Jellyfish: Networking Data Centers, Randomly

Ankit Singla[†,*] *Chi-Yao Hong*[†], *Lucian Popa*[♯], *P. Brighten Godfrey*[†]
[†] *University of Illinois at Urbana–Champaign*
[♯] *University of California, Berkeley*

1 Introduction

Data centers today form the backbone of cloud operations. A well provisioned data center network is important to ensure that servers do not face bandwidth bottlenecks to utilization; to isolate services from each other; and to gain more freedom in workload placement, rather than having to tailor placement of workloads to where bandwidth is available [16]. As a result, a significant body of work has tackled the problem of building high network capacity interconnects [1, 10–13, 26, 28, 29].

One crucial problem that has been ignored in these designs is that of incremental expansion of the network, *i.e.*, adding servers and network capacity incrementally to the data center. This may be motivated by growth of the user base, which requires more servers, or by the deployment of more bandwidth-intensive applications. Such expansion can be made feasible by either planned overprovisioning of space and power, or by an upgrade of the server base. The latter enables some of the old servers to be replaced by a larger number of more powerful and, at the same time, less power-consuming new servers.

Industry experience indicates that incremental expansion is an important problem. Consider the growth of Facebook's data center server population from $\sim$ 30,000 in November 2009 to more than 60,000 by June 2010 [24]. While Facebook has added new data center facilities too, much of this growth is more incremental in existing facilities ("adding capacity on a daily basis" [23]). For instance, Facebook announced that it will double the size of its facility at Prineville, Oregon by early 2012 [9]. Industry experts have also identified incremental build-out as a useful strategy to reduce cap-ex up-front [20].

Do current high-bandwidth data center network proposals allow incremental growth? Consider the fat-tree proposal [1] as an illustrative example. The entire structure is completely determined by the port-count k of the switches available. This is limiting in at least two ways. First, it makes the design space very coarse: full bisection bandwidth fat-trees can only be built at sizes 3456, 8192, 27648, and 65536 corresponding to the commonly available port counts of 24, 32, 48, and 64. Second, even if (for example) 50-port switches were available, the smallest incremental upgrade from the 48-port switch fat-tree would be 3,602 servers. Moreover, this "incremental" growth would require replacing all the 48-port switches by 50-port switches. There is, of course, the possibility of making localized changes like replacing a switch with one with larger port count – this necessarily requires either the servers in the vicinity to share capacity unevenly compared to the rest of the server pool. Thus, without compromise on its structure, bandwidth or cost, a fat-tree is not amenable to incremental growth. Other topologies have similar problems: a hypercube [3] allows power-of-2 sizes, a de Bruijn-like construction [27] allows power-of-3 sizes, etc.

Since it seems that *structure* hinders incremental expansion, we propose the opposite: a random network interconnect. The approach, which we call **Jellyfish**, constructs a *random graph* topology at the switch layer. The inherently sloppy nature of this design has the potential of being significantly more flexible than past designs. Additional components – racks of servers or switches to improve capacity – can be incorporated with a few random edge swaps. The design naturally supports heterogeneity, allowing the addition of newer network elements with higher port-counts as they become available, unlike past proposals which depend on certain regular port-counts [1, 11–13, 26, 28]. Jellyfish also allows construction of arbitrary-size networks, unlike past proposals discussed above which limit the network to very coarse design points dictated by their structure.

Are we sacrificing high bandwidth for incremental growth? No. To the contrary, we show that Jellyfish supports *more* servers at full bisection bandwidth than an equal-cost fat-tree [1] and with lower mean path length. Jellyfish also provides high connectivity and path diversity. In addition, as we discuss later, Jellyfish is resilient to failures and miswirings during construction.

On the other hand, a random data center network presents several interesting and important challenges that must be addressed for it to be viable. Among these are a suitable routing mechanism (schemes depending on a structured topology are not applicable), physical construction, and cabling layout. We discuss these chal-

[*] Fittingly, a coin-toss decided the first among the first two authors.

lenges and potential directions for solutions. We argue that given Jellyfish's dramatic flexibility and improved efficiency, these challenges are worth an exploration.

We next discuss related work (§2) and the Jellyfish topology (§3). We then outline opportunities which Jellyfish offers (§4), and the key challenges it faces (§5).

2 Related Work

Several recent data center network proposals [1, 11, 26] for full bisection bandwidth have been based on a special Clos network, called a fat-tree. Later work has employed servers as network forwarding elements [12, 13, 30]. All these designs are based on the application of special structure for the topology and routing. Moreover, carefully-structured expander graphs have been studied extensively in the high-performance computing literature [19].

However, none of these architectures address the issue of *incremental expansion* of the network. For some of these, adding servers while preserving the structural properties would require replacing a large number of network elements and extensive rewiring. MDCube [30] allows expansion at a very coarse rate (several thousand servers). DCell and BCube [12, 13] allow expansion to an *a priori* known target size, but require servers with free ports reserved for planned future expansion.

A recent approach to high-bandwidth interconnects exploits optical or wireless networking technology [10, 15, 18, 28, 29] to set up on-demand high-bandwidth network connections where required. However, this work does not address incremental network expansion. In addition, this work only applies to data centers which see *sparse* traffic patterns and not high volume all-to-all traffic (like high-performance computing [17]).

To the best of our knowledge, the only work which directly addresses incremental growth of a data center is LEGUP [8]. LEGUP starts with a Clos network and adds servers and switches by solving an optimization problem (approximately) to determine the best network updates. However, such approaches are fundamentally limited by the need to emulate a given network structure and by starting the upgrade process from a given rigid structure. For this reason, approaches like LEGUP can still result in significant overhauls of the topology even when adding just a few new servers, or can compromise the available bisection bandwidth. In this paper, we show that Jellyfish provides a simple model to expand the network to almost any desirable scale. In addition, we show that Jellyfish networks are inherently less expensive than fat trees, which are representative Clos topologies (widely used for datacenter interconnects [1, 11, 26]). A quantitative comparison with LEGUP would be an interesting subject of future work.

Scafida [14] proposes using a degree-constrained modification of scale-free graphs in the data center. This improves on a fat-tree because it can produce networks of any given size, and has lower mean path length. However, the topology construction does not support *incremental* expansion. Also, Scafida has slightly lower bisection bandwidth and marginally higher diameter than an equal-cost fat-tree.

Random graphs have been examined in the context of communication networks [22]. The novelty of our proposed research agenda is to apply random graphs to allow incremental expansion in data center networks.

3 Jellyfish Topology

The Jellyfish approach is to construct a random graph at the switch layer. Each switch i has some number k_i of ports, of which it uses r_i to connect to other switches, and uses the remaining $k_i - r_i$ ports for servers. In the simplest case, which we consider by default throughout this paper, every switch has the same number of ports and servers: for all i, $k = k_i$ and $r = r_i$. We let N be the number of switches, so the network supports $N(k - r)$ servers. In this case, the network is a *random regular graph*, which we denote as RRG(N, k, r). The particular topology is *uniform-randomly* sampled from the sample space of all such topologies. This is a well known construct in graph theory and has several desirable properties as we shall discuss later.

Why should this work? Intuitively, random graphs fulfill two key goals. First, they are very efficient: theoretical results show that almost every RRG has low diameter and high bisection bandwidth [4, 6]. Second, they are highly flexible: they can be built with any number of nodes and regular or heterogeneous degree distributions, and as we describe, are easy to modify incrementally.

Construction: Formally, RRGs are sampled uniformly from the space of all r-regular graphs. This is a complex problem in graph theory [21]; however, a simple procedure can produce a "uniform-enough" random graph which empirically gives us the desired bisection bandwidth and path length distribution. One can simply pick a random pair of nodes with free ports (preferring node-pairs that are not already neighbors), join them with an edge, and repeat until no further edges can be added. If a switch remains with ≥ 2 free ports, these can be incorporated by removing a random existing link, and linking its endpoints to two free ports. Thus only a single unmatched port might remain across the whole datacenter.

Using the above idea, we generate a topology blueprint for the physical interconnection. Note that allowing human operators to 'wire at will' may result in poor topologies due to the inherent bias involved (for instance, favoring shorter cables over longer ones). We briefly discuss

physical construction in terms of cabling later in §5.

4 Benefits of Jellyfish

We elaborate on the flexibility and efficiency of Jellyfish below. In general, we only sketch the benefits; quantitative measures and comparisons to other designs are a necessary avenue for a complete study in the future.

4.1 Flexibility

Incremental Expandability: Jellyfish's construction makes it amenable to incremental expansion by means of adding either servers and/or network capacity (if it's not full-bisection bandwidth already), with the increments being as small as just one switch or a switch together with some servers. Jellyfish can be expanded such that rewiring is limited to only twice the number of ports being added to the network; and the desirable properties are maintained: high bandwidth and short paths at low cost.

As an example, consider an expansion from an RRG(N, k, r) topology to RRG($N+1$, k, r). In other words, we are adding one switch u (that will be connected to k previously existing switches), together with r new servers. To connect the newly added switch, we pick a random link (v, w) such that this new switch is not already connected with either v or w, remove it, and add the two links (u, v) and (u, w), thus using 2 ports on u. This process is repeated until all ports are filled (or a single odd port remains, which could be matched with another free port on an existing switch, used for a server, or left free). This completes incorporation of the new switch (with servers), and can be repeated for as many new switches as desired.

We note that our expansion procedure may not produce uniform-random RRGs. Verifying that the expanded topologies maintain the desirable properties is the subject of future research. As preliminary evidence, we compared the average path length and diameter in RRGs which we generated from scratch at different sizes, with the average path length and diameter in an RRG which begins at 1,200 servers and evolves incrementally according to the above procedure. These turn out to be nearly identical (Fig. 1).

A similar procedure can be used to expand network capacity for an under-provisioned Jellyfish network. In this case, instead of adding a switch with servers, we only add the switch, connecting all its ports to the network.

Jellyfish also allows for heterogeneous expansion: nothing in the procedure above requires that the new switches have the same number of ports as the existing switches. Thus, as new switches with higher port-counts become available, they can be readily used, either with servers, or by themselves to augment the interconnect's bandwidth.

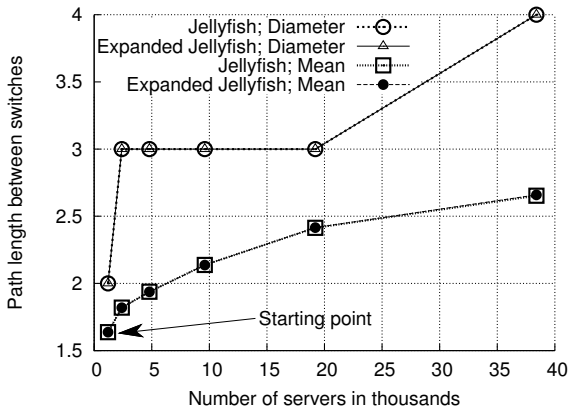


Figure 1: *RRG path length versus number of servers, with* $k = 48$ *port switches of which* $r = 36$ *connect to other switches and* 12 *connect to servers. Each data point is derived from* 10 *graphs.*

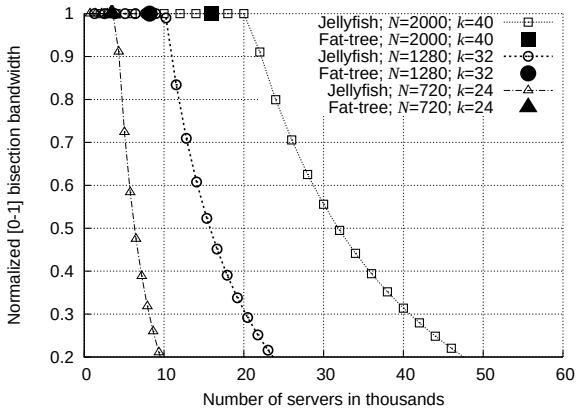
Arbitrary-sized Networks: Several existing proposals provide only the construction of interconnects with very coarse parameters. For instance, a 3-level fat-tree allows only $k^3/4$ servers with k being restricted to the port-count of available switches, unless some ports are left unused. This is an arbitrary constraint, extraneous to operational requirements. In contrast, Jellyfish permits any number of racks of servers to be networked efficiently.

Resilience to Failures: Jellyfish provides good path redundancy; in particular, an r-regular random graph is almost surely r-connected [5]. Also, the randomness of the topology implies that the graph maintains its structure (or rather, the lack of it!) in the face of link or node failures – a random graph topology with one switch failure is just another random graph topology of slightly smaller size, with a few unmatched ports on some switches.

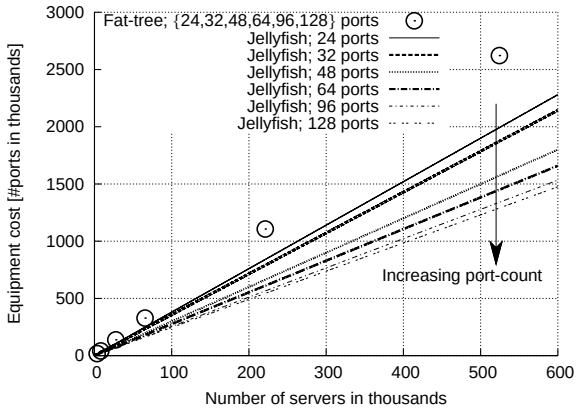
Resilience to miswirings: Intuitively, a few miswirings just make it a different random graph than the one 'planned'. Unless these are very numerous and biased, Jellyfish will continue to preserve its properties.

4.2 Efficiency

Capacity: In addition to Jellyfish's incremental expandability, Jellyfish is also a *more bandwidth-efficient* topology than a fat tree. We use both bisection bandwidth and throughput under random permutation traffic as metrics for topology capacity. Bisection bandwidth, denoted by B, measures the *worst-case* bandwidth between two equal-size partitions of the network. This can be normalized to a value between 0 and 1 by dividing it by the total line-rate bandwidth of the servers in one



(a)



(b)

Figure 2: *Jellyfish vs. fat-tree: (a) Normalized bisection bandwidth versus the number of servers supported; equal-cost curves, (b) Equipment cost versus the number of servers for commodity-switch port-counts* 24, 32, 48, 64, 96, 128.

partition.

Fig. 2(a) shows that to support a given number of servers (x axis) with full bisection bandwidth ($B = 1$), Jellyfish uses significantly fewer switches. For instance, at the same cost as a fat-tree with 16,000 servers, Jellyfish can support $>$ 20,000 servers at full bisection bandwidth. Also, Jellyfish allows the freedom to accept lower bisection bandwidth, in exchange for supporting more servers (as in Fig. 2(a)) or cutting costs by using fewer switches.

Fig. 2(b) shows that the cost of building a full bisection-bandwidth network increases more slowly with the number of servers for Jellyfish than for the fat-tree, especially for high port-counts. Also, the design choices for Jellyfish are essentially continuous, while the fat-tree allows only certain discrete jumps in size which are further restricted by the port-counts of available switches.

The above figures are computed via explicitly setting parameters for the fat tree. For Jellyfish they are computed via a lower bound of Bollobás [4]: in almost every r-regular graph with N nodes, every set of $u = N/2$ nodes is joined by at least $N(\frac{r}{4} - \frac{\sqrt{r \ln 2}}{2})$ edges to the rest of the graph. Thus, the bisection bandwidth B for RRG(N, k, r) is *at least*

$$\min\left(\frac{N(\frac{r}{4} - \frac{\sqrt{r \ln 2}}{2})}{N(k-r)/2}, 1\right) = \min\left(\frac{r/2 - \sqrt{r \ln 2}}{k-r}, 1\right).$$

As this expression is independent of N, bisection bandwidth stays constant as the network grows. However, the end-to-end capacity of the network is determined not only by the bisection bandwidth of the network, but also by the path length. As N increases and r remains constant, the average path length increases, which in turn reduces the capacity of the network. But since path length increases very slowly (as discussed below), bandwidth per server is likely to remain high even for relatively large factors of growth in N. Thus, operators can keep the servers-per-switch ratio constant even under large expansion, with only minor bandwidth loss.

Path Length: Short path lengths are important to ensure low latency, and to minimize network utilization. In this context, we note that the theoretical *upper-bound* on the diameter is fairly small: Bollobás and de la Vega [6] showed that in almost every r-regular graph with N nodes, the diameter is at most $1 + \lceil \log_{r-1}((2 + \epsilon)rN \log N) \rceil$ for any $\epsilon > 0$. Thus, the server-to-server diameter is at most $3 + \lceil \log_{r-1}((2 + \epsilon)rN \log N) \rceil$. Thus, the path length increases logarithmically (base r) with the number of nodes in the network.

The average path length and diameter (Fig. 1) in Jellyfish is much smaller than in the fat-tree. For example, for RRG(3200, 48, 36) with 38,400 servers, the average path length between switches is $<$ 2.7 (Fig. 1), while the fat-tree's average is $\sim$ 4. While the diameter for the largest Jellyfish topology in our experiment, RRG(3200, 48, 36), is the same as the fat-tree - 4, the 99.99$^{\text{th}}$ percentile switch-to-switch path-length across 10 runs did not exceed 3 for any of the topology sizes in Fig. 1.

5 Research Challenges

While random network topologies offer significant opportunities in flexibility, incremental expansion, and efficiency, they represent a dramatic departure from past data center network architectures, putting forth several challenges.

Routing: While structure has been the lead villain in our story so far, it lends itself to simple and efficient routing schemes. However, techniques such as spanning trees, which are used in switched networks, are not applicable to other recently proposed architectures either, because they do not exploit path diversity. Fat-tree-like

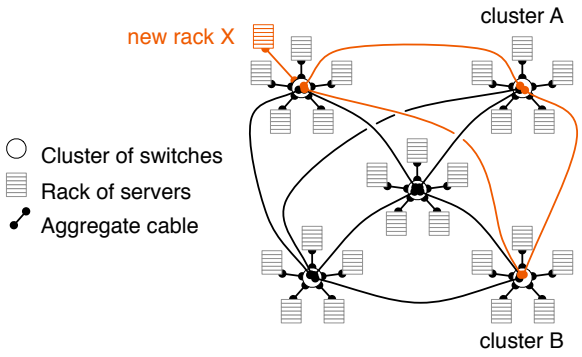


Figure 3: *Physical layout and incremental expansion.*

topologies can benefit from Valiant Load Balancing over ECMP [11] but even there, prior work has shown a gap of $\sim 20\%$ from the optimal throughput [2]. This has led recent work to propose a centralized controller to route *elephant* flows, and a randomization-based strategy for the *mice* [2, 25]. This should work well for Jellyfish too; a quantitative study remains the subject of future work.

Cabling: While the number of cables needed for a given bisection bandwidth in Jellyfish is smaller than that in the fat-tree, the randomness works against obvious cabling optimizations. However, prior work indicates that most of the cabling expenditure (which itself is only 3-8% of network equipment cost in typical data center designs) goes towards manual labor, and depends more significantly on the 'spread' of cabling, than on the number or length [27]. Schemes which result in a small number of parallel *aggregates* of cables are thus more desirable in comparison to schemes which run similar number/length of cables without aggregation. Cable aggregates are also likely to reduce spatial footprint of cabling and make cable management easier. With these motives, we discuss a first-cut approach to cabling in Jellyfish which attempts to organize cabling to target aggregation of cables.

Fig. 3 illustrates a physical Jellyfish layout. The core idea is to form clusters of racks with aggregated cables running between clusters. To connect N racks, we group them into N/T *clusters* of size T each. The switches for all racks of servers in a cluster could be placed at the cluster-center, with server racks around them. Using the interconnection blueprint, we can find the set of connections between each pair of clusters and run consolidated cable assemblies between these. This leads to all the *long* cables being in these cable assemblies. This motivates using a small number of *dense* clusters. However, packaging a large number of racks in a cluster is challenging because of physical space constraints. We consider $T = \sqrt{N}$ a reasonable middle-ground choice, which changes slowly with N and leads to a small number of cable aggregates; a small number of intra-cluster cables; and small lengths for intra-cluster cables. For example, consider a full-bisection Jellyfish data center with 30,000 servers built using 48-port switches. This requires at most 2,500 switches (based on the bisection bandwidth lower bound [4]), each connected to 12 servers. Thus, we create 50 clusters of 50 racks each (note that we use the term rack here to refer to the servers connected to the same switch rather than a physical rack). Compared with a fat-tree with 27,648 servers, this Jellyfish topology supports 8% more servers with 13% fewer cables, and the number of long cable assemblies which run between clusters is $\binom{50}{2} = 1{,}225$. The 12 cables from each server-rack to its corresponding switch in its cluster can also be aggregated and will be only a few meters long.

Incremental addition of racks requires either creation of new clusters (which need not be 'full') or expansion of existing clusters. The T parameter may diverge from $\sqrt{N}$, but given the $\sqrt{N}$ scaling, this should not cause significant problems for reasonable expansions. In Fig. 3, we add a new rack of servers X to a cluster. A new switch for this rack is added to the cluster's rack of switches, and additional cables are added to existing cable aggregates to connect X's switch to other clusters. The ports for these connections are vacated by picking random edges to break, as described in §4.

Debugging: Given the overwhelming size and complexity of data center networks, even structured topologies do not offer 'easy' debugging. This has prompted a trend towards automated debugging applicable to general topologies [7], from which Jellyfish will also benefit.

6 Conclusion

We argue that random graphs are a highly flexible architecture for data center networks. They represent a novel approach to the significant problems of incremental and heterogeneous expansion, still enabling high capacity, short paths, and resilience to failures and miswirings. While the issues of routing, cabling layout, multipath congestion control, etc. demand more analysis, the direction appears promising.

We thank Chandra Chekuri, Indranil Gupta, Gianluca Iannaccone, Steven Lumetta, Christos Papadimitriou, Sylvia Ratnasamy, Marc Snir, and the anonymous reviewers for helpful comments. This work was supported in part by National Science Foundation grant CNS 10-40396.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[2] T. Benson, A. Anand, A. Akella, and M. Zhang. The case for fine-grained traffic engineering in data-centers. In *INM/WREN*, 2010.

[3] L. N. Bhuyan and A. D. P. Generalized hypercube and hyperbus structures for a computer network. In *IEEE Transactions on Computers*, 1984.

[4] B. Bollobás. The isoperimetric number of random regular graphs. 1988.

[5] B. Bollobás. Random graphs, 2nd edition. 2001.

[6] B. Bollobás and W. F. de la Vega. The diameter of random regular graphs. In *Combinatorica 2*, 1981.

[7] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu. Generic and automatic address configuration for data center networks. In *SIGCOMM*, 2010.

[8] A. R. Curtis, S. Keshav, and A. Lopez-Ortiz. Legup: using heterogeneity to reduce the cost of data center network upgrades. In *CoNEXT*, 2010.

[9] Facebook. Facebook to expand prineville data center. http://goo.gl/fJAoU.

[10] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *SIGCOMM*, 2010.

[11] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. In *SIGCOMM*, 2009.

[12] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, 2009.

[13] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: A scalable and fault-tolerant network structure for data centers. In *SIGCOMM*, 2008.

[14] L. Gyarmati and T. A. Trinh. Scafida: A scale-free network inspired data center architecture. In *SIGCOMM CCR*, 2010.

[15] D. Halperin, S. Kandula, J. Padhye, V. Bahl, and D. Wetherall. Augmenting data center networks with multi-gigabit wireless links. In *SIGCOMM*, 2011.

[16] J. Hamilton. Datacenter networks are in my way. http://goo.gl/Ho6mA.

[17] J. Hamilton. High performance computing hits the cloud. http://goo.gl/ovylr.

[18] S. Kandula, J. Padhye, and V. Bahl. Flyways to de-congest data center networks. In *HotNets*, 2009.

[19] T. Leighton. Introduction to parallel algorithms and architectures: Arrays, trees, and hypercubes. 1992.

[20] A. Licis. Data center planning, design and optimization: A global perspective. http://goo.gl/Sfydq.

[21] B. D. McKay and N. C. Wormald. Uniform generation of random regular graphs of moderate degree. In *J. Algorithms*. Academic Press, Inc., 1990.

[22] A. B. Michael, M. Nolle, and G. Schreiber. A message passing model for communication on random regular graphs. In *International Parallel Processing Symposium (IPPS)*, 1996.

[23] R. Miller. Facebook now has 30,000 servers. http://goo.gl/EGD2D.

[24] R. Miller. Facebook server count: 60,000 or more. http://goo.gl/79J4.

[25] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee. Devoflow: cost-effective flow management for high performance enterprise networks. In *Hotnets*, 2010.

[26] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM*, 2009.

[27] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica. A cost comparison of datacenter network architectures. In *CoNEXT*, 2010.

[28] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang. Proteus: a topology malleable data center network. In *HotNets*, 2010.

[29] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan. c-through: Part-time optics in data centers. In *SIGCOMM*, 2010.

[30] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang. Mdcube: A high performance network structure for modular data center interconnection. In *CoNEXT*, 2009.