Comparative performance evaluation of Low Delay Routing on emulated environment

Ádler Oliveira Silva Neves
adler.neves@aluno.ufes.br

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

1 Introduction

On a world with web services billed by the millisecond, such as AWS Lambda[1], time is money. Accessing a remote database from such environment means that the time spent waiting the connection to establish is money lost. On the web, such delay in the response from the server side increases power consumption on mobile devices[2] and decreases user experience[5]. Therefore, reducing such latency is crucial and would bring clear benefits.

Gvozdiev et al. proposes Low Delay Routing[4] (LDR) as a solution for this problem, which is a network routing algorithm that got impressive results on the metrics that reflect its purpose. Despite its attractive results on latency, there's no in-depth evaluation of bandwidth and jitter. Storage and processing components usually requires that the owner chooses up to two attributes from high speed, high capacity and low cost, and, extending such dilemma to the network area, LDR had to give up somewhere. But where? Are there other limits which were not mentioned?

2 Materials and Methods

To answer that question, we set up a testing environment with Mininet, Ryu and some own code on a PC equipped with an i7 4790 (4 cores, 8 threads, 4GHz) and 16GB RAM DDR3@1600MHz, running Arch Linux. "Mininet is a system for rapidly prototyping large networks on the constrained resources of a single laptop"[6, p. 1], which was used to create the emulated network and run commands on the hosts it created. "Ryu is a component-based software defined networking framework"[3], which was used to define the behavior that the switches would have using OpenFlow. Our own code is a collection of Python scripts and a Makefile, including our re-implementation of Gvozdiev et al.'s as a Ryu controller, which were responsible for routing, monitoring, visualization, automatic testing and both table and chart generation. These tools generated the data that will be discussed on section 3, but, first, we will explain more on how our tool set does what it does on the next subsections.

2.1 Dividing the network for tests

Suppose we have an array of hosts, such as:

h1	h2	h3	h4	h5	h6	h7	h8	h9

An easy way to get pairs to test is split such array in path, discarding the extra member:

h1	h2	h3	h4	h5
h6	h7	h8	h9	

As `h1` and `h6` were, on some topologies, closer to each other than to `h9` (the last element), and the

first pair would be a latency test, it was chosen to reverse the second part. The final combination would be:

h1	h2	h3	h4
h9	h8	h7	h6

That said, `h1` and `h9` are latency tests and all other 3 pairs are bandwidth tests. All tests are run sequentially and then concurrently, in order to measure all links in its idle and overloaded state. Between every test there's a wait time that is exactly 2 times the monitoring interval, which is the minimal interval to make the routing algorithm see the entire network as unused. Latency is `avg` from `ping` command; jitter, `mdev` from the same command; bandwidth, the bottom line of the `iperf` command running as client, using TCP.

2.2 Representing the topologies

The topologies are stored as JSON files. Such file contains an array that contains 3 arrays. The first array is a list of strings that represents the list of hosts. The second array is a list of strings that represents the list of switches. The last array is a list of arrays that represents the list of links. Every array that represents a link contain an two strings that represents either a host or a switch, and the third element is a number that represents the maximum bandwidth of such link. Then, it comes the time to present the topologies.

2.3 The topologies

We tested 9 topologies. Switches (starts with "s", such as "s3") were represented by blue circles. Hosts (starts with "h", such as "h7") were represented by green circles. When there is no indication of speed on the edge, the speed is 1 mbps. The topologies are:

Triangle: This topology is a triangle that has a longer and slower path that only becomes a viable path when the shorter and faster is congested.

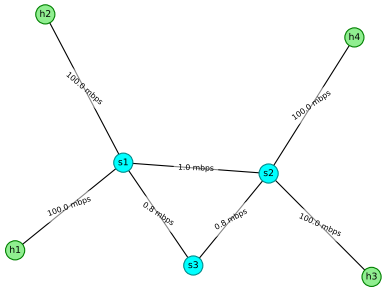

Figure 1: "Triangle" topology

Binary tree: asd

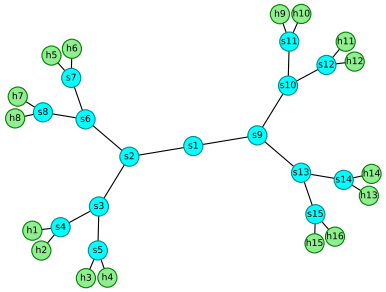

Figure 2: "Binary tree" topology

Fat tree: asd

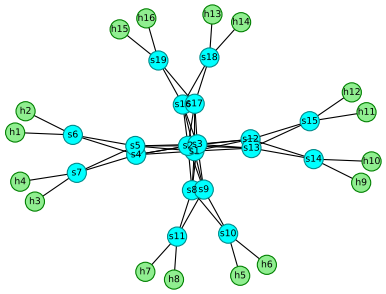

Figure 3: "Fat tree" topology

3-layered CLOS: asd

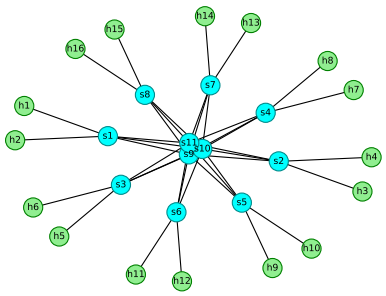

Figure 4: "3-layered CLOS" topology

Bipartite: A CLOS with an extra

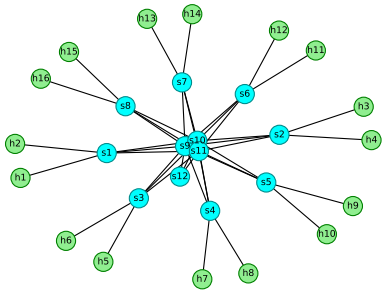

Figure 5: “Bipartite” topology

5-layered CLOS: asd

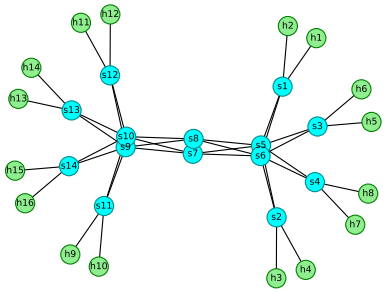

Figure 6: “5-layered CLOS” topology

Grid: asd

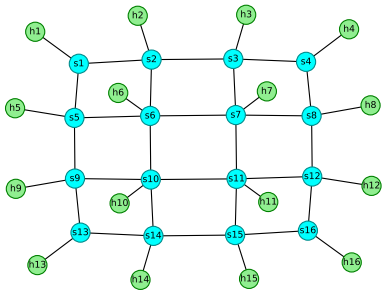

Figure 7: “Grid” topology

DCell: This topology, as presented in Pries et al.’s work [7],

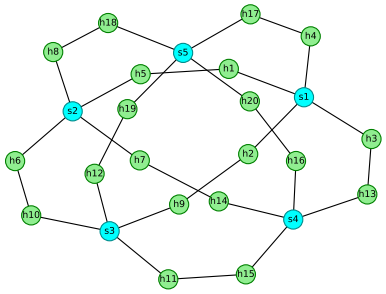

Figure 8: “DCell” topology

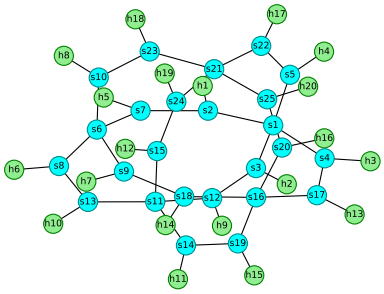

Figure 9: “DCell” topology with extra switches

BCube: This topology, as presented in Pries et al.’s work [7],

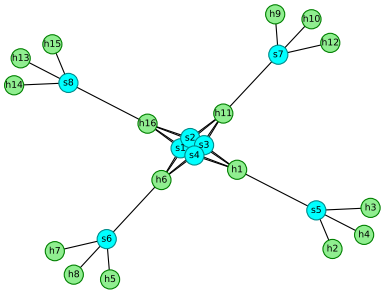

Figure 10: “BCube” topology

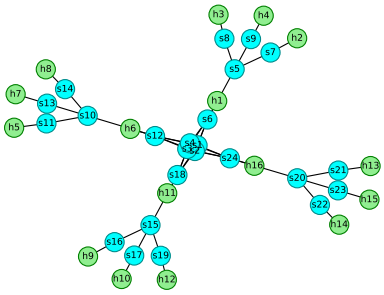

Figure 11: “BCube” topology with extra switches

2.4 The OpenFlow controller

Our Ryu controller got the shortcut of knowing the topology

3 Results

4 Discussion

5 Acknowledgments

References

[1] Amazon. *AWS Lambda pricing*. [Online; accessed 05-Jul-2019]. 2019. URL: https://aws.amazon.com/lambda/pricing/.

[2] Duc Hoang Bui et al. "Rethinking Energy-Performance Trade-Off in Mobile Web Page Loading". In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. MobiCom '15. Paris, France: ACM, 2015, pp. 14–26. ISBN: 978-1-4503-3619-2. DOI: 10.1145/2789168.2790103. URL: http://doi.acm.org/10.1145/2789168.2790103.

[3] Ryu SDN Framework Community. *Ryu SDN Framwork*. [Online; accessed 05-Jul-2019]. 2017. URL: http://osrg.github.io/ryu/.

[4] Nikola Gvozdiev et al. "On Low-latency-capable Topologies, and Their Impact on the Design of Intra-domain Routing". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. Budapest, Hungary: ACM, 2018, pp. 88–102. ISBN: 978-1-4503-5567-4. DOI: 10.1145/3230543.3230575. URL: http://doi.acm.org/10.1145/3230543.3230575.

[5] Conor Kelton et al. "Improving User Perceived Page Load Times Using Gaze". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 545–559. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kelton.

[6] Bob Lantz, Brandon Heller, and Nick McKeown. "A Network in a Laptop: Rapid Prototyping for Software-defined Networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 19:1–19:6. ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868466. URL: http://doi.acm.org/10.1145/1868447.1868466.

[7] Rastin Pries et al. "Power Consumption Analysis of Data Center Architectures". In: *Green Communications and Networking*. Ed. by Joel J. P. C. Rodrigues et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 114–124. ISBN: 978-3-642-33368-2.