AN EXPERIMENTAL EVALUATION OF DATA CENTER TOPOLOGIES

BY

FANG-KAI JAO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Assistant Professor P. Brighten Godfrey

ABSTRACT

A fair way to evaluate theoretical results related to the network topology design is running experiments on the topology built physically, such as by a network testbed. In this thesis, we introduce a testbed implementation and show the throughput results for the fat-tree and Jellyfish topologies using this implementation.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In network topology design, we expect that the theoretical results are as close to the real case as possible. A fair way of evaluation is running experiments on the exact network topology that is built physically. A *network testbed* is a platform that can run such experiments to evaulate theories, which typically include software, hardware, and networking components. In this thesis, we introduce a network testbed implementation, the *Ocean Cluster for Experimental Architectures in Networks* (OCEAN)[1].

Software-defined networking (SDN) is a technique in computer networking that allows network services to be managed through an application called a controller. By specifying through an SDN controller how the traffic should be forwarded according to the destination, a physical switch can behave as multiple switches; we call such a simulated switch a *virtual switch*.

A *hypervisor* is a program that allows multiple operating systems to share the hardware resources of a single machine. By binding the network interfaces of a single machine running a hypervisor to different operating systems on the hypervisor, a (physical) machine can behave as multiple nodes in a network. We call such an operating system simulating a node a *virtual host*. In Chapter 3, we will show how OCEAN is built with these two techniques, as well as the software and the networking components.

Jellyfish is a randomly constructed network topology introduced in [1].

[1] `http://ocean.cs.illinois.edu`

The authors showed that it can achieve similar or better throughput than fat-tree introduced in [2]. However, the results are made by software simulations. Using the OCEAN testbed, we evalute the throughput results of Jellyfish and fat-tree. Furthermore, to obtain full performance, the hosts in OCEAN are installed a kernel that supports Multipath TCP (MPTCP), where MPTCP is a development that allows a Transmission Control Protocol (TCP) connection to use multiple paths to maximize resource usage and increase redundancy. The results and discussions are in Chapter 4.

CHAPTER 2

RELATED WORK

To be realistic, results in network research have been evaluated by network emulation testbeds. The goals of a testbed can be different based on the purpose. A topology-oriented testbed would have a goal of emulating as many topologies as possible. As in its initial stage, OCEAN is capable of emulating multiple topologies and running throughput experiments.

Emulab[1] is one of the oldest and largest network testbeds; it is available to the public since 2000. The primary Emulab installation is run by the Flux Group, part of the School of Computing at the University of Utah. The group studied in [3] how to build a better testbed by analyzing more than 500,000 topologies from 13,000 experiments submitted to Emulab, where the switch connectivity is discussed. When OCEAN was built, the idea of random networking presented in [1] was used to connect the switches. More explicitly, except those cables ensuring that all switches are connected each other the rest cables are randomly wired. Similar to the Jellyfish topology, this can be studied in the aspect of network testbed design.

In [1], it is shown that Jellyfish can achieve similar or even better throughput than fat-tree. Also, MPTCP is used for the experiments, and the authors of [4] showed that most MPTCP flows get at least 90% of the available capacity with eight subflows. Since there has been no hardware (as opposed to simulation) evaluation of Jellyfish, as a possible pioneer testbed OCEAN

[1] `http://www.emulab.net`

has been used to evaluate Jellyfish and fat-tree of small size, with MPTCP support.

CHAPTER 3

THE OCEAN CLUSTER FOR EXPERIMENTAL ARCHITECTURES IN NETWORKS (OCEAN)

3.1 Hardware

The skeleton of OCEAN currently consists of 13 Pronto 3290 switches and four Dell PowerEdge T620 tower servers. Each switch has one management port and 48 10/100/1000 BASE-T *switch ports* used. And each server has two 6-core Intel Xeon E5-2630L 2.0 GHz CPU, 32 GB DDR3 RAM, two embedded 1 Gbps Ethernet ports (`em1` and `em2`). In each is installed five 6-port NIAGARA 32066 network interfaces, making a total of 30 *server ports*, and is running the VMWare ESXi hypervisor that is configured to have 30 virtual hosts, where each host has a single-core CPU, 512 MB RAM, and two network interfaces (`eth0` is bound to `em2` and `eth2` is bound to one of the server ports). Note that the embedded port `em2` is shared by all 30 virtual hosts, but no server port is shared.

3.2 Software

As mentioned, an SDN controller is the media that OCEAN uses to communicate with the switches. We use Floodlight[1] as the SDN controller, and our software[2] communicates with Floodlight via its REST APIs, by which we can discover the physical topology and add/remove flows to the switches to con-

[1] `http://www.openflowhub.org/display/floodlightcontroller`
[2] `https://github.com/kylejao/ocean`

trol packet forwarding. Currently, the software only runs experiments that measure throughput by running `iperf`[3]. The following is the basic workflow to measure the throughput over a target network.

1. Discover the current network topology
2. Find a mapping between the current and the target topologies
3. If such a mapping exists, slice the switches to form the target network
4. Define source-destination host pairs and find paths
5. Adding required host configurations (ARP, interfaces, routing tables)
6. Sending the traffic and measuring the throughput with iperf

3.3 Networking Components

There are two private networks in OCEAN. One is the *management network* (currently set to `192.168.2.024`), where we can reach each switch via its managemenet port and each virtual host via `eth0`. All these ports are physically linked to another management switch. The other network is the *testbed network* (currently set to `192.168.100.024`), where we can reach all switch ports and server ports. In OCEAN, only 10 of the switches are physically linked to server ports, each of which is linked to exactly 12 server ports. Also, each switch is linked to all other switches and itself at more than one switch port. The rest switch ports are wired pairly at random. Finally, in order to remotely access each hypervisor, the interface `em1` of each server is used to connect to the internet. In general, all these configurations can be set differently.

[3] `http://iperf.sourceforge.net`

In fact, some new networks will be created for subflows if MPTCP is enabled. Since MPTCP finds all paths between two hosts, if more than one path is available between two virtual hosts, we assign a new IP address bound to the same interface (`eth2`) on both source and destination virtual hosts for each such path, and use the new addresses as the source and destination when adding flows for the path. Also, since all virtual hosts are in the management network, we need to add routing table rules to block the connection over the management network. For example, if there are two paths from virtual host A (`192.168.100.101`) to virtual host B (`192.168.100.102`), then we add the flows for the path with `192.168.100.101` as the source and 192.168.100.102 as the destination and the flows for the other path with `192.168.101.101` as the source and `192.168.101.102` as the destination, as well as the firewall rules that block the traffic between A and B over the management network.

It is worth noting that we drop all broadcasting packets in OCEAN, since ARP broadcast storms can be caused as we have internal loop-like links. Alternatively, for each source-destination pair we add MAC addresses to the ARP cache.

CHAPTER 4

FAT-TREE VERSUS JELLYFISH ON OCEAN

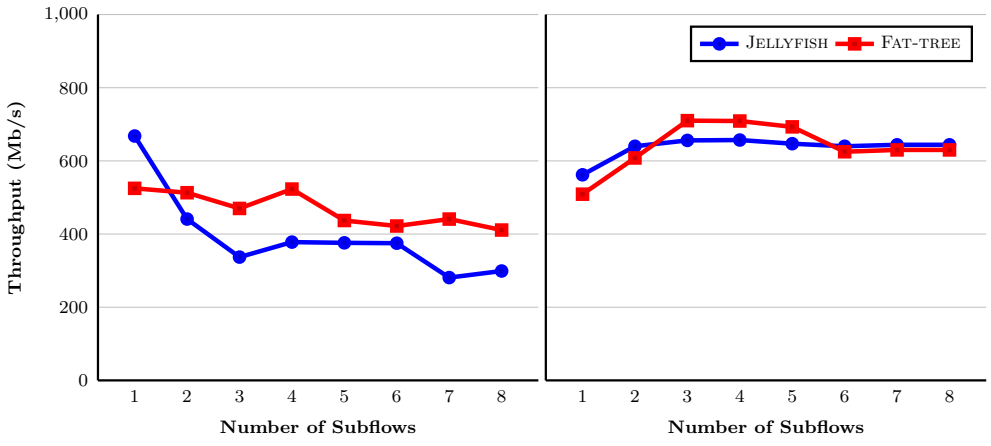
In this chapter, we show the throughput results of 3-layer fat-tree networks introduced in [2] and the Jellyfish networks introduced in [1] built with OCEAN. All results shown in this chapter are based on the top-k-shortest-path routing algorithm and the *random permutation traffic pattern*, namely each virtual host sends traffic to a single other virtual host and receives from a single other virtual host, where the permutation is chosen uniformly randomly. We will show results using TCP and MPTCP, as well as MTU = 1500 bytes and 9000 bytes, where MTU is the maximum transmission unit.

In each section, we show the throughput results of a fat-tree network and a Jellyfish network, and they both have the same number of switches, nodes and links. Futhermore, the Jellyfish is built in the way that nodes are evenly connected to the switches. In particular, each switch in the fat-tree or the Jellyfish in Section 4.1 has four ports, and hence both networks have 20 switches, 16 nodes, and 32 internal links. And each switch in the fat-tree or the Jellyfish in Section 4.2 has six ports, and hence both networks have 45 switches, 36 servers, and 108 internal links. Recall that the bandwidth of each link is 1 Gbps.

4.1 4-Port Switches

Figure 4.1a shows the throughput results of a 3-layer fat-tree network and a Jellyfish network built with OCEAN, where TCP is used and MTU = 1500 bytes. The fat-tree has better throughput than the Jellyfish when more subflows are used, but the overall performance is less than 70% of the available capacity. Also, the performance does not improve as we use more subflows. One reason for this is that when more subflows are used, more links are shared and shared by more subflows. This suggests to optimize the usage of links.

Figure 4.1b shows the throughput results using MPTCP. As shown, the performance is better than using TCP, and furthermore, it improves moderately as we use more subflows. Although the two networks seem competitive, they both have only about 70% of the available capacity. Also, we noticed that having more subflows improves the performance only at the beginning. We will discuss this later.



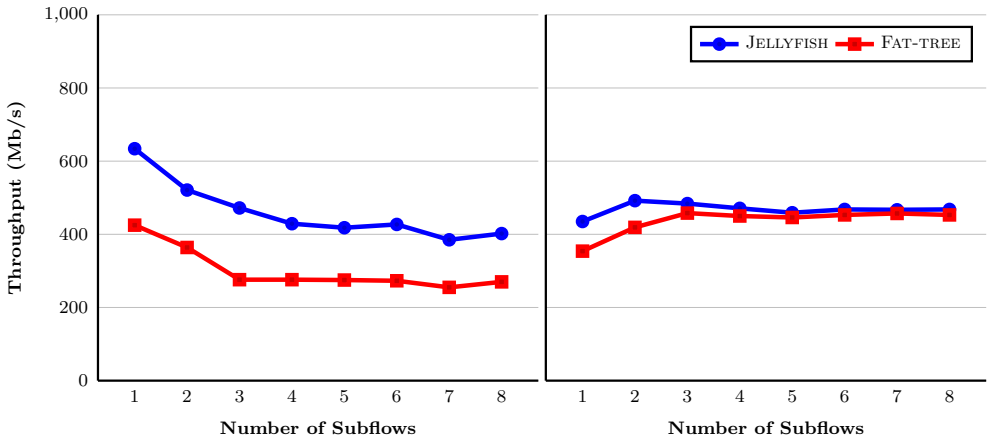
(a) 4-port, TCP, MTU=1500 (b) 4-port, MPTCP, MTU=1500

Figure 4.1

4.2 6-Port Switches

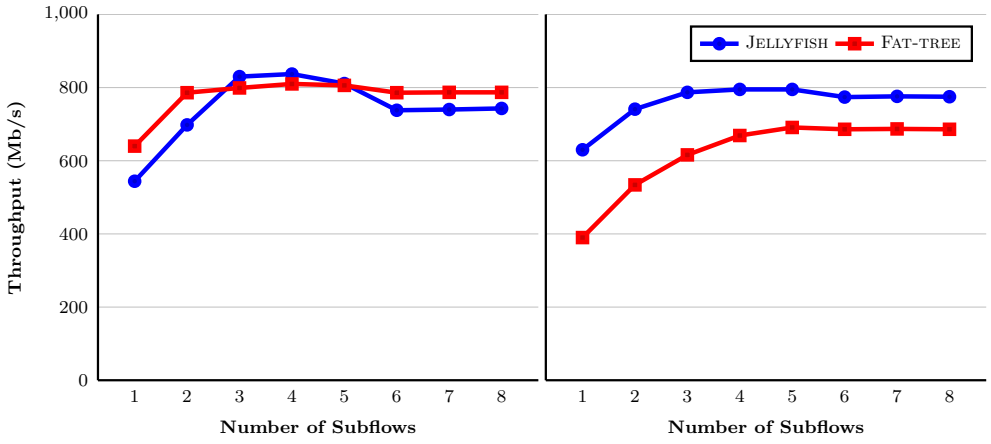
Figures 4.2a and 4.2b show the throughput results of a 6-port fat-tree and a Jellyfish built with OCEAN. In all cases, the Jellyfish has better throughput than the fat-tree. However, the overall performace not only does not improve as we use more subflows, but is even less than 60% of the available capacity. And again, when using MPTCP we noticed that the performance peaks at the beginning. In fact, we found that many packets were dropped in this case. Due to the fact that MPTCP uses more CPU-cycles, this seems to be caused by the hardware constraint. To confirm this, we changed the MTU to 9000 bytes, and the results are shown in Figures 4.1b and 4.3b.

Using MPTCP and MTU = 9000 bytes, the performance is about 80% of the available capacity in the 4-port case. In the 6-port case, when more subflows are used the performance of the Jellyfish was measured to be about 80% and that of the fat-tree was measured to be about 70%. But again, we can see that the performance still peaks before more subflows are used. So we believe that the performance is still limited by the hardware contraint in both cases.



(a) 6-port, TCP, MTU=1500 (b) 6-port, MPTCP, MTU=1500

Figure 4.2



(a) 4-port, MPTCP, MTU=9000 (b) 6-port, MPTCP, MTU=9000

Figure 4.3

CHAPTER 5

CONCLUSION

With the configurations described in 3, the OCEAN testbed can emulate small networks with flexibility. And from the results shown in 4, we can see that OCEAN can run experiments with choices of routing algorithms (different numbers of subflows) and transport protocols (TCP or MPTCP). Although part of the results were limited by the hardware constraint, the results still show that OCEAN runs reasonably fair experiments.

For future work, in addition to supporting more SDN controllers, routing algorithms, and traffic patterns by the software, we can improve the flexibility in order to flexibly emulate more networks. When a target network can not be emulated by the current topology, we might be able to rewire the cables to make it possible. More explicitly, how can we rewire the cables to make the emulation possible or decide that the emulation is not possible?

REFERENCES

[1] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, April 2012.

[2] 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," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, August 2009.

[3] F. Hermenier and R. Ricci, "How to build a better testbed: Lessons from a decade of network experiments on emulab," vol. 44, pp. 287–304, 2012.

[4] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 266–277, August 2011.