Analysis of Jellyfish Topology over Traffic Load

Muzamil Yahia
Dept. Computer Science
University of Hawaii at Manoa
muzamil@hawaii.edu

***Abstract*—Traffic sharing through Equal Cost Multiple Path (ECMP) is a common practice in datacenters to exploit the available resources according to the load demand. Being able to incrementally expand these networks is essential for industry to adapt to increasing demands when needed. Having fixed topology of the network may hinder the traffic sharing and network expansion. Jellyfish topology introduced in 2012 by Ankit Singla et.al. to increases the throughout of available network and allows for incremental expansion of network [3].**

In this report, we study three types of Jellyfish topologies, analyse their performance over heavy and half heavy traffic load using ECMP protocol. We notice that random Jellyfish topology offers better if not equivalent performance than Incrementing Jellyfish and Bipartite Jellyfish in both models of heavy and half heavy traffic loads.

***Index Terms*—Networking, distributed algorithms, Network Topology, Traffic Load, ECMP, Routing Protocol.**

I. Introduction

The rapid growth of online services (with heavy streaming, video-games, and virtual worlds) places huge demand over the underlying network. Being able to meet these demands is essential to remain in the market. Traffic Engineering using Equal Cost Multiple Path (ECMP) which splits equally the load between the shortest paths between two nodes in the network helps overall to reduce the traffic load over links and increase the throughput [4]. Despite the limitations of ECMP, it still remains popular in industry due to other features like being simple to implement, with predictable shortest paths, low protocol overhead and scalable to additional nodes in network [2].

Scalable low overhead software solutions to reduce traffic problem need to be accompanied with scalable low overhead hardware architecture. Jellyfish networks introduced by Ankit Singla et.al. offer such needed architecture for the network topology [3]. Its ability to reduce traffic relies on the flexibility of re-routing the network on demands. As we witnessing new programmable switches, flexible network architecture can be build with such devices with little overhead see [1]. In this report, we study three types of Jellyfish network: Random Jellyfish, Incrementing Jelly fish and Bipartite Jellyfish. We compare the compare the performance of these network over heavy load, and lesser load using metrics of maximum throughout over ECMP protocol.

II. Jellyfish topologies

Jellyfish is an undirected graph with random links between its vertices. Given $|V| = n$, and max degree d. Jellyfish topology can be constructed in three different ways:

A. Random Jellyfish

In this topology, we start with an empty set of edges E. We then choose two random nodes u and v from V and connect them through a link $e = (u, v)$ and update $E = E \cup \{e\}$. We keep repeating this process till almost every node in V has degree d.

Algorithm 1

1: **function** RandomJellyFish($V, E = \emptyset$)
2: let $degreeSeq$ be an array of size $|V|$
3: **while** $\exists u \in V$ with $deg(u) < d$ **do**
4: choose at random $v \neq u \in V$
5: **if** $deg(v) < d$ **then**
6: $E = E \cup \{(u, v)\}$
7: **else**
8: **if** $deg(u) == d - 2$ **then**
9: find $(v, w) \in E$ such that $v, w \notin N(u)$
10: $E = E - \{(v, w)\}$
11: $E = E \cup \{(u, v), (u, w)\}$

This approach will populate the set E with set of edges till it can no longer find a node with degree less than d. In practice, this might take a while when the network becomes saturated, i.e. too many nodes with degree d. We can limit the number of trials in line 5 and 9 by (10000 trials) and we get almost a full network of degree d. We notice that there are (0–14) nodes with degree less than d in all sizes of V implemented.

B. Incrementing Jellyfish

In this topology, we create a complete graph $G_{d+1} = (V_{d+1}, E_{d+1})$ with $V_{d+1} = \{0, \ldots, d\}$ and $E_{d+1} = \{(i, j) \mid 0 \leq i \neq j \leq d\}$. We then add node $u \in \{d + 1, \ldots, |V| - 1\}$ in sorted order by unlinking $d/2$ random edges $(v, w) \in E_{u-1}$ and connecting (u, v) and (v, w) as shown below:

Algorithm 2

1: **function** IncrementingJellyFish($V, E \neq \emptyset$)
2: let $V_{d+1} = \{0, \ldots, d\}$
3: let $E_{d+1} = \{(i, j) \mid 0 \leq i \neq j \leq d\}$
4: **for** $u = d + 1$ to $|V| - 1$ **do**
5: $V_u = V_{u-1} \cup \{u\}$
6: **for** $i = 1$ to d with step 2 **do**
7: choose $(v, w) \in E_{u-1}$ at random
8: $E_u = E_{u-1} \cup \{(u, v), (v, w)\} - \{(v, w)\}$
9: $E = E_{|V|-1}$

In this topology, we guarantee there every node $v \in V$ has degree d.

C. Bipartite Jellyfish

In this topology, we create a complete bipartite graph from V by choosing $L_d = \{0, \ldots, d-1\}$ and $R_d = \{|V| - d, \ldots, |V| - 1\}$ with $E_d = \{(i,j) \mid 0 \leq i \leq d-1, |V| - d, \leq j \leq |V| - 1\}$. We then choose two random nodes u, v not in $V_d = L_d \cup R_d$, link them together, and unlink $d/2$ links in E_d and reconnect their adjacent nodes with u and v as shown in the algorithm below:

Algorithm 3

function BIPARTITEJELLYFISH($V, E \neq \emptyset$)
  let $V_d = \{0, \ldots, d-1\} \cup \{|V| - d, \ldots, |V| - 1\}$
  let $E_d = \{(i,j) \mid 0 \leq i \leq d-1, |V|-d, \leq j \leq |V|-1\}$
  for $u = d$ to $|V|/2 - 1$ **do**
    $V_u = V_{u-1} \cup \{u, |V| - u - 1\}$
    $E_u = E_{u-1} \cup \{(u,v)\}$
    for $i = 1$ to d **do**
      choose $(v,w) \in E_{u-1}$ at random
      $E_u = E_{u-1} - \{(v,w)\}$
      $E_u = E_u \cup \{(u,w), (v, |V| - u - 1)\}$
    $E_u = E_u \cup \{(u,v)\}$
  $E = E_{|V|/2}$

Similar to Incrementing Jellyfish, Bipartite Jellyfish has degree d for every node $v \in V$.

III. Equal Cost MultiPath (ECMP)

In a network setting, where the number of hops is more significant than delay over link (fibre optics for example), shortest path based on number of hops plays an important role in determining how to route elements with least delay. However, choosing pure shortest path may congest certain network with the load of the entire network. Therefore a way to share load between shortest possible paths (paths with least amount of hops from source to target) is needed. ECMP does this type of routing as follows:

Algorithm 4

function ECMP($G = (V, E)$, trafficMatrix)
  for any $u \neq v \in V$ **do** ▷ Double loop
    traf$[u][v] = 0$ ▷ initial traffic is zero
    load $= trafficMatrix[u][v]$
    dist$[u, v] = dijikstra(G, u, v)$
    pAdj$[u, v] = \{w \in V \mid dist(u, w) + dist(w, v) = dist(u, v)\}$
    INCREASECAP(G, traf, u, v, load)
  return traffic

Algorithm 5

function INCREASECAP(G, traf, u, v, pAdj, load)
  if traf$[u][v] \neq 0$ or $u \neq v$ **then**
    nload $=$ traffic$[u][v]/pAdj[u][v].size$
    for $w \in pAdj[u][v]$ **do**
      traffic$[u][w] +=$ nload
      traffic$[w][u] +=$ nload
      INCREASECAP(G, traf, w, v, pAdj, nload) ▷ BFS traversal

This function works in recursive manner where we increase the traffic of each link while doing BFS toward neighbours with equal distance from target.

Throughput is defined as follows:

$$Throughput = \frac{1}{max\{traf[u][v] \mid u, v \in V\}} \quad (1)$$

IV. Performance Analysis

We use the measure of throughout in Equation 1 above, and we test with traffic matrix where $trafficMatrix[u, v] = 1$ if $u \neq v$. This traffic matrix simulates a heavy loaded network where everyone is speaking to everyone. We also test with another traffic matrix where half of the nodes are sending to $|V|/2$ random nodes. This simulates a less heavy network. The resulted performance is shown in the following figure

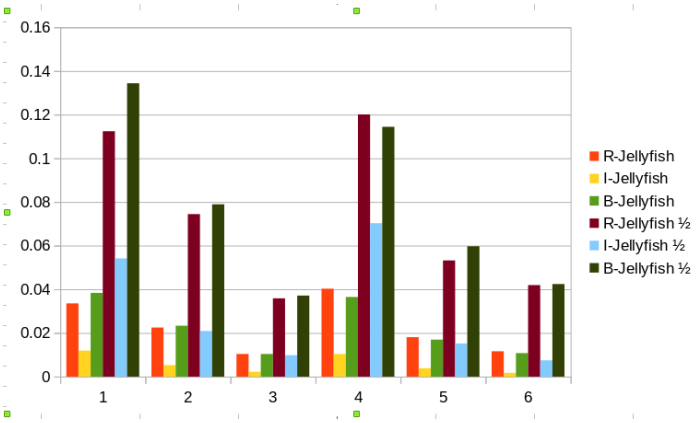

Fig. 1. Bar chart that shows the throughput performance over fully congested network and half congested network

We notice that when the network is congested, Random Jellyfish offers better if not an equivalent performance than other types of Jellyfish topologies. We believe the effect of missing out few links from Random Jellyfish (due to the way of constructing them) let Bipartite Jellyfish topology do slighter better than Random Jellyfish. Similar behaviour observed with half congested network. We notice also increase in throughput almost by two thirds when we half the congestion.

V. Conclusion

ECMP is a stable and scalable method to split load equally across shortest paths. Combining ECMP with flexible network architecture like Jellyfish helps to balance the load across

the network. In heavy congested network where every node is speaking to any other node at rate r, Random Jellyfish overcomes Incrementing Jellyfish and Bipartite Jellyfish in throughput performance. Such flexibility requires a flexible way to re-route the network and may require programmable switches or routers. We observe that this model has a limitation where we assume a router knows all nodes in the network. It is interesting to consider ECMP routing with Jellyfish topology where we know only a partial links in the network as the case in real life. Can Jellyfish perform better with biased traffic matrix?

REFERENCES

[1] Ran Ben-Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. Efficient measurement on programmable switches using probabilistic recirculation. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 313–323. IEEE, 2018.

[2] M. Chiesa, G. Kindler, and M. Schapira. Traffic engineering with equal-cost-multipath: An algorithmic perspective. *IEEE/ACM Transactions on Networking*, 25(2):779–792, 2017.

[3] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 225–238, San Jose, CA, 2012. USENIX.

[4] D. Thaler and C. Hopps. Rfc2991: Multipath issues in unicast and multicast next-hop selection, 2000.