11	96%	96%	68%	60%
11/17	97%	97%	93%	73%
17/26	96%	96%	102%	85%
19/29	104%	104%	101%	90%
25/38	99%	99%	107%	96%
29/44	99%	99%	115%	98%
35/51	102%	102%	120%	102%
43/65	99%	99%	117%	107%
47/71	99%	99%	122%	108%
55/83	96%	96%	120%	112%

Table E15: Jellyfish vs. Slim-Fly.

F. ADDITIONAL FIGURES

We present additional figures below, the X axis is the number of switches in the simulated network and their switch-to-switch degree d, we also show results for other deterministic expander datacenter using a network topology called LPS [30].

F.1 All-to-All

We present additional results for the all-to-all simulations, as discussed in Section 4.2, in Figure F18 and Figure F19.

	5m	10m	15m	20m	25m	30m	35m
FatTree	8,192	2,048	3,072	2,048	1,024	0	0
Xpander	1,536	2,880	3,264	2,496	1,728	960	384

Table E13: Amount of cables used by $k = 32$ fat-tree vs. matching Xpander

Switch Degree	#Switches	#Servers	Path Length Distribution in % (Between ToRs) [1,2,3,4]	Avg. Path Between ToRs	Path Length Distribution in % (Between Servers) [0,1,2,3,4]	#Edge Disjoint Paths Between Tors
30 vs. 32 (93.75%)	1,152 vs. 1,280 (90%)	8,064 vs. 8,192 (98.44%)	[2,32.32,65.56,0.12] vs. [0,2.9,97.1,0]	2.6379 vs. 3.9413 (66.93%)	[0.08,2,32.3,65.51,0.11] vs. [0.19,0,2.93,0,96.88]	23 vs. 16 (143%)

Table E14: Xpander vs. fat-tree, theoretical analysis

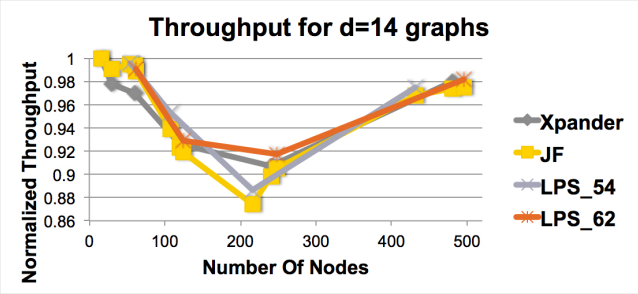


Figure F18: Results for all-to-all throughput ($d = 14$).

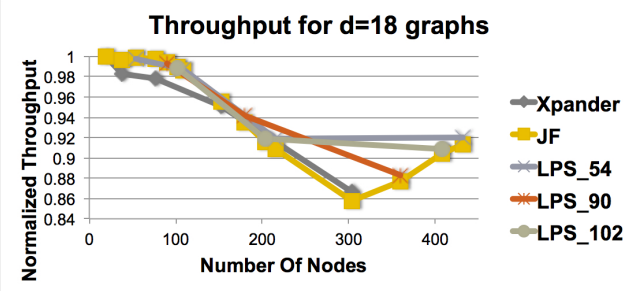


Figure F19: Results for all-to-all throughput ($d = 18$).

F.2 Resiliency To Failures

In section 4.3 we compared the ability of Xpander, Jellyfish and Fat-Tree networks to withstand random link failures. We present additional results for these simulations in Figure F20.

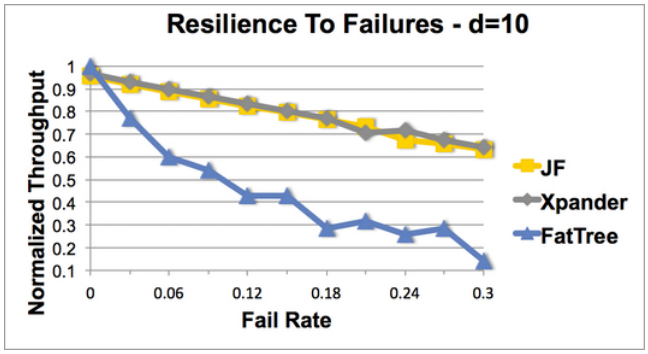


Figure F20: Server-to-server throughput degradation with failures ($d = 10$).