Cloud Computing Final Project

Liwei Cui, Yifeng Yin, Mou Zhang

May 1st, 2020

1 Introduction

Burst tolerance is one of the biggest bottlenecks of constructing modern cloud computing clusters. Burst flow is a sudden, uncertain and unpredictable spike in flow. It can cause a lot of trouble for the network, especially in a cluster of cloud computing.
Noticing the high capacity in Jellyfish, we tends to deal with burst traffic on Jellyfish with new routing algorithm. In Jellyfish paper, they have propose an routing algorithm - K shortest path(KSP) - to replace the traditional routing algorithm - ECMP. This KSP algorithm largely increases the path diversity and throughput.
We propose a new routing algorithm, K-non-overlapping paths(KNOP), to handle burst flows in Jellyfish. Comparing to the ECMP and KSP algorithm, our algorithm improves the overall path diversity and throughput of single source transmission in Jellyfish. In addition, we present a theoretical analysis of our algorithm, which proves that our algorithm is mathematically superior to ECMP and KSP.

2 Motivation

The high capacity mentioned in the Jellyfish paper inspires us to explore its potential to tackle burst flow. We want to maximize the average throughput at the expense of tolerable latency.

3 Algorithm

Instead of looking for the shortest path in ECMP, or the K shortest paths in the KSP, our algorithm KNOP looks for the K-shortest paths with no overlapping links. With this idea, we have largely avoided burst traffic and achieved higher path diversity and throughput.

4 Experiment

We put forward and implemented a new routing algorithm named K-non-overlapping Path(KNOP), which guarantees all links on paths from A to B have no overlapping. Figure 1 shows different paths under different algorithms in the same graph.

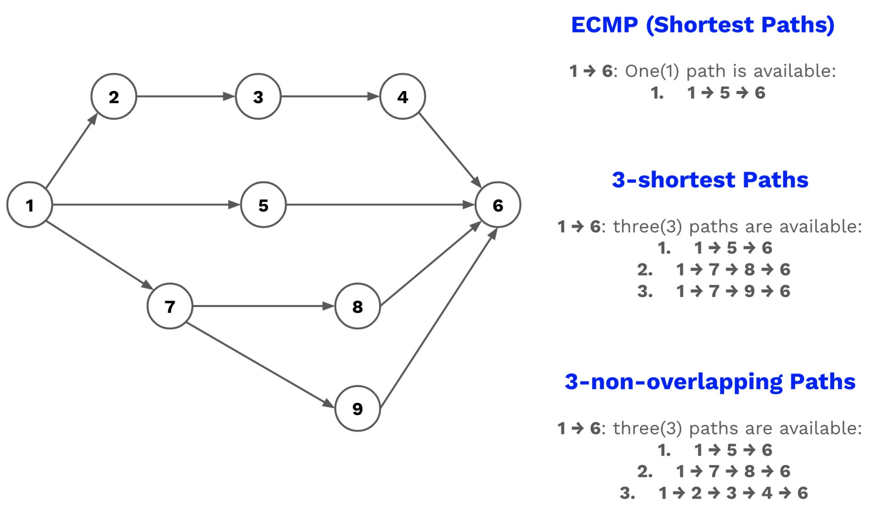

Figure 1: Different paths under different algorithms

Path Diversity

We implement out experiment based on Jellyfish Paper Figure 9: Inter-switch link's path count in ECMP, k-shortest-path(KSP) and k-non-overlapping(KNOP) routing for random permutation traffic at the server-level on a Jellyfish of 50 servers. For each link, we count the number of distinct paths it is on. Each network cable is considered as two links, one for each direction. The result in shown in Figure 2.

Average Throughput per Server

We conducted several experiments and averaged the results and marked the range of fluctuations. The results of the experiment fully illustrate the validity of our algorithm.

All experiments were conducted under the following conditions. Jellyfish network with 50 switches (8 ports connecting peer switches and 1 for host). Links between switches are 10 Mbps.

The result of our experiments in shown in Figure 3.

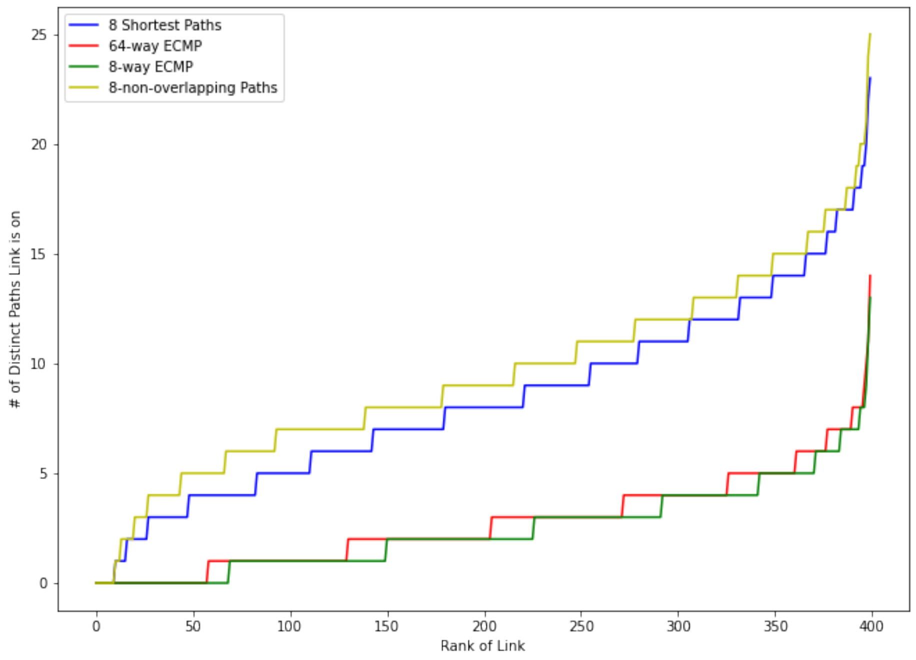

Figure 2: Path diversity in different algorithms

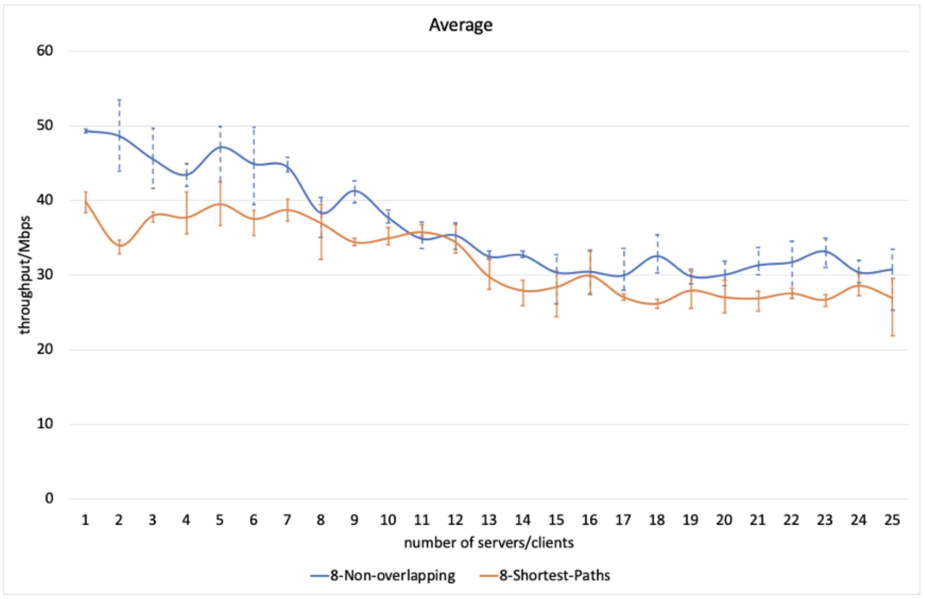

Figure 3: Average throughput of different algorithms

5 Mathematical Analysis

We abstract the Jellyfish topology in graph theory terms: We have n nodes, which are switches, and each node are linked to k random other nodes with edges, where k is the number of ports available for our switches of choice, and we assume $n > k$, which almost always happen in reality. Thus, we have ourselves a k-regular random graph with degree n. In this setting we assume that all switches have the same number of ports. In real life applications where we have different switches with different number of ports, we can set k as the average number of ports and achieve the same result. In this analysis, we denote the cost of a path as the number of hops, therefore we set all edges with a weight/cost of 1.
It is easy to see that there are kn edges in the graph and $\binom{n}{2}$ distinct pairs of nodes. Thus, for a random given pair of nodes, the probability that there's a direct edge connecting them is therefore:

$$P_d = \frac{kn}{\binom{n}{2}}$$

In other words, for any given node pair (u, v), the probability that the shortest path between them has cost 1 is P_d
Now, consider a random pair of nodes (u, v) and a number x where $0 \leq x \leq n-2$ be the number of switches other than u and v in a simple path with source u and destination v(as with an efficiency-oriented routing route, a simple path does not visit the same node twice. Thus the maximum number of switches to visit other than u, v is $n-2$); Then, there must be $x+2$ nodes in total including u, v involved in any simple path of cost $x+1$. For any simple path of cost $x+1$, there are $\binom{n-2}{x}x!$ distinct arrangements to pick x nodes from the $n-2$ reserve nodes(those that are not u or v). The rational here is, there are $\binom{n-2}{x}$ ways to pick x nodes from the $n-2$ reserve nodes without order, and we add ordering by multiply the result by $x!$. For example, if $x = 0$, the number of switches between u and v is 0, and the cost of this path is $x + 1 = 1$ which means u and v are directly connected.
We mentioned the probability that there's a direct edge connecting a pair of nodes is P_d, therefore, all of these $\binom{n-2}{x}x!$ simple paths with cost $x + 1$ has a probability $(P_d)^{x+1}$ to exist. The rational here is, for a path to exist, there must be an edge between each of the consecutive nodes; there are $x + 2$ nodes in the path, which gives us $x + 1$ edges, and each edge has a probability of P_d to exist independently.
Putting what we have together, by partitioning the sample space with the number of switches within the path(denoted as x), and the life-saving linearity of expectation, we have the expected number of simple paths $E[\#P]$ between two arbitrary nodes as:

$$E[\#P] = \sum_{x=0}^{n-2} x! \binom{n-2}{x} (P_d)^{x+1}$$

The above can be interpreted as, when x loop through all possible number of switches within a path, there are estimated $x!\binom{n-2}{x}(P_d)^{x+1}$ paths with cost $x+1$. that is, by setting the set of all possible number of switches in the path as X, we partition the sample space by conditioning on X; and by the law of total expectation:

$$E[\#P] = \sum_X E[\#of\ path|X = x]P(X = x)$$

Where the $x!\binom{n-2}{x}$ is $E[\#of\ path|X = x]$ and $(P_d)^{x+1}$ is $P(X = x)$.
Because within the summation, these paths have a cost of $x+1$, we have the expected total cost of all paths between any two arbitrary nodes as:

$$E[TC] = \sum_{x=0}^{n-2} x!\binom{n-2}{x}(x+1)(P_d)^{x+1}$$

The expected cost of any path between any two nodes is therefore the expected total cost divided by the expected number of paths:

$$E[C] = \frac{E[TC]}{E[\#P]}$$

Unfortunately the only direct way to know the number of shortest paths between any two nodes is to run an algorithm(like Dijkstra's Algorithm) on a set network, and there's no viable way(yet) to give a exact mathematical expectation of the number of shortest paths along with their costs. Thus here, we take a indirect approach by providing a(somewhat loose) bound.
By[3], we have the diameter of the graph $d(G)$ as:

$$d_1 \leq d(G) \leq d_2$$

Where d_1 and d_2 are the maximal integer that satisfies, for a small $\epsilon > 0$:

$$k(k-1)^{d_1-2} \leq (2-\epsilon)n\log n$$

$$k(k-1)^{d_2-1} \leq (2+\epsilon)n\log n$$

Which means, Generally Speaking, we have the diameter:

$$d(G) \leq \left\lceil \log_{k-1}\left[\frac{(2+\epsilon)n\log n}{k} + 1\right]\right\rceil$$

We have now 3 bounds on the length of the shortest paths between any 2 node: namely the smallest possible value is 1 with probability $P_d = \frac{kn}{\binom{n}{2}}$; the maximum length is $d(G)$; and since we have the expectation of the number of simple path already, by Markov's inequality, we have:

$$P(Length \geq l) \leq \frac{E[C]}{l}$$

Thus, the length of the shortest path sl between any two nodes is bounded by:

$$1 \leq sl \leq Min(d(G), l)$$

Where l is the biggest integer that satisfies:

$$\frac{E[C]}{l} \leq 1$$

Which is basically the smallest integer less than $E[C]$.
By assuming any path who's cost is within this rage might be the shortest path, and by the same method we calculated the expected number of path, we have the(loose) upper bound of the number of shortest paths $\#SP$ between any two nodes as:

$$[\#SP]_{UB} = \sum_{x=0}^{Min(d(G),l)-1} x! \binom{Min(d(G), l) - 1}{x} (P_d)^{x+1}$$

Do recall that the inequality $1 \leq sl \leq Min(d(G), l)$ is a bound for path length but we are partitioning the sample space with the number of switches in the path, which is always the path length minus one!
The upper bound of the total cost of shortest paths is:

$$[SPTC]_{UB} = \sum_{x=0}^{Min(d(G),l)-1} x! \binom{Min(d(G), l) - 1}{x} (x+1)(P_d)^{x+1}$$

Similarly, an approximation($\frac{upper\ bound}{upper\ bound}$ is not an upper bound but an approximation!) of the average cost of shortest paths is:

$$E[SPC]_{approx} = \frac{[SPTC_{UB}]}{[\#SP_{UB}]}$$

As we can easily see, for a fixed k, $E[SPC]_{approx}$ grows very sub-linearly with respect to n. Which means, for those paths that are just a little longer than the shortest path, their approximate expected cost is $E[SPC]_{approx} + \hat{o}(n)$.

Armed with our calculations, we now have the tools to compare and contrast our algorithm against ECMP and KSP(k-shortest paths). consider all links are with bandwidth b and of the same cost(time takes to transmit):
If the number of shortest path $\#SP$ is less than k, the approximate 'useful' throughput T of ECMP is:

$$T \approx \frac{\#SP * b}{E[SPC]_{approx}} - c$$

Where c is the congestion factor. That is, if we are routing TCP data, with two or more links of bandwidth b converges at a single link with bandwidth b,

there might be congestion, causing re-transmission and therefore lowering the 'useful' working throughput(we see re-transmission as 'not useful'). We denote the bandwidth wasted by re-transmission due to congestion as c.
For our algorithm, we have:

$$T \approx \frac{\#SP * b}{E[SPC]_{approx}} + \frac{(k - \#SP) * b}{E[SPC]_{approx} + \hat{o}(n)}$$

That is, we take $\#SP$ shortest paths and $(k - \#SP)$ 'not so shortest' paths, better than ECMP.
For KSP, we have:

$$T \approx \frac{\#SP * b}{E[SPC]_{approx}} + \frac{(k - \#SP) * b}{E[SPC]_{approx} + \hat{o}(n)} - c$$

We have the same base throughput as KSP but due to possible choke point created by overlapping, the 'Useful' throughput of KSP must be subtracted by the congestion factor. in this case, our algorithm is at least as good as KSP(c could be 0).

If the number of shortest path $\#SP$ is larger or equal to k, because there are only k links directly attached to the destination node, thus, no matter how many paths taken, we will always have to traverse these k links to get to the destination. Thus, for both ECMP and KSP, we have the useful throughput as:

$$T \approx \frac{k * b}{E[SPC]_{approx}} - c$$

For our algorithm, due to non-overlapping, we can run all k links that attached with the destination node in full useful bandwidth, thus we have:

$$T \approx \frac{k * b}{E[SPC]_{UB}}$$

In conclusion, our algorithm behaves at least as good as ECMP and KSP in all cases.

6 Conclusion

We propose a new routing algorithm, K-non-overlapping paths(KNOP), to boost the single source transmission throughput of cloud computing cluster based on Jellyfish. After rigorous mathematical proof and experimentation, we determined that this new routing algorithm KNOP is superior to the original ECMP and KSP algorithms. We believe this new algorithm will play a role in the construction of data centers and the transmission of data in the future.

Contribution

Liwei Cui: Reproduced Jellyfish network; Implemented and benchmarked routing algorithms; Analyzed and visualized data
Mou Zhang: Performed experiments under various circumstances; Migrated the environment to the Cloud
Yifeng Yin: Provided mathematical analysis

Acknowledgement

Mininet library for network emulation
Pox library for OpenFlow controller
RipL library for simplifying data center code
RipL-POX library for controller built on RipL
Austin Poore and Tommy Fan's open-source code for inspiration to reproduce Jellyfish and k-shortest-paths routing

Github Repository

https://github.com/Lw-Cui/Non-overlapping-Path-in-Jellyfish

Reference

[1]Singla A, Hong C, Popa L, Godfrey PB. Jellyfish: Networking data centers randomly. 2012:225-238.
[2]Singh A, Ong J, Agarwal A, et al. Jupiter rising: A decade of Clos topologies and centralized control in google's datacenter network. ACM SIGCOMM computer communication review. 2015;45(4):183-197.
[3]Bollobás, B., & De La Vega, W. F. (1982). The diameter of random regular graphs. Combinatorica, 2(2), 125-134.