Jellyfish: Networking Data Centers Randomly

Singla, Hong, Popa, and Brighten Godfrey.
Originally presented in NSDI 2012.

presented by Harrison Chandler

Overview

- Motivation
- Prior work
- Jellyfish
- Evaluation
- Cabling
- Conclusion

Overview

- Motivation
- Prior work
- Jellyfish
- Evaluation
- Cabling
- Conclusion

Motivation

Industry desires incremental expansion in data centers

- Facebook "adding capacity on a daily basis"
- 84% of enterprises surveyed planned on expanding data centers
- Ice-Cube (SGI) and EcoPod (HP) advertise as incrementally expandable

Data centers need to maintain high throughput

Prior work

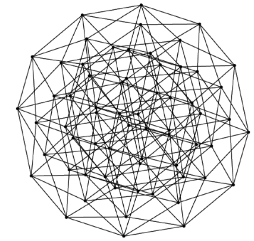
Highly structured topologies

- Clos/Fat-tree
  - LEGUP: finds optimal upgrades for Clos networks; needs free ports to exist in network

Random topologies

- Scafida: builds scale-free network; not evaluated for incremental deployment
- Small-World Data Center: uses regular lattice, still structured

Structure constrains expansion



Coarse design points

- Hypercube: 2^k switches
- 3-level Fat-Tree: $5k^2/4$ switches

3-Level Fat-Tree, commodity switches

- 24-port switch -> 3,456 servers
- 32-port switch -> 8,192 servers
- 48-port switch -> 27,648 servers

Workarounds exist, but unclear how to maintain structure incrementally

- Overutilize network? Uneven / constrained bandwidth
- Overprovision for later? Wasted investment

Slide contents from Chi-Yao Hong, “Jellyfish: Networking Data Centers Randomly.” https://www.usenix.org/conference/nsdi12/jellyfish-networking-data-centers-randomly

Overview

- Motivation
- Prior work
- Jellyfish
- Evaluation
- Cabling
- Conclusion

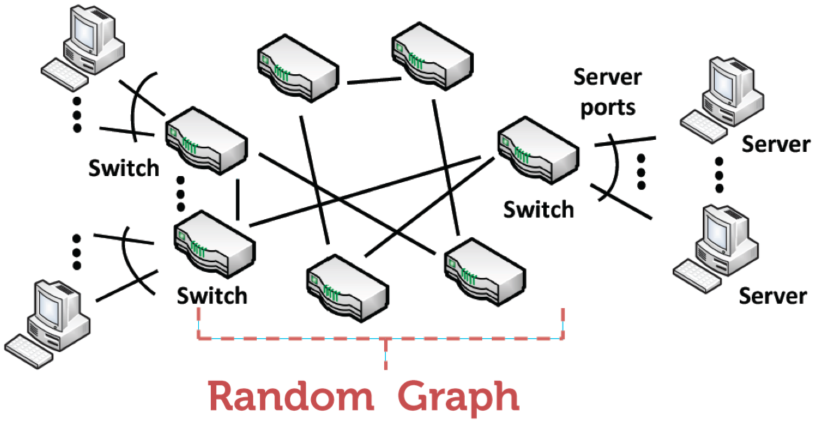
Jellyfish

Solves incremental expansion problem by eliminating structure

Builds a random graph between top-of-rack (ToR) switches

- switch i has k_i ports
- use r_i ports to connect to other ToR switches
- use k_i - r_i ports to connect to servers
- every switch will have degree r_i

Jellyfish topology

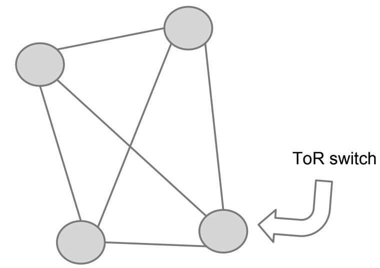


Slide contents from Chi-Yao Hong, “Jellyfish: Networking Data Centers Randomly.” https://www.usenix.org/conference/nsdi12/jellyfish-networking-data-centers-randomly

Constructing Jellyfish

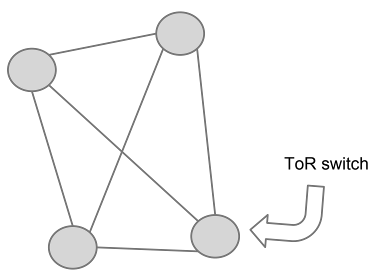
Pick a random pair of switches with open ports and connect them

Continue until no further links can be added



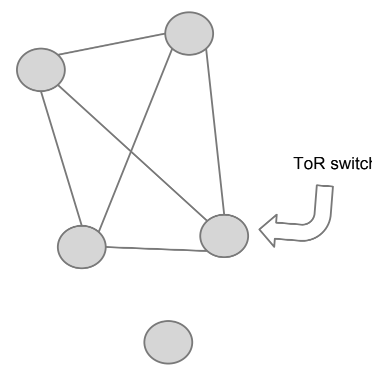
Constructing Jellyfish

If a switch exists with two or more free ports, break an existing link and insert two new links



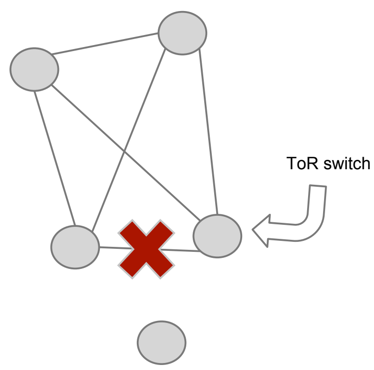
Constructing Jellyfish

If a switch exists with two or more free ports, break an existing link and insert two new links



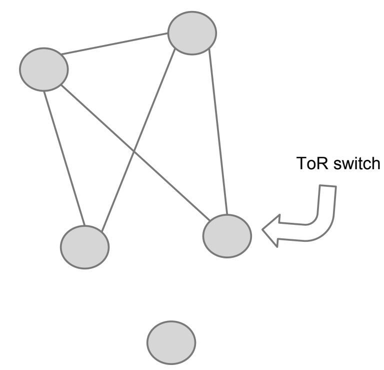
Constructing Jellyfish

If a switch exists with two or more free ports, break an existing link and insert two new links



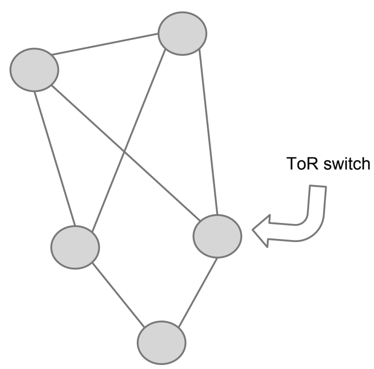
Constructing Jellyfish

If a switch exists with two or more free ports, break an existing link and insert two new links



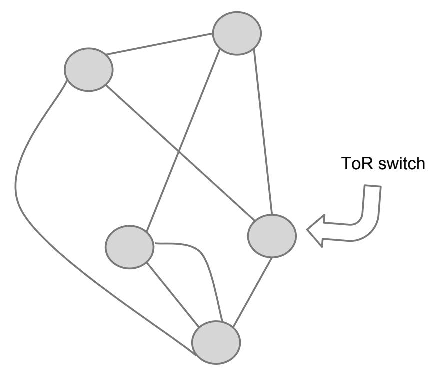
Constructing Jellyfish

If a switch exists with two or more free ports, break an existing link and insert two new links



Constructing Jellyfish

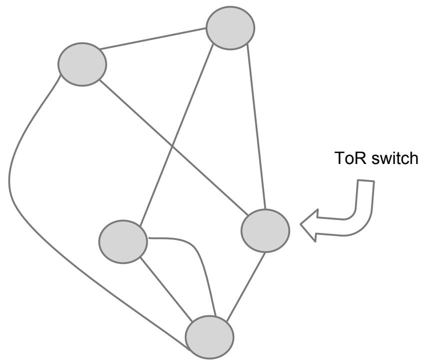
If a switch exists with two or more free ports, break an existing link and insert two new links



Constructing Jellyfish

If a switch exists with two or more free ports, break an existing link and insert two new links

This also works for incremental expansion



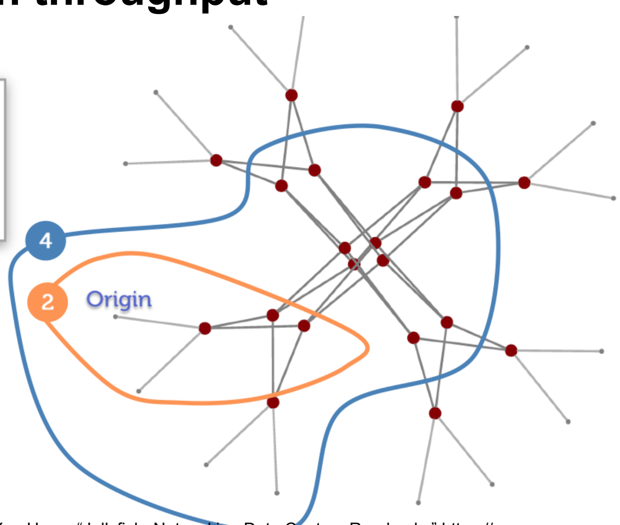
Jellyfish throughput

Intuition: end-to-end throughput inversely proportional to resources used to deliver data

=> Minimizing path lengths will improve throughput

Jellyfish throughput

Fat-tree:
3 of 15
in < 6 hops

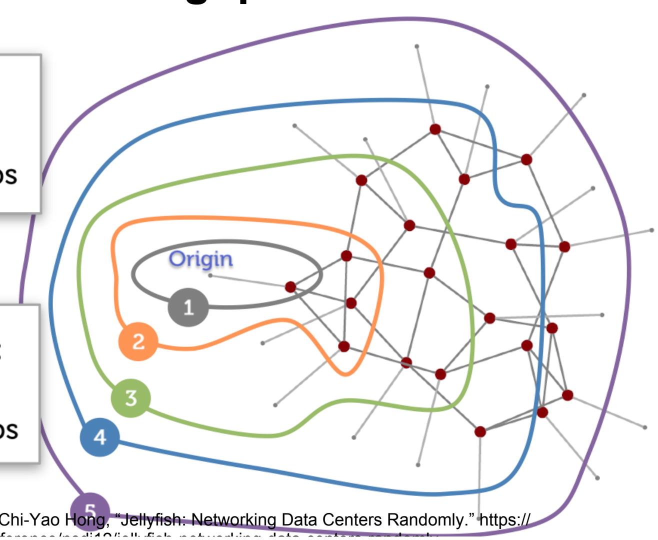


Slide contents from Chi-Yao Hong, “Jellyfish: Networking Data Centers Randomly.” https://

Jellyfish throughput

Fat-tree:
3 of 15
in < 6 hops

Jellyfish:
11 of 15
in < 6 hops



Slide contents from Chi-Yao Hong, “Jellyfish: Networking Data Centers Randomly.” https://

Overview

- Motivation
- Prior work
- Jellyfish
- **Evaluation**
- Cabling
- Conclusion

Evaluation

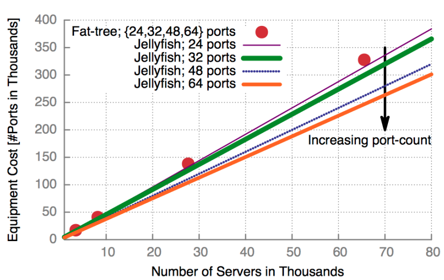
Jellyfish evaluated in two parts

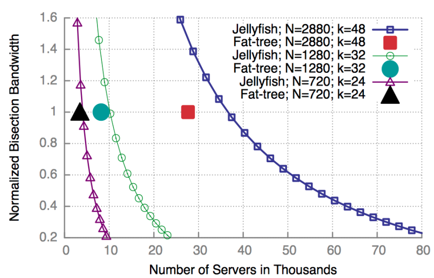
1) Topology: analyze raw capabilities of the network, assume optimal routing

2) Routing/Congestion control: analyze impact of routing choices

Random permutation traffic used for all throughput tests

Evaluation

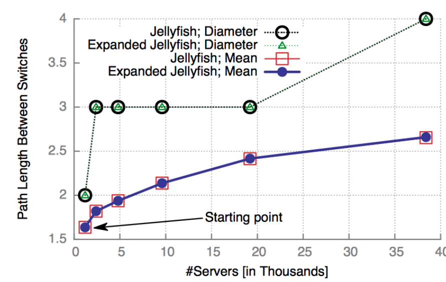


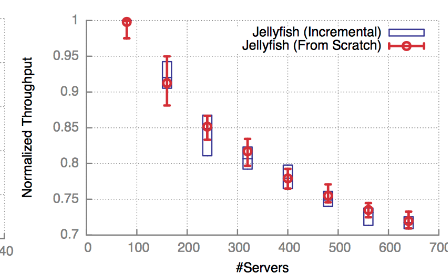


1) Jellyfish can connect more servers at lower cost

2) Jellyfish can provide higher bisection bandwidths at same network cost

Evaluation





Incrementally expanding Jellyfish is just as effective as building the network from scratch

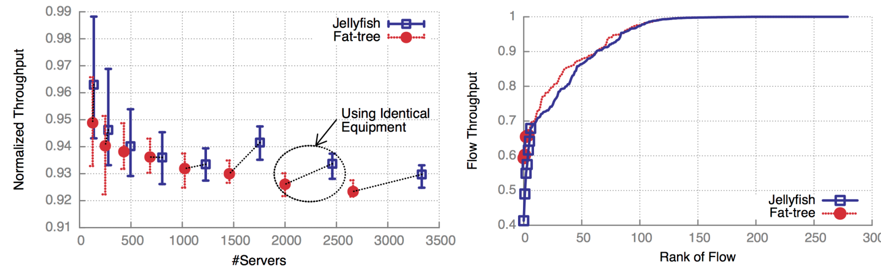
Evaluation

Routing: tested with ECMP and k shortest paths

Congestion control: tested with TCP and multipath TCP

Congestion control	Fat-tree (686 svrs) ECMP	Jellyfish (780 svrs) ECMP	Jellyfish (780 svrs) 8-shortest paths
TCP 1 flow	48.0%	57.9%	48.3%
TCP 8 flows	92.2%	73.9%	92.3%
MPTCP 8 subflows	93.6%	76.4%	95.1%

Evaluation



1) Jellyfish has better throughput than Fat-tree, even with sub-optimal routing

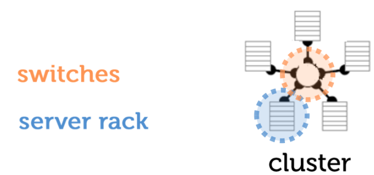
2) Both networks exhibit flow fairness

Overview

- Motivation
- Prior work
- Jellyfish
- Evaluation
- Cabling
- Conclusion

Cabling

Place switch racks in the physical center, aggregate cables run between switches and server racks



Large data centers: have multiple clusters, localize some links within a cluster

- only slightly reduces throughput

Overview

- Motivation
- Prior work
- Jellyfish
- Evaluation
- Cabling
- Conclusion

Strengths

Simple method of building network topology

Adding additional capacity to the data center seems very easy

Topology analysis was thorough

Weaknesses

Evaluation doesn't account for traffic locality (biases results in favor of Jellyfish)

No comparison to Scafida

k shortest paths routing implementation