Comparative performance evaluation of Low Delay Routing on emulated environment

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

Abstract

On a computer network, packets hop from network element to network element as a way to propagate a message that is expected to reach at its destination. How the network is doing this, however, is constantly changing, as Software Defined Networks brought programmability to the switches that forward packets. With some web services getting billed by the millisecond, the load speed influencing both user experience and battery usage on mobile devices, latency became crucial. There's a routing algorithm by Gvozdiev et al. (2018) which has promising but untested results, which may be subject to unknown limitations. Here we show that Low Delay Routing doesn't handle extreme cases of overload well. We found that when all routes are congested, Low Delay Routing prioritize bandwidth over both latency and jitter. Our results demonstrate that such routing algorithm achieves low latency as a side-effect of maximizing bandwidth, which isn't guaranteed happen on all cases. These cases in which low latency was still achieved were with CLOS and B-Cube topologies. We anticipate that the major contribution of our article to be the framework used to compare more algorithms on more topologies.

1 Introduction

On a world with web services billed by the millisecond, such as AWS Lambda[2], 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[3] and decreases user experience[7]. Therefore, reducing such latency is crucial and would bring clear benefits.

Gvozdiev et al. proposes Low Delay Routing[6] (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"[8, 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"[5], which was used to define the behavior that the switches would have using OpenFlow. Our own code[10] is a collection of Python scripts and a Makefile, including our partial re-implementations of Gvozdiev et al.'s algorithm 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 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.

2.2 The topologies

We tested 9 topologies. Switches (starts with "s", as in "s3") were represented by blue circles. Hosts (starts with "h", as in "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 (figure 1) is a triangle that has a longer and slower path that only becomes a viable path when the shorter and faster is congested.

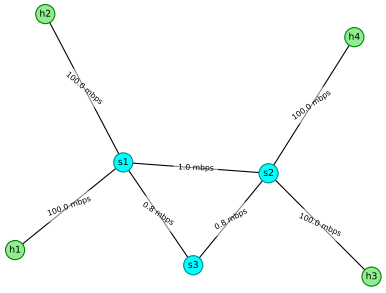

Figure 1: "Triangle" topology

Fat tree: Essentially, a complete bipartite graph $k_{3,8}$ where every pair of 2 switches of the group with 8 nodes make another $k_{2,2}$ graph where the new switches which are more distant from the 3 ones from the core gets 2 hosts connected to each one. This topology (figure 2) is equivalent to Pries et al.'s[11] one.

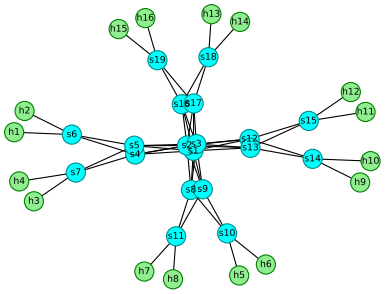

Figure 2: "Fat tree" topology

3-layered CLOS: A 3-stage CLOS fabric as defined by Al-shawi[1], which has 3 switches acting as spine and 8 as access leaves, with 2 hosts per access leaf. It can also be seen as an $k_{3,8}$ (as in Fat tree, but without that $k_{2,2}$ process) with 2 nodes on every one of those 8 nodes, as expressed on figure 3.

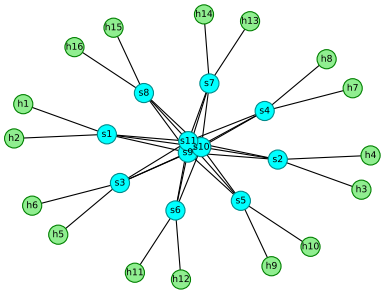

Figure 3: "3-layered CLOS" topology

Bipartite: The same as the previous one (3-layered CLOS), but with an extra switch connecting all 3 spine switches, as expressed on figure 4.

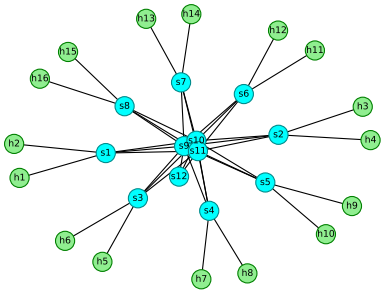

Figure 4: "Bipartite" topology

Binary tree: A full binary tree with $depth = 5$ where the childless nodes are hosts and the other ones are switches, as seen in figure 5.

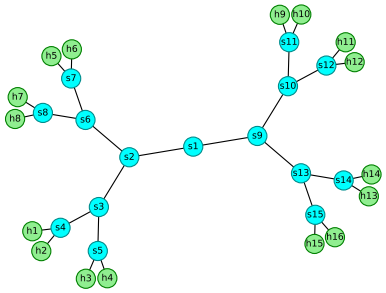

Figure 5: "Binary tree" topology

5-layered CLOS: A 5-stage CLOS fabric as defined by Al-shawi[1], which has 2 rows of 4 access leaves each, 3 rows of 2 spine switches each, and 2 hosts on each access leaf, as seen in figure 6.

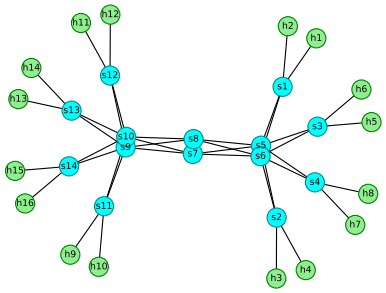

Figure 6: "5-layered CLOS" topology

Grid: This topology (figure 7) aims to represent a mesh of squares, which is a trivial solution for physically interconnecting columns rows and rows full of servers.

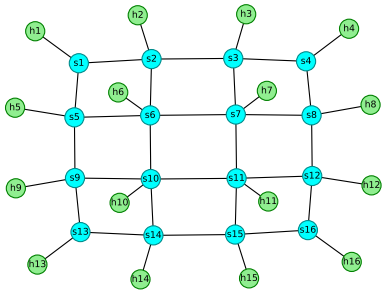

Figure 7: "Grid" topology

DCell: This topology is the same as the one presented in Pries et al.'s work [11] and replicated on figure 8, however Mininet hosts don't forward packets to other hosts by default and don't obey OpenFlow rules, so a change was needed.

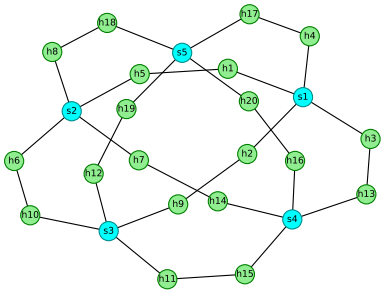

Figure 8: "DCell" topology

After such change, every host has become a switch with a host connected to it, and all connections the host had, the switch gets them all, as seen in figure 9.

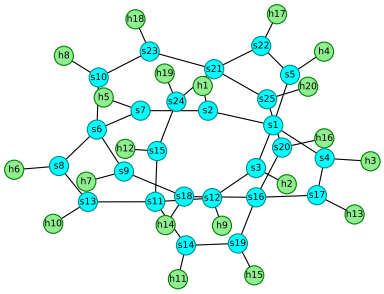

Figure 9: "DCell" topology with extra switches

BCube: This topology is the same as the one presented in Pries et al.'s work [11] and replicated on figure 10, however the same problem as the one present on DCell repeats.

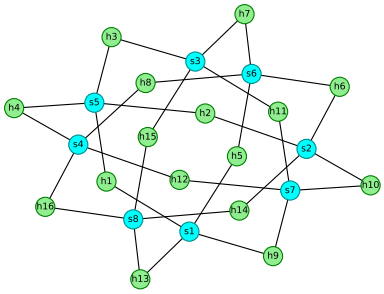

Figure 10: "BCube" topology

The same solution as applied on DCell also repeats here, as seen in figure 11.

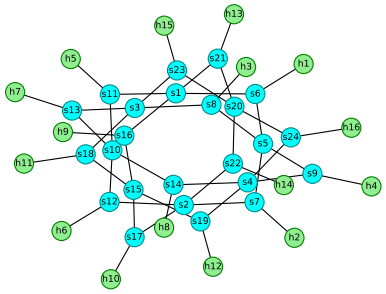

Figure 11: "BCube" topology with extra switches

With these 9 topologies defined, we chose to run 30 tests on them. However, there was no defined method of testing those topologies in a way that the numbers would reflect routing algorithms' ability to maintain low latency, low jitter and high bandwidth even when congestion is unavoidable.

2.3 Testing the topologies

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	

That done, `h1` and `h6` would be latency tests and all other 3 pairs are bandwidth tests. This would be a great simple solution if `h1` and `h6` could be more distant to each other on Grid topology (figure 7). Keeping this as is would bring a latency test which result would reflect more about the capacity of the routing algorithm on making a better use of the network. A host which is consistently the most distant is the last one. As `h1` and `h6` were, on some topologies, closer to each other than to `h9` (the last element). Therefore, 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.

Our testing process starts the topology, wait the controller be ready, run the tests, write the results and finishes itself. All tests are run sequentially and then concurrently, in order to measure all links in its idle and overloaded states. Between every test there's a wait time that is exactly 2 times the monitoring interval, which is the minimal interval to let 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. The bulk testing script we have also write controller's configuration to change the routing algorithm being tested, starts the controller, run a giver number of tests (we chose 30) for each pair routing algorithm, ends the execution of the controller and destroys the topology and, after all tests were ran, store all results in a single file.

Such file that contains the results of all tests was later used to generate the tables, box and whisker plots and a ternary plot that will be presented on section 3.

2.4 The OpenFlow controller

Our OpenFlow controller was written using Ryu as framework. To avoid going over the task of discovering the topology and recovering from topology changes at run-time, it was chosen to let the topology be static and pass the topology (section 2.1) as a command-line argument because this wouldn't impact the tests. Translating Gvozdiev et al.'s Points of Presence to Mininet's hosts, it was possible to map the concepts the article mentions into the framework we have. It was tested 5 routing algorithms: Spanning Tree (Dijkstra-based), 2 implementations of LDR, ECMP (equal-cost multiple path) and a reimplementation of MinMax from Gvozdiev et al.'s textual description.

Gvozdiev et al.'s LDR algorithm is inherently multi-path, using different paths as some sort of link aggregation, which means that packets may not necessarily arrive at the order it was sent. So, keeping this nature, the linear program from Gvozdiev et al.'s figure 12 [6, p. 96] was reimplemented using the library PuLP to solve such linear program, using link load obtained from monitoring and latency on link derives from; figure 13's [6, p. 96] conditional path set exten-

sion was adapted to always give the linear program the paths the pass a filter that only allow a path stretch up to 1.4 [6, p. 89] after applied Gvozdiev et al.'s algorithm 1 [6, p. 94]. This can be summed up as the flow: Monitoring → Algorithm 1 → APA-based filter doing the role of figure 13 → Figure 12 → Flow table. The steps about the "iteration to assess statistical multiplexing" as expressed on Gvozdiev et al.'s figure 14 [6, p. 96] were simply ignored because we considered it to be very abstract with many possible interpretations and, as consequence, hard to re-implement without re-implementing something the authors did it in a very different manner and achieving completely different results. We called this "LDR (multi-path)" on charts.

However, not all applications handles packets arriving out of their order well. So, instead of introducing a packet queue (increasing delay and possibly introducing buffer bloat as a concern), we could also change the routing algorithm to choose only one path and change later, as needed. Instead of solving a linear program, Gvozdiev et al.'s figure 12 [6, p. 96] could be interpreted as a cost function, where we should pick the path on the aggregate which is the minimum. Such cost function would be $C(p) = d_p + \frac{d_p M_1}{S_a} + M_2 O_{max} + \sum_l O_l$, which is the per-aggregate part of the sum which is part of the linear program from Gvozdiev et al.'s figure 12 with the restriction of choosing only one path ($x_{ap} = 1$). Then, Gvozdiev et al.'s figure 13 [6, p. 96] can be applied by interpreting the path set extension as a path change, and the decision if there's any link overloaded as a decision if there's a change that figure 12 suggested that would make the routing better (less congested). As there is only one path being used by aggregate, there is no statistical multiplexing to be assessed on Gvozdiev et al.'s figure 14 [6, p. 96], so it was ignored. We called this "LDR (single-path)" on charts.

Gvozdiev et al.'s figure 12 [6, p. 96] uses 2 constants: M_1 and M_2. The only hint to give M_1 a value was "to ensure this is only a tie-break, M_1 is a very small constant"[6, p. 96]. As a wild guess over that vague description and seeing its use on Gvozdiev et al.'s figure 12 [6, p. 96] as being multiplied by a fraction that ranges from 0 to 1, we started Mininet with its default topology and ran a `ping` between the two hosts; as the value was $0.6ms$, then we set M_1 to be 0.0006. M_2, on its time, acts over the maximum overload of a given path "[...] subject to the constraint that they avoid congestion; M_2 is large to ensure this term dominates"[6, p. 95]. As another wild guess, we set that value to be 120; with particular reason and with no clue if such value being too large could have negative side-effects, but with the suspicion that M_2 should not be a constant, but a metric of such node in a graph that represents the network, however we didn't investigate that possibility on this article. That said, M_1 and M_2 lacks better explanation on what it does and how could we calculate them.

Our Dijkstra-based Spanning Tree algorithm takes the advantage of knowing the entire topology (which is also static) and calculates the shortest path for each pair of hosts using Dijskstra. The advantage of doing Spanning Tree this way and not through OSPF is that we will not be sending the topology over the links periodically, being able to keep the topology completely idle while no host is sending or receiving packets. No routing difference is expected to exist between this approach and OSPF. Even that it's not OSPF on a strict sense, we called this "OSPF (single-path)" on charts.

Our ECMP implementation lists all paths between two hosts, discards the longer paths which have conflicting flow directions over the same link as shorter ones (or else we would have packet loss) and creates a static link aggregation that uses all remaining flows to deliver packages. Similarly as "LDR (multi-path)", this routing algorithm is expected to have issues with applications which doesn't handle packets arriving out of their order well. We called this "ECMP (multi-path)" on charts.

Gvozdiev et al. describes "[...] a pure MinMax approach [as one that] optimizes traffic placement so as to minimize the maximum link utilization[...]"[6, p. 96]. Therefore, we wrote an algorithm that takes every host pair from $\wp(H) \setminus \{(h,h) \forall h \in H\}$ (where H is a set of hosts), simulate all possible paths for that source-destination pair, pick the path that has the minimum maximum path load and update the current network model to have that traffic moved to that chosen path; after all source-destination pairs have been processed, our network model will hold the future network topology. We called this "MinMax (single-path)" on charts.

With all topologies, algorithms and testing

methodology defined, we set our topology update cycle to 1 second and ran our tests over some days and we can, now, proceed to the results.

3 Results

After all tests ran, we got the 20 tables, 170 box plots and 10 ternary scatter plots. Due to space restrictions, lack of relevance of some collected data and limited ability to analyze so many facets from a data matrix with 4 dimensions (routing algorithm, topology, metric and sample), we will only display here 7 tables, 33 box plots and a single ternary plot.

The tables 1 to 4 have a line and a column named "All", which represents the union of the data from all neighbor lines and columns, respectively. Such union was, later, used to quickly represent all topologies on a single plot. This can be observed on the box plot figures 12 to 14, which were generated using matplotlib[4], displaying the median as a green line, the mean as a blue diamond and the outliers as red "×"s. Figure 15 was also plotted using such from "All" column, combining individual values of concurrent latencies, jitter and bandwidth, plotted using python-ternary[9].

Spotting out the outliers on the tables 1 to 3 is difficult, the values from the columns were subtracted from the column "All". Therefore, the tables 5 to 7 contain the deviation of such algorithm's average on a given topology from all topologies' algorithm average. On the table 5, we can see that on "B-Cube" and on "3-layered CLOS" the values of both "LDR (single-path)" and "MinMax (single-path)" on the table are negative while all others are positive, "5-layered CLOS", "B-Cube" and "Triangle" are positive on "LDR (multi-path)" while all others are negative, "Binary Tree", "B-Cube" and "3-layered CLOS" are positive on "ECMP (multi-path)" while all others are negative; the value "ECMP (multi-path)" × "5-layered CLOS" is the only 3-digit number (after truncation) of the entire table. On the table 6, we can see that on the line "Binary Tree", "B-Cube" and "D-Cell" the values of both "LDR (single-path)" and "MinMax (single-path)" on the table are negative while all others are positive; on "LDR (multi-path)", "5-layered CLOS", "B-Cube" and "Triangle" are positive while other values are negative; on "ECMP (multi-path)", "Binary Tree", "B-Cube", "D-Cell" and "3-layered CLOS" are positive while other values are negative; again, "ECMP (multi-path)" × "5-layered CLOS" is the most significative value of the table. On table 7, we can see that on the line "B-Cube" and "Triangle" the values of both "LDR (single-path)" and "MinMax (single-path)" on the table are positive while all others are negative; on "LDR (multi-path)", "Binary Tree", "5-layered CLOS" and "B-Cube" are negative while other values are positive; on "ECMP (multi-path)", "Binary Tree", "B-Cube", "D-Cell" and "Triangle" are negative while other values are positive; "ECMP (multi-path)" × "5-layered CLOS" isn't the most significative value of the table, but "Bipartite" × "LDR (multi-path)". Therefore, we also included the box plot figures 16 to 30, which contains boxes and whiskers blots previously mentioned (which are redundant on table 8 for readability).

Table 8: Outliers identified on tables 5 to 7 which are represented on figures 16 to 30, highlighting line with different values

Latency	Jitter	Bandwidth
Binary Tree	Binary Tree	Binary Tree
B-Cube	B-Cube	B-Cube
3-layered CLOS	D-Cell	D-Cell
5-layered CLOS	5-layered CLOS	5-layered CLOS
Triangle	Triangle	Triangle

The figures 31 to 45 are slices from the aforementioned 4-dimensional matrix, by algorithm and metric, containing the topologies on the horizontal axis, the values of each measurement on the vertical axis, grouped as a box and whiskers plot, also generated using matplotlib[4]. Such view will be relevant on discussion section.

Something we didn't mention on section 2.2 is that our 5-layered CLOS (figure 6) originally had 3 switches on each spine row. The problem with that is that there are more paths in that topology than our computer had memory to represent them. To solve that, we had to reduce it to 2 switches on each spine row. This may suggest some limitations on routing algorithm applicability.

Something we noticed during the tests was that routing algorithm response time got slower as the topology had more paths available. Such im-

Table 1: Average of all concurrent latency tests

Latency (concurrent)	OSPF (single-path)	LDR (single-path)	MinMax (single-path)	LDR (multi-path)	ECMP (multi-path)	All
Binary Tree	343.69 ms	344.13 ms	344.33 ms	343.25 ms	344.24 ms	343.93 ms
5-layered CLOS	343.65 ms	342.78 ms	343.36 ms	306.31 ms	146.70 ms	296.56 ms
B-Cube	11.66 ms	17.53 ms	12.68 ms	67.84 ms	33.48 ms	28.64 ms
Fat Tree	156.51 ms	156.23 ms	156.90 ms	68.09 ms	53.67 ms	118.28 ms
Grid	193.80 ms	180.29 ms	187.01 ms	153.61 ms	116.54 ms	166.25 ms
D-Cell	107.61 ms	106.78 ms	107.33 ms	89.34 ms	90.61 ms	100.34 ms
Bipartite	19.91 ms	19.93 ms	20.39 ms	9.31 ms	8.33 ms	15.58 ms
3-layered CLOS	20.39 ms	19.86 ms	19.78 ms	13.58 ms	45.66 ms	23.85 ms
Triangle	76.26 ms	78.36 ms	75.98 ms	76.97 ms	23.13 ms	66.14 ms
All	141.50 ms	140.66 ms	140.86 ms	125.37 ms	95.82 ms	128.84 ms

Lower is better

Table 2: Average of all concurrent jitter tests

Jitter (concurrent)	OSPF (single-path)	LDR (single-path)	MinMax (single-path)	LDR (multi-path)	ECMP (multi-path)	All
Binary Tree	106.37 ms	106.26 ms	106.30 ms	106.27 ms	106.51 ms	106.34 ms
5-layered CLOS	107.63 ms	106.66 ms	107.42 ms	92.26 ms	37.76 ms	90.35 ms
B-Cube	5.36 ms	8.07 ms	5.66 ms	25.86 ms	13.72 ms	11.73 ms
Fat Tree	38.86 ms	38.85 ms	38.77 ms	19.95 ms	17.10 ms	30.71 ms
Grid	49.68 ms	47.61 ms	48.24 ms	39.11 ms	31.28 ms	43.19 ms
D-Cell	23.05 ms	23.15 ms	23.17 ms	18.26 ms	28.71 ms	23.27 ms
Bipartite	12.32 ms	12.29 ms	12.37 ms	6.04 ms	4.34 ms	9.47 ms
3-layered CLOS	12.13 ms	12.35 ms	12.29 ms	9.27 ms	15.23 ms	12.25 ms
Triangle	40.77 ms	44.84 ms	39.90 ms	40.61 ms	21.98 ms	37.62 ms
All	44.02 ms	44.45 ms	43.79 ms	39.74 ms	30.74 ms	40.55 ms

Lower is better

Table 3: Average of all concurrent bandwidth tests

Bandwidth (concurrent)	OSPF (single-path)	LDR (single-path)	MinMax (single-path)	LDR (multi-path)	ECMP (multi-path)	All
Binary Tree	210.5 kbps	210.1 kbps	209.8 kbps	209.6 kbps	209.4 kbps	209.9 kbps
5-layered CLOS	216.2 kbps	214.0 kbps	214.5 kbps	235.6 kbps	443.7 kbps	264.8 kbps
B-Cube	987.6 kbps	975.8 kbps	961.7 kbps	601.3 kbps	600.3 kbps	825.4 kbps
Fat Tree	378.3 kbps	378.2 kbps	379.5 kbps	535.0 kbps	591.3 kbps	452.5 kbps
Grid	474.3 kbps	446.7 kbps	462.1 kbps	481.7 kbps	518.5 kbps	476.7 kbps
D-Cell	562.5 kbps	563.0 kbps	561.9 kbps	663.6 kbps	542.4 kbps	578.7 kbps
Bipartite	706.7 kbps	706.5 kbps	706.9 kbps	1033.9 kbps	806.4 kbps	792.1 kbps
3-layered CLOS	706.8 kbps	706.3 kbps	706.7 kbps	981.9 kbps	857.9 kbps	791.9 kbps
Triangle	1168.0 kbps	1168.0 kbps	1170.0 kbps	1170.0 kbps	1102.9 kbps	1155.8 kbps
All	601.2 kbps	596.5 kbps	597.0 kbps	657.0 kbps	630.3 kbps	616.4 kbps

Higher is better

Table 4: Average routing algorithm's first response time on controller initialization

Routing response time	OSPF (single-path)	LDR (single-path)	MinMax (single-path)	LDR (multi-path)	ECMP (multi-path)	All
Binary Tree	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
5-layered CLOS	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
B-Cube	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
Fat Tree	20.9 s	19.7 s	21.2 s	8.8 s	14.9 s	17.1 s
Grid	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
D-Cell	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
Bipartite	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
3-layered CLOS	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
Triangle	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
All	2.3 s	2.2 s	2.4 s	1.0 s	1.7 s	1.9 s

Lower is better

Table 5: Deviation from average of all concurrent latency tests

Latency (concurrent)	OSPF (single-path)	LDR (single-path)	MinMax (single-path)	LDR (multi-path)	ECMP (multi-path)
Binary Tree	-0.24 ms	0.20 ms	0.40 ms	-0.68 ms	0.31 ms
5-layered CLOS	47.09 ms	46.22 ms	46.80 ms	9.75 ms	-149.86 ms
B-Cube	-16.98 ms	-11.11 ms	-15.96 ms	39.20 ms	4.84 ms
Fat Tree	38.23 ms	37.95 ms	38.62 ms	-50.19 ms	-64.61 ms
Grid	27.55 ms	14.04 ms	20.76 ms	-12.64 ms	-49.72 ms
D-Cell	7.27 ms	6.45 ms	7.00 ms	-11.00 ms	-9.72 ms
Bipartite	4.34 ms	4.36 ms	4.82 ms	-6.27 ms	-7.24 ms
3-layered CLOS	-3.47 ms	-4.00 ms	-4.07 ms	-10.27 ms	21.81 ms
Triangle	10.12 ms	12.22 ms	9.84 ms	10.83 ms	-43.01 ms
All	12.66 ms	11.81 ms	12.02 ms	-3.47 ms	-33.02 ms

Lower is better

Table 6: Deviation from average of all concurrent jitter tests

Jitter (concurrent)	OSPF (single-path)	LDR (single-path)	MinMax (single-path)	LDR (multi-path)	ECMP (multi-path)
Binary Tree	0.03 ms	-0.09 ms	-0.04 ms	-0.07 ms	0.17 ms
5-layered CLOS	17.29 ms	16.31 ms	17.08 ms	1.91 ms	-52.58 ms
B-Cube	-6.38 ms	-3.67 ms	-6.07 ms	14.13 ms	1.99 ms
Fat Tree	8.15 ms	8.14 ms	8.07 ms	-10.76 ms	-13.60 ms
Grid	6.49 ms	4.43 ms	5.06 ms	-4.07 ms	-11.90 ms
D-Cell	-0.22 ms	-0.12 ms	-0.10 ms	-5.01 ms	5.44 ms
Bipartite	2.85 ms	2.82 ms	2.89 ms	-3.44 ms	-5.13 ms
3-layered CLOS	-0.12 ms	0.09 ms	0.04 ms	-2.98 ms	2.98 ms
Triangle	3.15 ms	7.22 ms	2.28 ms	2.99 ms	-15.64 ms
All	3.47 ms	3.90 ms	3.24 ms	-0.81 ms	-9.81 ms

Lower is better

Table 7: Deviation from average of all concurrent bandwidth tests

Bandwidth (concurrent)	OSPF (single-path)	LDR (single-path)	MinMax (single-path)	LDR (multi-path)	ECMP (multi-path)
Binary Tree	0.6 kbps	0.2 kbps	-0.1 kbps	-0.2 kbps	-0.4 kbps
5-layered CLOS	-48.6 kbps	-50.8 kbps	-50.3 kbps	-29.2 kbps	178.9 kbps
B-Cube	162.2 kbps	150.5 kbps	136.4 kbps	-224.0 kbps	-225.0 kbps
Fat Tree	-74.2 kbps	-74.3 kbps	-72.9 kbps	82.6 kbps	138.8 kbps
Grid	-2.3 kbps	-30.0 kbps	-14.6 kbps	5.1 kbps	41.8 kbps
D-Cell	-16.2 kbps	-15.7 kbps	-16.8 kbps	84.9 kbps	-36.2 kbps
Bipartite	-85.4 kbps	-85.6 kbps	-85.2 kbps	241.8 kbps	14.3 kbps
3-layered CLOS	-85.1 kbps	-85.6 kbps	-85.2 kbps	189.9 kbps	66.0 kbps
Triangle	12.2 kbps	12.2 kbps	14.2 kbps	14.2 kbps	-52.9 kbps
All	-15.2 kbps	-19.9 kbps	-19.4 kbps	40.6 kbps	13.9 kbps

Higher is better

pact was perceived by watching the topologies being tested on our real-time visualizer, which took more time than 1 second to update the file that the visualizer refreshed 10 times per second. Such observation suggested that "MinMax (single-path)" is slower than "LDR (single-path)", which is slower than "LDR (multi-path)", which is slower than all the others, which were imperceptibly fast. This may be important for some applications with already-deployed topologies.

It's worth highlighting that every source value on table 4 was measured with 2 second intervals between every observation of the controller state. For example, if a routing algorithm took 4.1 seconds to get ready, the test would have registered it took 6 seconds. With some variability on the measurements over the 30 samples, the average should be able to smooth those minor imperfections.

After mentioning all the data we obtained and behaviors we observed, we have content to discuss on section 4.

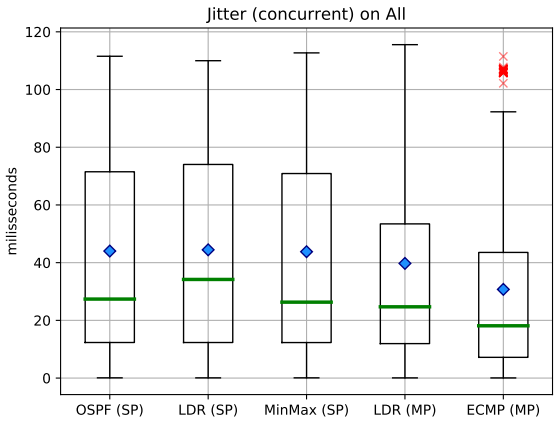

Figure 13: Box and whisker plot of all jitter samples from concurrent tests

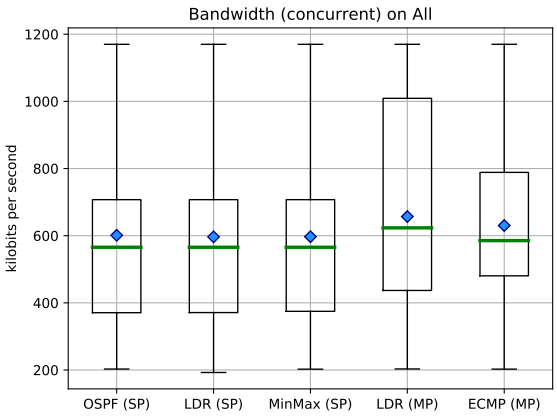

Figure 14: Box and whisker plot of all bandwidth samples from concurrent tests

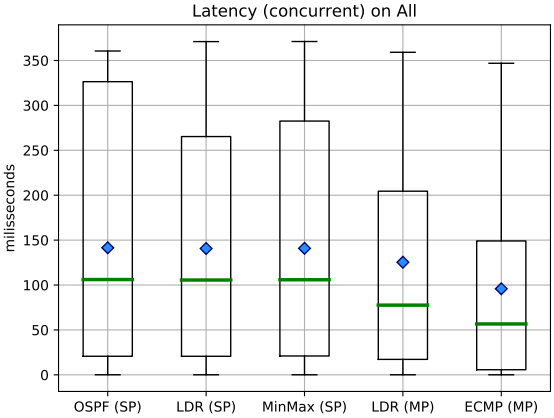

Figure 12: Box and whisker plot of all latency samples from concurrent tests

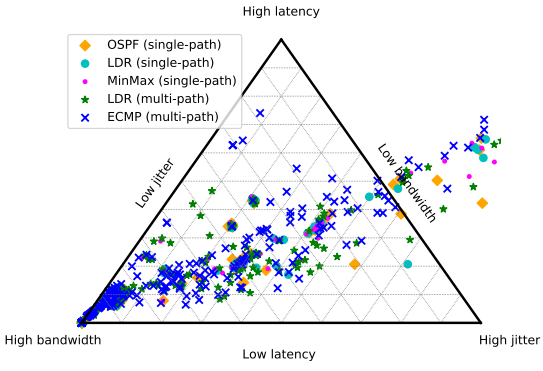

Figure 15: Ternary scatter plot of the union of all samples from all topologies

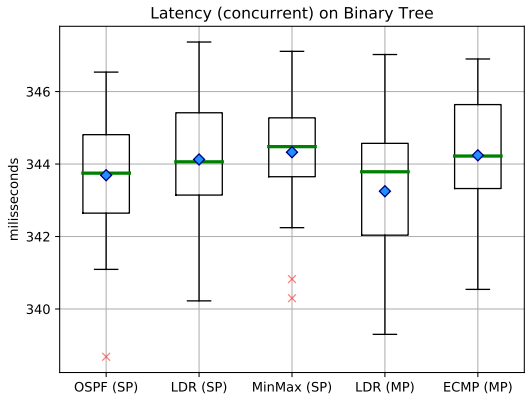

Figure 16: Box and whisker plot of latency samples from concurrent tests on “Binary Tree” topology

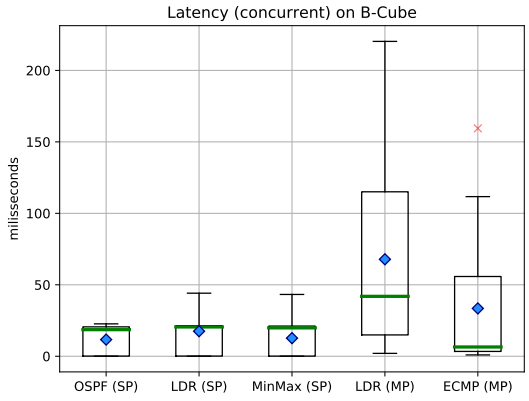

Figure 17: Box and whisker plot of latency samples from concurrent tests on “B-Cube” topology

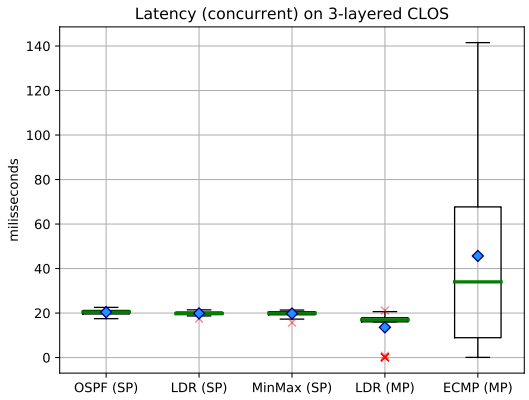

Figure 18: Box and whisker plot of latency samples from concurrent tests on “3-layered CLOS” topology

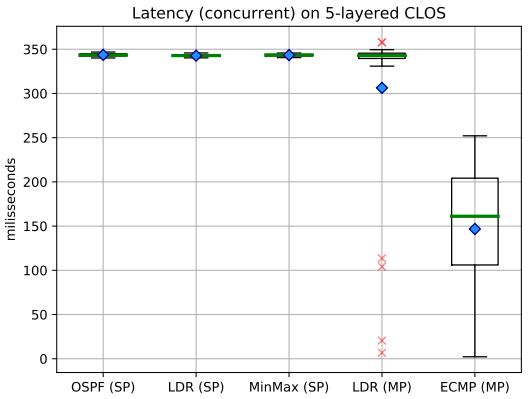

Figure 19: Box and whisker plot of latency samples from concurrent tests on “5-layered CLOS” topology

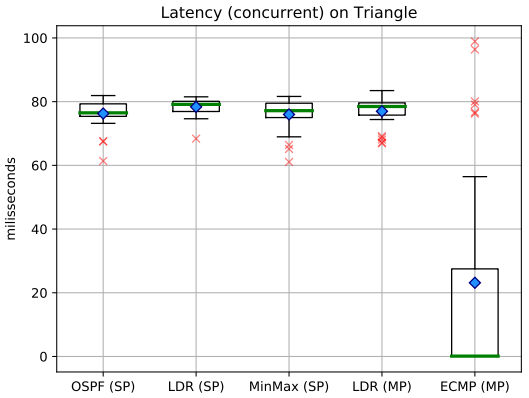

Figure 20: Box and whisker plot of latency samples from concurrent tests on “Triangle” topology

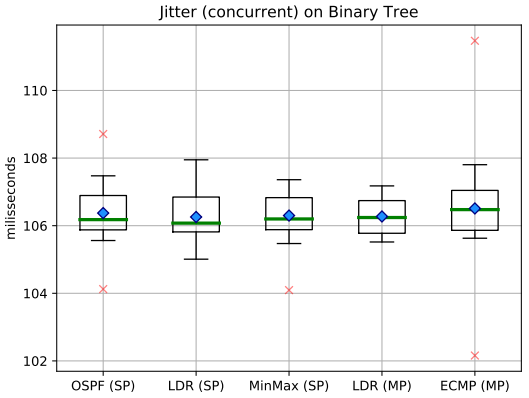

Figure 21: Box and whisker plot of jitter samples from concurrent tests on “Binary Tree” topology

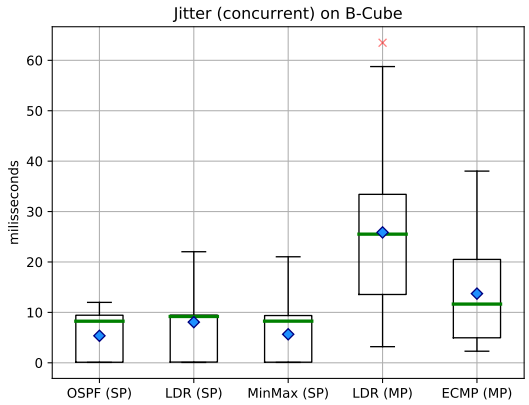

Figure 22: Box and whisker plot of jitter samples from concurrent tests on “B-Cube” topology

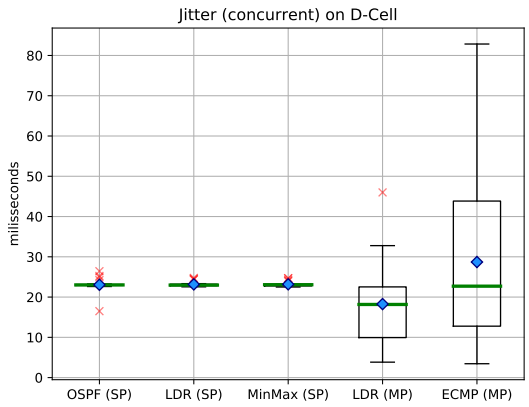

Figure 23: Box and whisker plot of jitter samples from concurrent tests on “D-Cell” topology

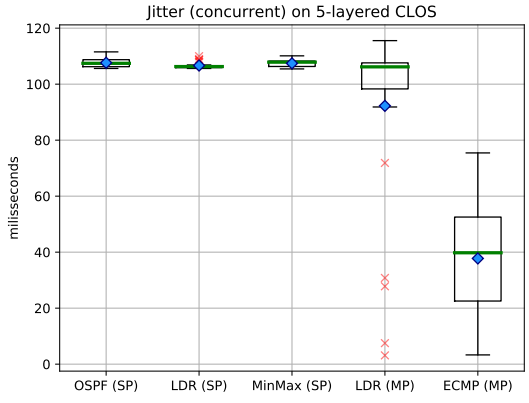

Figure 24: Box and whisker plot of jitter samples from concurrent tests on “5-layered CLOS” topology

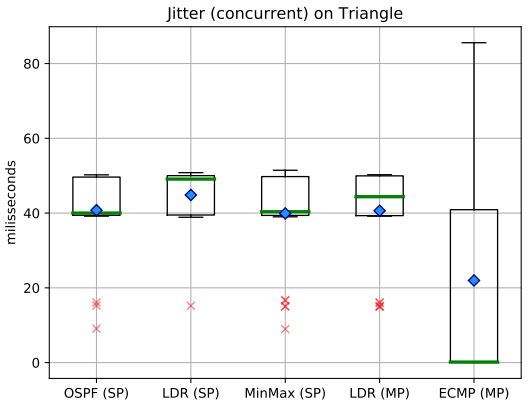

Figure 25: Box and whisker plot of jitter samples from concurrent tests on “Triangle” topology

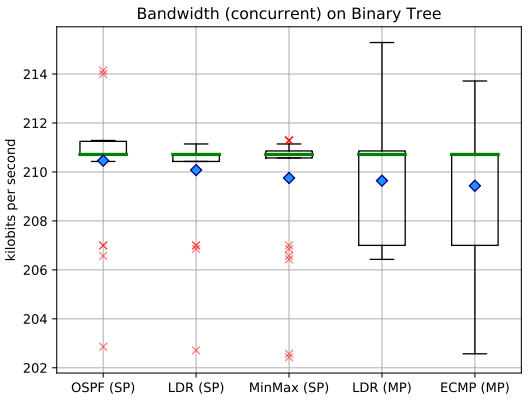

Figure 26: Box and whisker plot of bandwidth samples from concurrent tests on “Binary Tree” topology

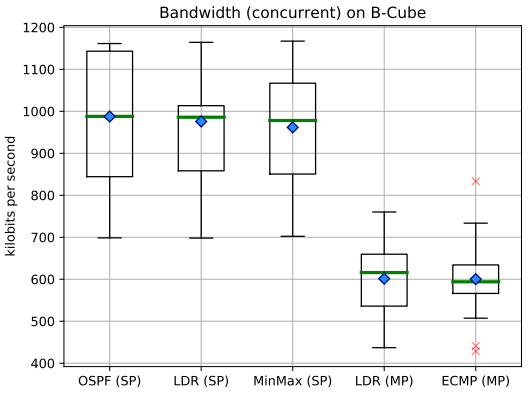

Figure 27: Box and whisker plot of bandwidth samples from concurrent tests on “B-Cube” topology

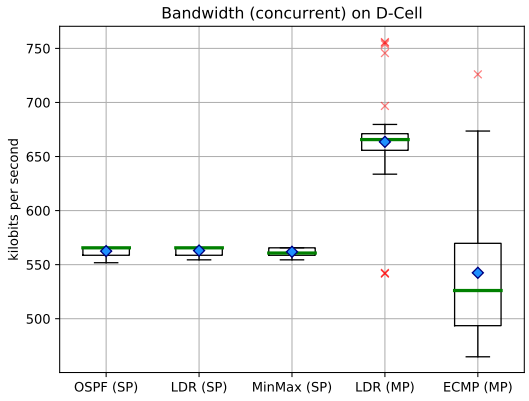

Figure 28: Box and whisker plot of bandwidth samples from concurrent tests on “D-Cell” topology

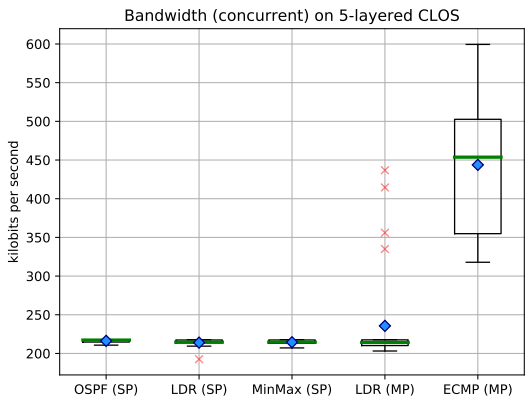

Figure 29: Box and whisker plot of bandwidth samples from concurrent tests on “5-layered CLOS” topology

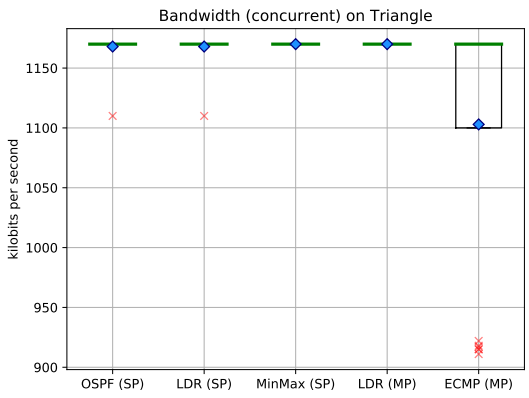

Figure 30: Box and whisker plot of bandwidth samples from concurrent tests on “Triangle” topology

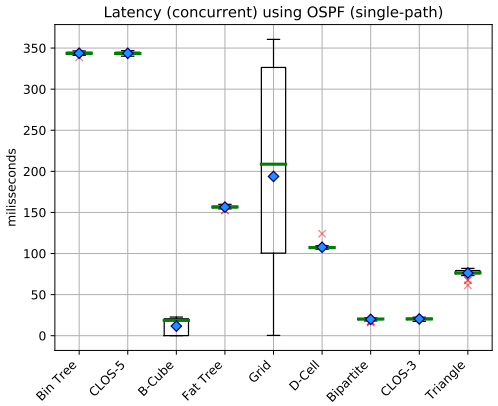

Figure 31: Box and whisker plot of latency samples from concurrent tests on “OSPF (single-path)” algorithm

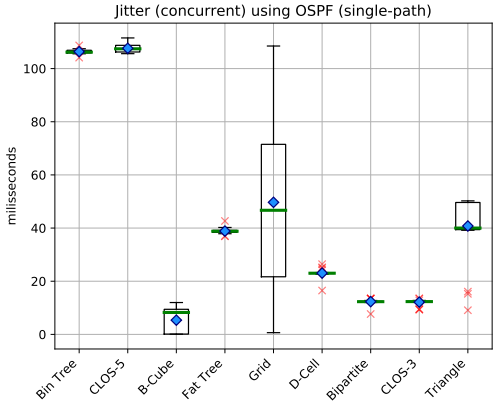

Figure 32: Box and whisker plot of jitter samples from concurrent tests on “OSPF (single-path)” algorithm

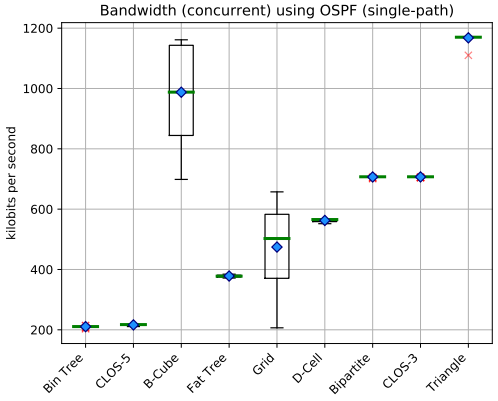

Figure 33: Box and whisker plot of bandwidth samples from concurrent tests on “OSPF (single-path)” algorithm

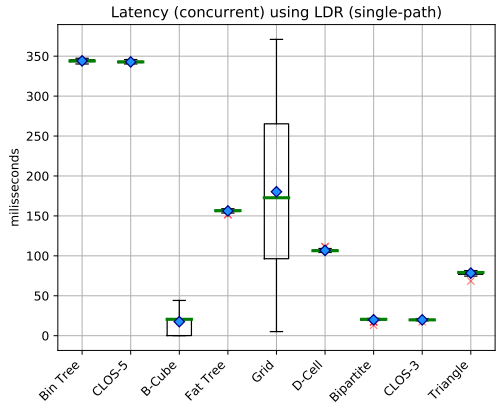

Figure 34: Box and whisker plot of latency samples from concurrent tests on “LDR (single-path)” algorithm

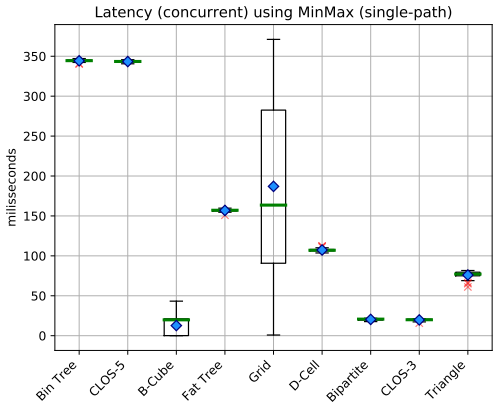

Figure 37: Box and whisker plot of latency samples from concurrent tests on “MinMax (single-path)” algorithm

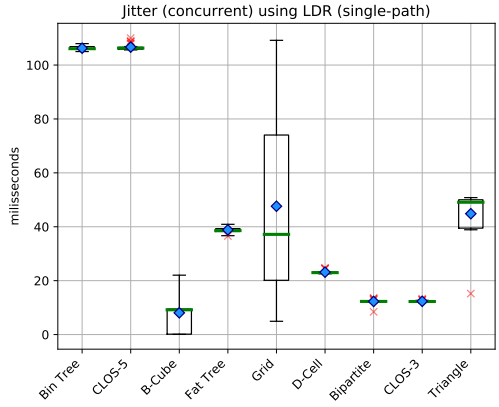

Figure 35: Box and whisker plot of jitter samples from concurrent tests on “LDR (single-path)” algorithm

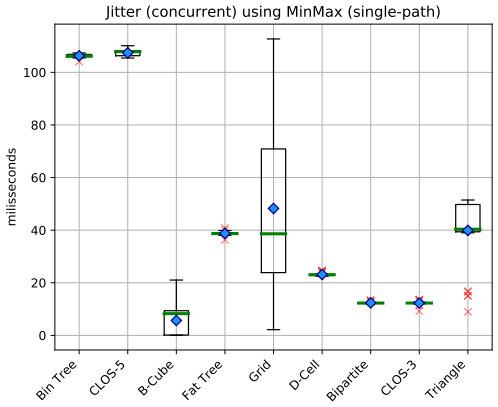

Figure 38: Box and whisker plot of jitter samples from concurrent tests on “MinMax (single-path)” algorithm

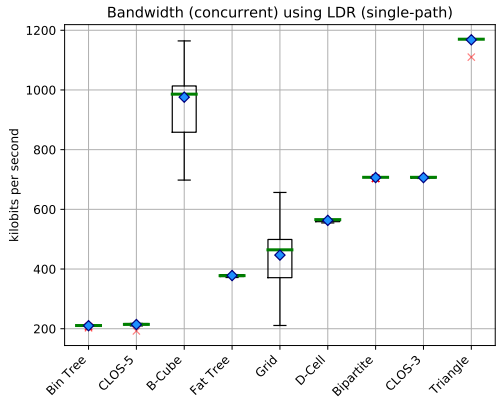

Figure 36: Box and whisker plot of bandwidth samples from concurrent tests on “LDR (single-path)” algorithm

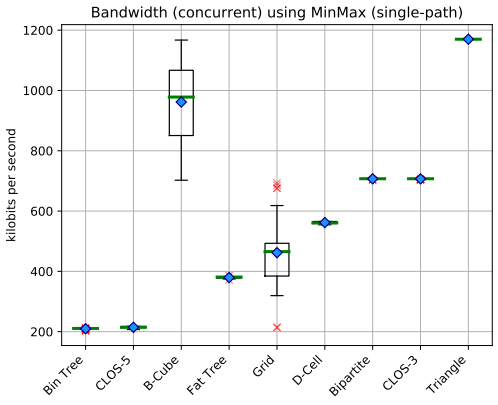

Figure 39: Box and whisker plot of bandwidth samples from concurrent tests on “MinMax (single-path)” algorithm

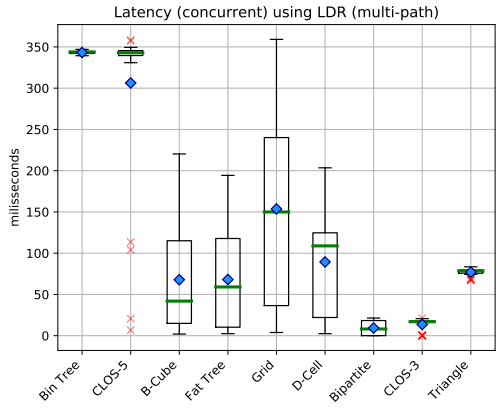

Figure 40: Box and whisker plot of latency samples from concurrent tests on “LDR (multi-path)” algorithm

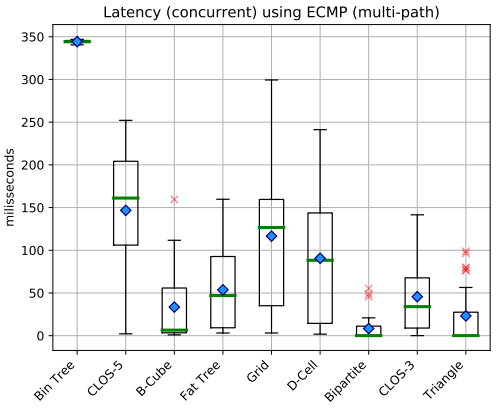

Figure 43: Box and whisker plot of latency samples from concurrent tests on “ECMP (multi-path)” algorithm

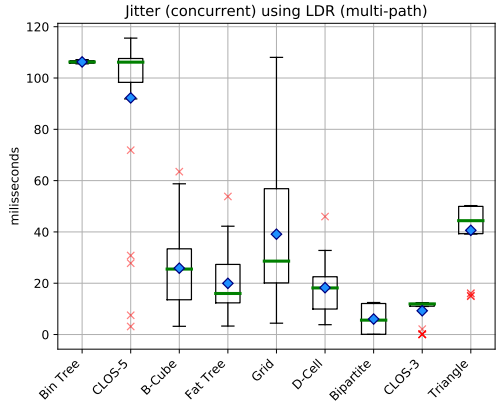

Figure 41: Box and whisker plot of jitter samples from concurrent tests on “LDR (multi-path)” algorithm

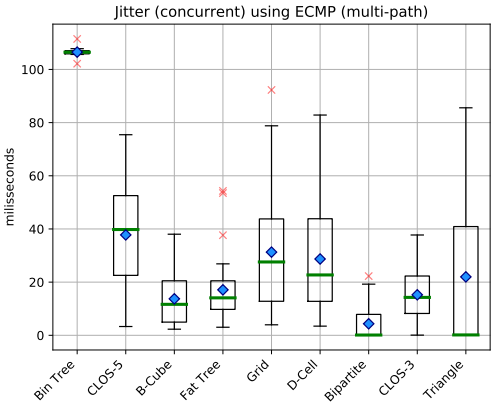

Figure 44: Box and whisker plot of jitter samples from concurrent tests on “ECMP (multi-path)” algorithm

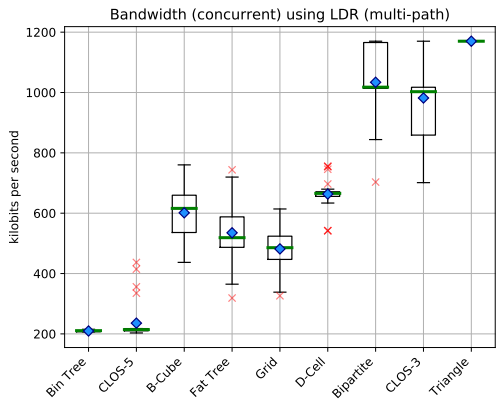

Figure 42: Box and whisker plot of bandwidth samples from concurrent tests on “LDR (multi-path)” algorithm

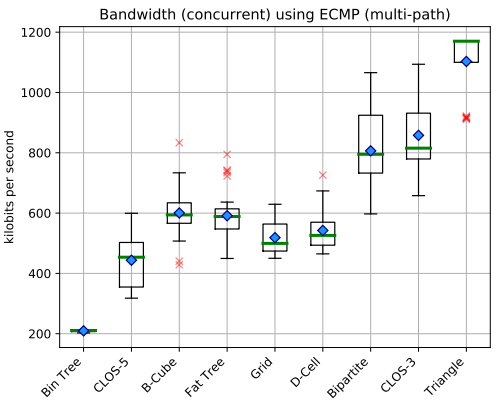

Figure 45: Box and whisker plot of bandwidth samples from concurrent tests on “ECMP (multi-path)” algorithm

4 Discussion

Before we start, we should remember that all paths from all topologies had $1mbps$ bandwidth, with "Triangle" being the only exception, but it had overcapacity. This is the ideal scenario for the ECMP we implemented; not for LDR. This suggests that another question very similar to Gvozdiev et al.'s question "Does routing influence topology?"[6, p. 99], but worded as "Does topology influence routing?". In no moment we expected LDR to be consistently faster, with lower latency and lower jitter than ECMP; and that's the strategy behind the tests: what attributes it gives up to become better at what else?

With S as the set of switches and H as the set of hosts to be routed, $|S| \times |\wp(H) \setminus \{(h,h) \forall h \in H\}|$ is the sum of the number of entries of every switch of the network. This number of entries can be feasible for testing few hosts on an emulated environment (our case). However, our implementation is not suitable for production.

From the data we collected, we see that "LDR (multi-path)" gained, on the average case, advantage on bandwidth (figure 14), but lost performance on latency and jitter (figures 12 and 13) over other algorithms. Even that this is not true for all cases, such as in figures 16, 18, 23, 29, this is the opposite of what we were expecting. To understand why this happened, we need to remember that Gvozdiev et al.'s proposed routing algorithm tries, first, to eliminate congestion, then choose the path with lower latency. Eliminating congestion on one overloaded link by sharing its load with multiple underused link sounds like an obvious solution. The problem starts when you have more demand than capacity. In such scenario, data shows that this algorithm will attempt keep link usage evenly balanced to keep bandwidth high, even if this means increasing latency and jitter. If this gets combined with an latency-dependent application that generates more traffic when the latency requirements aren't met (like an hypothetical video chat application that tries compensating a higher delay by increasing the frame rate), a scenario for thrashing appears. Such behavior, even if it happens under extremely rare and abnormal situations, will be more likely to happen over the time, as technology advances and new requirements are set, enabling DDoS[1] attacks to use even more bandwidth and new internet applications that make a more intense use of the available infrastructure. The upcoming 5G technology (which have already arrived on few cities) enables low-latency and near $1gbps$ mobile connections, while some parts of the infrastructure are still from 4G period; if adding such routing algorithm introduces the possibility of severe service degradation, it's understandable why ISPs wouldn't deploy it in real world. Finally, this goes the will expressed on the end of Gvozdiev et al.'s discussion section: "We hope our work can be a first step toward enabling the deployment of ISP topologies that are better than today's for the provision of low-latency service [...]"[6, p. 100].

From the data we collected "LDR (single-path)" exchanges, on the average case, a slightly higher jitter for the average latency variability between measurements to reduce, while keeping the bandwidth nearly the same as "OSPF (single-path)", however . As for an overloaded network, it's performing as expected: changing routes to attempt (in vain) to reduce latency. However, it didn't perform completely as expected. As soon as monitoring started inserting monitoring data, response time increased. On topologies with high path diversity, it meant path changing stopped right after the first monitoring. As the same thing happened to "MinMax (single-path)", it suffered from the same problems, but reducing latency while keeping jitter and bandwidth stable, but not as reduced as in "LDR (single-path)". As the routing decisions are time-dependent, this diminished their performance; specially because each test individual test took 10 seconds.

The figure 15 shows a dispersion of all collected data. How the data organizes itself on that ternary plane shows how a incomprehensible mess looking at everything at once is. Therefore the average used previously may be prone to oversimplification. A way to don't oversimplificate, in this case, is looking at what topologies each routing algorithm works the best and ignoring the cases where the topology doesn't work well with the algorithm.

Figures 31 to 45 brings, then, the required performance information to assist a data center architect to decide which topology and routing algorithm to deploy; "Principle" should be ignored, as it was intended to be a proof of concept; "D-Cell" should

[1] **D**istributed **D**enial **o**f **S**ervice

also be ignored, as having 5 cells with 4 hosts each and dividing the hosts in half for testing gave that topology the advantage of having 2 pairs doing bandwidth tests in which both client and server were inside the same cell. Looking at "OSPF (single-path)" (figures 31, 32 and 33), "B-Cube" brings high bandwidth at the lowest latency and jitter. Looking at "LDR (single-path)" (figures 34, 35 and 36) and "MinMax (single-path)" (figures 37, 38 and 39), "B-Cube" brings high bandwidth at the lowest latency and jitter, but these last two suffer from variability in a way that it can perform worse than other topologies; if having a stable latency and/or jitter is a requirement that makes a loss on bandwidth around 27% tolerable, "Bipartite" and "CLOS-3" are nearly-equivalent options. Looking at "LDR (multiple-path)" (figures 40, 41 and 42), "Bipartite" brings the lowest latency and jitter with the highest bandwidth. Looking at "ECMP (multiple-path)" (figures 43, 44 and 45), "Bipartite" brings the lowest latency and jitter, but a slightly higher bandwidth is found with "CLOS-3". As this didn't take into account cabling costs, generated heat by equipment, cooling, etc, other factors may influence the final choice, but great benefits are expected if the routing algorithm is tuned for the network it'll route and vice-versa.

Finally, we can answer the questions made on the introduction of the article. The compromises were made on the case when there is no alternative path available and all paths are overloaded, but that may not be the case if the topology is similar to a "3-layered CLOS". Such compromise, when present, becomes a limitation that makes it doesn't handle full-network overload well. We hope that this article left, at least, an useful comparative testing code artifact for routing algorithms, which can be reused on future comparative works.

5 Acknowledgments

We thank CAPES for the financial support given during this research, Magnos Martinello for pointing us towards CLOS topologies, Maxwell Monteiro for pointing us towards ECMP, BCube and DCell, Marin Vlastelica Pogančić for the comprehensible tutorial on HackerNoon on how to use PuLP and Wildan Maulana Syahidillah for the explanatory multi-path routing tutorial using Ryu.

References

[1] Marwan Al-shawi. *Clos (Spine & Leaf) Architecture – Overview*. [Online; accessed 08-Jul-2019]. Nov. 2018. URL: http://netdesignarena.com/index.php/2018/11/05/clos-spine-leaf-architecture-overview/.

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

[3] 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.

[4] Thomas A Caswell et al. *matplotlib/matplotlib: REL: v3.1.1*. July 2019. DOI: 10.5281/zenodo.3264781. URL: https://doi.org/10.5281/zenodo.3264781.

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

[6] 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.

[7] 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.

[8] 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.

[9] Marc et al. *marcharper/python-ternary: Version 1.0.6*. Apr. 2019. DOI: 10.5281/zenodo.2628066. URL: https://doi.org/10.5281/zenodo.2628066.

[10] Ádler Neves. *Mininet+Ryu Routing algorithm Comparator*. July 2019. DOI: 10.5281/zenodo.3273440. URL: https://doi.org/10.5281/zenodo.3273440.

[11] 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.