Flat-tree: A Convertible Data Center Network Architecture from Clos to Random Graph

Yiting Xia T. S. Eugene Ng
Rice University

Abstract

Clos networks are easy to implement, whereas random graphs have good performance. We propose flat-tree, a convertible data center network architecture, to combine the best of both worlds. Flat-tree can change the network topology dynamically, so the data center can be implemented as a Clos network and be converted to approximate random graphs of different sizes. To serve the heterogeneous workloads in data centers, flat-tree can organize the network into functionally separate zones each having a different topology. Workloads are placed into suitable zones that best optimize the performance. Simulation results demonstrate that flat-tree has similar performance to random graphs.

CCS Concepts

•**Networks** → **Network architectures; Physical topologies; Data center networks;**

Keywords

Convertible data center; Clos networks; Random graph

1. INTRODUCTION

The fundamental trade-off in data center network design is easy implementation versus good performance. Clos, or multi-rooted tree, is the *de-facto* standard data center network architecture because of easy implementation [1, 21]. Figure 2b shows an example Clos network. The central wiring between switches in adjacent layers are relatively easy to manage, and the network can be expanded to arbitrary size by adding stages. Bandwidth oversubscription can occur at any layer of switches to save cost. Modular Pods are usually adopted as building blocks to further ease network deployment and management. However, Clos networks have suboptimal throughput, as traffic needs to traverse up and down the network hierarchy and the resulting inefficiency exacerbates oversubscription.

In contrast, random graphs are proven to have optimal throughput [22, 23]. Without rigid structures, switches are more directly connected at shorter path lengths. If implemented using the same switches and servers as a Clos network, a random graph can provide richer bandwidth and effectively alleviate the oversubscription problem. To address the heterogeneous workloads in data centers, it is desirable to construct random graphs at different scales to adapt to the various service cluster sizes [6, 8, 9, 17], e.g. a network-wide random graph to serve large clusters and regional random graphs as part of the network to serve small ones. Yet the neighbor-to-neighbor wiring between random switch pairs are complicated, making real-world implementation a daunting task.

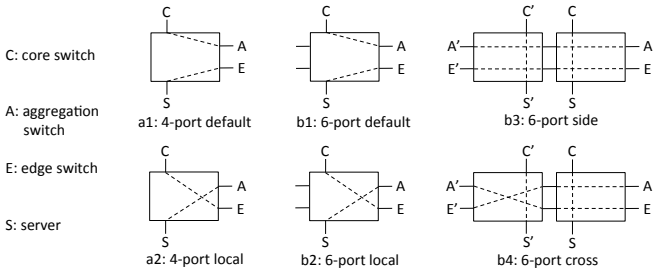

Figure 1: Converter switch configurations

This dilemma poses a natural question: is it possible to have random-graph-like performance at various scales with Clos-like implementation simplicity?

We address this question by an unconventional proposal: a convertible data center network architecture called flat-tree[1], which converts topologies between Clos and approximated random graphs. We combine the best of both worlds by building the data center as a Clos network and converting it to approximate random graphs at different scales.

Flat-tree leverages inexpensive small port-count converter switches to convert topologies dynamically. By changing the configurations of the converter switches, cables are rewired to different outgoing connections, as if they were unplugged and replugged manually. Flat-tree takes a pragmatic approach to start from a Clos network and addresses challenges of flattening the tree structure to approximate random graphs. Specifically, how to equalize switches in different layers and relocate servers from edge to aggregation and core switches? How to break the hierarchy and connect the network core and edge directly? How to enable connections between switches in the same layer at minimum wiring complexity?

Flat-tree inherits the merits of packaging and wiring from Clos networks. It adopts the modular Pod design. Additional hardware and wiring are packaged in Pods, leaving the same external connectors as a Clos counterpart. Pods are connected to core switches with a customized regular wiring pattern. Adjacent Pods are interconnected through multi-link side connectors to allow simple neighbor-wise wiring.

HotNets-XV, November 09 - 10, 2016, Atlanta, GA, USA

 ISBN 978-1-4503-4661-0/16/11…$15.00

DOI: http://dx.doi.org/10.1145/3005745.3005763

[1]The name "flat-tree" captures the dual nature of the proposed architecture. It can function as approximated random graphs ("flat" networks) and Clos (multi-rooted "tree"). It is as easy to implement as a "tree" network and has good performance as "flat" networks.

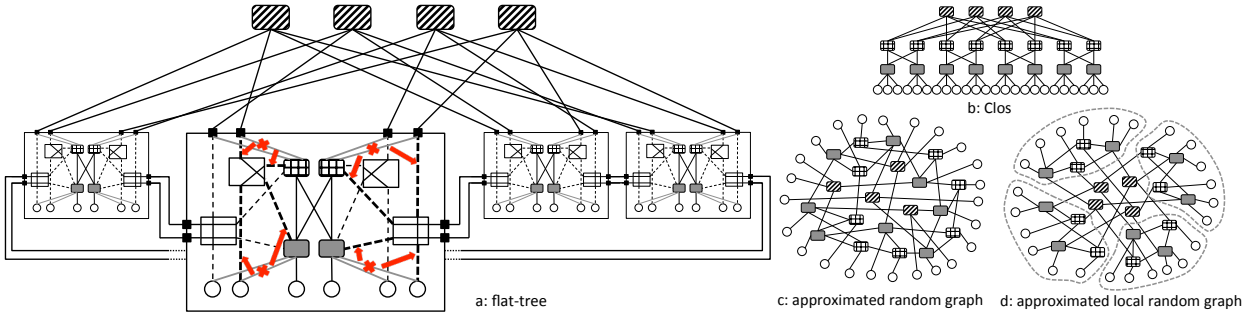

Figure 2: Example flat-tree network and some achievable topologies. Core switches in stripe, aggregation switches in grid, edge switches in shade, and servers as circles. Gray lines are connections in the original Clos network, which are replaced with the dashed links connected to converter switches to form flat-tree. The converter switches show the configuration for approximated random graph. Flat-tree uses a customized wiring pattern to connect Pods to core switches.

Flat-tree can approximate random graphs at different scales, ranging from a Pod, to a subnetwork comprising multiple Pods, to the entire network. It can also function as Clos, which benefits applications that require rich equal-cost redundant links, predictable path length, and rack-level locality. Flat-tree can operate in hybrid mode: the network is organized into functionally separate zones each having a different topology. Workloads are placed into suitable zones to optimize their performance. As the workloads change, the network can be reorganized to adapt to the new requirements.

Simulation results show that the performance of flat-tree is close to random graphs. Compared to a network-wide random graph, the difference in average path length is within 5% and the difference in throughput for large-clustered traffic is negligible. Flat-tree in hybrid mode optimizes traffic in different zones without interference and achieves the same throughput as separate flat-tree networks.

2. FLAT-TREE ARCHITECTURE

2.1 Motivating Example

We use the simple flat-tree example in Figure 2 to demonstrate how to convert a Clos network to an approximated random graph. The gray lines represent original connections in the Clos Pod that need to be replaced by the dashed links in the flat-tree Pod. The most notable differences between Clos and random graphs are server distribution and types of wires. In Clos networks, servers are attached to edge switches only and all links are hierarchical, either between edge and aggregation switches or between aggregation and edge switches. All switches are equal in random graphs. Servers are uniformly distributed to the switches, and the links are between random switch pairs. So, the first step of conversion is to relocate servers to aggregation and edge switches and to diversify the types of links.

These can be achieved by small port-count converter switches. As shown in the zoomed-in Pod, flat-tree breaks an edge-server link and an aggregation-core link in the Clos network, and connects the corresponding server, edge, aggregation, and core switches to a converter switch. Figure 1 illustrates the valid configurations of 4-port and 6-port converter switches. The "default" configuration enables the original Clos connections. The "local" configuration relocates the server to the aggregation switch and connects the core and edge switches directly. This change is local in the Pod.

4-port converter switches should not be used to relocate servers to core switches. If we connect the server and the core switch, the edge and aggregation switches must be connected as well, otherwise we waste a link. There are sufficient edge-aggregation links in the Pod, so this change fails to diversify the types of links. 6-port converter switches introduces side ports, through which two converter switches can be interconnected. The "side" and "cross" configurations both relocate servers to core switches, but connect edge and aggregation switches to their peers in different ways. We only allow 6-port converter switches in adjacent Pods to be interconnected for simple neighbor-to-neighbor wiring.

The number of 4-port and 6-port converter switches are determined by the layout of the Clos network. In Figure 2, each pair of edge and aggregation switches are connected to a 4-port converter switch and a 6-port converter switch, which show the approximated random graph configuration. Converter switches and the additional wiring are packaged in the Pod, keeping the same core connectors as a Clos Pod. The side connectors of 6-port converter switches are bundled as multi-link connectors to simplify inter-Pod wiring. Flat-tree Pods are connected to core switches via a customized wiring pattern (details in Section 2.3). In this example, the uplinks from Pods are swapped in different ways, so that servers are distributed uniformly across the core switches.

Flat-tree converts between multiple topologies with different converter switch configurations. Figure 2b shows the Clos network, when all converter switches take the "default" configuration. Figure 2c shows an approximated global random graph, with the 4-port "local" and 6-port "side" configurations. In practice, we can also use the 6-port "cross" configuration to swap connections. Figure 2d shows approximated local random graphs in each Pod. It is configured in a way that half servers are connected to the edge switches and half to the aggregation switches. In this example, we use 4-port "local" and 6-port "default" configurations. Flat-tree can also operate in hybrid mode, with different combinations of the above topologies each in a number of Pods.

This paper limits the discussion to one Pod layer connected by core switches. Flat-tree can be extended to multi-stages of Pods: the lower-layer Pods consider the edge switches in

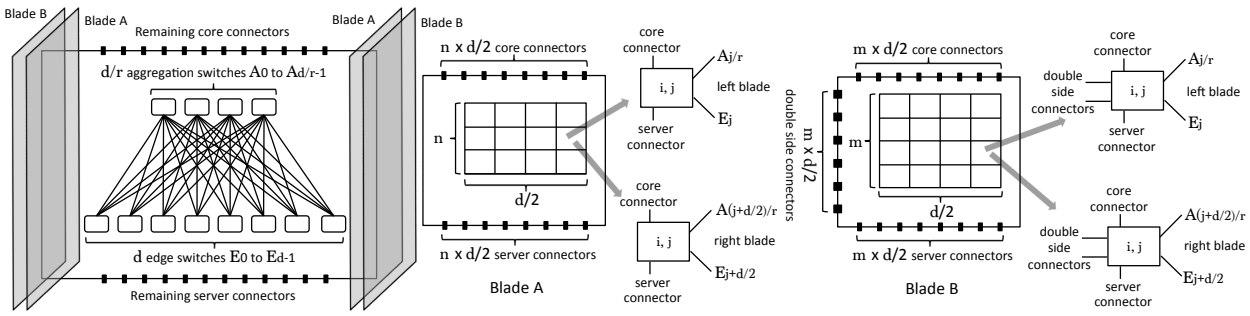

Figure 3: A flat-tree Pod. A pair of edge switch E_j and aggregation switch $A_{j/r}$ connected to n 4-port converter switches and m 6-port converter switches. Converter switches are placed evenly on both sides as matrices. Blade A and B has 4-port and 6-port converter switches respectively.

the upper-layer Pods as core switches; intermediate switch-only Pods take relocated servers from lower-layer Pods as their own servers. We leave the details to future work.

2.2 Flat-tree Pod

Figure 3 depicts a flat-tree Pod. Without loss of generality, we assume the number of edge switches is a multiple of the number of aggregation switches. There are d edge switches and d/r aggregation switches. We pair up each edge switch E_j with aggregation switch $A_{j/r}$ and connect them to n 4-port converter switches and m 6-port converter switches. n and m represent the number of servers that can be relocated dynamically to aggregation and core switches. We place the converter switches evenly on the two sides of the Pod: those connected to E_0 to $E_{d/2-1}$ locate on the left of the Pod and those connected to $E_{d/2}$ to E_{d-1} locate on the right. This forms a $n \times d/2$ matrix of 4-port converter switches, i.e. blade A in figure, and a $m \times d/2$ matrix of 4-port converter switches, i.e. blade B in figure, on each side of the Pod.

For both types of blades, converter switch $\langle i, j \rangle$ on the left blade is connected to edge switch E_j and aggregation switch $A_{j/r}$, and that on the right is connected to edge switch $E_{j+d/2}$ and aggregation switch $A_{(j+d/2)/r}$. Each 4-port converter switch connects to a core switch and a server, so blade A has $n \times d/2$ core connectors and server connectors. Each 6-port converter switch has a pair of side connectors as well, so blade B has $m \times d/2$ core connectors, server connectors, and double side connectors. There may be remaining core connectors on the aggregation switches and server connectors on the edge switches. The total number of core connectors and server connectors are equal to those in a Clos counterpart. If d is odd, a middle converter switch can be on either side, but the side connectors of the 6-port converter switch are unused.

2.3 Pod-Core Wiring

In a Clos network, all Pod-core connections are between aggregation and core switches. Suppose each aggregation switch has h uplinks. As Figure 4a illustrates, aggregation switches with the same index i in different Pods are connected to the same group of h core switches via the aggregation connectors. Repeatedly for each Pod, this wiring pattern links the h connectors for each aggregation switch consecutively to core switches.

In flat-tree, as shown in Figure 3, there are 3 types of core connectors. Core switches can be connected to servers via blade B connectors, to edge switches via blade A connectors, and to aggregation switches via aggregation connectors. The Pod-core wiring determines the distribution of servers and different types of links (to an edge or aggregation switch) across the core switches, thus affecting how close flat-tree approximates a random graph.

As each aggregation switch corresponds to r edge switches, the h aggregation connectors in Clos are replaced with $n \times r$ blade A connectors, $m \times r$ blade B connectors, and $h - m \times r - n \times r$ aggregation connectors. The Clos wiring pattern is based on aggregation switches, each connected to h core switches. Since flat-tree has edge-core connections, its wiring pattern should be based on edge switches. Each edge switch corresponds to n blade A connectors, m blade B connectors, and $h/r - m - n$ aggregation connectors, which connects to overall h/r core switches.

We offer two wiring options, shown in Figure 4b and 4c. Connectors corresponding to the edge switches with the same index j in different Pods are connected to the same group of h/r core switches. Both wiring patterns connect the group of core switches consecutively to blade B connectors, followed by blade A connectors and aggregation connectors. They rotate in different ways across Pods. Pattern 1 packs blade B connectors continuously Pod by Pod throughout the set of core switches. Pattern 2 moves them forward by one more core switch as the Pod index grows. Both patterns wrap around within the group.

Physically, we suggest wiring Pod 0 first, by linking every m blade B connectors, n blade A connectors, and $h/r - m - n$ aggregation connectors in turn to core switches consecutively. We start from the left blades and move on to the right blades, until all connectors in the Pod are consumed. In this process, we mark the mapping between each edge switch and the corresponding group of h/r core switches. For the following Pods, connectors corresponding to each edge switch are connected to the marked h/r core switches according to the rotating patterns.

These wiring patterns have the following properties:

Property 1: For both wiring patterns, servers are distributed uniformly across the core switches.

Property 2: For both wiring patterns, the core switches have equal number of links of the same type.

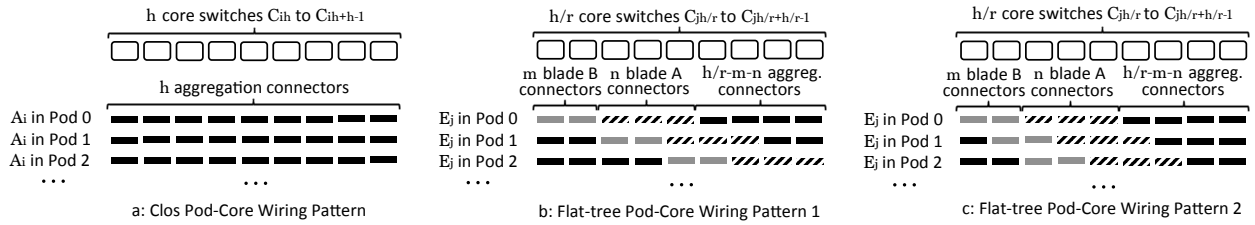

Figure 4: Pod-Core wiring for the same set of connectors across Pods. All connectors are on aggregation switches in Clos; flat-tree has 3 types of connectors on blade A, B, and aggregation switches, enabling core-server, core-edge, and core-aggregation connections respectively.

Flat-tree maintains structures to ease implementation, so servers and links must be permuted by wiring. These properties ensure that flat-tree well-approximates a random graph.

Because these patterns follow straightforward rules, they have low wiring complexity. Pattern 1 has better performance, because a core switch does not connect to servers from adjacent Pods at the same time, thus it takes advantage of side connections between adjacent Pods to the greatest extent. Yet when h/r is a multiple of m, different Pods are likely to repeat the same pattern, thus reducing the wiring diversity. In this case, pattern 2 is more favorable.

2.4 Server Distribution

In a random graph, servers are distributed uniformly across the switches, because the random links roughly connect the switches in a uniform manner. Yet flat-tree maintains structures, e.g. the Clos connections between edge and aggregation switches, core switches connected to the Pods, though using customized wiring patterns, and the neighbor-to-neighbor wiring restricted to adjacent Pods. The path length of switch pairs is not uniform for flat-tree, so we should place servers intelligently to leverage the shorter paths in the network.

Recall that 6-port converter switches can relocate servers to core switches, and 4-port ones can relocate servers to aggregation switches, so the server distribution is determined by the choice of m and n. Because flat-tree aims at converting generic Clos networks, which may have very different layouts, it is difficult to pre-define the m and n values for optimal transmission performance. We suggest a profiling scheme: under the preferred Pod-core wiring pattern described in Section 2.3, vary m and n until they result in the shortest average path length over all server pairs. We show an example of the profiling process in Section 3.2.

2.5 Inter-Pod Wiring

For adjacent Pods p and $p+1$, the 6-port converter switches on the left blade B of Pod $p+1$ are connected to those on the right blade B of Pod p by the side connectors. Recall from Figure 3 that the converter switches in the same column connect to the same pair of edge and aggregation switches. We want to connect an edge/aggregation switch to as many different switches as possible in the adjacent Pod, so we design a shifting wiring pattern such that the converter switches in the same column of the right Pod are connected to converter switches each in a different column of the left Pod. Specifically, let i and j be the row and column of the converter switch matrices, converter switch $\langle i, j\rangle$ on the left of Pod $p+1$ is connected to converter switch $\langle i, (d/2-1-j+i)\%(d/2)\rangle$ on the right of Pod p, which represents the converter switch in the same row i and in the column i slots shifted from the mirrored column $d/2-1-j$. We want the converter switches to be interconnected by different configurations, so we have both peer-wise and edge-aggregation connections across Pods. If i is even, they take the 6-port "side" configuration (in Figure 1); if i is odd, they take the 6-port "cross" configuration. To streamline the connection of adjacent Pods, the side connectors on the same side of a Pod are bundled as a multi-link connector that integrates this wiring pattern.

2.6 Control Plane

Flat-tree requires a control plane to change the network topology and to conduct routing accordingly. Because a data center is administered by a single authority, we follow the recent trend of using a centralized network controller for global network management. Flat-tree has several operation modes with pre-known topologies. The controller changes among these options to optimize workloads, either as explicitly instructed by the network manager or in an adaptive manner through network measurement. It may coordinate with workload placement software to take advantage of the topologies. The topology is changed by configuring converter switches, via specific control mechanisms depending on the realization technology. For instance, most optical switches can be programmed via a software interface. We adopt the suggested routing schemes for each network topology. For Clos, we use ECMP [15], two-level routing [5], or customized SDN routing with pre-computed paths [21]. We use k shortest paths routing for approximated random graphs [23]. Because flat-tree maintains structures when approximating random graphs, instead of learning routes, it is possible to have prior knowledge of the shortest paths and program the routing decisions via SDN.

2.7 Cost Analysis

Converter switches can be realized by various switching technologies, as long as they are software configurable. As Figure 1 shows, they have only a few configurations. If implemented using packet switches, traffic can be piped point to point with no bandwidth contention. This bare-minimum switching functionality does not require expensive processor/buffering, sophisticated routing protocols, or general-purpose OS, etc. Cheap switching chips with support for simple port-to-port forwarding rules would suffice. If implemented using

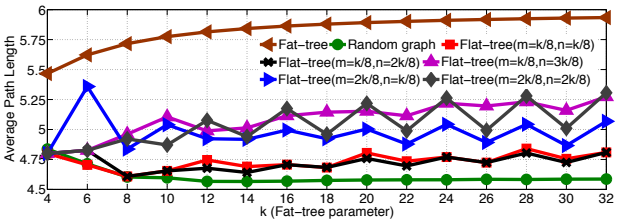

Figure 5: Average path length of server pairs in the entire network

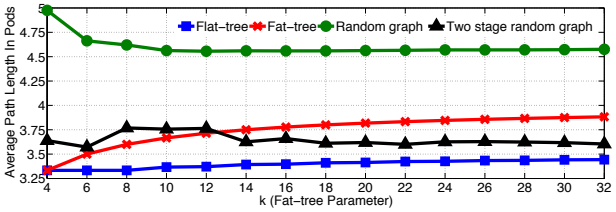

Figure 6: Average path length of server pairs in each Pod

circuit switches, the cost is related to switching delay and port count. Flat-tree changes topology infrequently, so it imposes no rigid restriction on switching delay. For some low-cost circuit switches, the signal losses limit the port count to modest scale [28, 12]. Because converter switches have small port-count, these technologies can apply. Given the large number of eligible low-cost switching technologies, we envision the cost of converter switches in flat-tree to be minimal compared to that of the high-end servers and switches in data centers.

3. EVALUATION

3.1 Simulation Methodology

We evaluate the performance of flat-tree by simulations. Although flat-tree targets at converting generic, especially oversubscribed, Clos networks, our evaluations are based on fat-tree [5]. Because generic Clos networks can have very different layouts, e.g. arbitrary number of switches and servers, oversubscription at any possible layer, it is difficult to have a "typical" example for evaluation. Fat-tree gives the upper-bound performance for Clos networks, thus serving as a stress test for our solution. We construct fat-tree, random graph, and our flat-tree using the same equipments with the variance of k, the fat-tree parameter that defines the switch port count, the number of Pods, as well as the number of switches in each layer. We consider two flat-tree configurations: approximated network-wide random graph and approximated local random graphs in each Pod. When flat-tree approximates local random graphs, we compare it with two-stage random graph, which first forms random graphs in each Pod with the same number of links as flat-tree, and takes the Pods as super nodes to form another layer of random graph together with core switches. We also evaluate flat-tree in hybrid mode: having different proportions of the network functioning as approximated global and local random graph respectively.

We use average path length in hops and throughput as the evaluation metrics. We assume converter switches function in the physical layer and do not contribute to path length. The throughput experiments follow a well-adopted methodology [22, 23]. We assume optimal routing and solve the maximum concurrent multi-commodity flow problem [18] using a linear programming solver. All links have one unit bandwidth. We relax the bandwidth constraints at the servers to show the switch-level capacity, which is relevant to the maximum number of servers a topology can accommodate. Measurement studies show two pervasive traffic patterns in data centers: broadcast/incast traffic from/to hot spots to/from a large number of servers, and all-to-all traffic within small clusters [6, 9, 17]. We simulate them by broadcast/incast traffic from/to a random target in 1000-server clusters and all-to-all traffic in 20-server clusters. We consider strong, weak, and no locality of workload placement to evaluate the topologies' sensitivity to it. Specifically, the workload is placed continuously across servers, randomly in Pods, or randomly in the entire network.

3.2 Average Path Length

We first determine m and n for flat-tree through the profiling mechanism described in Section 2.4. Flat-tree has the same equipments as the fat-tree counterpart, so $m+n \leq k/2$. We vary m and n at the interval of $k/8$, rounded to the closest integer if fractional. This process can happen at finer granularity with smaller intervals. We use Pod-core wiring pattern 2 when k is a multiple of 4 and pattern 1 otherwise for reason discussed in Section 2.3.

Figure 5 compares the average path length of flat-tree under the settings of different m and n against that of fat-tree and random graph. The desirable values for m and n are $k/8$ and $2k/8$, when flat-tree has the minimal average path length. It is notably shorter than that of fat-tree, and within only 5% difference to random graph. k as multiples of 4 are hard cases where pattern 1 tends to repeat frequently. Pattern 2 successfully maintains the average path length at a relatively low level. These results demonstrate that with the right choice of m and n, flat-tree approximates global random graph well and it improves against fat-tree significantly. We set $m = k/8$ and $n = 2k/8$ for the rest experiments.

We further evaluate the average path length between server pairs in the same Pod, when flat-tree functions as approximated local random graphs within each Pods. Figure 6 shows the result against fat-tree, global random graph, and two-stage random graph. Random graph performs the worst as servers scatter around the network, followed by fat-tree whose servers at the edge switches have locality. Flat-tree moves half the servers from edge to aggregation switches, reducing the distance between servers connected to different types of switches. Surprisingly, it outperforms two-stage random graph. In flat-tree, servers evenly distributed over edge and aggregation switches are connected by the regular Clos edge-aggregation links, which is more efficient than pure randomness in server distribution and inter-switch connections.

3.3 Throughput

We create 1000-server clusters, each server being involved in a single cluster. One random server in each cluster is the source/destination of broadcast/incast traffic to/from all the

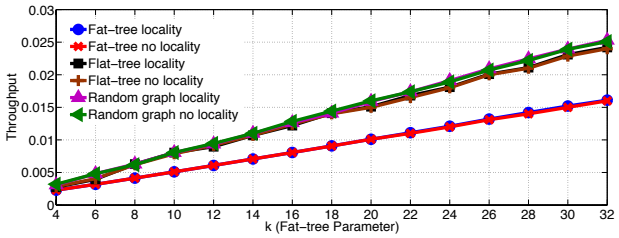

Figure 7: Throughput of broadcast/incast traffic in 1000-server clusters

other servers in the same cluster. In the locality case, we pack clusters continuously across the servers; in the no locality case, we place them randomly throughout the network. Each Pod has $k^2/4$ servers, so one cluster spans multiple Pods even for large k under the locality setting. Flat-tree approximates a global random graph to accommodate such large clusters, so we compare its throughput with fat-tree and random graph. As shown in Figure 7, the throughput of flat-tree is very close to that of random graph and is $1.5\times$ that of fat-tree. The throughput grows linearly with k, the switch port count, as the few hot spots have increasing sending/receiving capacity. None of the topologies is sensitive to locality, due to heavy cross-Pod traffic. This set of results demonstrate that with arbitrary workload placement, flat-tree can achieve near-optimal performance for the prevalent broadcast/incast traffic at hot spots.

Then we create 20-server clusters featuring all-to-all traffic, which can fit in the Pod for most k. Flat-tree approximates local random graphs within each Pod to accommodate small clusters in Pods. So, besides fat-tree and random graph, it is also compared with two-stage random graph. We consider workload with locality, clusters packed continuously across servers, and workload with weak locality, clusters packed randomly in Pods as long as there are remaining servers. Weak locality is the worst-case simulation of resource fragmentation in workload placement.

Figure 8 shows flat-tree well approximates local random graph. It outperforms two-stage random graph for small networks ($k \leq 14$), and the difference in throughput is less than 6% and 9% respectively with strong and weak workload locality for larger networks. Flat-tree has shorter average path length in Pods, as shown in Figure 6, whereas two-stage random graph forms closer inter-Pod connections. The result is the outcome of the interplay between these factors. Traffic locality has greater impact on flat-tree, because the regular direct links between adjacent Pods are more likely to benefit consecutively packed servers. Fat-tree is highly sensitive to workload placement. It has sweet spots, such as $k = 20$, when most traffic is local in Pods. Its throughput drops significantly for weak locality, as even local traffic in Pods takes more hops through aggregation switches. Random graph has moderate throughput as it does not specialize in local clustered traffic, but it is the least sensitive to workload locality. In reality, we expect the performance of flat-tree to be between the locality and the weak locality curves, given a reasonable level of fragmentation. In summary, flat-tree optimizes all-to-all traffic in small clusters effectively.

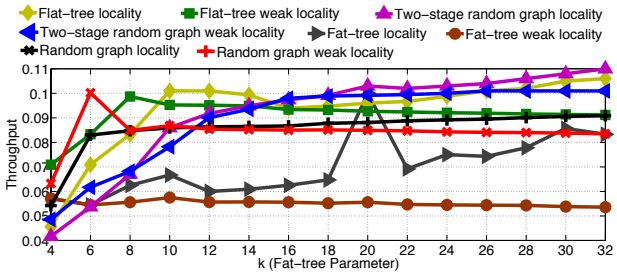

Figure 8: Throughput of all-to-all traffic in 20-server clusters

3.4 Hybrid Flat-tree

Flat-tree can work in hybrid mode with different topologies each in a number of Pods. Workloads placed in different zones share the network core. We use experiments to answer the question whether flat-tree can optimize multiple workloads in separate zones without interfering with each other.

We construct flat-tree with 30 Pods, i.e. $k = 30$, and organize the network into two separate zones with varying proportions at an interval of 10%. We let flat-tree operate as an approximated global random graph in one zone and as approximated local random graphs within each Pod in the other zone. Each topology gets the same traffic pattern as the corresponding complete network as described in Section 3.3. We observe that regardless of the proportion, each zone constantly achieves the same throughput as that of the corresponding complete network under the same locality setting. Therefore, flat-tree in hybrid mode is as effective as building separate flat-tree networks, and the workloads in different zones can be segregated perfectly.

4. RELATED WORK

Flat-tree is distinguished from other data center network architectures such as [5, 23, 14, 13, 3, 4] by its convertibility. Each of these fixed topologies has sweet spots for particular traffic patterns [22], whereas flat-tree is able to convert the topology to adapt to different workloads. Flat-tree also goes beyond the recent proposals of configurable data center network architectures. Some solutions provide local remedy by adding or changing a small number of connections to alleviate hot spots [24, 11, 10, 30, 29, 26, 27, 16, 25, 7]. Others create a flexible network core for small scale networks [20, 19, 2]. Flat-tree has a more ambitious goal of creating globally convertible data center networks at any size.

5. CONCLUSION

The concept of convertible data center networks is powerful. Flat-tree is merely one design point in the broad space for exploration. It motivates the further study of relationships between different network topologies, those that complement each other, beyond Clos and random graph, with desirable properties that cannot coexist traditionally. Besides traffic optimization, convertibility can play a broader role in network management, e.g. self-recovery of the topology from failures and automatic up/down-scale the network at busy/idle time. Convertible network topology gives greater flexibility to routing and workload placement, making the joint optimization a potentially interesting research topic.

Acknowledgement

We would like to thank the anonymous reviewers for their thoughtful feedback. This research was sponsored by the NSF under CNS-1422925, CNS-1305379 and CNS-1162270, an IBM Faculty Award, and by Microsoft Corp.

6. REFERENCES

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

[2] Plexxi, http://www.plexxi.com/.

[3] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly. Symbiotic Routing in Future Data Centers. In *SIGCOMM '10*, pages 51–62, New Delhi, India, August 2010.

[4] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. HyperX: Topology, Routing, and Packaging of Efficient Large-scale Networks. In *SC '09*, pages 41:1–41:11, Portland, Oregon, USA, November 2009.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM '08*, pages 63–74, Seattle, Washington, USA, August 2008.

[6] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. DCTCP: Efficient Packet Transport for the Commoditized Data Center. In *SIGCOMM'10*, August 2010.

[7] H. H. Bazzaz, M. Tewari, G. Wang, G. Porter, T. S. E. Ng, D. G. Andersen, M. Kaminsky, M. A. Kozuch, and A. Vahdat. Switching the optical divide. In *SOCC '11*, pages 1–8, Cascais, Portugal, October 2011. ACM Press.

[8] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding Data Center Traffic Characteristics. *SIGCOMM CCR*, 40(1):92–99, January 2010.

[9] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica. Surviving Failures in Bandwidth-constrained Datacenters. In *SIGCOMM '12*, pages 431–442, Helsinki, Finland, August 2012.

[10] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. In *NSDI '12*, San Joes, CA, April 2012.

[11] 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 '10*, pages 339–350, New Delhi, India, August 2010.

[12] M. Fokine, L. E. Nilsson, Å. Claesson, D. Berlemont, L. Kjellberg, L. Krummenacher, and W. Margulis. Integrated Fiber Mach–Zehnder Interferometer for Electro-Optic Switching. *Optics Letters*, 27(18):1643–1645, September 2002.

[13] 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 '09*, pages 63–74, Barcelona, Spain, August 2009.

[14] 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 '08*, pages 75–86, Seattle, Washington, USA, August 2008.

[15] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. *RFC 2992*, 2000.

[16] X. S. Huang, X. S. Sun, and T. S. E. Ng. Sunflow: Efficient Optical Circuit Scheduling for Coflows. In *CoNEXT '16*, Irvine, CA, December 2016.

[17] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The Nature of Data Center Traffic. In *IMC '09*, pages 202–208, Chicago, Illinois, USA, November 2009.

[18] T. Leighton and S. Rao. Multicommodity Max-flow Min-cut Theorems and Their Use in Designing Approximation Algorithms. *J. ACM*, 46(6):787–832, November 1999.

[19] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav. Quartz: A New Design Element for Low-latency DCNs. In *SIGCOMM '14*, pages 283–294, Chicago, Illinois, USA, August 2014.

[20] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat. Integrating Microsecond Circuit Switching into the Data Center. In *SIGCOMM '13*, pages 447–458, Hong Kong, China, August 2013.

[21] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *SIGCOMM '15*, pages 183–197, London, United Kingdom, August 2015. ACM.

[22] A. Singla. *Designing Data Center Networks for High Throughput*. Ph.D. Thesis, University of Illinois at Urbana-Champaign.

[23] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking Data Centers Randomly. In *NSDI '12*, pages 1–14, San Jose, California, USA, April 2012.

[24] 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 '10*, pages 327–338, New Delhi, India, August 2010.

[25] G. Wang, T. S. E. Ng, and A. Shaikh. Programming Your Network at Run-time for Big Data Applications. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 103–108, Helsinki, Finland, August 2012. ACM.

[26] H. Wang, Y. Xia, K. Bergman, T. S. E. Ng, S. Sahu, and K. Sripanidkulchai. Rethinking the Physical Layer of Data Center Networks of the Next Decade: Using Optics to Enable Efficient *-cast Connectivity. *SIGCOMM Comput. Commun. Rev.*, 43(3):52–58, July 2013.

[27] D. Wu, X. Sun, Y. Xia, X. Huang, and T. S. E. Ng. HyperOptics: A High Throughput and Low Latency Multicast Architecture for Datacenters. In *HotCloud '16*, Denver, CO, June 2016.

[28] M. C. Wu, O. Solgaard, and J. E. Ford. Optical MEMS for Lightwave Communication. *Journal of Lightwave Technology*, 24(12):4433–4454, December 2006.

[29] Y. Xia, T. S. E. Ng, and X. Sun. Blast: Accelerating High-Performance Data Analytics Applications by Optical Multicast. In *INFOCOM '15*, pages 1930–1938, Hong Kong, China, April 2015.

[30] Y. Xia, M. Schlansker, T. S. E. Ng, and J. Tourrilhes. Enabling Topological Flexibility for Data Centers Using OmniSwitch. In *HotCloud '15*, Santa Clara, CA, July 2015.