Jellyfish: Networking Data Centers Randomly

Ankit Singla, Chi-Yao Hong, Lucian Popa, P. Brighten Godfrey

University of Illinois at Urbana–Champaign

Presented by *Xiang Wang*

December 12, 2012

Outline

- Introduction
- Jellyfish Topology
- Jellyfish Topology Properties
- Routing & Congestion Control
- Physical Construction and Cabling
- Conclusion

Outline

- Introduction
- Jellyfish Topology
- Jellyfish Topology Properties
- Routing & Congestion Control
- Physical Construction and Cabling
- Conclusion

Existing Design

- Bandwidth bottlenecks:
  - Restrict workload placement
  - QoS, SLA
- Structure datacenter network:
  - FatTree, Dcell, Bcube …
- Problem:

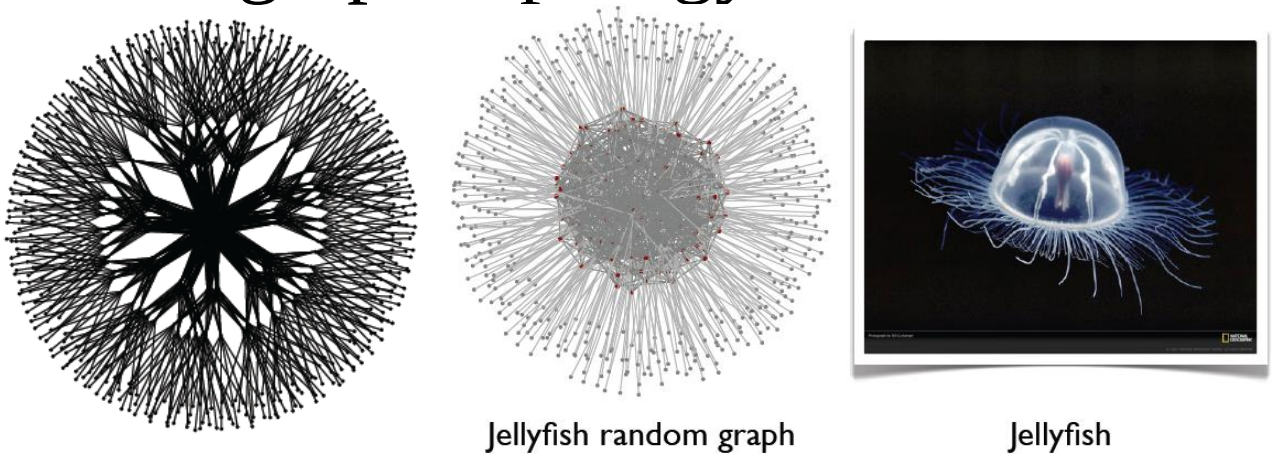
  Incremental expansion

Restriction

- Take FatTree as example:
  - Coarse design space (24:3456, 32:8192, 48:27648)
  - Replacing every switch
- Some Compromise:
  - Allow oversubscribe
  - Port reservation
- *Structure* hinders incremental expansion

Jellyfish

- A random graph topology network fabric



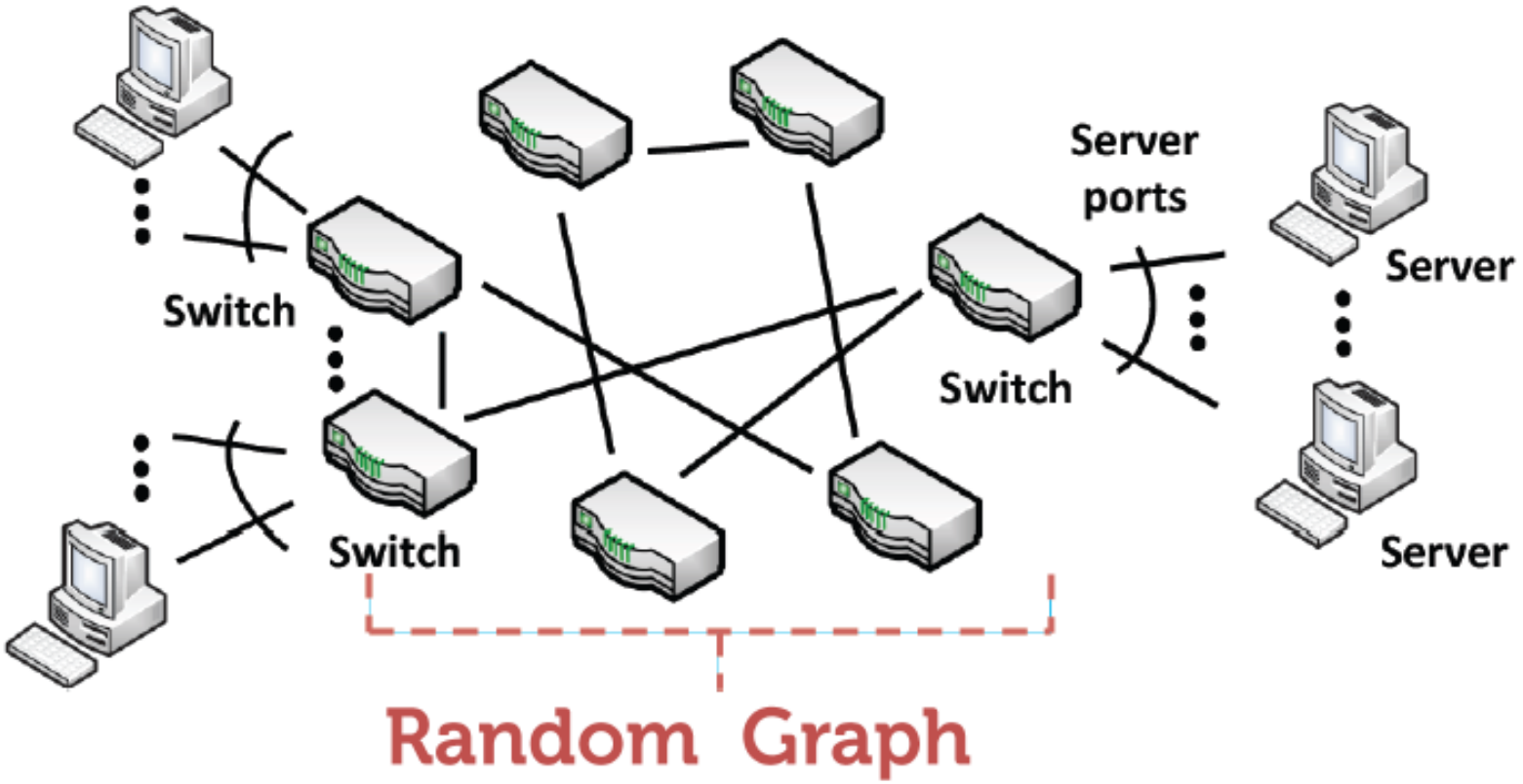
Jellyfish random graph Jellyfish

  - Connection among ToR switches
  - Racks of servers or switches to improve capacity
  - Support heterogeneity
  - Allow arbitrary-size network construction
  - More efficient than FatTree

Outline

- Introduction
- Jellyfish Topology
- Jellyfish Topology Properties
- Routing & Congestion Control
- Physical Construction and Cabling
- Conclusion

Jellyfish Topology

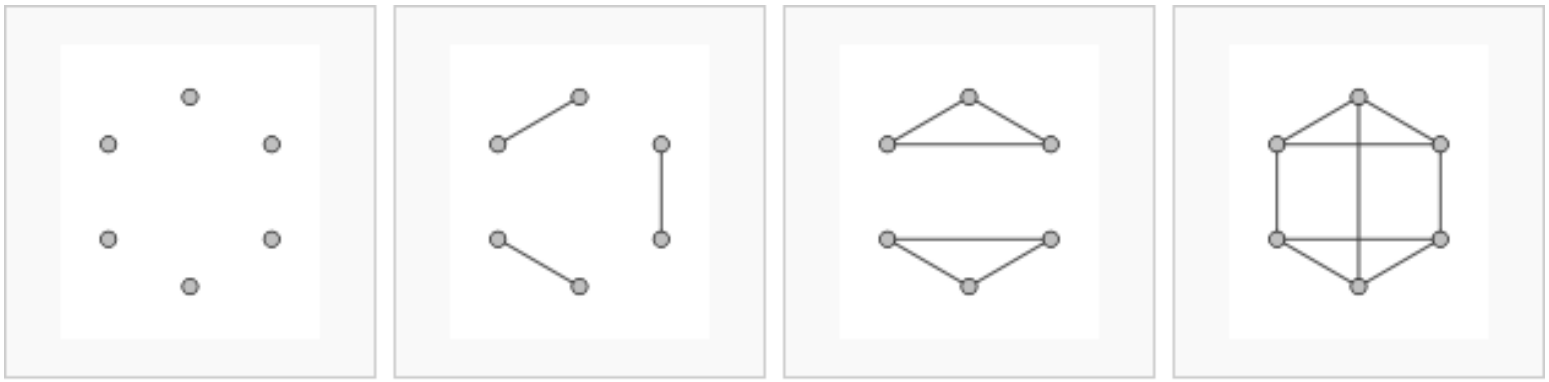


Randomly selected
from all valid graphs

Switches are nodes

Topology Construction

- Based on Random Regular Graph
  - Regular Graph
    - A graph where each vertex has the same degree, i.e., every vertex has the same number of neighbors.



0-regular graph | 1-regular graph | 2-regular graph | 3-regular graph

Topology Construction

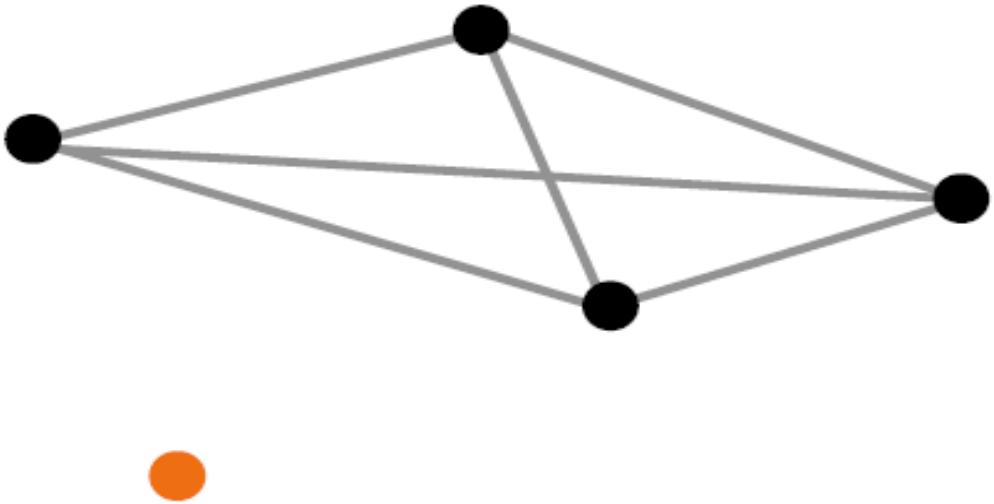
- Based on Random Regular Graph
  - Random Regular Graph
    - Let $0 < d < n$ be two positive integers and $S(n,d)$ be the set of all simple d-regular graphs on n vertices.
    - A random regular graph $G(n,d)$ is obtained by sampling from $S(n,d)$ with respect to the uniform distribution.
    - Complex problem in graph theory

Topology Construction

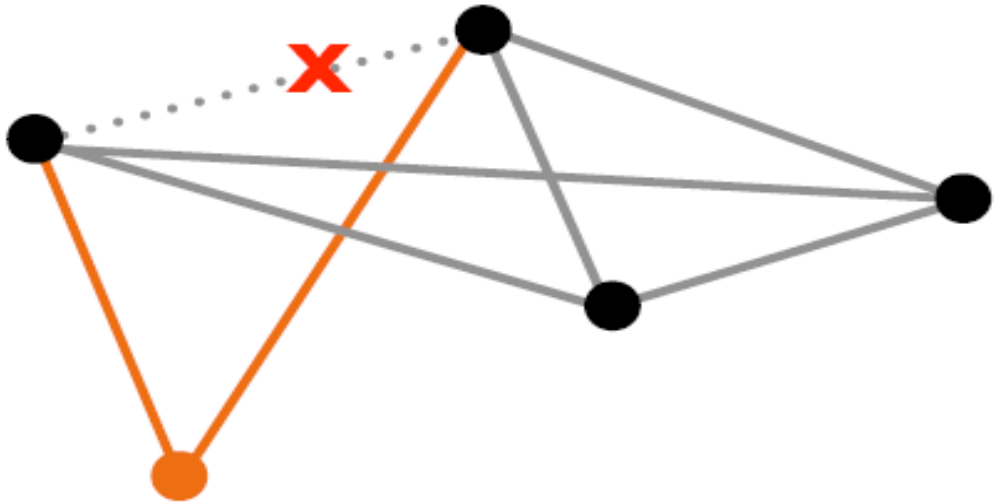
- Pick a random pair of switches with free ports
- Join them with a link
- Repeat until no further links can be added

- If a switch remains with >=2 free ports (p1,p2)
- Removing a random existing link (x, y)
- Adding links (p1, x) and (p2, x)

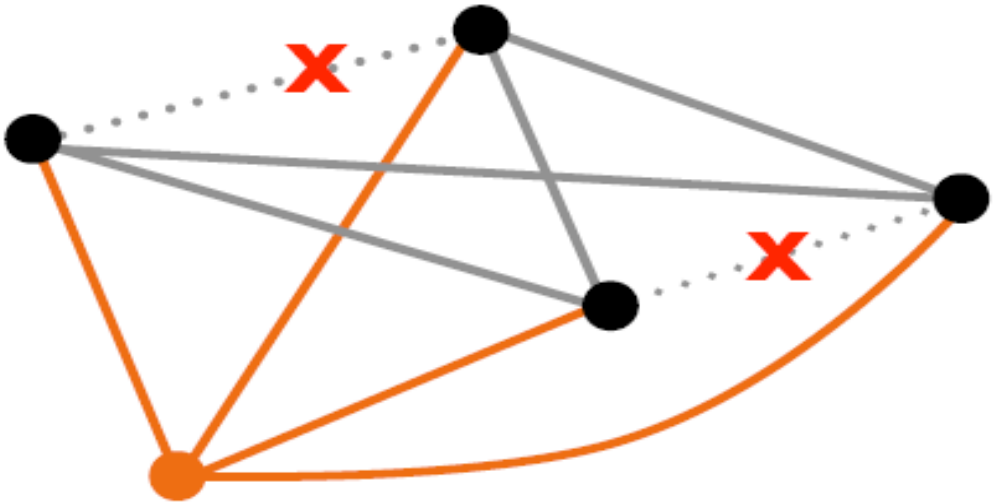
Topology Construction



Topology Construction



Topology Construction



Same procedure for initial construction
and incremental expansion

Jellyfish Topology

- Does Jellyfish meet design goal?
  - High bandwidth
  - Flexibility of incremental expansion

- The end-to-end throughput depends
  - the capacity of the network
  - *amount of network capacity consumed to deliver each byte, i.e., average path length*

Jellyfish Topology

If we fully utilize all available capacity ...

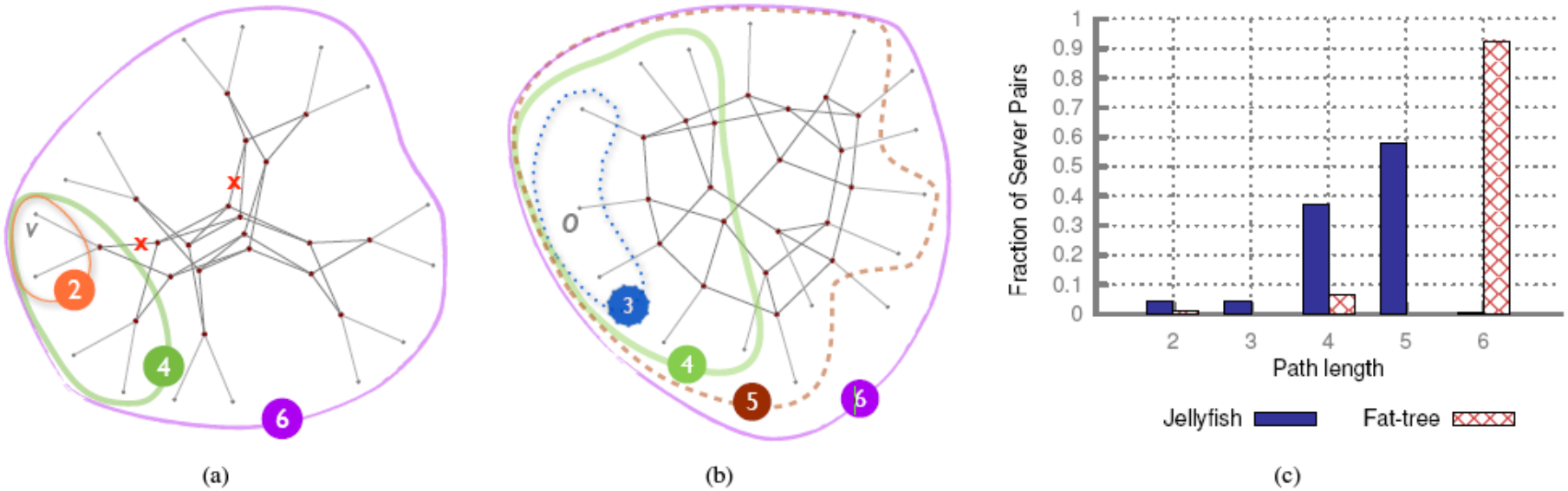
$$\sum_{\forall \text{links}} \text{capacity}(link)$$

$$\text{Number of flows at full throughput (1 Gbps)} = \frac{\text{total network capacity}}{\text{capacity used per flow}}$$

$$= 1\ \text{Gbps} \bullet \text{mean path length}$$

Mission:
minimize average path length

Jellyfish Topology



(a) (b) (c)

Figure 1: *Random graphs have high throughput because they have low average path length, and therefore do less work to deliver each packet. (a): Fat-tree with 16 servers and 20 four-port switches. (b): Jellyfish with identical equipment. The servers are leaf nodes; switches are interior nodes. Each 'concentric' ring contains servers reachable from any server v in the fat-tree, and an arbitrary server o in Jellyfish, within the number of hops in the marked label. Jellyfish can reach many more servers in few hops because in the fat tree, many edges (like those marked "X") are redundant from a path-length perspective. (c): Path length distribution between servers for a 686-server Jellyfish (drawn from 10 trials) and same-equipment fat-tree.*

Outline

- Introduction
- Jellyfish Topology
- Jellyfish Topology Properties
- Routing & Congestion Control
- Physical Construction and Cabling
- Conclusion

Topology Properties

- Some key findings
  - Support 27% more servers
  - Identical throughput and path len when expanding
  - Shorter average path, max path is same or lower
  - Highly failure resilient

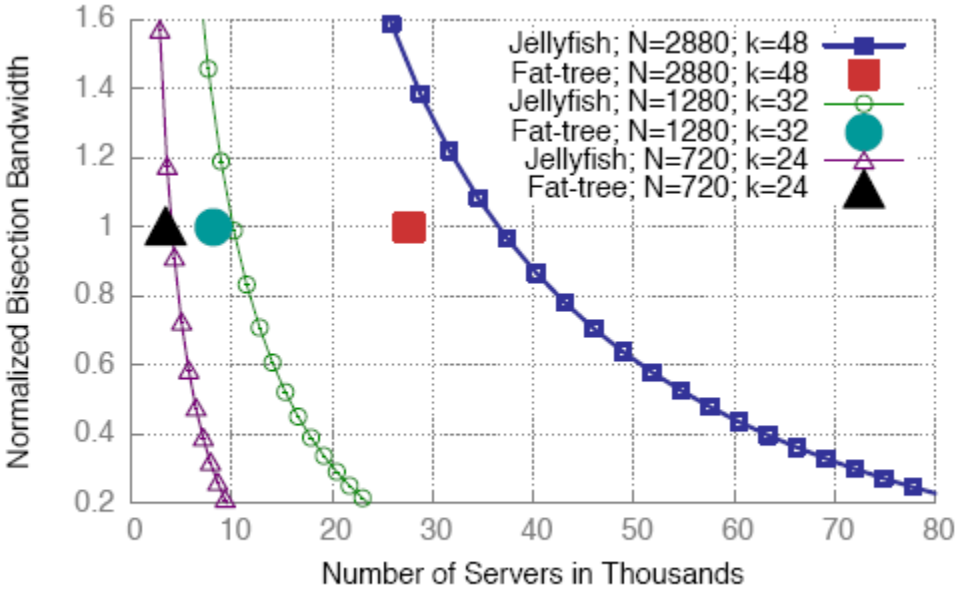
- Evaluation methodology
  - Capacity: calculations of theoretical bounds
  - Throughput: random permutation traffic

Efficiency

- Bisection Bandwidth vs. Fat-Tree
  - The worst-case bandwidth spanning any two equal-size partitions of a network
  - Fat-Tree: calculate from its parameters
  - Jellyfish: calculate the lower bound of *Bollobas*
  - Normalize by dividing bisection bandwidth by the total line-rate bandwidth of the servers in one partition

Efficiency

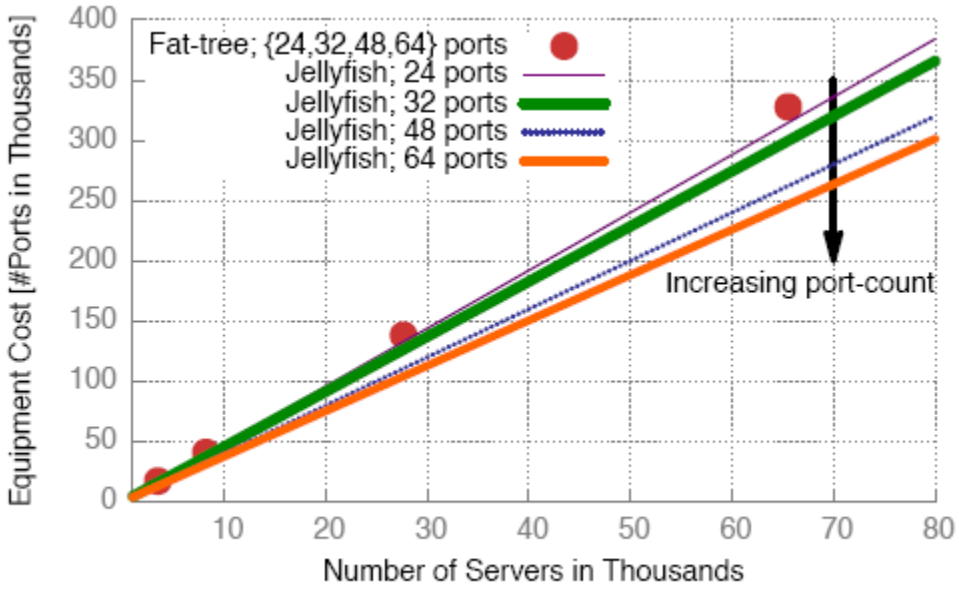
- Bisection Bandwidth vs. Fat-Tree
  - Jellyfish supports more servers
  - Jellyfish allows freedom to adjust



(a)

Efficiency

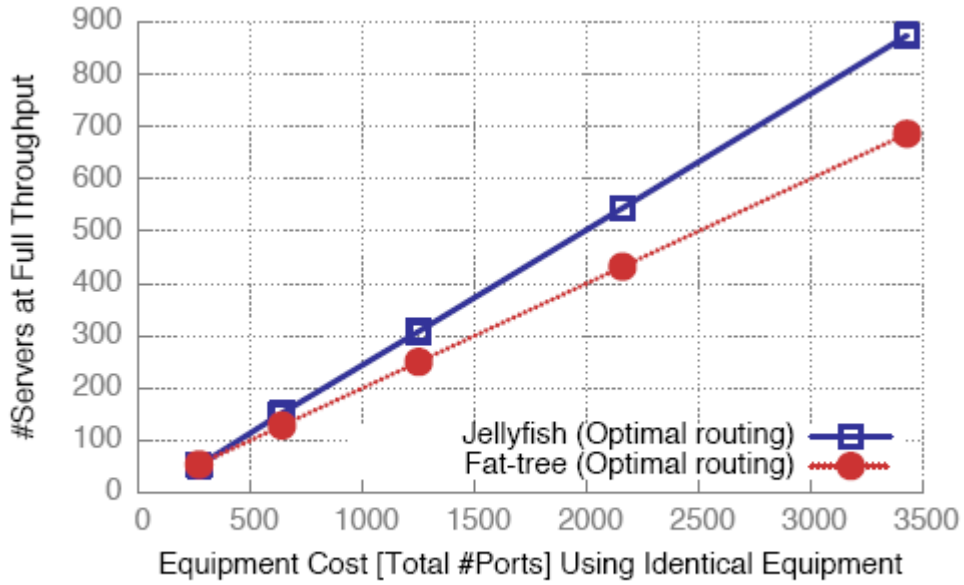
- Bisection Bandwidth vs. Fat-Tree
  - The cost increases slowly
  - Arbitrary size construction



(b)

Efficiency

- Throughput vs. Fat-Tree
  - The improvement increases with scales



(c)

Efficiency

- Throughput vs. Degree-Diameter Graphs
  - Degree-Diameter problem: the problem of finding the largest possible graph G of diameter k such that the largest degree of any of the vertices in G is at most d.
  - The largest degree-diameter graph known to be optimal has N = 50 nodes, with degree 7 and diameter 2.

Degree	2	3	4
3	10	20	38
4	15	41	98
5	24	72	212
6	32	111	390
7	50	168	672
8	57	253	1 100

Efficiency

- Throughput vs. Degree-Diameter Graphs
  - In worst case, Jellyfish achieves 91% throughput of degree-diameter graph
  - Little room for improvement

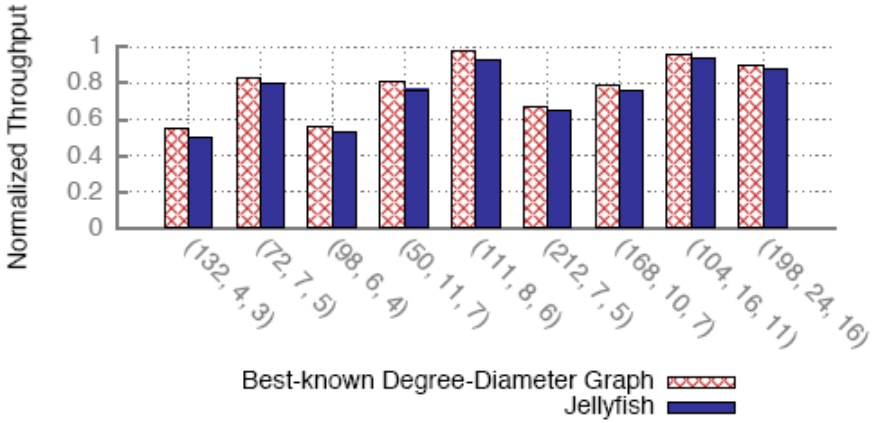
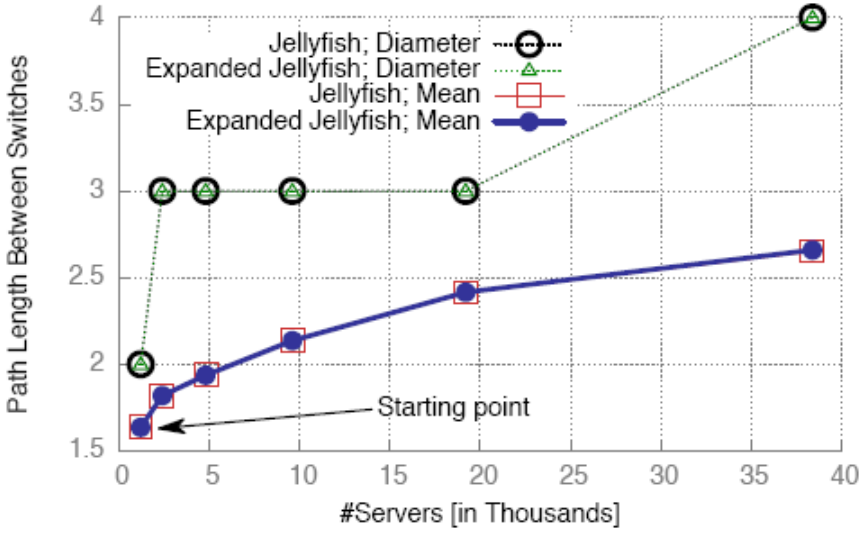


Figure 3: *Jellyfish's network capacity is close to (i.e., ~91% or more in each case) that of the best-known degree-diameter graphs. The x-axis label (A, B, C) represents the number of switches (A), the switch port-count (B), and the network degree (C). Throughput is normalized against the non-blocking throughput. Results are averaged over* 10 *runs.*

Efficiency

- Path Length
  - Short path lengths means low latency and network utilization
  - The upper-bound on the diameter of random regular graph (r-regular graph with N nodes):

$$1 + \lceil \log_{r-1}((2+\varepsilon) r N \log N) \rceil \text{ for any } \varepsilon > 0$$



Flexibility

- Arbitrary-sized Networks:
  - 3-level fat-tree allows only $k^3/4$ servers with k being restricted to the port-count of available switches
- Incremental Expandability:
  - Increments as small as one rack or one switch
  - Rewiring is limited to the number of ports being added to the network
  - Allows Heterogeneous expansion

Flexibility

- Incremental Expandability:
  - The properties are maintained

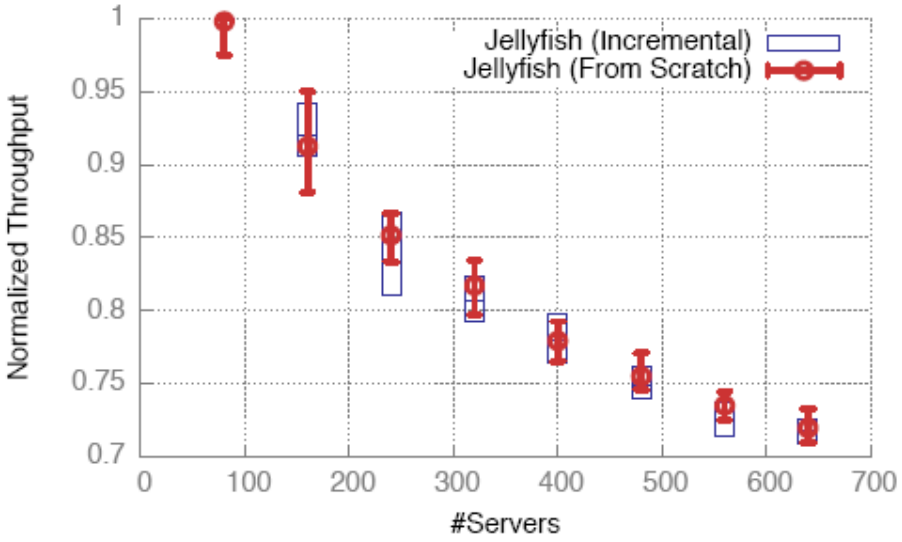
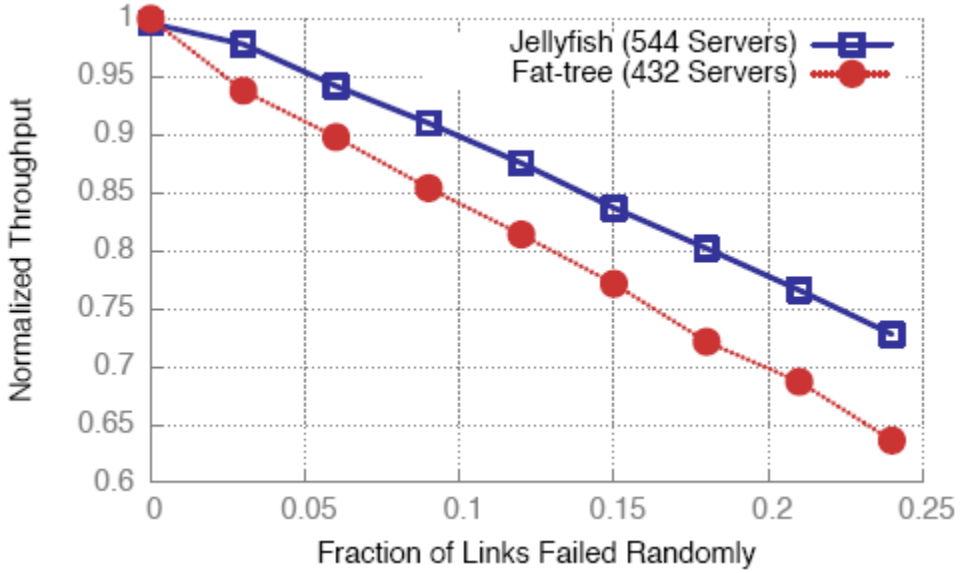


Figure 6: *Incrementally constructed Jellyfish has the same capacity as Jellyfish built from scratch: We built a Jellyfish topology incrementally from* 20 *to* 160 *switches in increments of* 20 *switches, and compared the throughput per server of these incrementally grown topologies to Jellyfish topologies built from scratch using our construction routine. The plot shows the average, minimum and maximum throughput over* 20 *runs.*

Flexibility

- Failure Resilience:
  - r-regular random graph is almost r-connected
  - A random graph topology with a few failures is another random graph topology of smaller size



Outline

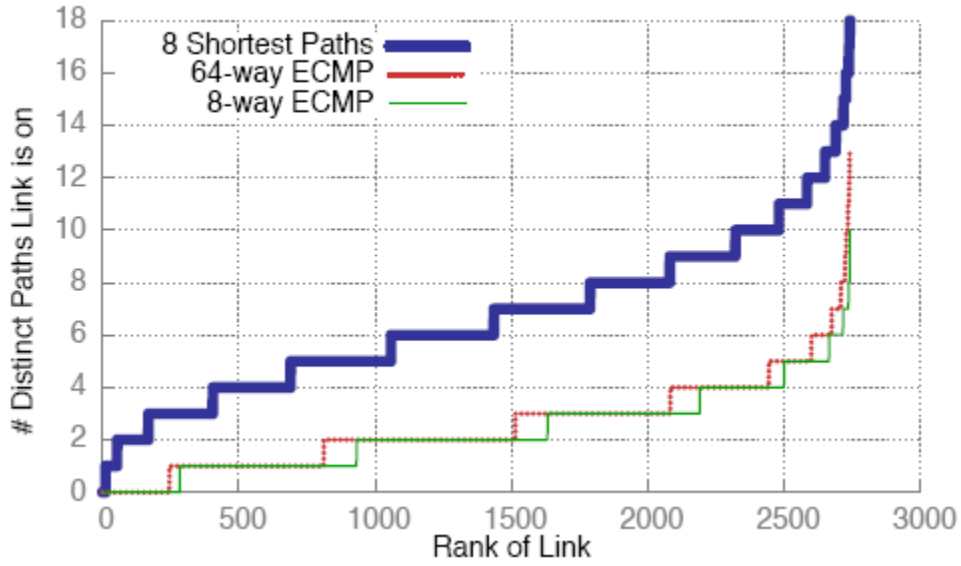
- Introduction
- Jellyfish Topology
- Jellyfish Topology Properties
- Routing & Congestion Control
- Physical Construction and Cabling
- Conclusion

Routing & Congestion Control

- For routing:
  - Compare ECMP and k-shortest paths (k=8)
  - 64-way ECMP does not perform much better than 8-way ECMP
- For congestion control:
  - Compare standard TCP and MPTCP

Routing & Congestion Control

- ECMP is not enough:
- k-shortest path is better for Jellyfish



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%

Routing & Congestion Control

- Routing and congestion control efficiency:

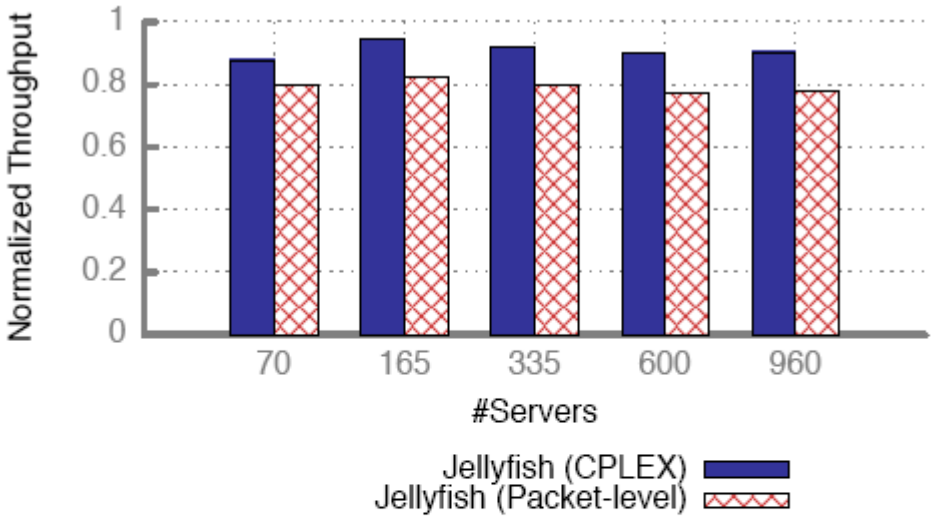


Figure 10: *Simple k-shortest path forwarding with MPTCP exploits Jellyfish's high capacity well: We compare the throughput using the same Jellyfish topology with both optimal routing, and our simple routing mechanism using MPTCP, which results in throughput between 86% – 90% of the optimal routing in each case. Results are averaged over 10 runs.*

Routing & Congestion Control

- FatTree throughput comparison:

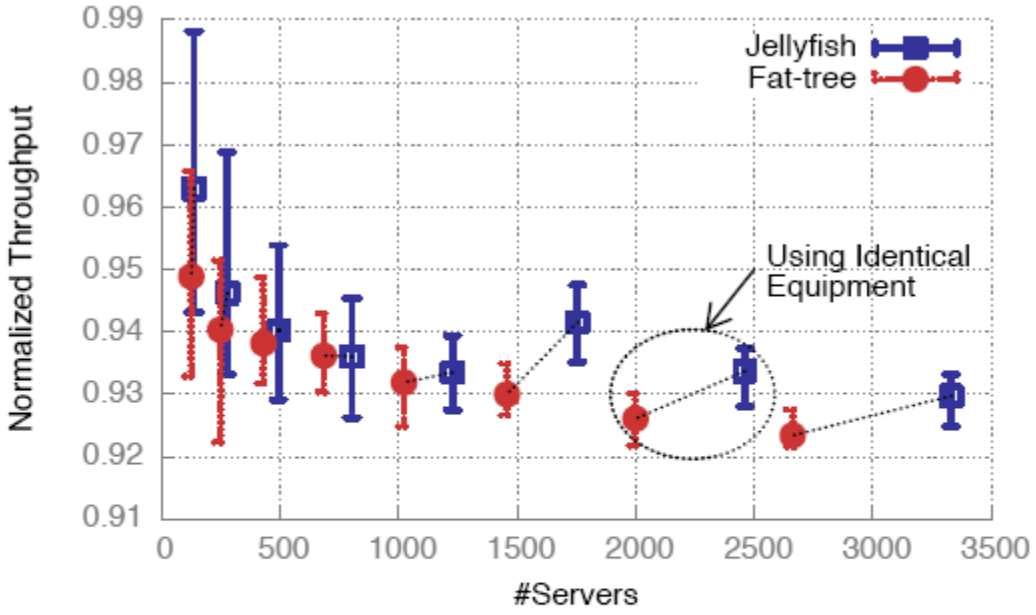


Figure 12: *The packet simulation's throughput results for Jellyfish show similar stability as the fat-tree. (Note that the y-axis starts at 91% throughput.)*

Routing & Congestion Control

- Implementing k-shortest path routing:
  - OpenFlow
    - End-to-end routing rules
  - SPAIN
    - Merge paths into multiple trees, then map to VLANs
  - MPLS
    - MPLS tunnels between switches

Outline

- Introduction
- Jellyfish Topology
- Jellyfish Topology Properties
- Routing & Congestion Control
- Physical Construction and Cabling
- Conclusion

Key considerations

- Number of cables:
  - each cable represents a material and a labor cost
- Length of cables:
  - electrical/optical cable
  - optical transceiver
- Cabling complexity:
  - Is it a significant factor?
  - Layout design that aggregates cables in bundles

Handling Wiring Errors

- Jellyfish cabling:
  - Blueprint automatically generated based on topology and the physical datacenter layout
  - Wiring errors are easy to detect (LLDP) and fix
  - A small number of miswirings may not even require fixing in many cases
  - Labor cost of cabling is about 10% of total cabling cost; total cabling cost is 50% of the network cost at the worst.

Small Clusters and CDCs

- Small clusters form a significant section of market for datacenters
- Key observation in high-capacity Jellyfish
  - More than twice as many cables running between switches than from servers to switches
- Place all switches close to each other
  - Reduce cable length and manual labor
  - Simplify rewiring for expansion and fixing errors

Small Clusters and CDCs

- Number of cables:
  - Fewer cable (about 15% - 20%) than FatTree
- Length of cables:
  - using above optimization
- Complexity:
  - 3~5 standard racks for switches in physical center
- Cabling under expansion:
  - Leaving space near switch-cluster
  - Switch-clusters need to be placed within 10 meters

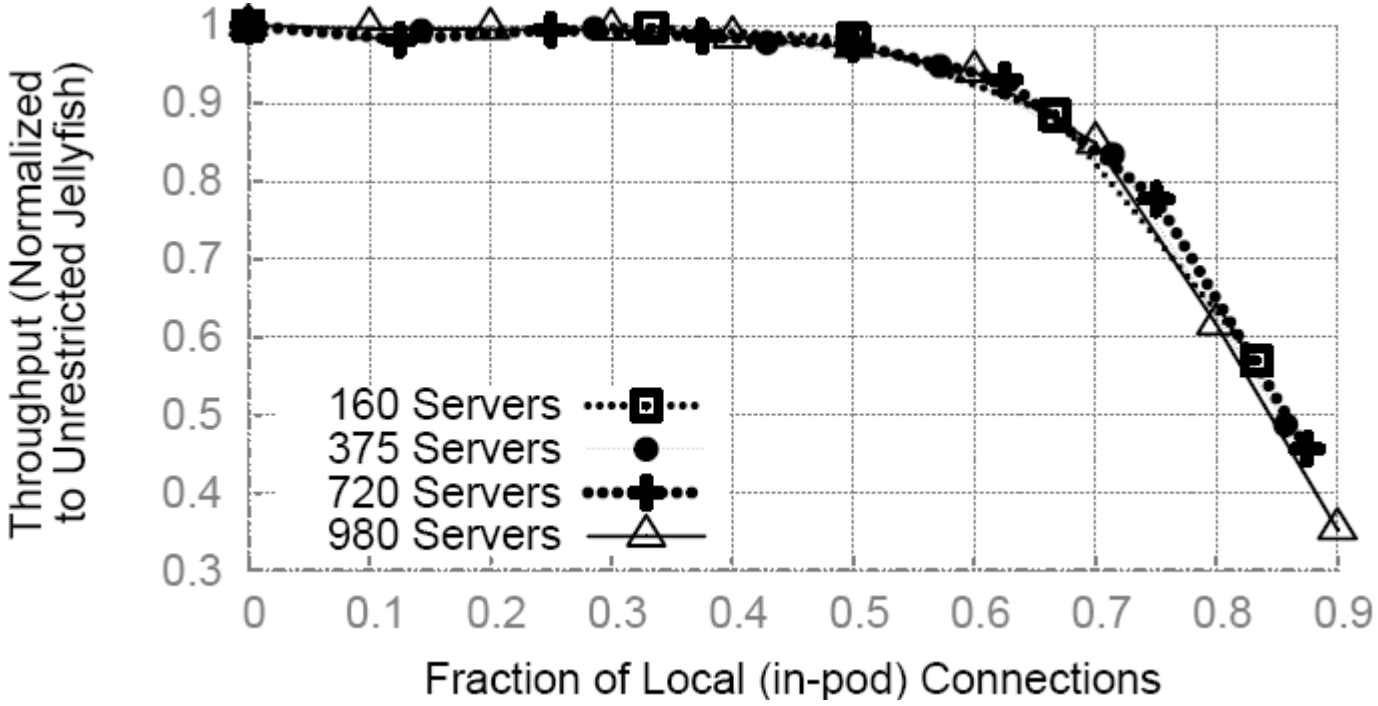
Jellyfish in Massive-Scale DCs

- Containers interconnect
  - Inter-container cables require optical connectors
  - Excessive cabling costs compared to FatTree

- 2-layered random graph
  - a random graph within each container
  - a random graph between containers

Jellyfish in Massive-Scale DCs

- Network Capacity:
  - Local connection: intra-container cables
  - Global connection: inter-container cables



Jellyfish in Massive-Scale DCs

- Complexity:
  - 100,000 server datacenter (~40 containers)
  - ~800 cable assemblies in the worst case
  - 10GBASE-SR external diameter 245um
  - Packed within a pipe of radius < 1cm

Outline

- Introduction
- Jellyfish Topology
- Jellyfish Topology Properties
- Routing & Congestion Control
- Physical Construction and Cabling
- Conclusion

Conclusion

- Random graphs are a highly flexible architecture for datacenter network
- Novel approach to the significant problems of incremental and heterogeneous expansion
- High capacity, short path, resilience to failures and miswirings

Acknowledgements

- Almost the whole content comes from authors' paper and also their slides
- This slides is only for seminar use in NSLab
- For more information, please refer to the following links:
  - https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final82.pdf
  - https://www.usenix.org/sites/default/files/conference/protected-files/jellyfish_nsditalk.pdf

Thanks