

Performance Evaluation of Data Center Topologies

Sahiti Bommareddy

Hemasagar Babu Reddy

1 Introduction

Prior to the introduction of the fat-tree topology to network architecture, data centers employed tens of thousands of computers with significant aggregate bisection bandwidth requirements. The network architecture typically consisted of a two or three-level tree of routing with progressively more specialized and expensive equipment moving up the topology. However, with a tree topology it was observed that only about 50% of the aggregate bandwidth was available at the edge of the network in spite of deploying high-end IP switches and routers. In paper [1], the authors Al-Fares et.al. present a fat-tree topology and explains how we can leverage commodity Ethernet switches to support the full aggregate bandwidth of networks consisting of tens of thousands of elements. They argue that an appropriately architected and interconnected network of commodity switches may achieve better performance at a lower cost than possible with high-end IP switches/routers interconnected in the form of a tree.

The authors in [1] identify the principal bottleneck in large-scale clusters to be inter-node communication bandwidth. A multitude of applications are required to exchange information with remote nodes to perform local computations. While one option for building the communication network for large-scale clusters is to use specialized hardware and communication protocols [2, 3], they point out that for reasons of compatibility and economy, most cluster communication systems employ commodity Ethernet switches and routers to interconnect cluster machines.

The authors identify the following three principle goals involved in designing a data center communication architecture: scalable interconnection bandwidth, i.e. it should be possible for an arbitrary host in the data center to communicate with any other host in the network at the full bandwidth of its local network interface; leverage economies of scale to make the most of cheap off-the-shelf Ethernet switches the basis for large-scale data center networks; the entire system should be backward compatible with hosts running Ethernet and IP. They show that by interconnecting commodity switches in a fat-tree architecture, the full bisection bandwidth of clusters containing tens of thousands of elements can be achieved. Additionally, since the implementation used commodity Ethernet switches, they achieved lower costs than existing solutions at the time while also delivering more bandwidth.

1.1 Motivation

For our work, we chose to compare the performance of a simple tree topology with that of a 4-ary fat-tree that uses the same topology as described in [1]. For a large part of our work, we chose to reproduce the results of the work conducted by Al-Fares et.al. relevant to this

problem. The primary motivation for us to study the data center topologies and evaluate their performances is to gain a better insight into the functioning of large-scale clusters and to better understand the requirements and trade-offs pertaining to performance and economics. Since the publishing of the paper [1] in 2008, data centers around the world have opted to implement the fat-tree architecture for their interconnection networks. During the course of our work, we faced multiple challenges and found various possible solutions to each of those problems. By investigating and evaluating data center topologies, we have learnt a great deal about the requirements of data centers and the challenges faced in satisfying those requirements. We have also enhanced our knowledge in the vast field of computer networks and network architectures.

The rest of the paper is organized as follows. In the next section, we briefly describe other relevant work and give an introduction to some work preceding that of the authors in [1]. In subsequent sections, we describe the general tree topology and the fat-tree topology, and describe our implementations of both. We then describe the two-level routing table for static load scheduling used in the fat-tree. We also explain how we implemented it on Mininet [6] using the OpenFlow [10] switches' flow table. We compare and contrast the hash based approach for our static routing with that employed by the authors in [1]. In the penultimate section we provide an overview of the work in progress and conclude the report in the section after that.

2 Related Work

The work of the authors in [1] builds on consolidates work in a number of related areas. There have been various efforts towards building scalable interconnects, primarily involving super-computer and massively parallel processing (MPP) communities. Many MPP interconnects have been previously organized as fat-trees in systems from Thinking Machines [14, 9] and SGI [15]. Thinking Machines, whose systems employed this architecture, also used pseudo-random forwarding decisions to perform load balancing among fat-tree links. Myrinet switches [3] also employed fat-tree topologies and have been a popular choice for cluster-based supercomputers. Infiniband [12], the authors say, was another popular interconnect for high-performance computing environments and has been migrated to data center environments. Infiniband also uses Clos topologies [4] to achieve scalable bandwidths. Torus [5] was another popular MPP interconnect topology. It directly connects a processor to some number of its neighbors in a k -dimensional lattice. In an MPP application, Torus offered the benefit of not having any dedicated switching elements along with electrically simpler point-to-point links. However, in a cluster, the wiring complexity of Torus was found to be prohibitive.

The authors' proposed routing techniques in [1] are related to techniques such as OSPF2 and Equal-Cost Multipath (ECMP) [11, 13, 7]. Relevant to their work, ECMP, the authors note, proposes three classes of stateless forwarding algorithms: (i) Round-robin and randomization; (ii) Region splitting where a prefix is split into two with a larger mask length; and (iii) A hashing technique that splits flows among a set of output ports based on source and destination addresses. Each of the three approaches mentioned above have limitations. For instance, (i) suffers from packet reordering issues which is especially troubling for TCP; (ii) can lead to an explosion in the number of prefixes which will increase cost and significantly increase table lookup latencies; (iii) does not account for flow bandwidth in making allocation

decisions, which they say can quickly result in oversubscription.

3 Topologies and Implementation

For our work, we used the tree topology and the fat-tree topology. The topologies and traffic flows are generated using Mininet. The performance of the fat-tree was compared to that of the tree to highlight the improvements made possibly by the fat-tree.

3.1 Baseline Topology: Hierarchical Tree

A three-tiered tree has a core tier in the root of the tree, an aggregation tier in the middle and an edge tier as the last level at the leaves of the tree. Switches at the leaves of the tree have some number of ports as well as some number of uplinks to the upper layers of network elements that aggregate and transfer packets between the leaf switches. In the higher levels, there are switches with same number of ports and significant switching capacity to aggregate traffic between the edges. Over-subscription is introduced to lower the total cost of building the network interconnections. Over-subscription is the ratio of the worst-case achievable aggregate bandwidth among the end hosts to the total bisection bandwidth of a topology.

For the experiment implementation, we have four hosts in each pod at the edge and four pod switches. The four pod switches are connected to a 4-port core switch through which interpod traffic is routed. To enforce the 3.6:1 oversubscription on the uplinks from the aggregation switches to the core switch, these links are bandwidth-limited to 106.67 Mbps, while all the other links are limited to 96 Mbps. An oversubscription of 3.6:1 means that only 27.78% of available host bandwidth is available for some communication patterns. Each host generates constant outgoing traffic at 96 Mbps. The bisection bandwidth is measured by measuring the minimum aggregate incoming traffic of all bijective communications.

3.2 Fat-Tree

The authors argue that the price difference between existing commodity and non-commodity switches at the time was a strong incentive to build large-scale networks from small commodity switches. They adopt a special case of a Clos topology called the fat-tree to interconnect commodity Ethernet switches. This topology and its performance evaluation formed the basis of their work.

A k -ary fat-tree topology is shown in Figure 1. The topology consists of k pods, each containing two layers of $k/2$ switches each. Each of the two $k/2$ switches in the lower layer are connected directly to $k/2$ hosts. Each of the remaining $k/2$ ports is connected to $k/2$ of the k ports in the higher aggregation layer of hierarchy. It is important to note that, all the commodity switches used in the topology are k -port switches. At the highest level, there are $(k/2)^2$ k -port switches. One port of each of the core switches is connected to each of the k pods. It can be deduced that, in general, a fat-tree built with k -port switches supports $k^3/4$ hosts. We implement a 4-ary fat-tree with 16 hosts for our work.

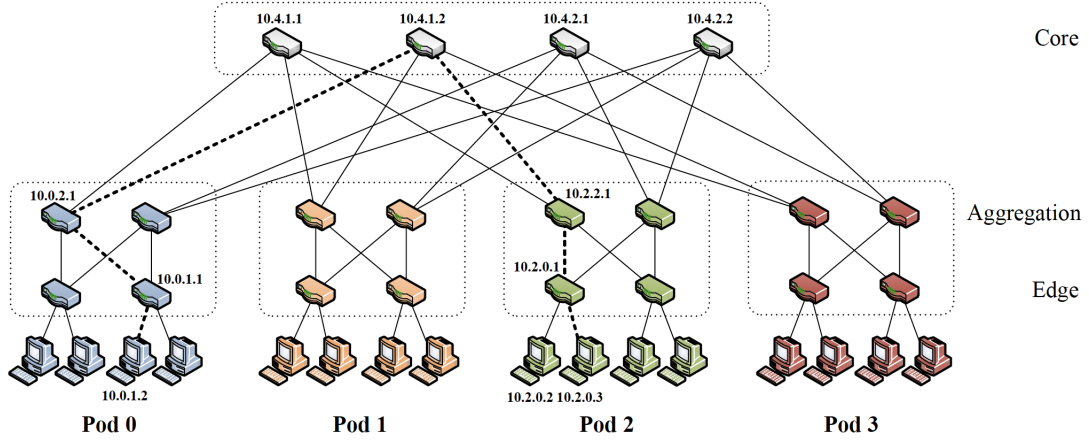


Figure 1: A simple 4-ary fat-tree topology.


Addresses of the switches are defined as illustrated for a 4-ary fat-tree in Figure 1. All the IP addresses in the network are allocated within the 10.0.0.0/8 private block. Pod switches take addresses of the form $10.pod.switch.1$, where *pod* denotes the pod number (0 to $k-1$) and *switch* denotes the position of that switch in the pod starting from left to right, bottom to top. Core switches take address of the form $10.k.j.i$, where j and i denote the switches coordinates in the $(k/2)^2$ core switch grid, starting from the top-left. Hosts have address of the form $10.pod.switch.ID$, where ID is the host's position in that subnet (2 to $k/2+1$), going from left to right.

As shown in the figure 1, the 4-port fat-tree consists of 16 hosts, 4 pods and 4 core switches. There is a total of 20 switches and 16 end hosts. This topology was implemented in Mininet. All the individual links between pairs of switches are bandwidth limited to 96 Mbit/s as in the authors' original work [1].

4 Routing Implementation

In order to achieve maximum bisection bandwidth in the fat-tree network, outgoing traffic from any given pod must be spread as evenly as possible among core switches. A simple, fine-grained method of traffic diffusion that takes advantage of the structure of the topology is defined in [1]. This is achieved by using two-level routing tables that spread outgoing traffic based on the lower-order bits of the destination IP address. Routing tables are modified to allow two-level prefix lookup. Each entry in the primary table can have an additional pointer to a small secondary table of (*suffix*, *port*) entries. Entries in the first-level table are left-handed, i.e. $/m$ prefix masks of the form $1^m 0^{32-m}$, and the entries in the second-level table are right-handed, i.e. $/m$ suffix masks of the form $0^{32-m} 1^m$. A first-level prefix is said to be terminating if it does not point to any second level suffixes. If a longest-matching prefix search in the primary table yields a non-terminating prefix, the longest-matching suffix in the secondary table is found and used. This is illustrated in Figure 3.

Prefix	Output port
10.2.0.0/24	0
10.2.1.0/24	1
0.0.0.0/0	



Suffix	Output port
0.0.0.2/8	2
0.0.0.3/8	3

Figure 2: A two-level table example [1].

4.1 Our Implementation of two level table

The two-level routing table was implemented with the help of OpenFlow switch flow table. The priority field in the flow table was used accordingly to implement the functionality of the two-level table. Prefix masks in the flow table were assigned a higher priority than the suffix masks. As a result, prefixes are searched initially for a terminating prefix, and if one is found, the traffic is forwarded on the associated output port. If a prefix search does not yield a terminating prefix, the lower priority suffixes are searched for a match. The traffic is forwarded on the output port of the corresponding matching suffix. This implementation is illustrated in Figure 3.

Category	Priority	Source IP	Destination IP	Action
Prefix	32768	*	10.k.0.0/24	Port 0
Prefix	32768	*	10.k.1.0/24	Port 1
Suffix	100	*	0.0.0.2/8	Port 2
Suffix	100	*	0.0.0.3/8	Port 3

Figure 3: An example flow table set-up on OpenFlow to implement a two-level table.

4.2 Hash based approach for static load balancing with multipath routing

At edge and aggregate layers in pod switches due to availability of choice we can distribute the up-link traffic. Our first implementation choice was to utilize *add-group* command available in open flow. To state it explicitly, we defined a group with id 5 with available up-link ports. When we direct the suffix traffic to group id 5, the switch uses inbuilt hash property based on source and destination IPs to split flows on up-links. This ensures static load balancing with multipath routing harvesting the potential of fat trees. The figure 4 gives a visual representation of this implementation.

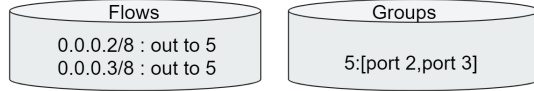


Figure 4: Multipath routing with add-group command

4.3 Implementation described in the paper for static load balancing with multipath routing

For their work [1], the authors use Click [8] to build a prototype of the two-level routing table. Click is a modular software routing architecture that supports implementation of experimental router designs. The authors build a Click element called *TwoLevelTable* which implements the functionality of a two-level routing table described previously. They initialize the contents of the table using an input file that provides all the prefixes and suffixes. For every packet, the *TwoLevelTable* looks up the longest matching prefix. If the prefix is terminating, the packet is forwarded on the corresponding output port. If no terminating prefix matches, it will perform a right-handed search for the longest-matching suffix on the secondary table and forward the packet on the corresponding port of the matching suffix.

5 Traffic patterns used for evaluation

We simulate both the hierarchical tree and the fat-tree for a set of traffic patterns and compare their performances in each case. We generated traffic matrices for Mininet for the following traffic flows:

- Random: A host sends to any other host in the network with uniform probability.
- Stride(i): A host with index x sends to a host with index $(x+i) \bmod 16$.
- Staggered Probability (*SubnetP*, *PodP*): A host sends to another host in its subnet with probability *SubnetP*, and to its pod with probability *PodP*, and to any other host with probability $1 - \text{SubnetP} - \text{PodP}$.
- Inter-pod Incoming: Multiple pods send to different hosts in the same pod through the same core switch.
- Same-ID Outgoing: Hosts in the same subnet send to different hosts elsewhere in the network such that the destination hosts have the same host ID byte.

6 Results and Observations

The first benchmark results was to measure aggregate bandwidth of hierarchical tree described above for all traffic patterns. The results are plotted in figure 5 and it is observed that they match those in the paper with a very close fit.

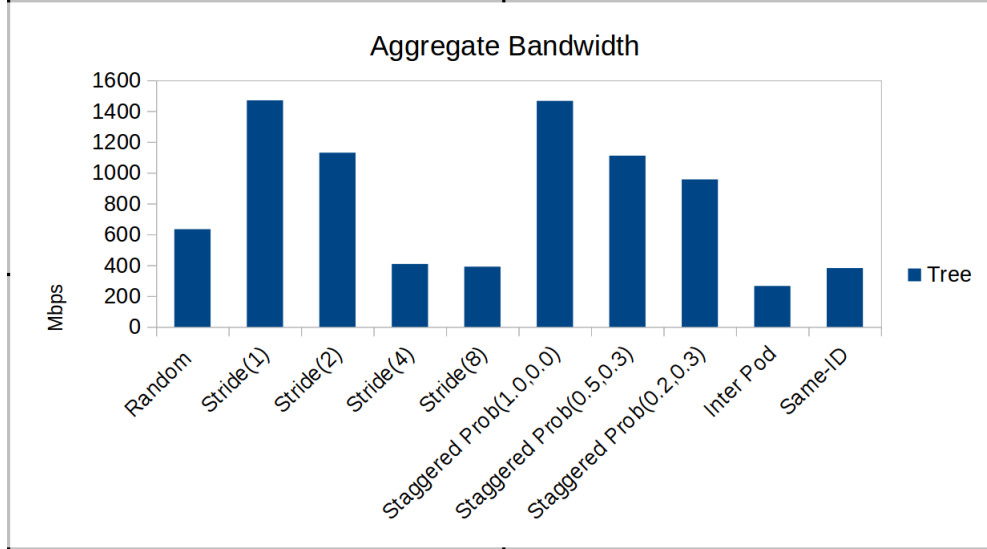


Figure 5: Hierarchical Tree aggregate Bandwidth

After the aforementioned result that stays consistent to that of paper measurements, we measure aggregate bandwidth in context of fat tree. The results in figure 6 correspond to hash based static load balancing implementation. Firstly, in case of any traffic pattern, fat-tree performs better than tree. This pattern is expected of fat-tree. Secondly, We have observed that there is an unexpected trend with stride traffic patterns. This stride trend is highlighted in figure 7. It is clear that this doesn't match the results we are trying to reproduce. Though fat-tree performs better than tree, it is not as promising as the results described in the paper for two-level routing. Upon investigation, we found that this trend is seen in *M Al-Fares et al* (ecmp in fig 8), confirmed by same author in [2]. This pattern is due to the implementation choice for static load balancing.

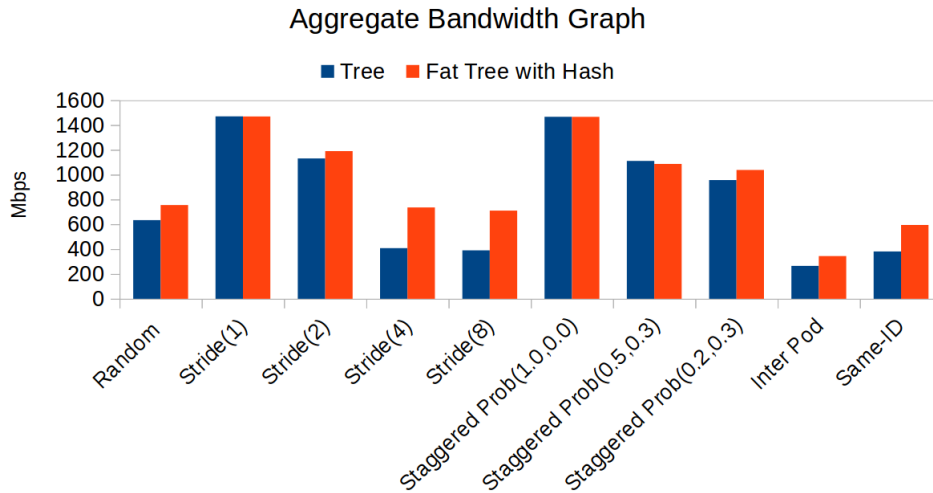


Figure 6: Aggregate Bandwidth of fat tree

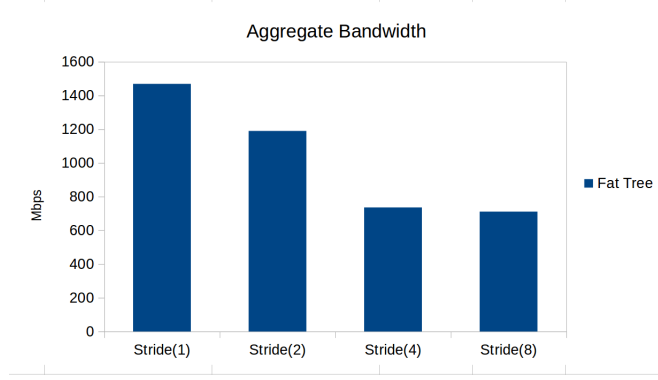


Figure 7: Aggregate bandwidth for Stride traffic patterns of fat tree with hash based approach

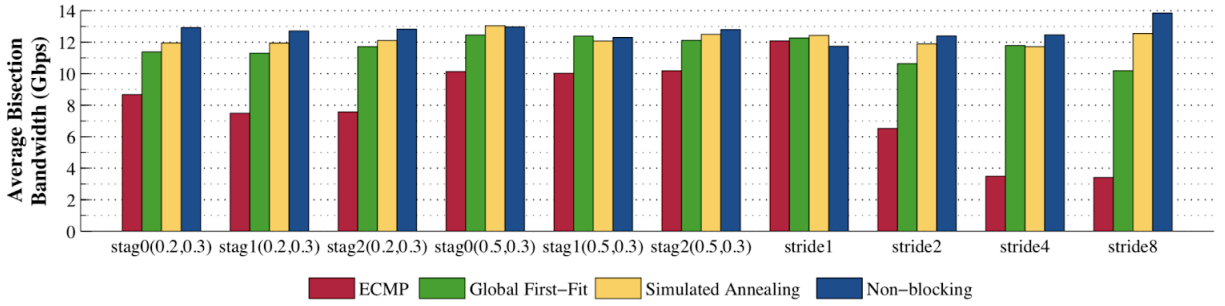


Figure 8: Network bisection bandwidth achieved for staggered, stride, and randomized communication patterns for fat tree using three routing methods vs. a non-blocking switch (figure 9 from Hedera [2])

Logically, the next step was to implement and evaluate the routing scheme as illustrated in figure 3 and described in section 4.1. These results, plotted in figure 9, had some interesting observations. Firstly, in stride(2) pattern we noted that, though hash based approach splits traffic, it does so unevenly, resulting in reduced aggregate bandwidth. It can be noticed that in the case of the suffix match approach, the traffic generated is split more evenly and hence the improvement. This holds true for same-ID pattern for which traffic for suffix 0.0.02/8 and 0.0.0.8/8 is equal, obtaining equal distribution on uplinks in this static load balancing technique. Secondly, one of the runs for random patterns generated a traffic matrix where majority of destination host ids were 2. In such a biased traffic matrix case, suffix based approach cannot spread the traffic flow evenly leading to hash based approach faring better.

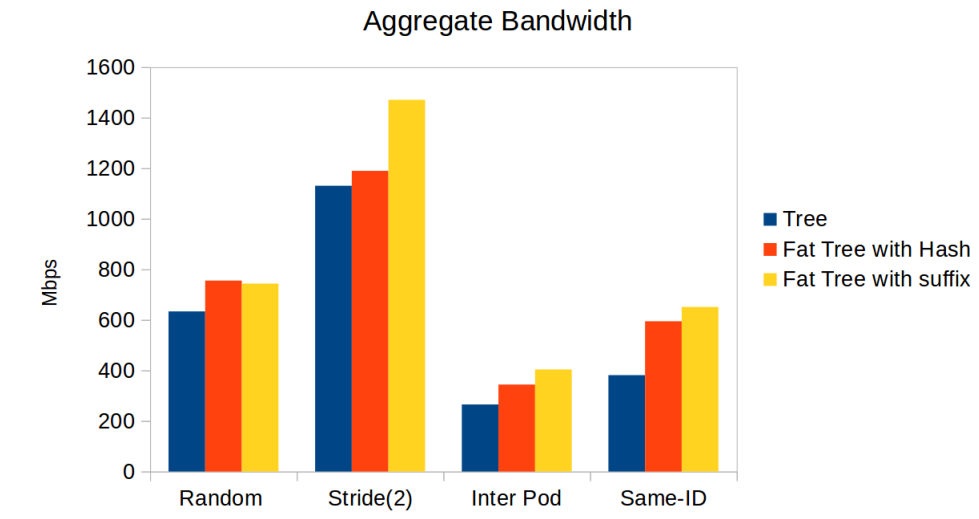


Figure 9: Aggregate bisection bandwidth for tree, fat tree with two routing methods

Upon trying other schemes and analyzing them, we believe that any approach in static load balancing will not be sufficient to harvest the performance potential of fat-trees. This definitely calls for dynamic load balancing techniques.

7 Future Work : Dynamic load balancing

We have successfully implemented static load balancing using the two-level table as described in the previous sections. The performance improvements of the fat-tree over that of a hierarchical tree for a set of traffic patterns have been demonstrated. It is shown that the aggregate bandwidth that can be achieved using the fat-tree is higher than that possible using the hierarchical tree. Owing to time constraints, we were not able to complete our implementations of dynamic load balancing using Flow Classification and Flow Scheduling as described in the paper [1]. We expect our observations for dynamic routing to follow the trend presented in the paper, as is the case for two-level static routing.

We would still like to highlight that a significant amount of work has been completed towards reproducing all the results and findings of the paper [1] in our project.

8 Conclusion

In this paper, we present our performance evaluation of data center network topologies, specifically the hierarchical tree and the fat-tree. The fat-tree uses commodity Ethernet switches to deliver scalable bandwidth for large-scale clusters. We reproduce the two-level static routing technique described in paper [1] to perform scalable routing for the fat-tree. Dynamic routing remains a work in progress as of the time of writing this paper.

We find that we achieve better aggregate bandwidth with the fat-tree than with the hierarchical tree using only commodity Ethernet switches. This demonstrates the better scalability

of fat-tree for large clustered networks. To achieve ideal bisection bandwidth using hierarchical tree would need the use of specialized switches in the higher levels of the hierarchy. Fat-tree which leverages commodity Ethernet switches offers a clear economic benefit and reduces the total cost of implementation significantly.

References

- [1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, August 2008.
- [2] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92, 2010.
- [3] Nanette J Boden, Danny Cohen, Robert E Felderman, Alan E. Kulawik, Charles L Seitz, Jakov N Seizovic, and Wen-King Su. Myrinet: A gigabit-per-second local area network. *IEEE micro*, 15(1):29–36, 1995.
- [4] Charles Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.
- [5] William J Dally and Brian Towles. Route packets, not wires: on-chip interconnection networks. In *Proceedings of the 38th annual Design Automation Conference*, pages 684–689. ACM, 2001.
- [6] Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6. IEEE, 2014.
- [7] Christian Hopps. Analysis of an equal-cost multi-path algorithm. Technical report, 2000.
- [8] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.
- [9] Charles E Leiserson, Zahi S Abuhamdeh, David C Douglas, Carl R Feynman, Mahesh N Ganmukhi, Jeffrey V Hill, W Daniel Hillis, Bradley C Kuszmaul, Margaret A St Pierre, David S Wells, et al. The network architecture of the connection machine cm-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, 1996.
- [10] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
- [11] J Moy. Rfc 2328. *OSPF version*, 2, 1998.
- [12] Gregory F Pfister. An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O*, 42:617–632, 2001.

- [13] Dave Thaler and C Hopps. Multipath issues in unicast and multicast next-hop selection. Technical report, 2000.
- [14] Lewis W Tucker and George G Robertson. Architecture and applications of the connection machine. *Computer*, 21(8):26–38, 1988.
- [15] Michael Woodacre, Derek Robb, Dean Roe, and Karl Feind. The sgi altixtm 3000 global sharedmemory architecture. *Silicon Graphics, Inc.(2003)*, 2005.

A Appendix: Instructions for running our code

- To run baseline tree measurements:
 - Start controller of your choice in a terminal.
 - Wait until controller starts running and then run *sudo python treetest.py*.
- To run measurements on fat-tree with hash based static load balancing:
 - Run *sudo python fatTreeTest.py*.
- For static load balancing using implementation described in section 4.1:
 - Run *sudo python fatTreeTest_RoutingPaper.py*.

The raw bandwidth results will be stored in treeResultsBW and fatTreeResultsBW folders.