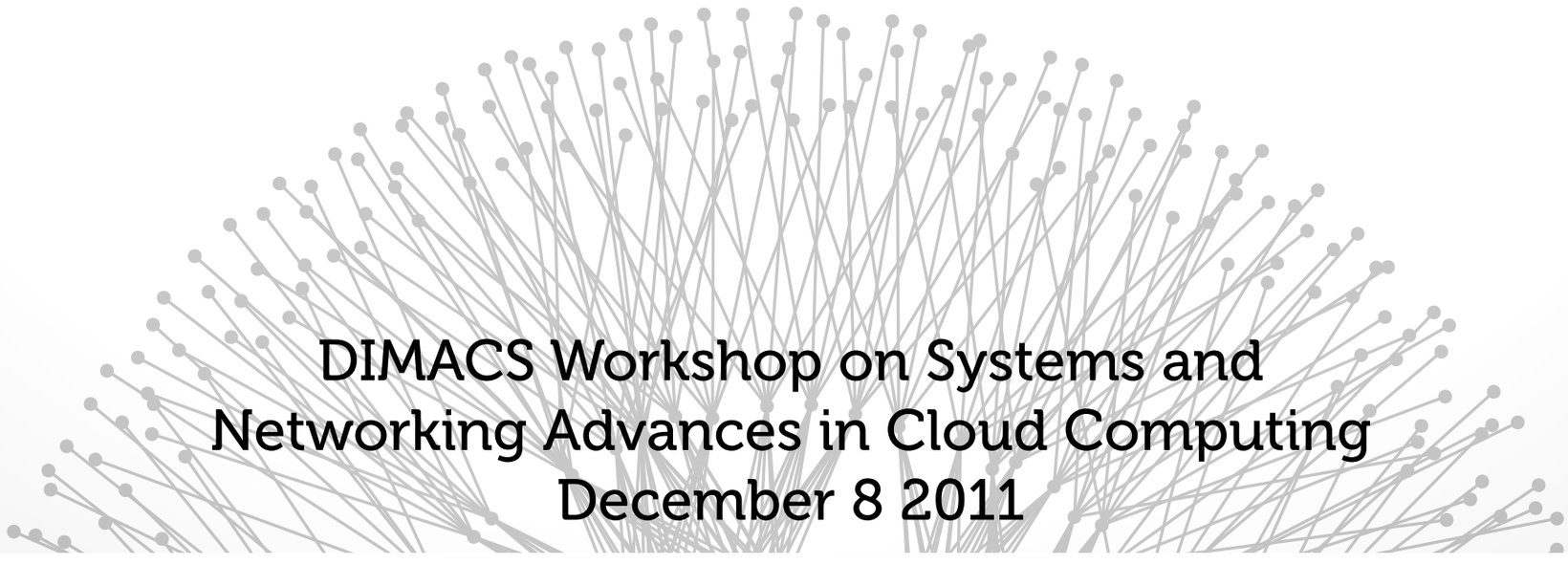
Jellyfish: Networking Data Centers Randomly

Ankit Singla Chi-Yao Hong Lucian Popa
Brighten Godfrey



DIMACS Workshop on Systems and
Networking Advances in Cloud Computing
December 8 2011

The real stars...

Ankit Singla

UIUC

Chi-Yao Hong

UIUC

Lucian Popa

HP Labs

“It is anticipated that the whole of the populous parts of the United States will, within two or three years, be covered with net-work like a spider's web.”

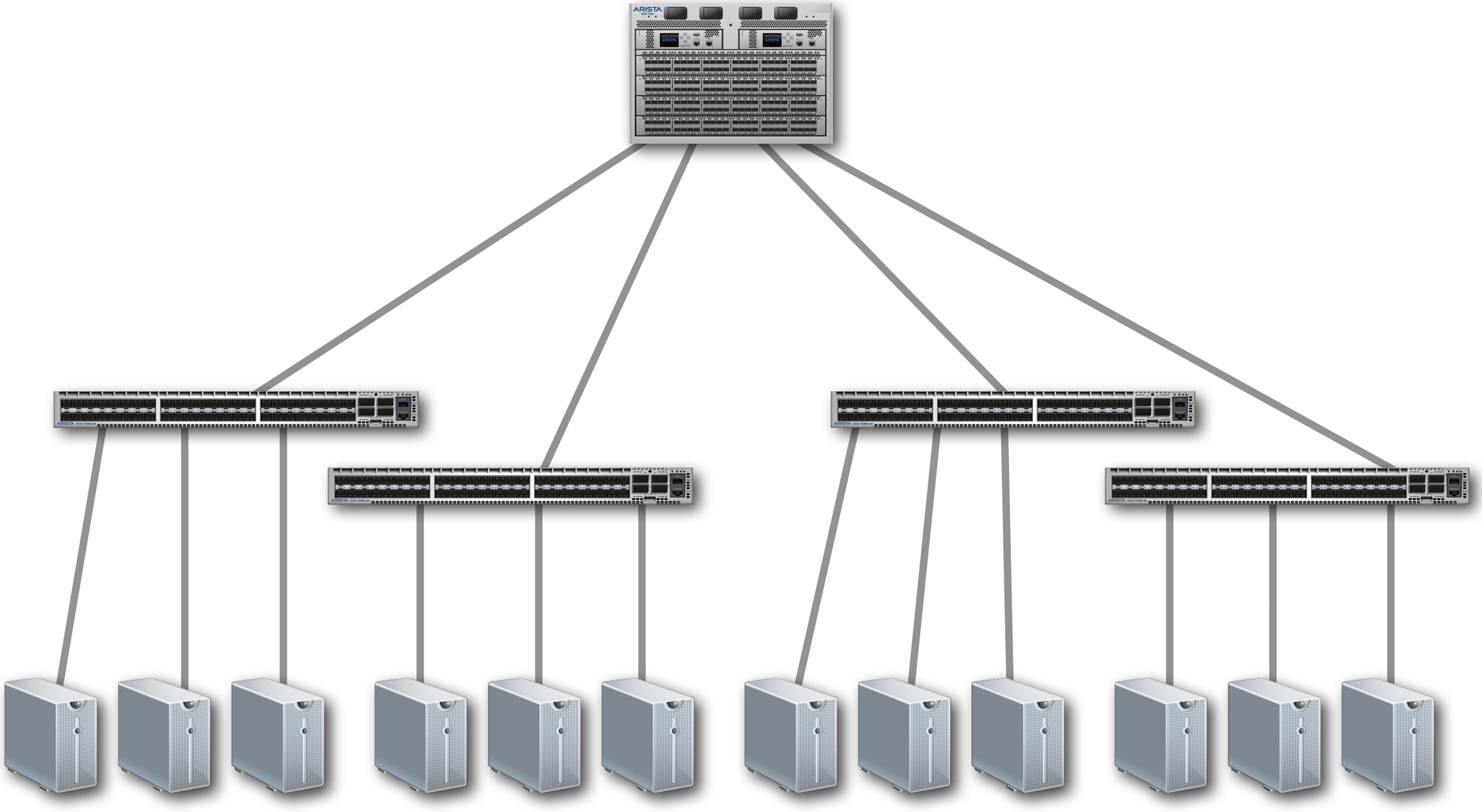
Let’s start with a prediction: Any guesses what date this is from?

“It is anticipated that the whole of the populous parts of the United States will, within two or three years, be covered with net-work like a spider's web.”

— *The London Anecdotes*, 1848

This talk is about network topology, and as this quote illustrates people have been designing network topologies for hundreds of years. But in the past, they have been constrained in some way. If you’re building a wide area network, you need to build a network constrained by, for example, where the population is located ...

... If you’re building a supercomputer, you need a network with a highly regular structure so routing is easy and can’t get deadlocked. ...



... and if you're building a traditional data center, you need a design like a tree so the Spanning Tree Protocol can work right.

But now, in data centers particularly, we have the tools to build topologies with a great deal more freedom than we've ever had before. And I intend to use it! We're going to take that freedom and do something a little radical with it. So, what do we want out of a network topology?

Two goals

High throughput

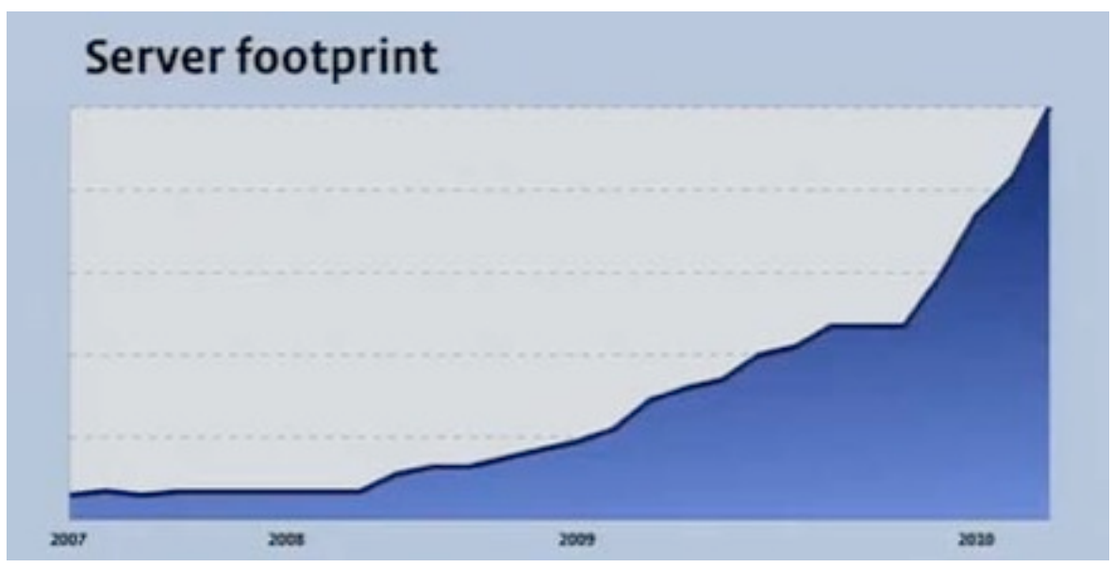
Eliminate bottlenecks
Agile placement of VMs

Incremental expandability

Easily add/replace
servers & switches

Incremental expansion

Facebook "adding capacity on a daily basis"



2007 08 09 10

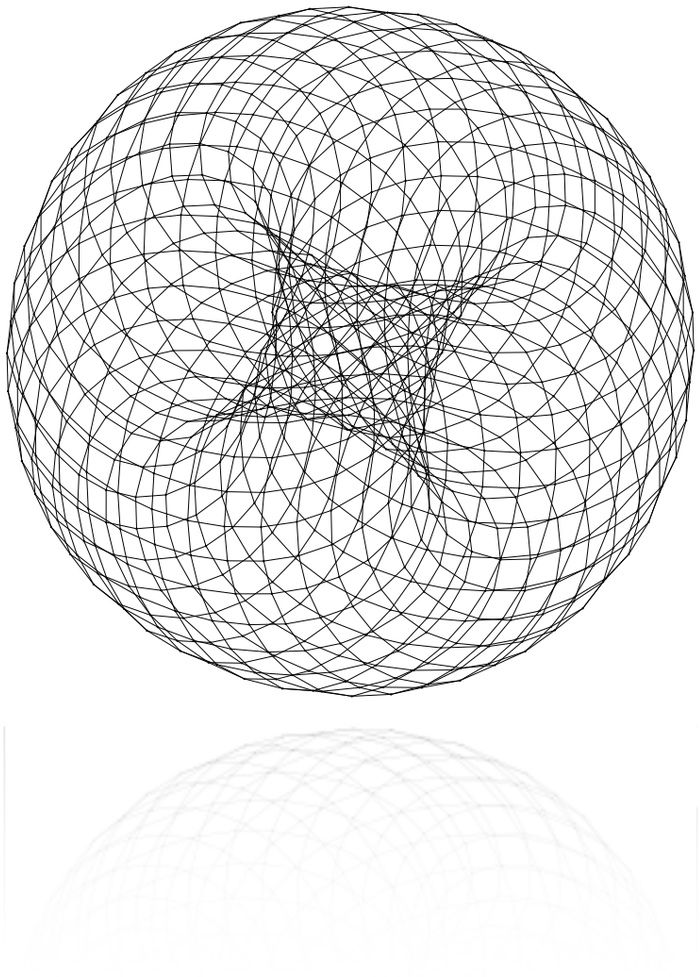
Commercial products

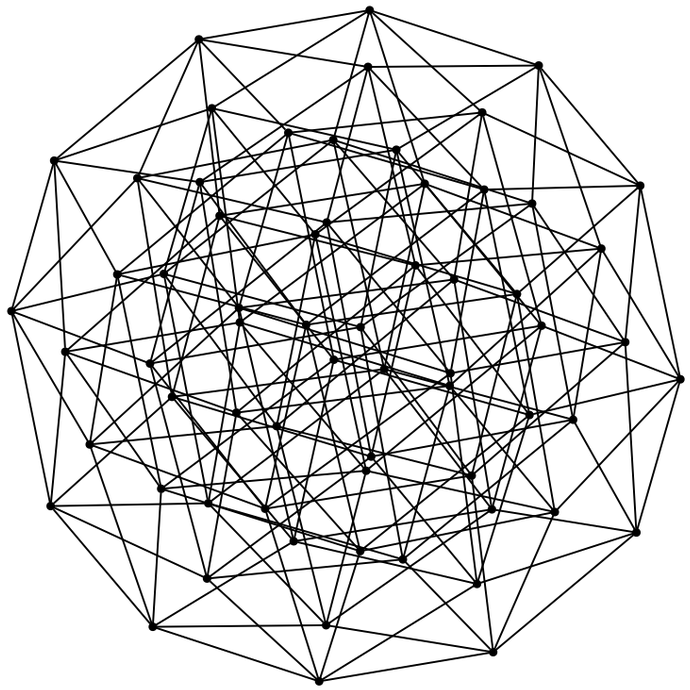
- SGI Ice Cube ("Expandable Modular Data Center")
- HP EcoPod ("Pay-as-you-grow")

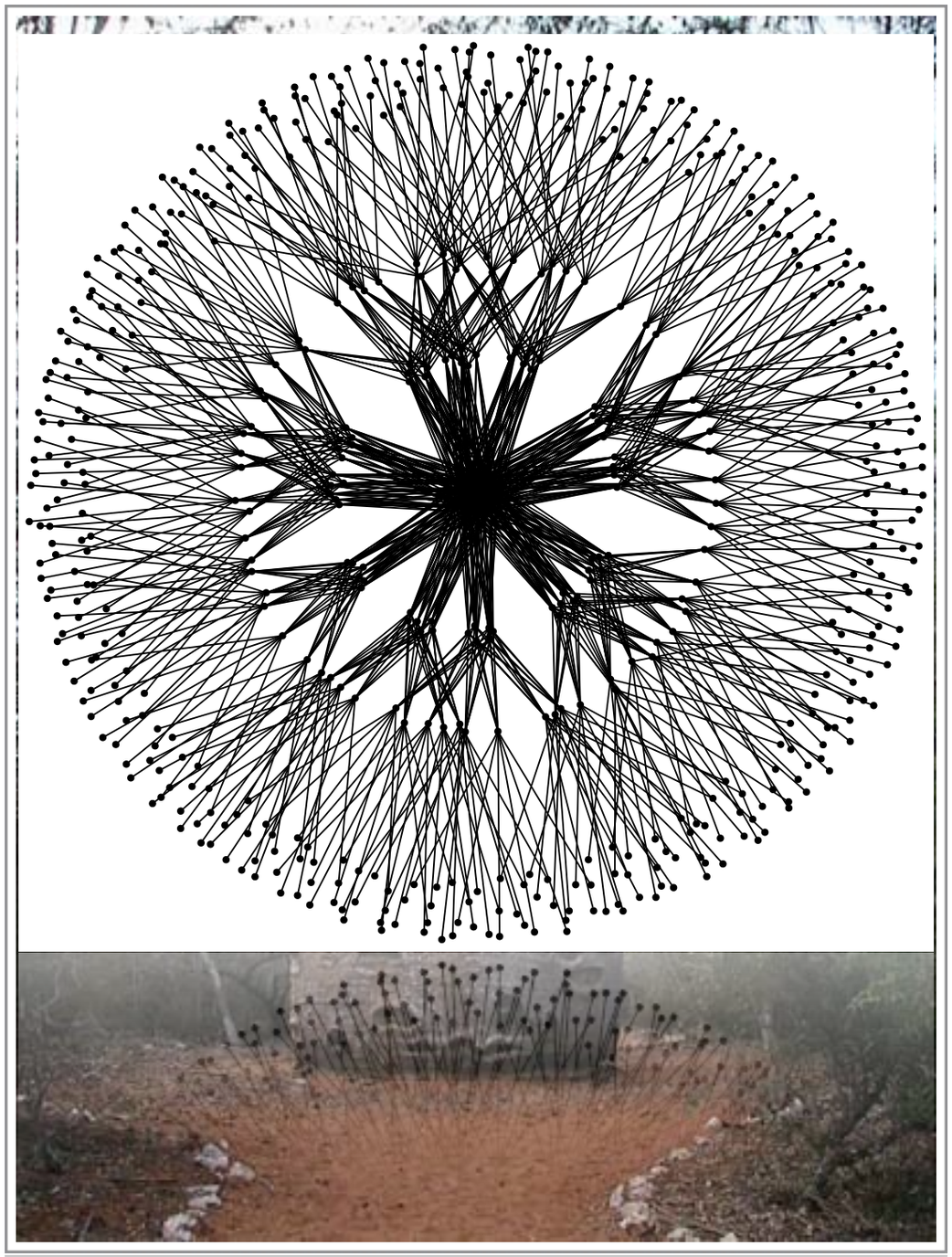
You can add servers, but what about the network?

(http://tinyurl.com/2ayeu4f) These commercial products let you add servers, but expanding high bandwidth network interconnects turns out to be rather tricky.

Today's structured networks







Structure constrains expansion

Coarse design points

- Hypercube: 2^k switches
- de Bruijn-like: 3^k switches
- 3-level fat tree: $5k^2/4$ switches

Fat trees by the numbers:

- (3-level, with commodity 24, 32, 48, ... port switches)
- 3456 servers, 8192 servers, 27648 servers, ...

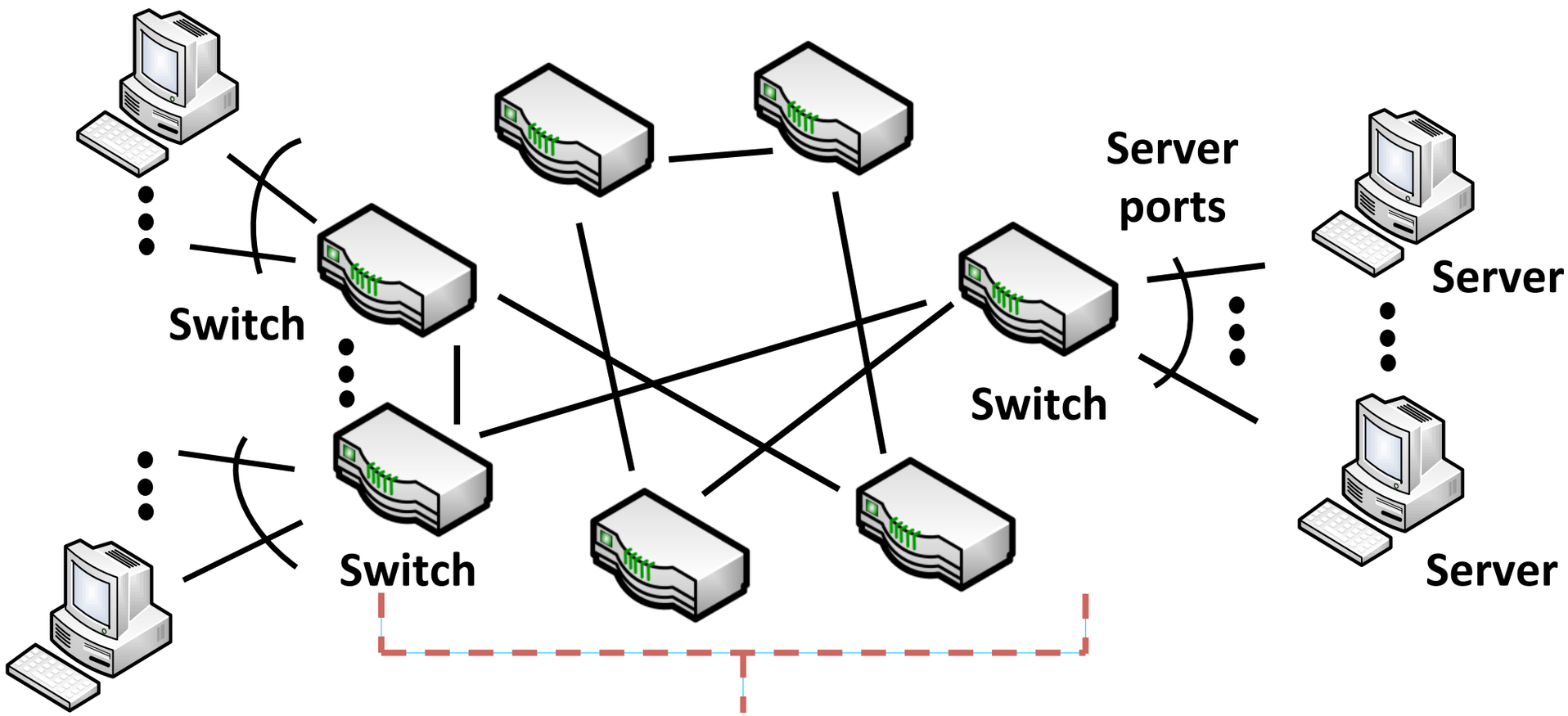
Unclear how to maintain structure incrementally

Our Solution

Forget about structure –
let's have no structure at all!

Jellyfish: The Topology

Jellyfish: The Topology



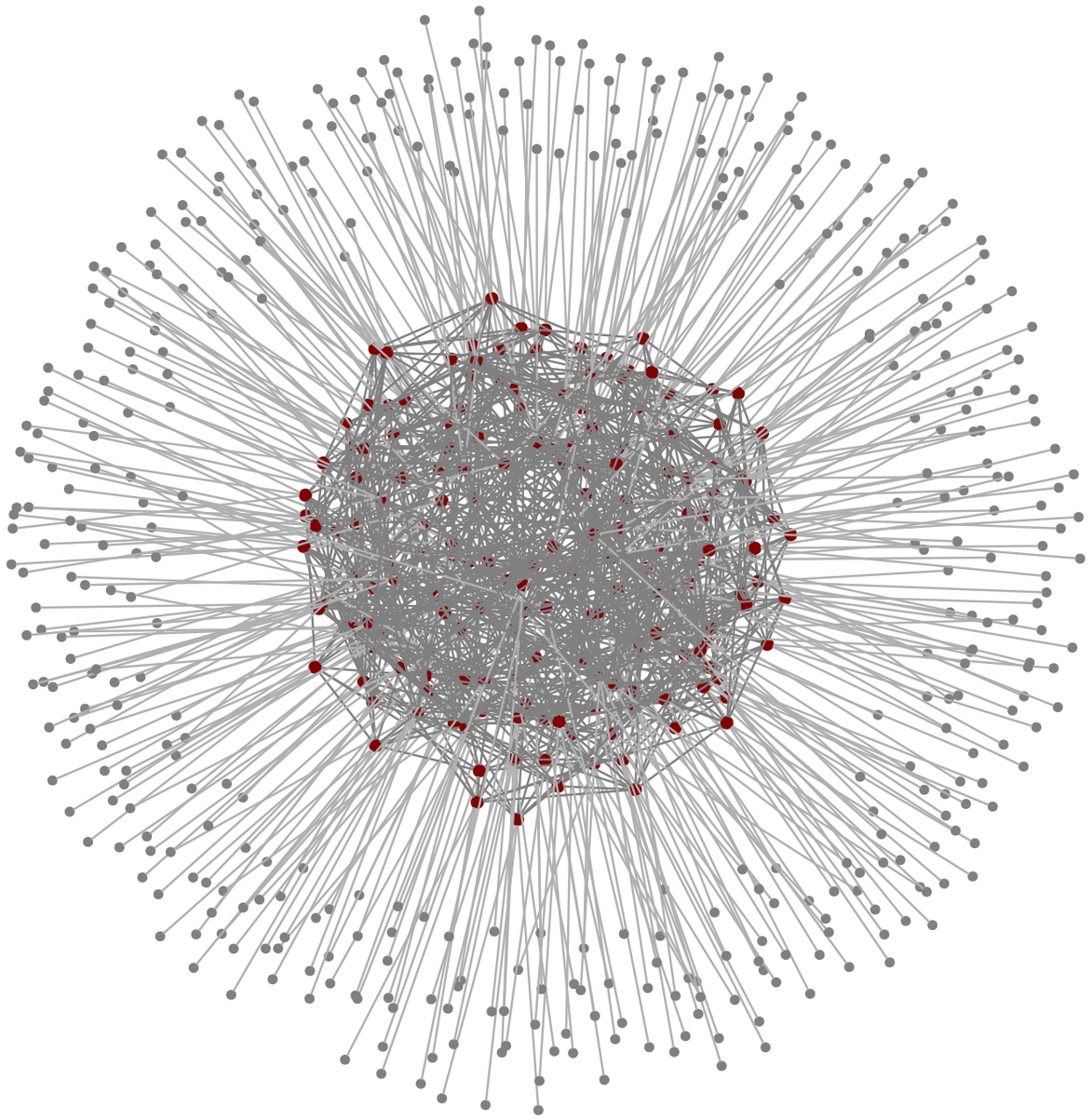
Random Regular Graph

Uniform randomly selected from all regular graphs

Each node has the same degree

Switches are nodes

Capacity as a fluid

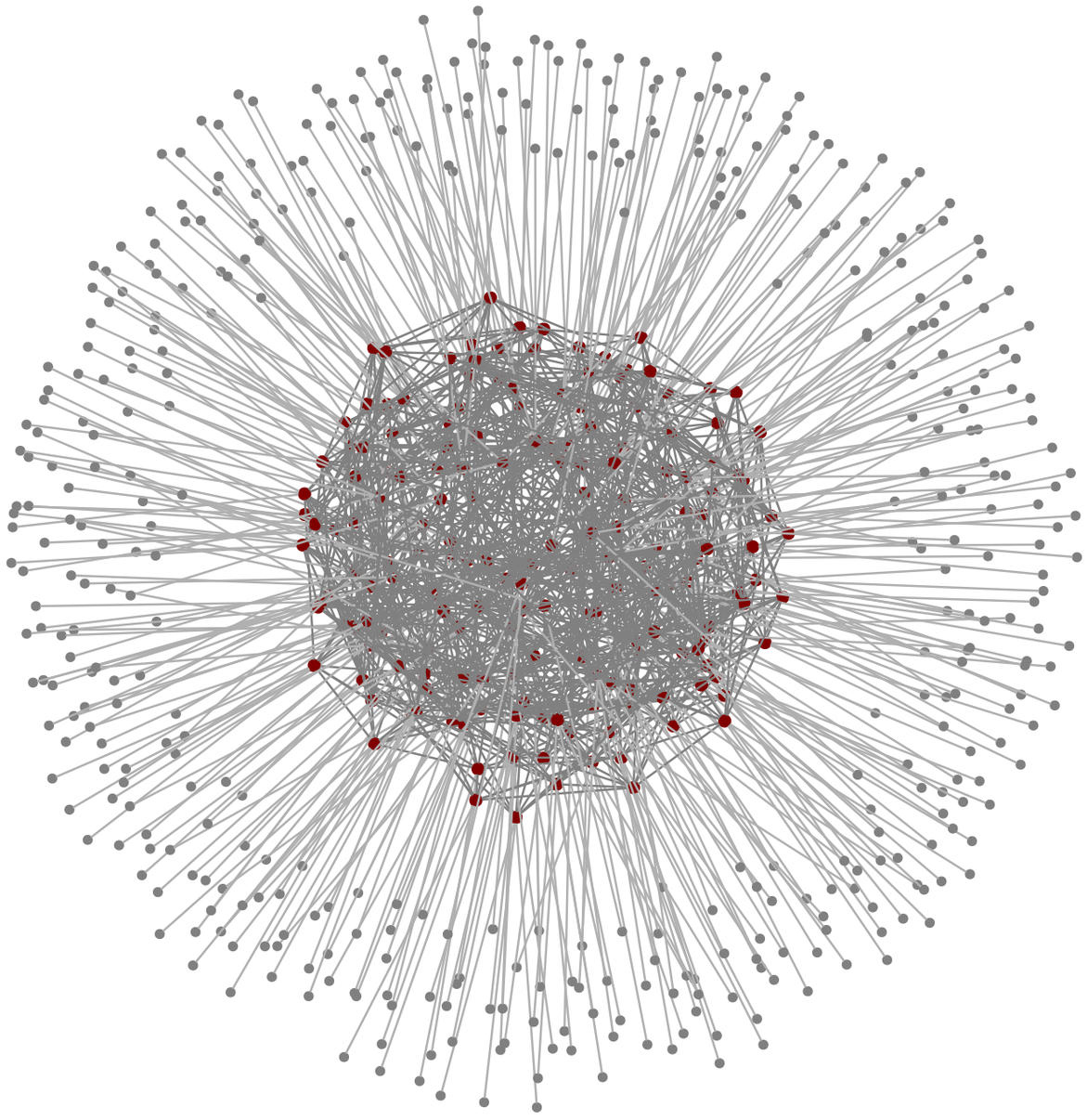


Jellyfish random graph

432 servers, 180 switches, degree 12

The name Jellyfish comes from the intuition that Jellyfish makes network capacity less like a structured solid and more like a fluid.

Capacity as a fluid



Jellyfish random graph

432 servers, 180 switches, degree 12

Jellyfish

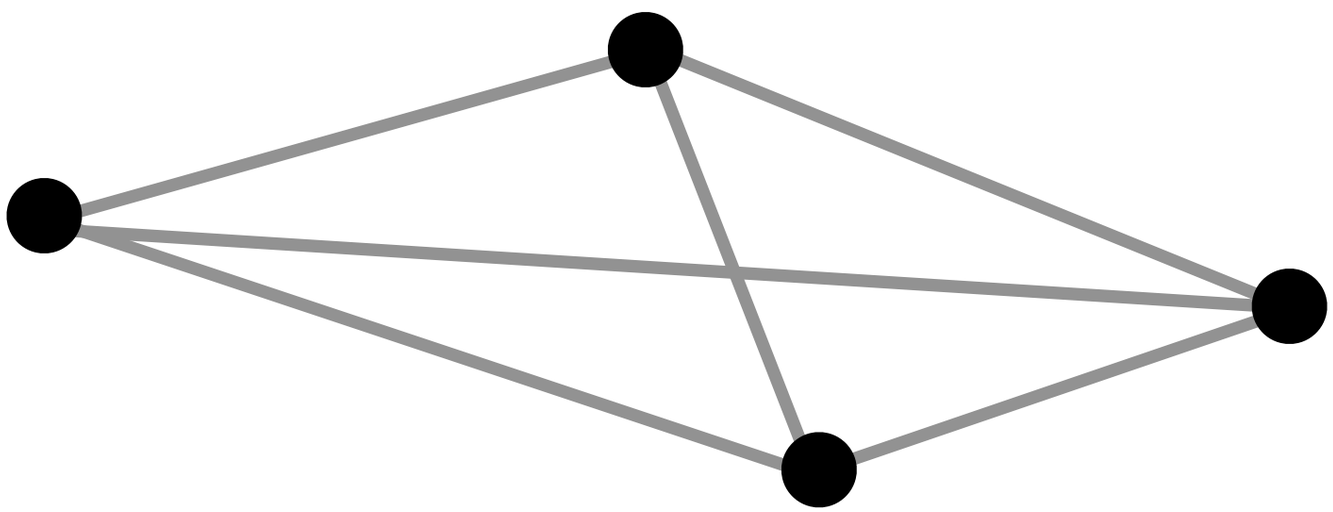
Arctapodema (http://goo.gl/KoAC3)

[Photo: Bill Curtsinger, National Geographic]

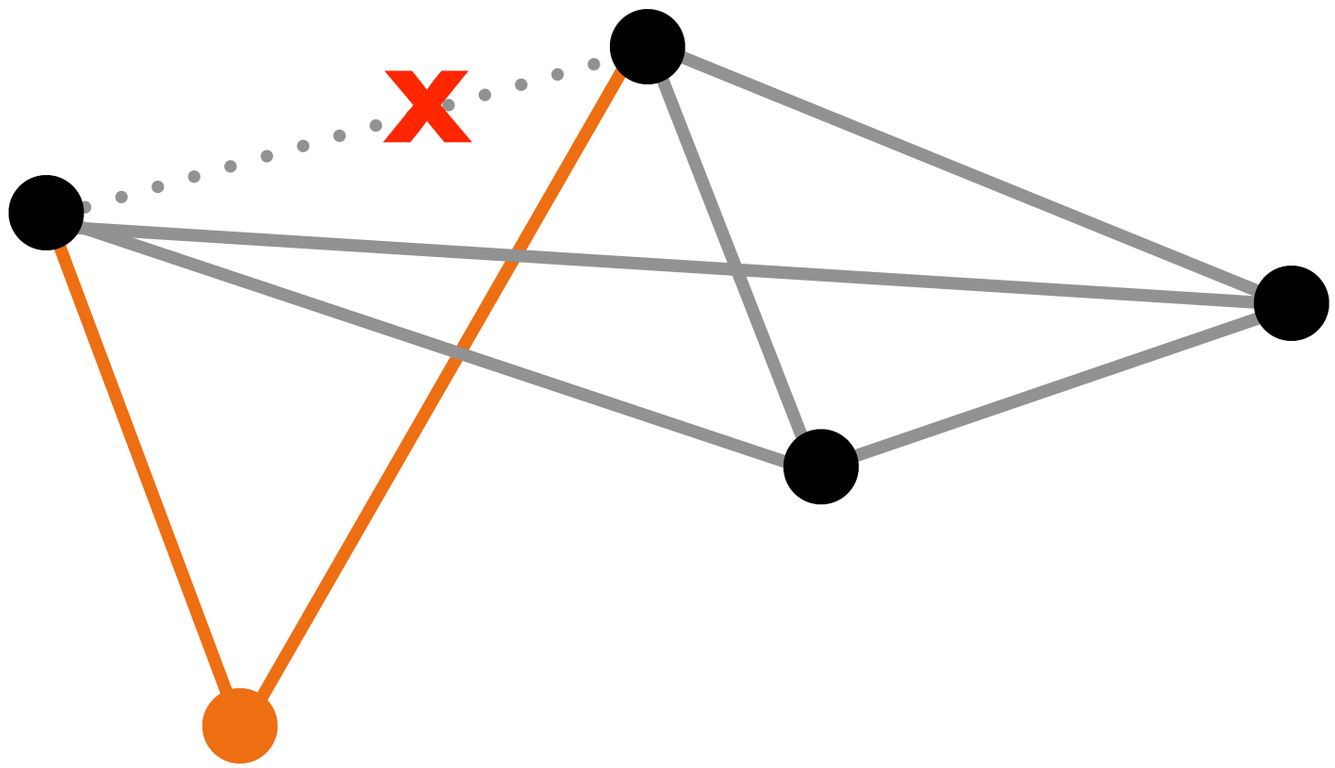
But it also looks like a jellyfish...

Construction & Expansion

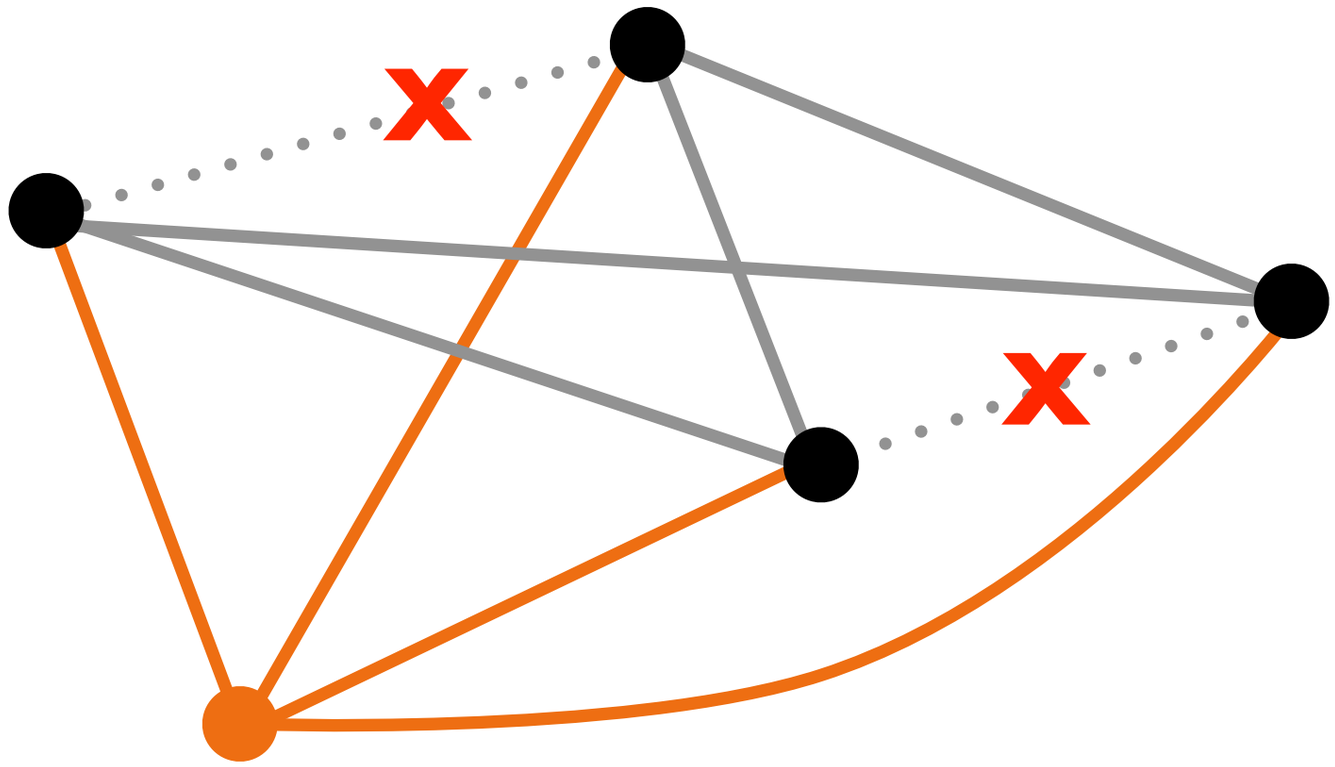
Building Jellyfish



Building Jellyfish

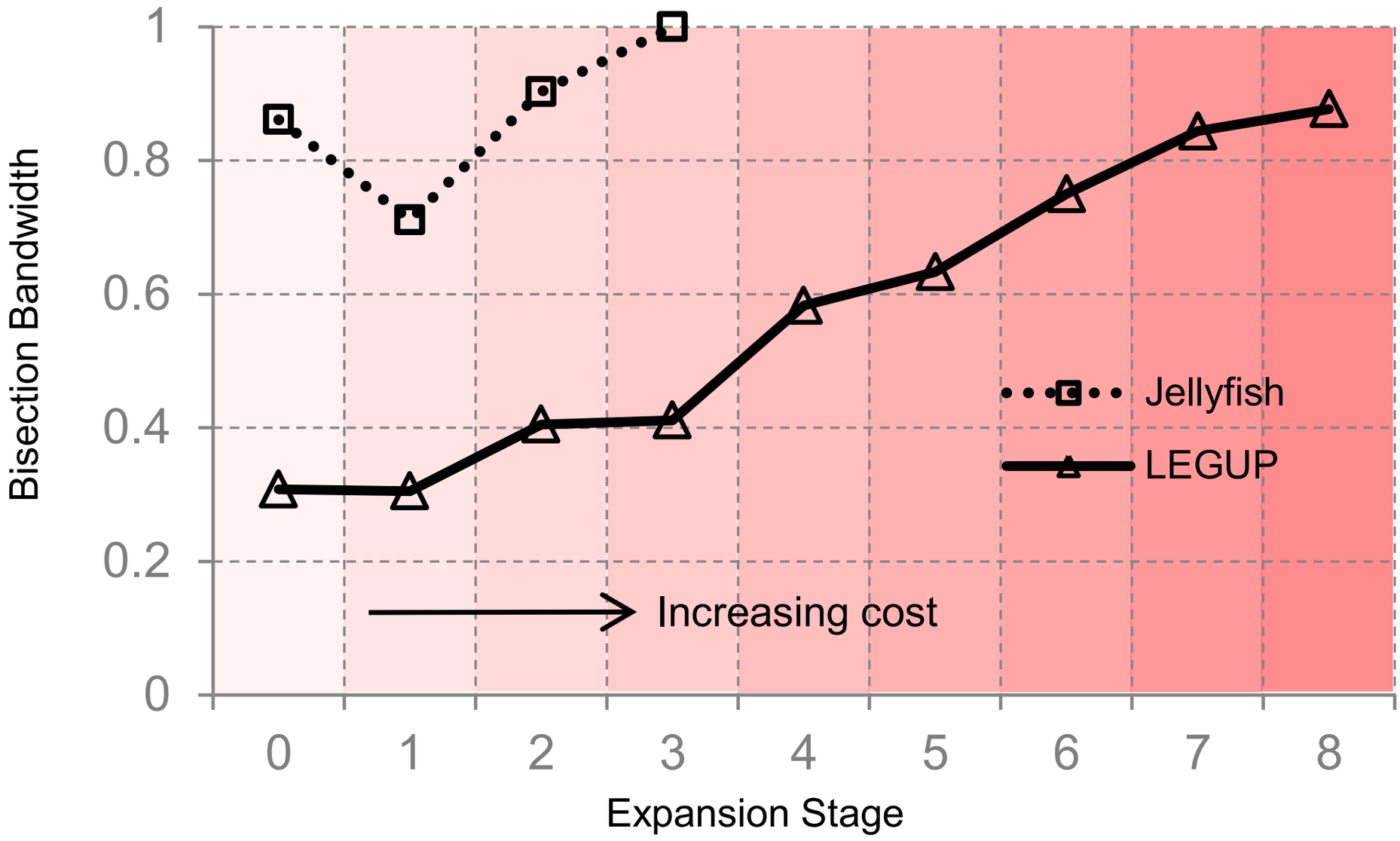


Building Jellyfish



Same procedure for initial construction
and incremental expansion

Quantifying expandability



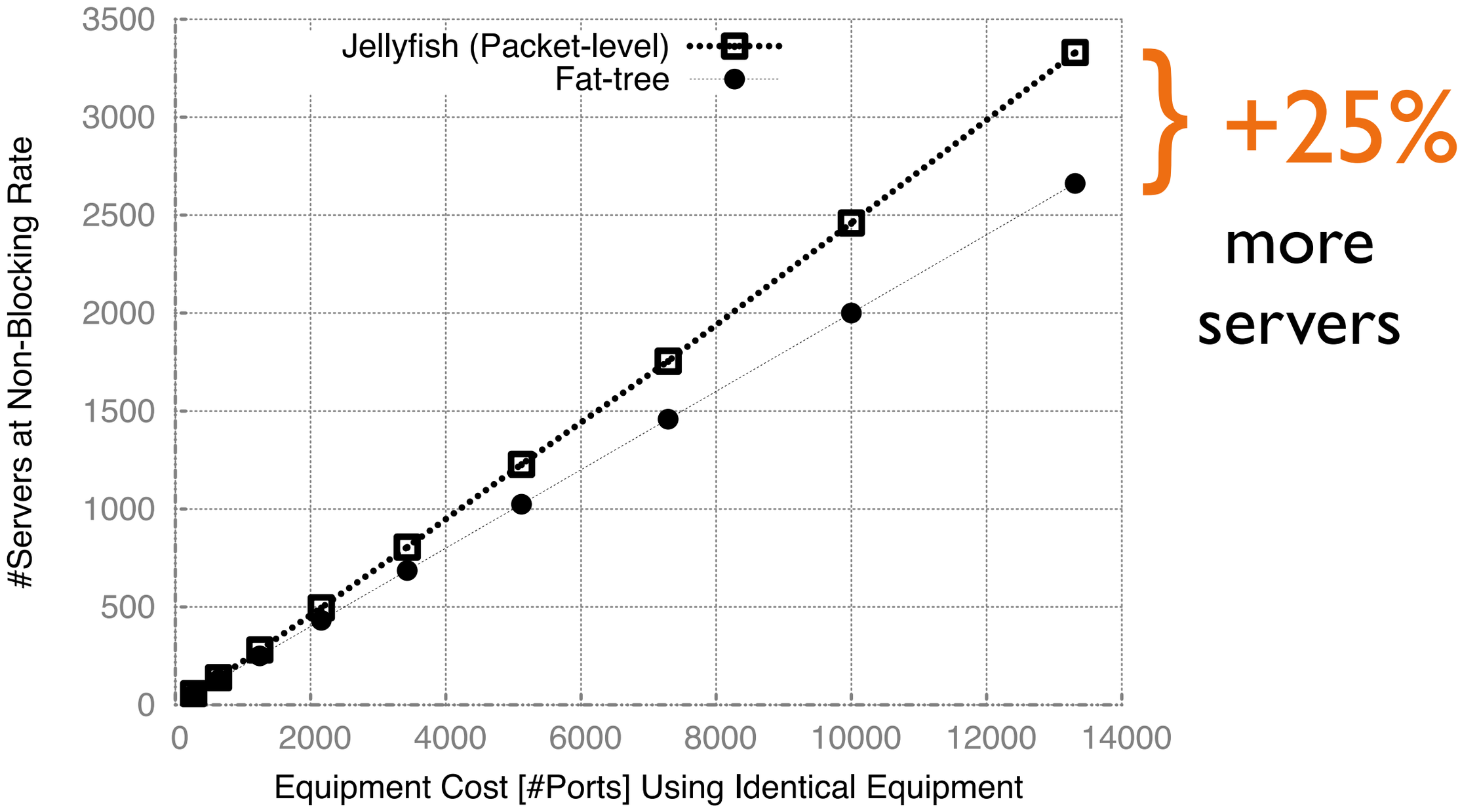
LEGUP: [Curtis, Keshav, Lopez-Ortiz, CoNEXT'10]

Main reason this happens: LEGUP needs to leave some ports free to be able to scale out, while Jellyfish can use them all. The point is not that LEGUP is bad -- it's trying its best, but it has to stay within a Clos-like topology, and to do that, it has to leave some ports free for later expansion.

Throughput

So we got higher bisection bandwidth here because we're using all ports. But, what if we forget about expandability for a moment, and just compare two topologies with equivalent equipment: By giving up a carefully planned structure, do we take a hit on throughput?

Throughput: Jellyfish vs. fat tree



Packet-level simulation

About half the people we talk to think this is obvious, and half think it's surprising. So, let's get some intuition for why Jellyfish has higher throughput.

Intuition

if we fully utilize all available capacity ...

$$\#\ 1\ \text{Gbps flows} = \frac{\text{total capacity}}{\text{used capacity per flow}}$$

Intuition

if we fully utilize all available capacity ...

$$\#\ 1\ \text{Gbps flows} = \frac{\sum_{\text{links}} \text{capacity}(\textit{link})}{\text{used capacity per flow}}$$

Intuition

if we fully utilize all available capacity ...

$$\#\ 1\ \text{Gbps flows} = \frac{\sum_{\text{links}} \text{capacity}(\textit{link})}{1\ \text{Gbps} \bullet \text{mean path length}}$$

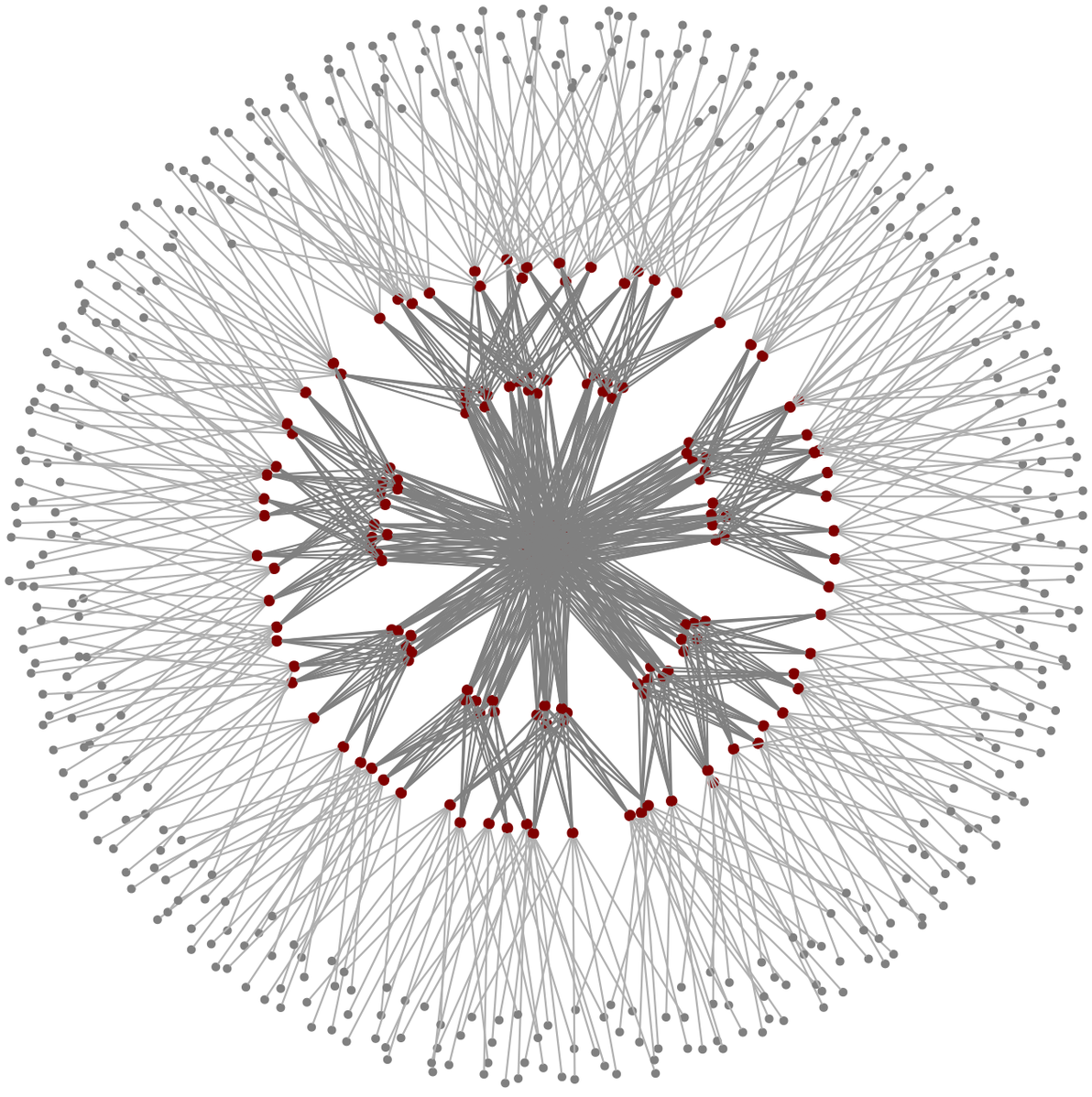
Intuition

if we fully utilize all available capacity ...

$$\#\ 1\ \text{Gbps flows} = \frac{\sum_{\text{links}} \text{capacity}(\textit{link})}{1\ \text{Gbps} \bullet \text{mean path length}}$$

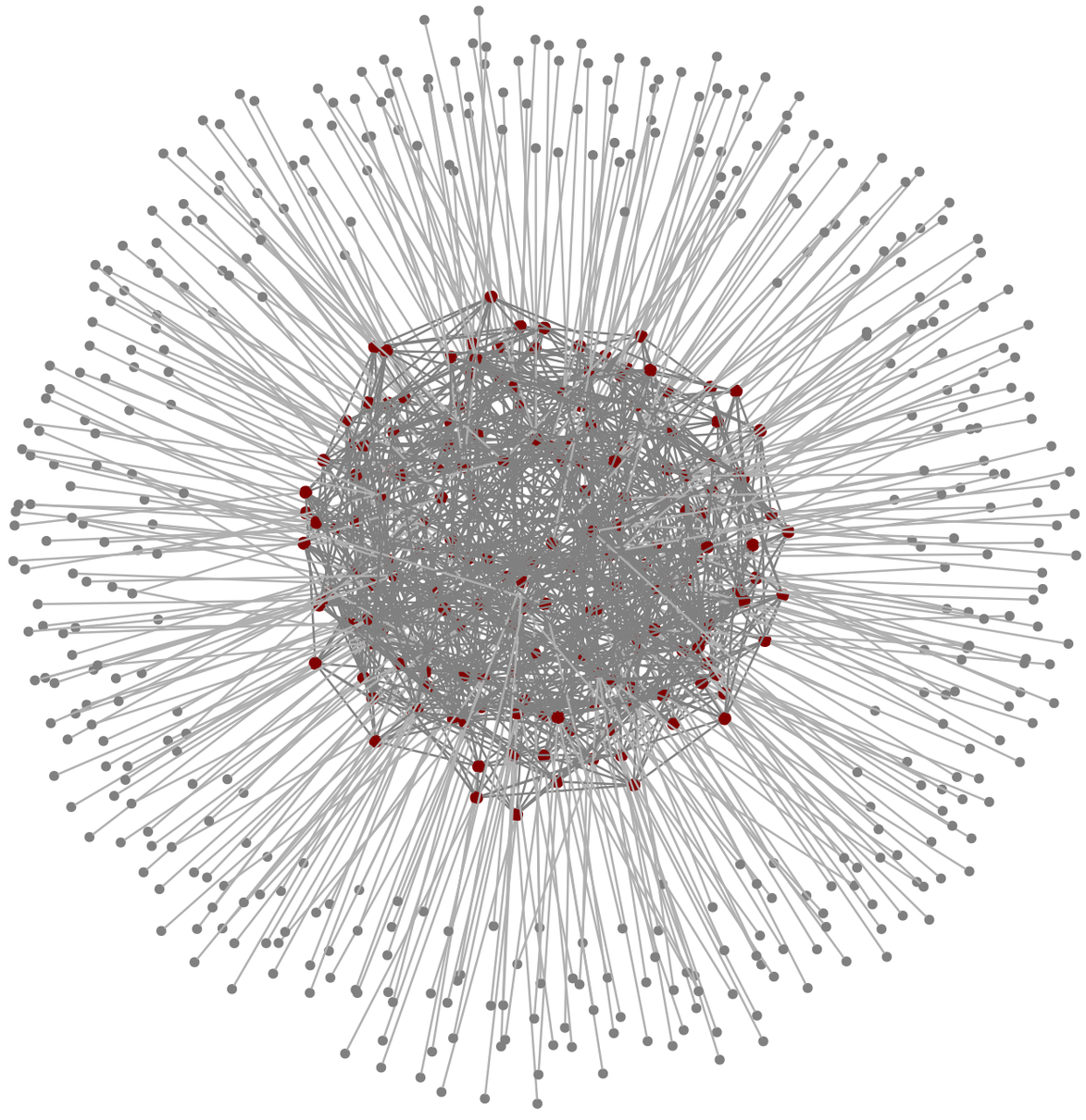
Mission:
minimize average path length

Example



Fat tree

432 servers, 180 switches, degree 12

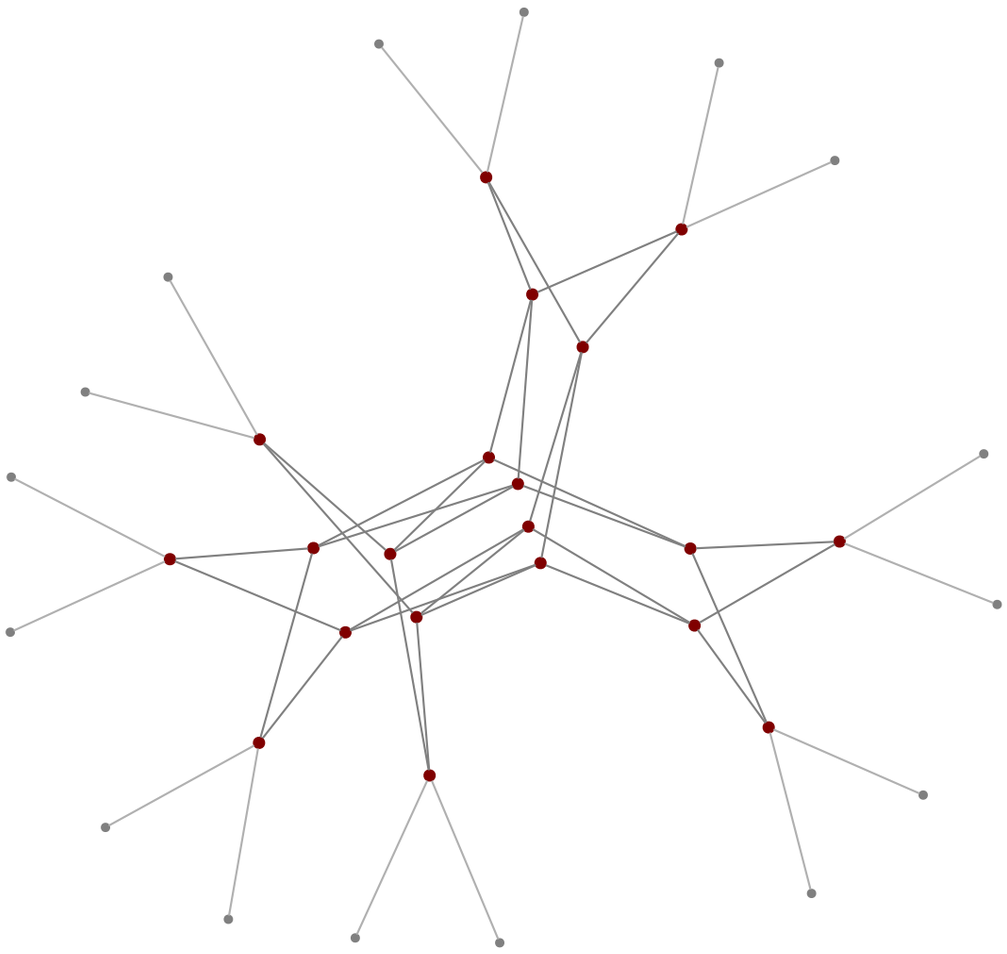


Jellyfish random graph

432 servers, 180 switches, degree 12

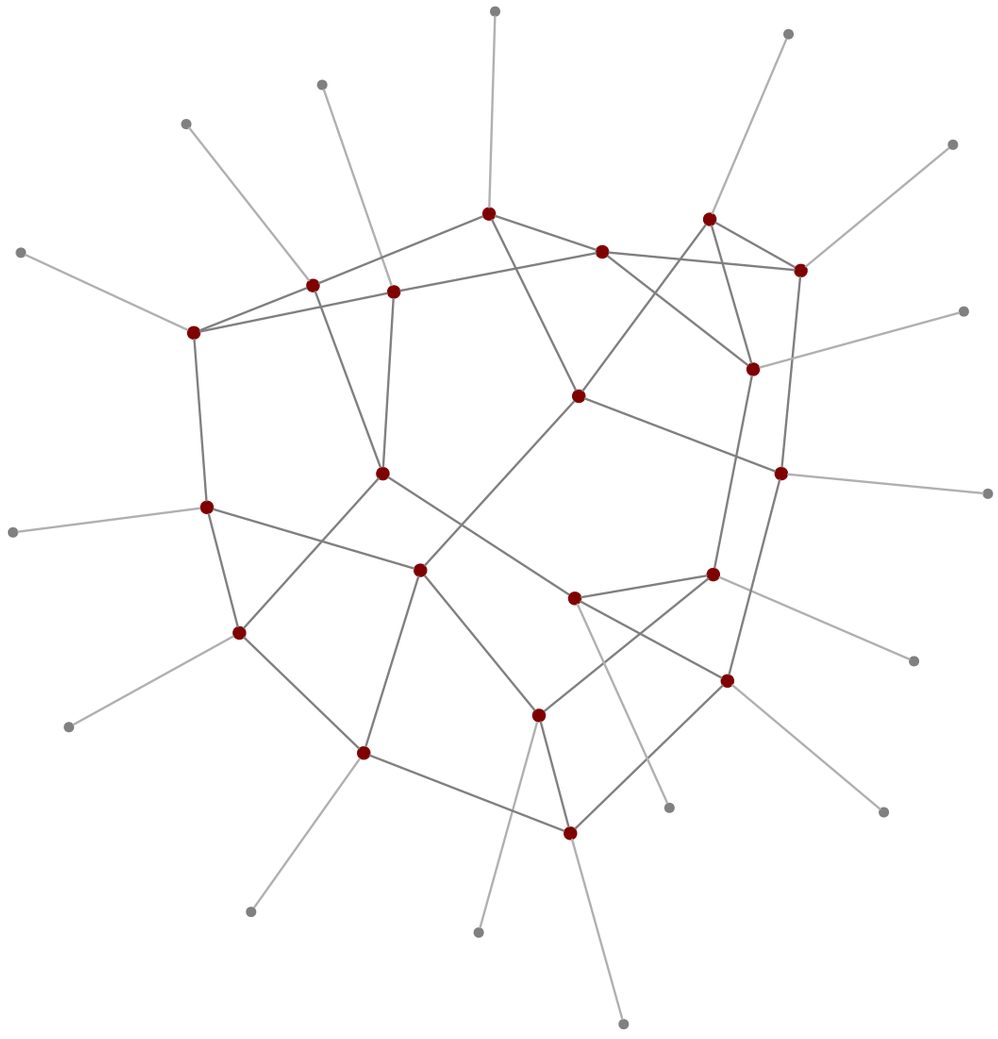
Let’s take an example...

Example



Fat tree

16 servers, 20 switches, degree 4

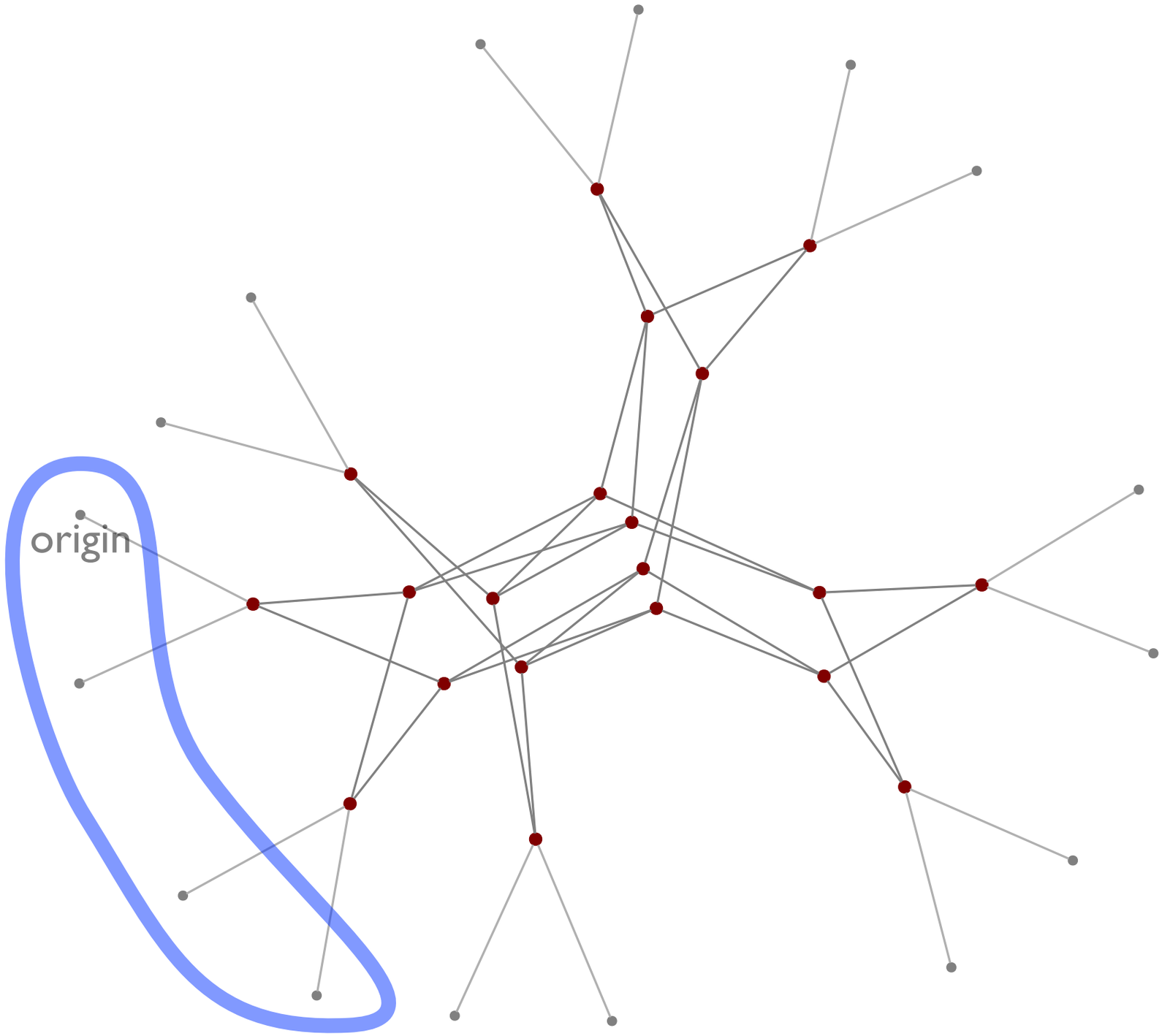


Jellyfish random graph

16 servers, 20 switches, degree 4

A more manageable example, actually...

Example: Fat Tree

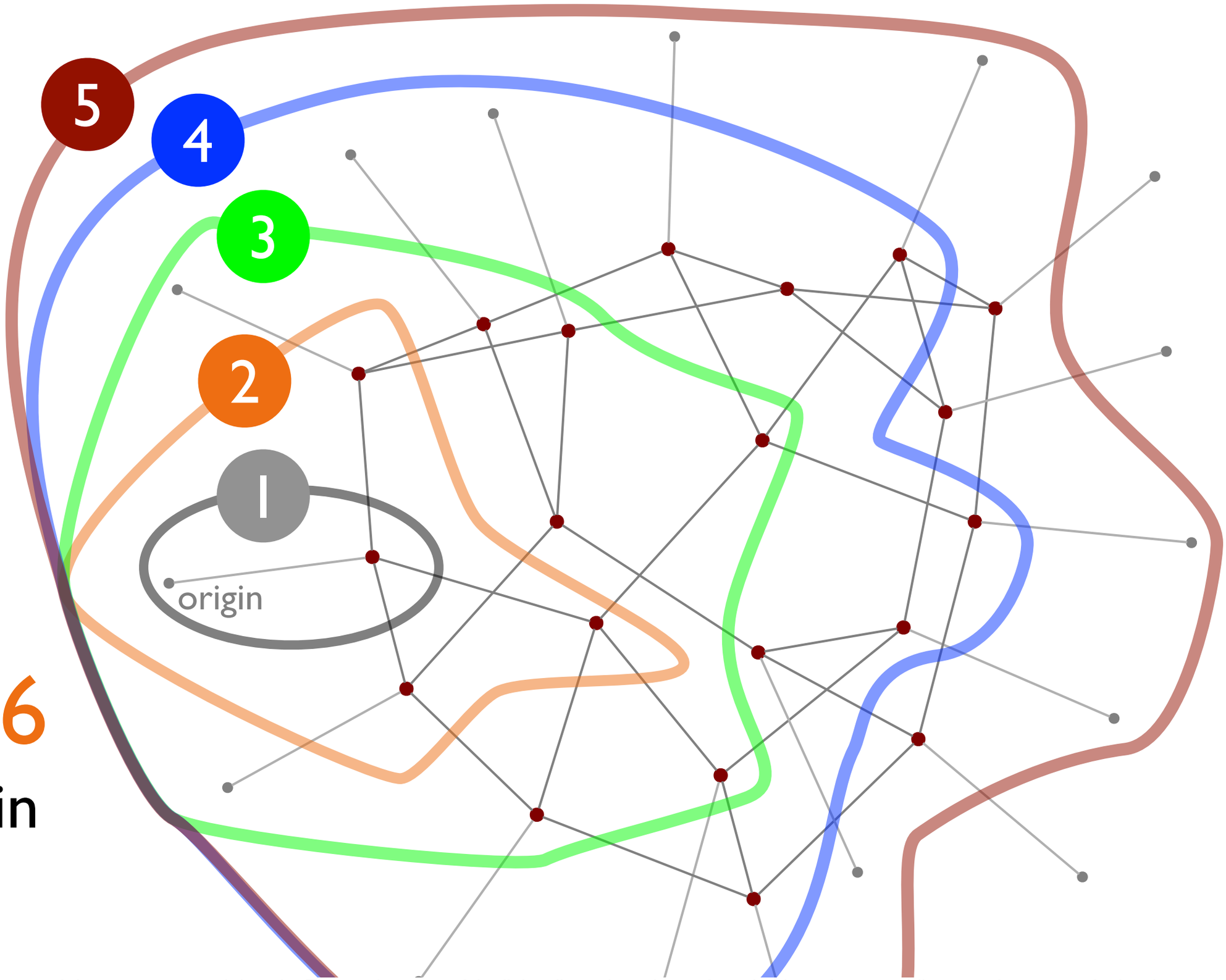


4 of 16

reachable

in < 6 hops

Example: Jellyfish



13 of 16

reachable in

< 6 hops

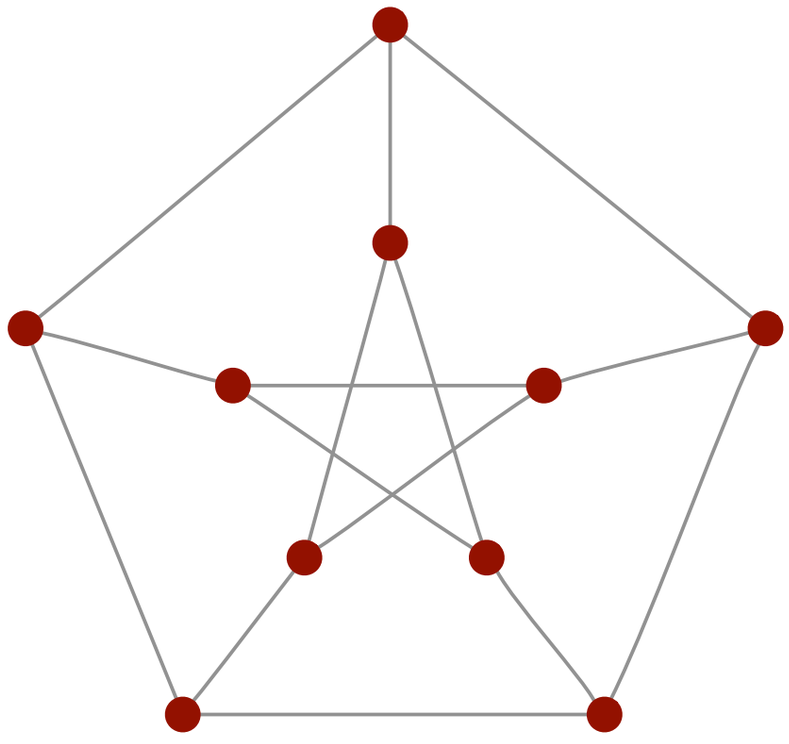
The example demonstrates that Jellyfish has much lower average path length. The randomness of the links allows the sphere of reachable nodes to rapidly expand as we get farther from the origin. (Formally, the random graph is a good expander graph.)

Can we do even better?

What is the maximum number of nodes
in any graph with degree ∂ and diameter d?

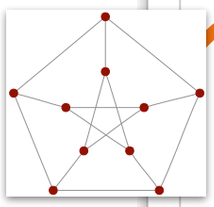
Can we do even better?

What is the maximum number of nodes
in any graph with degree 3 and diameter 2?



Peterson graph

Degree-diameter problem



LARGEST KNOWN (Δ,D)-GRAPHS. June 2010.

Diameter

Degree \ D	2	3	4	5	6	7	8	9	10
3	**10**	**20**	**38**	70	132	196	336	600	1 250
4	**15**	41	98	364	740	1 320	3 243	7 575	17 703
5	**24**	72	212	624	2 772	5 516	17 030	53 352	164 720
6	**32**	111	390	1 404	7 917	19 282	75 157	295 025	1 212 117
7	**50**	168	672	2 756	11 988	52 768	233 700	1 124 990	5 311 572
8	57	253	1 100	5 060	39 672	130 017	714 010	4 039 704	17 823 532
9	74	585	1 550	8 200	75 893	270 192	1 485 498	10 423 212	31 466 244
10	91	650	2 223	13 140	134 690	561 957	4 019 736	17 304 400	104 058 822
11	104	715	3 200	18 700	156 864	971 028	5 941 864	62 932 488	250 108 668
12	133	786	4 680	29 470	359 772	1 900 464	10 423 212	104 058 822	600 105 100
13	162	851	6 560	39 576	531 440	2 901 404	17 823 532	180 002 472	1 050 104 118
14	183	916	8 200	56 790	816 294	6 200 460	41 894 424	450 103 771	2 050 103 984
15	186	1 215	11 712	74 298	1 417 248	8 079 298	90 001 236	900 207 542	4 149 702 144
16	198	1 600	14 640	132 496	1 771 560	14 882 658	104 518 518	1 400 103 920	7 394 669 856

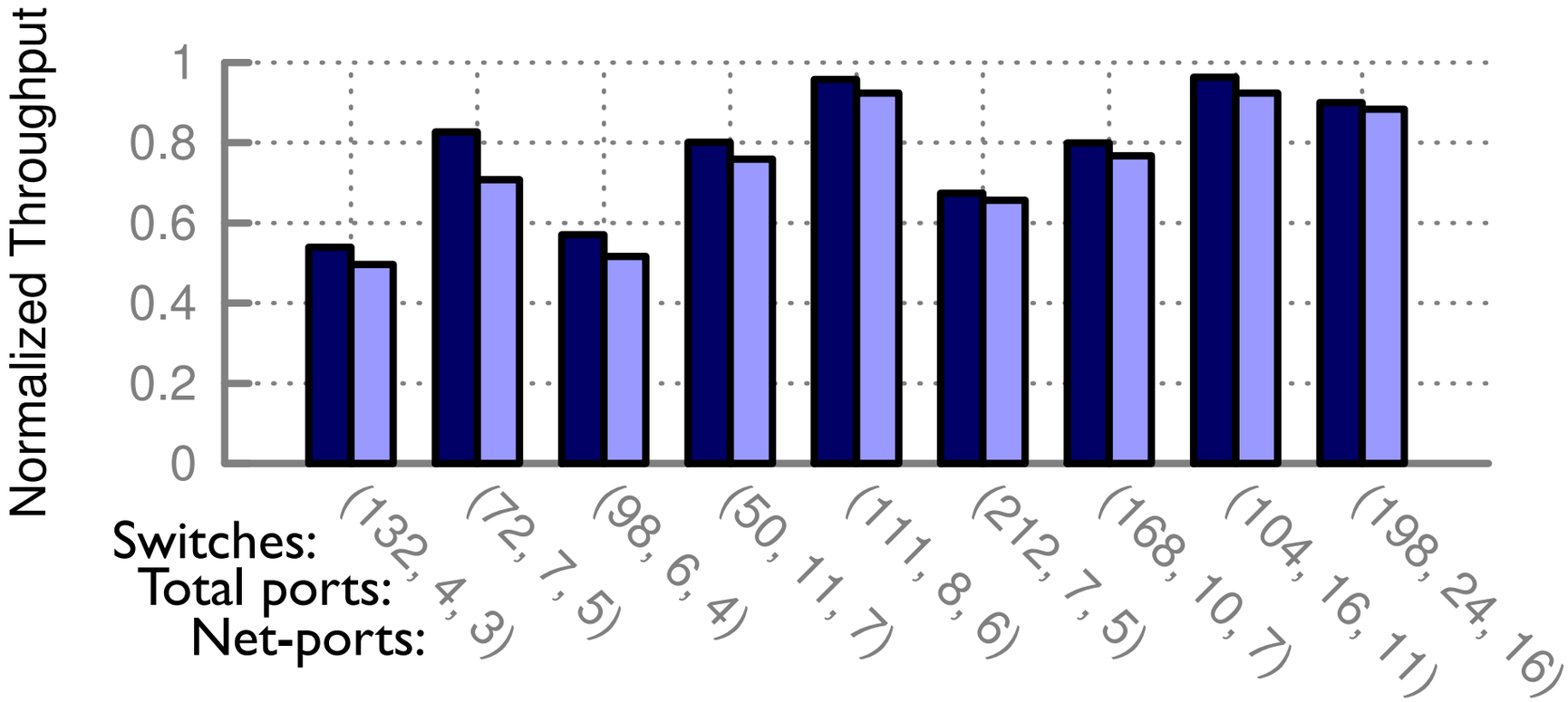
[Delorme & Comellas: http://www-mat.upc.es/grup_de_grafs/table_g.html/]

This is not an easy problem! Only the values in bold are known to be optimal. But people have put in a lot of time to find good graphs in clever ways. Can we make use of this?

Degree-diameter problem

Do the best known degree-diameter graphs also work well for high throughput?

Degree-diameter vs. Jellyfish



Best-known Degree-Diameter Graph

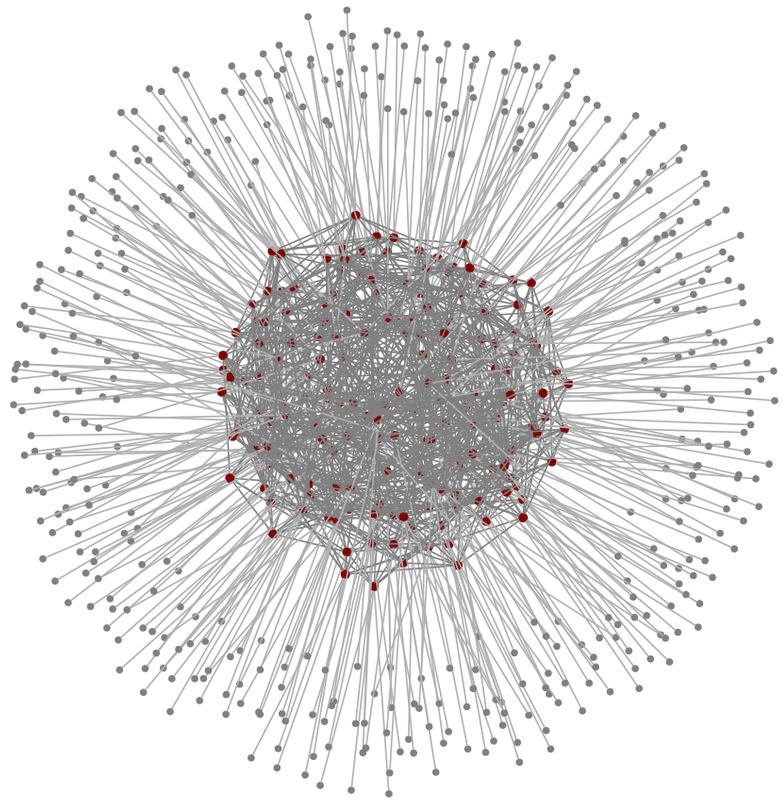
Jellyfish

D-D graphs ***do*** have high throughput

Jellyfish within 15%!

Two interesting things come out of this: (1) Our hypothesis was right, D-D do have high throughput, which might be useful as a benchmark or to build DCs that don't need to expand like maybe in a container. (2) Randomness is competitive, always within 15% of these carefully-optimized topologies. And of course, Jellyfish has the advantage of easy incremental expandability.

What we know so far



flexible, expandable

high throughput

“OK, but…”

Now, this is the point in the talk when you might be saying, “OK, but, what about X?” I’d like to talk about two values of X: Routing and Cabling.

Routing

Intuition

if we fully utilize all available capacity ...

$$\#\ 1\ \text{Gbps flows} = \frac{\text{total capacity}}{\text{used capacity per flow}}$$

How do we effectively utilize capacity without structure?

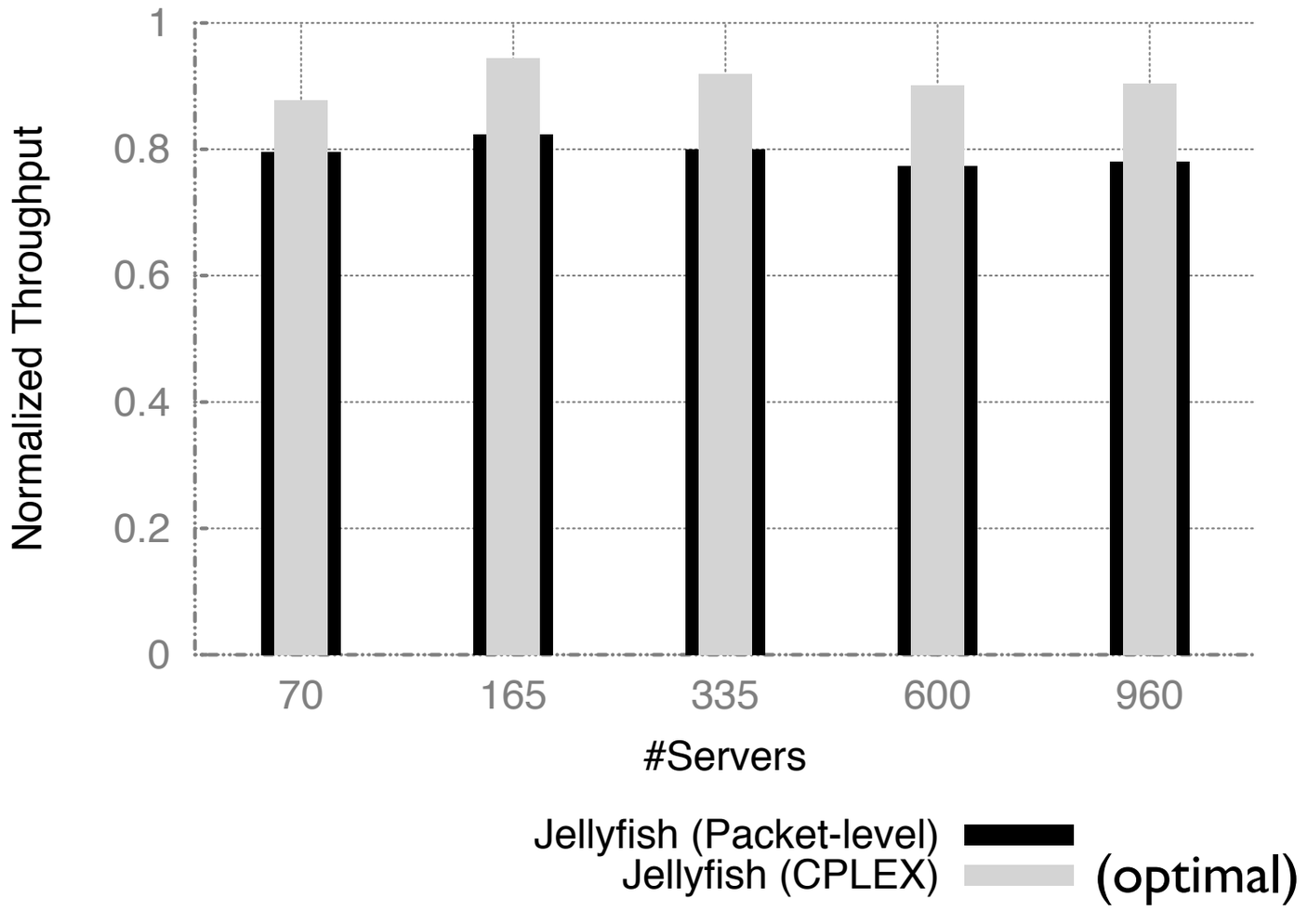
Well, that's a big if...
So, how do we fully utilize the capacity? Tree-like networks have nice structure and we can use something like ECMP or Valiant load balancing, spraying packets or flows randomly to the core switches. But now we don't have any structure of "core" switches. What do we do?

Routing: a simple solution

Find *k* shortest paths

Let Multipath TCP do the rest

- [Wischik, Raiciu, Greenhalgh, Handley, NSDI'10]

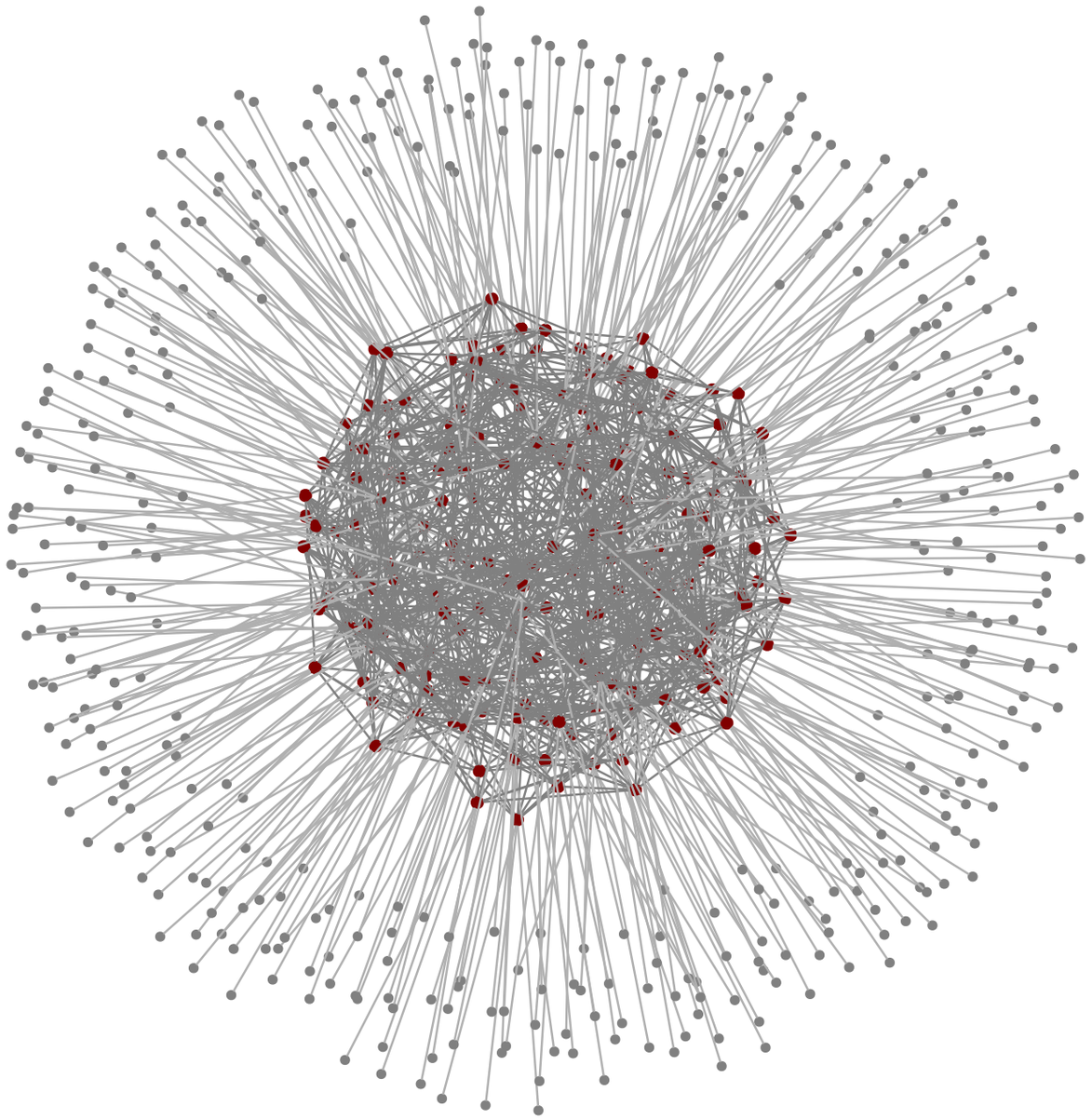


86-90% of optimal

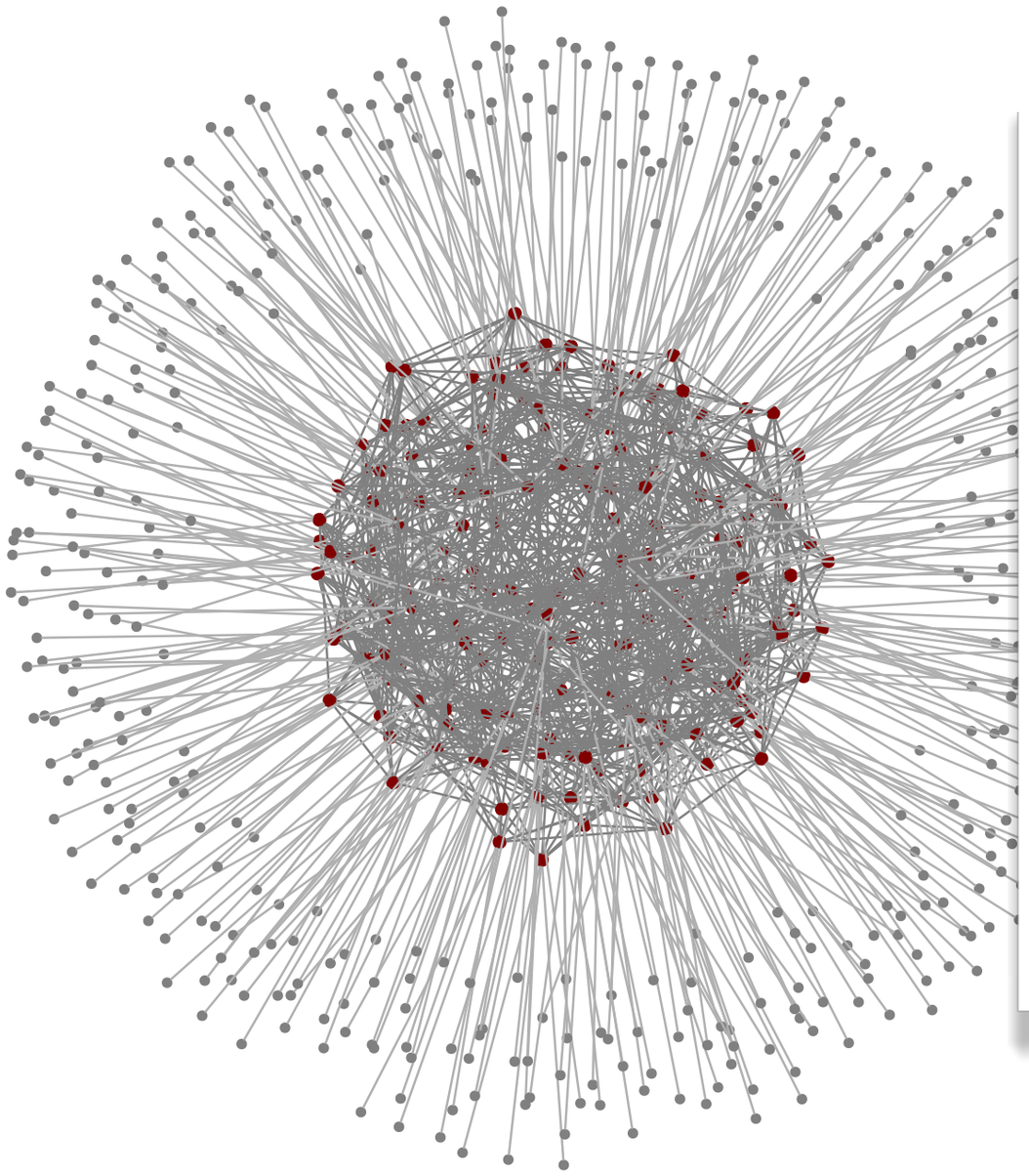
We are happy to stand on the shoulders of giants.

Perhaps not literally giants, but Mark Handley *is* pretty tall...

Cabling



Cabling



[Photo: Javier Lastras / Wikimedia]

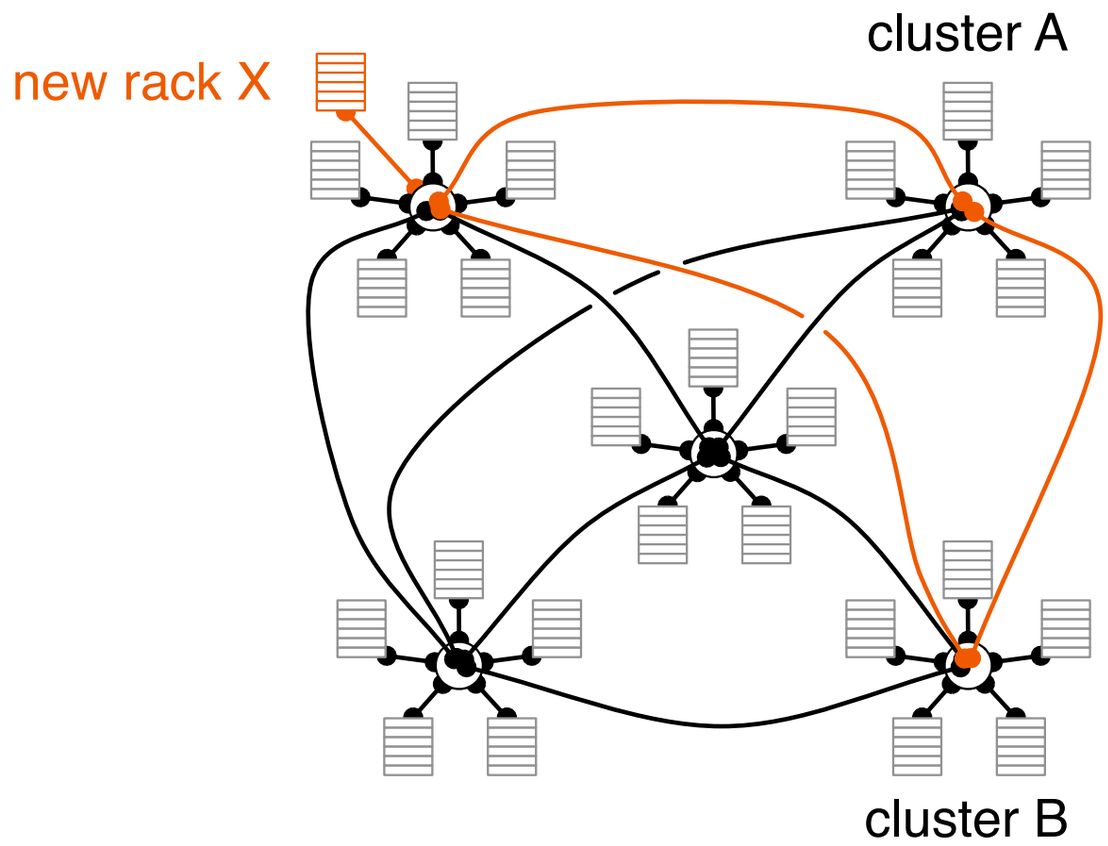
You'll note that Jellyfish bears more than a passing resemblance to a bowl of spaghetti.

Cabling solutions

Fewer cables

for same # servers as fat tree

Aggregate bundles



Avoid long cables

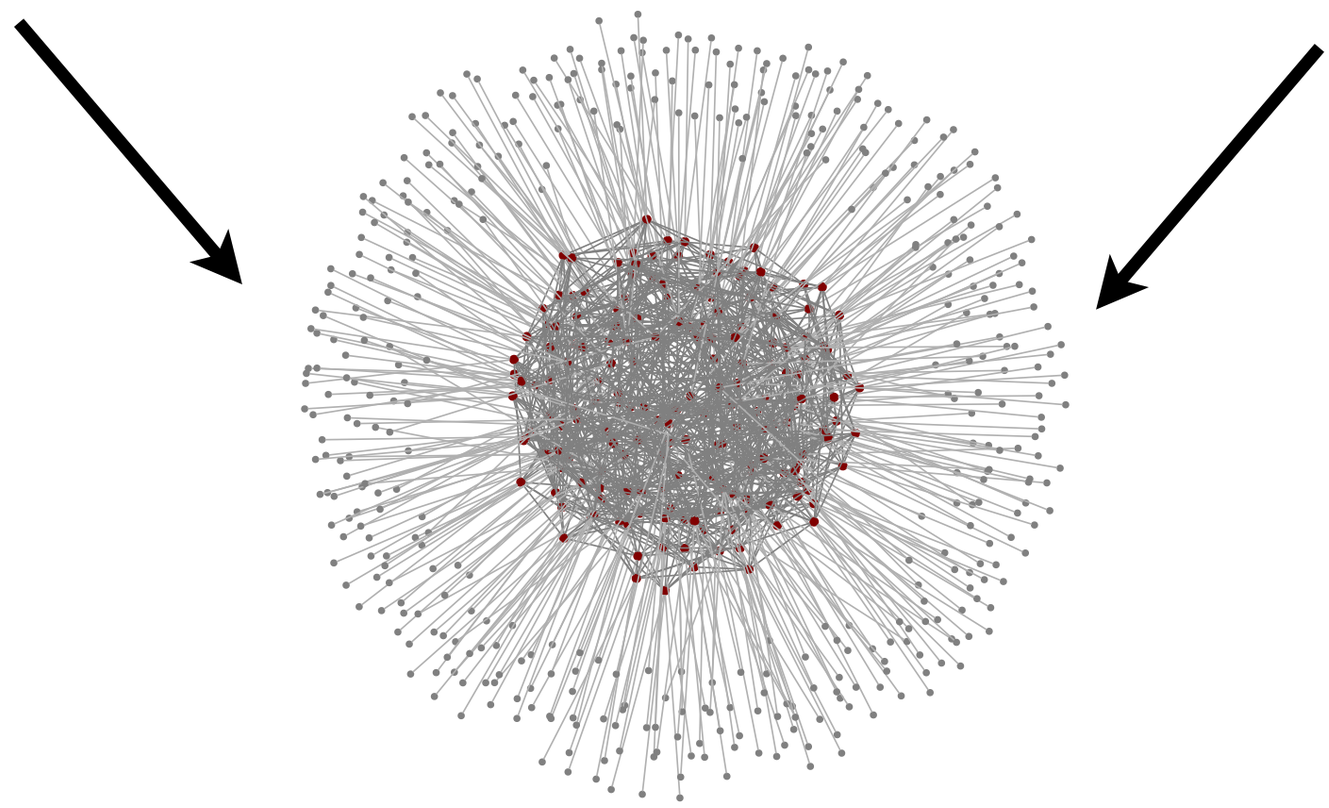
< 5% loss of throughput

It might seem that randomness means there's no way to organize cables. But we note that (1) Jellyfish has about 20% fewer switches and cables than an equivalent fat tree with the same number of servers, (2) It is possible to cluster servers in a 'pod' or perhaps a container, and run bundles of cables between the pods, (3) cable length is also an issue since long cables can be significantly more costly; but we can restrict the number of short vs. long cables to match the fat tree with less than 5% throughput loss (details omitted).

Conclusion

High throughput

Expandability



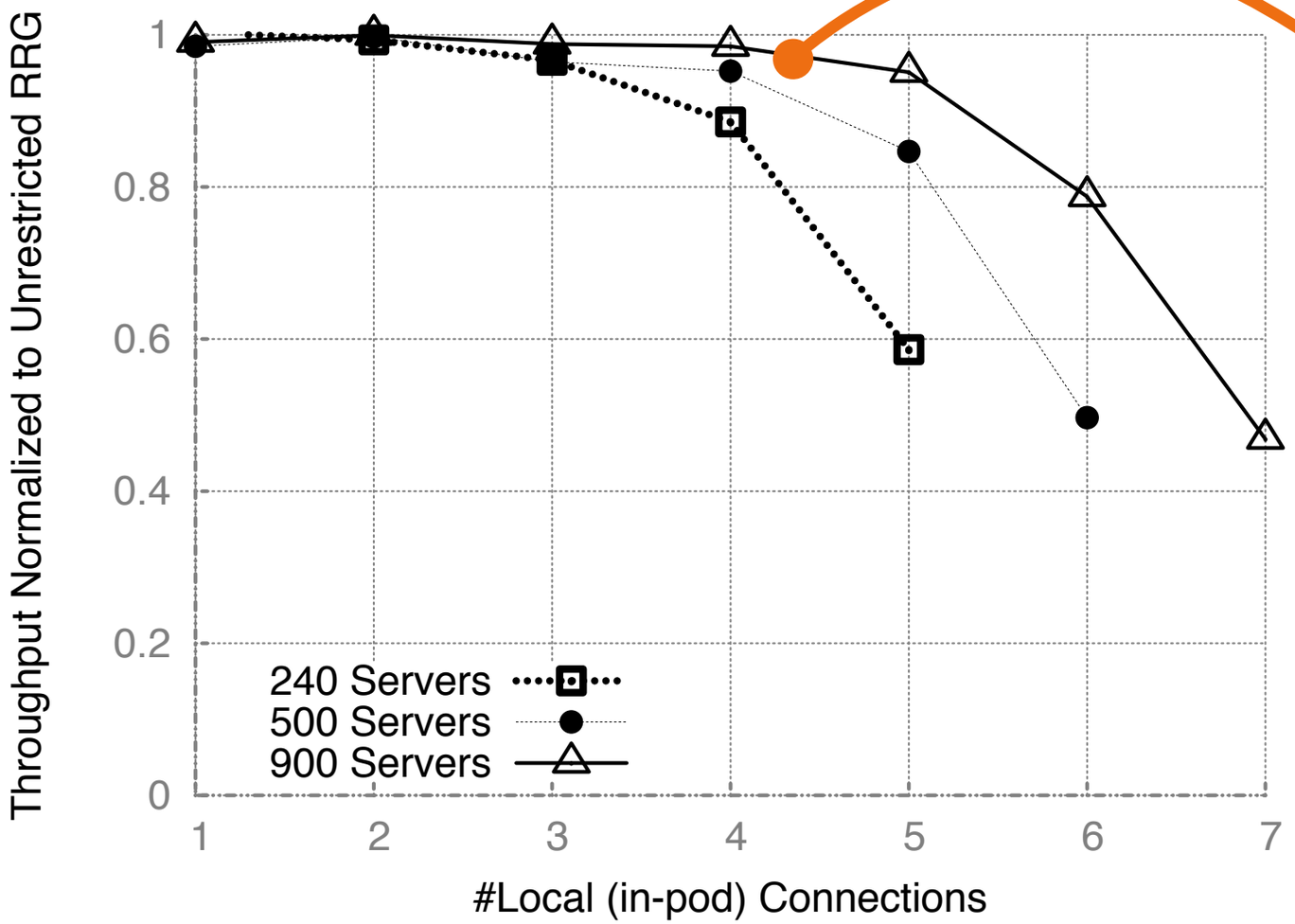
Sometimes in systems design you have to carefully navigate a tradeoff space. But here it seems that we can get the best of both worlds.

Backup

Cabling geometry

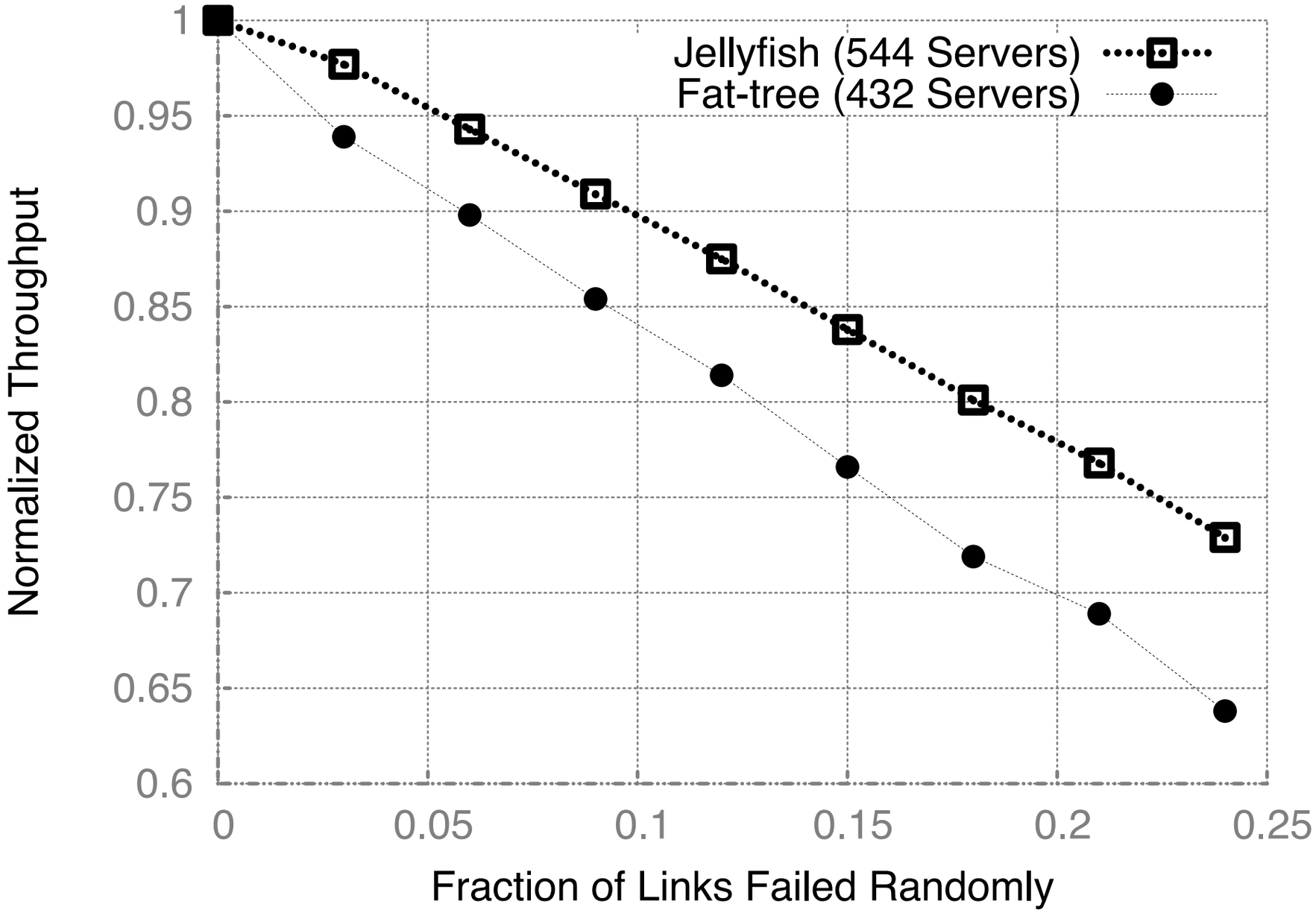
Long optical cables: cost += ~$200

Idea: random with constraint on # of long cables

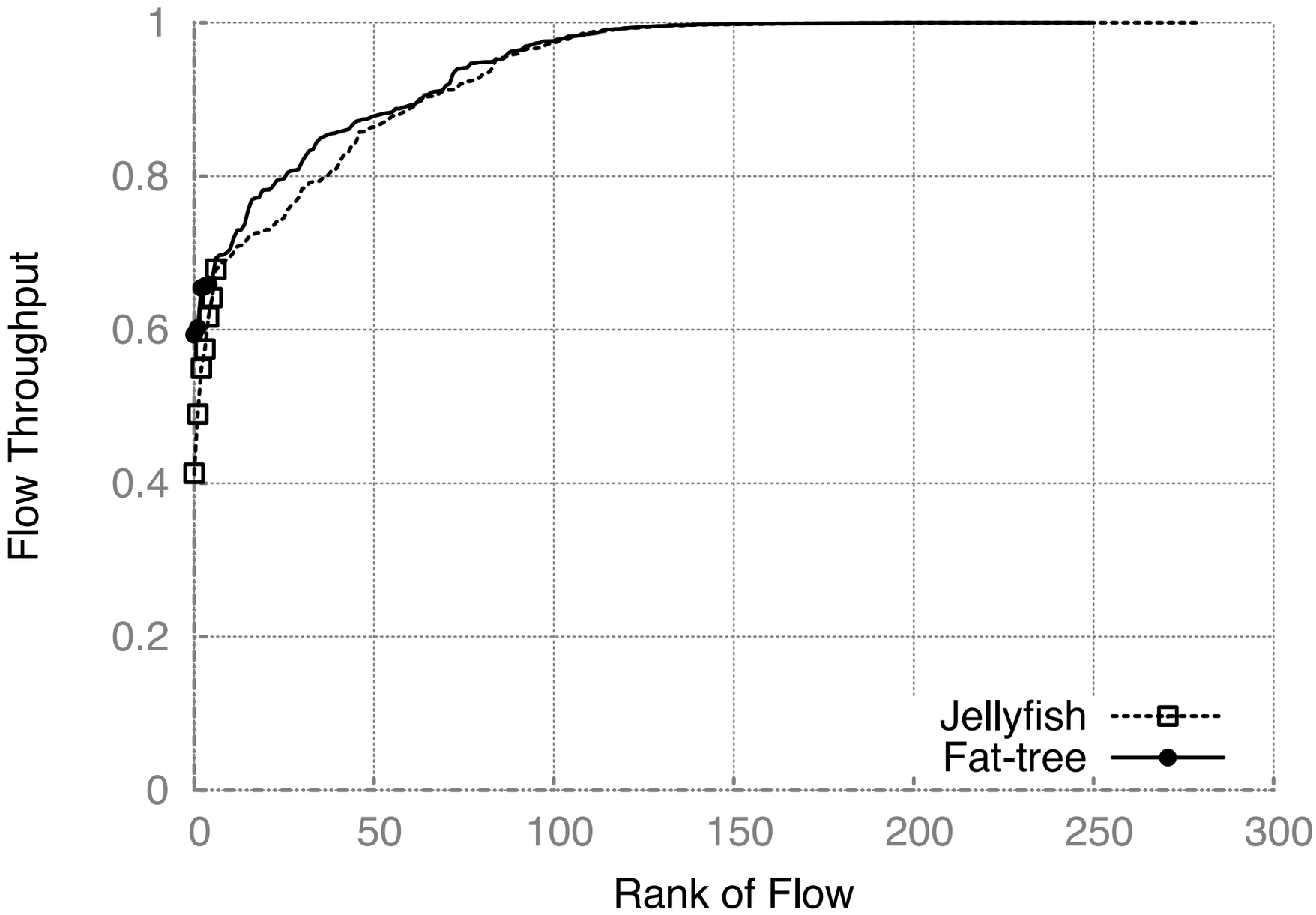


< 5% throughput loss with same equipment *and* cable lengths as fat tree

Robustness


Jellyfish (544 Servers)
Fat-tree (432 Servers)
Normalized Throughput
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0
0.05
0.1
0.15
0.2
0.25
Fraction of Links Failed Randomly

Fairness


1
0.8
0.6
0.4
0.2
0
0
50
100
150
200
250
300
Flow Throughput
Rank of Flow
Jellyfish
Fat-tree