Reproducing “Jellyfish: Networking Datacenters Randomly”

Bruce Spang, 4/14/2020

Plan

Paper overview

Assignment overview

Your Questions

Jellyfish: Networking Data Centers Randomly

Ankit Singla[†,*], Chi-Yao Hong[†,*], Lucian Popa[♯], P. Brighten Godfrey[†]
[†] University of Illinois at Urbana–Champaign
[♯] HP Labs

Abstract

Industry experience indicates that the ability to incrementally expand data centers is essential. However, existing high-bandwidth network designs have rigid structure that interferes with incremental expansion. We present Jellyfish, a high-capacity network interconnect which, by adopting a random graph topology, yields itself naturally to incremental expansion. Somewhat surprisingly, Jellyfish is more cost-efficient than a fat-tree, supporting as many as 25% more servers at full capacity using the same equipment at the scale of a few thousand nodes, and this advantage improves with scale. Jellyfish also allows great flexibility in building networks with different degrees of oversubscription. However, Jellyfish's unstructured design brings new challenges in routing, physical layout, and wiring. We describe approaches to resolve these challenges, and our evaluation suggests that Jellyfish could be deployed in today's data centers.

1 Introduction

A well provisioned data center network is critical to ensure that servers do not face bandwidth bottlenecks to utilization; to help isolate services from each other; and to gain more freedom in workload placement, rather than having to tailor placement of workloads to where bandwidth is available [22]. As a result, a significant body of work has tackled the problem of building high capacity network interconnects [6, 17–20, 38, 42, 43].

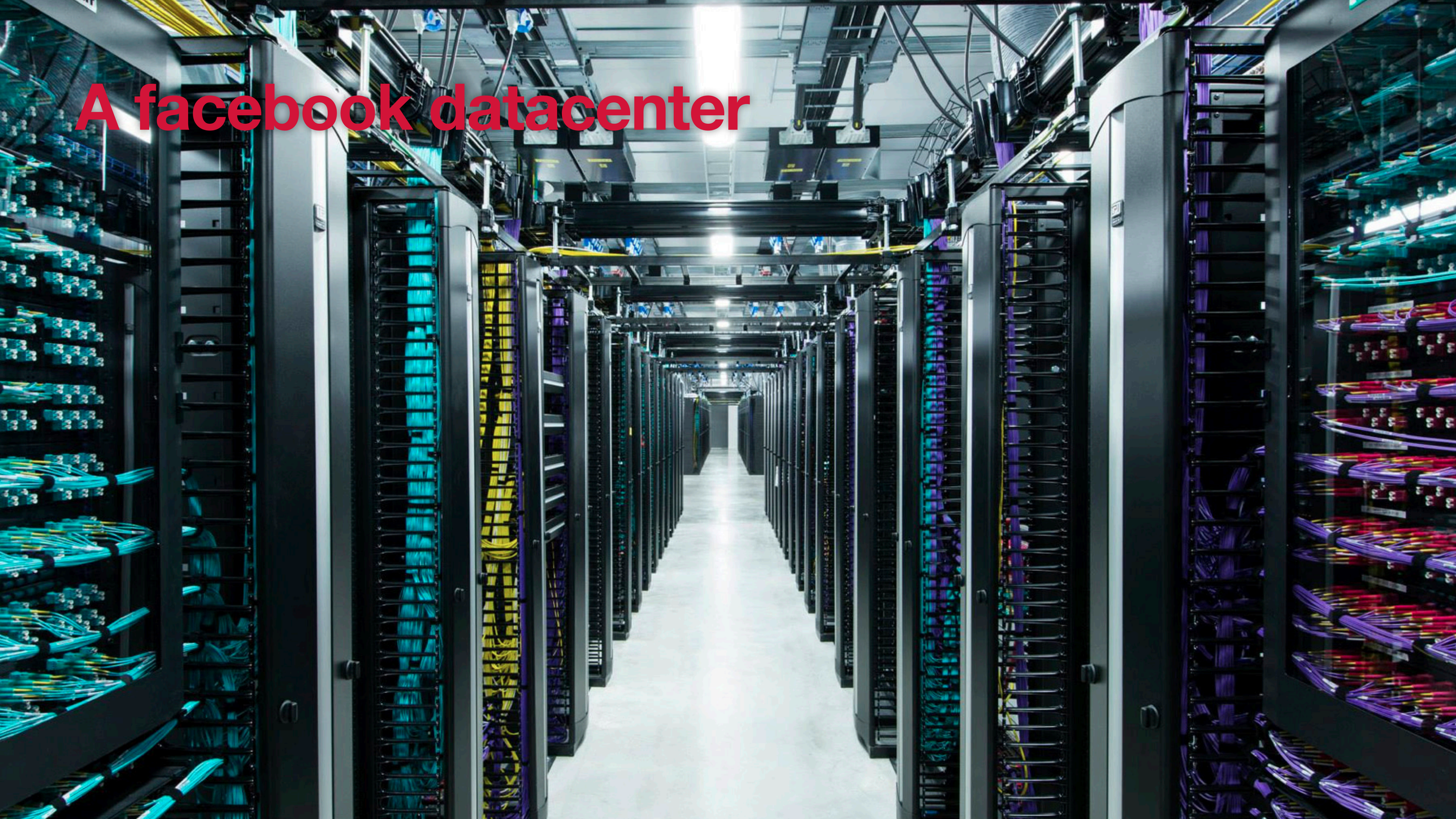
One crucial problem that these designs encounter is incremental network expansion, *i.e.*, adding servers and

Industry experience indicates that incremental expansion is important. Consider the growth of Facebook's data center server population from roughly 30,000 in Nov. 2009 to >60,000 by June 2010 [34]. While Facebook has added entirely new data center facilities, much of this growth involves incrementally expanding existing facilities by "adding capacity on a daily basis" [33]. For instance, Facebook announced that it would double the size of its facility at Prineville, Oregon by early 2012 [16]. A 2011 survey [15] of 300 enterprises that run data centers of a variety of sizes found that 84% of firms would probably or definitely expand their data centers in 2012. Several industry products advertise incremental expandability of the server pool, including SGI's IceCube (marketed as "The Expandable Modular Data Center" [5]; expands 4 racks at a time) and HP's EcoPod [24] (a "pay-as-you-grow" enabling technology [23]).

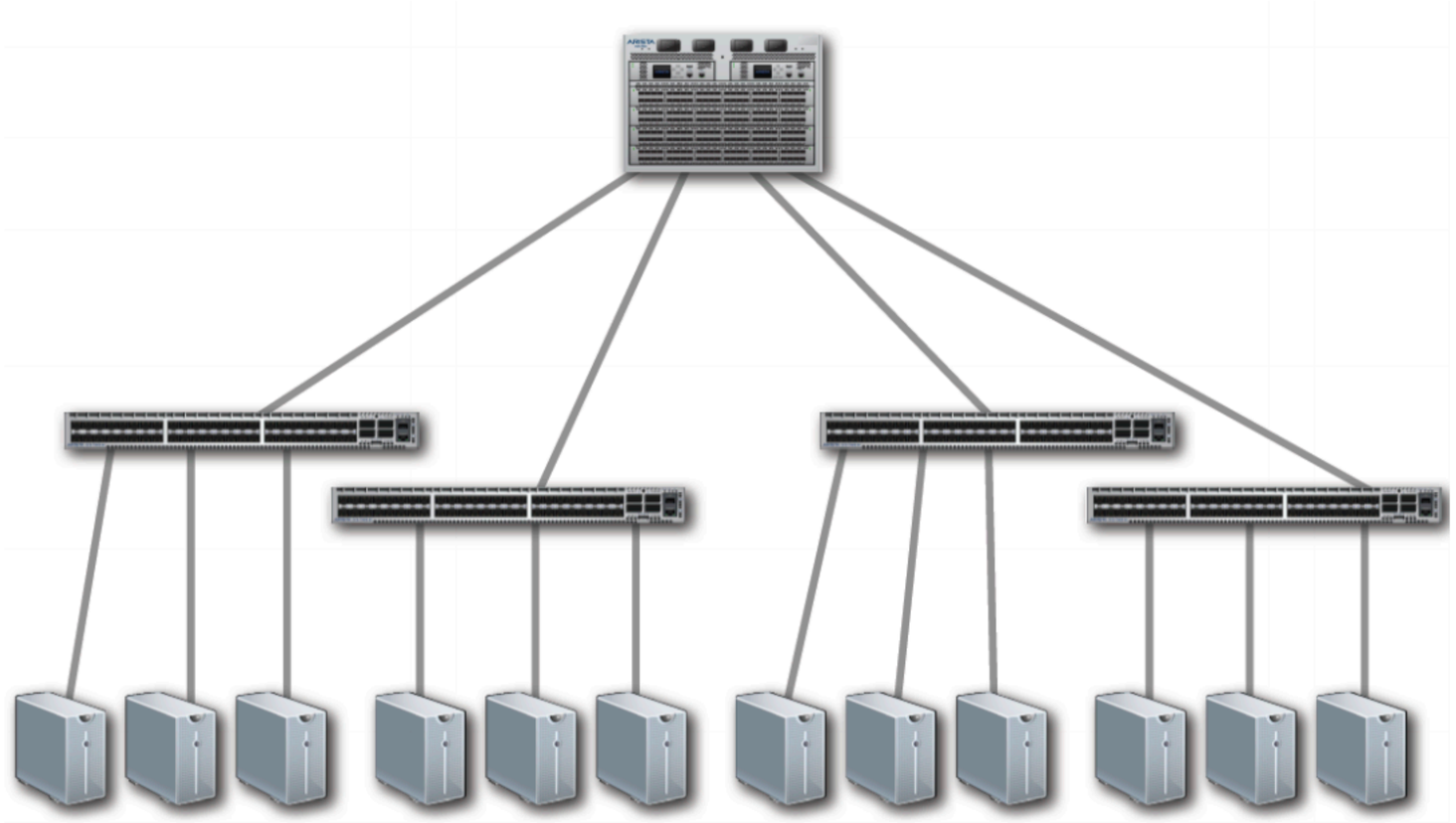
Do current high-bandwidth data center network proposals allow incremental growth? Consider the fat-tree interconnect, as proposed in [6], as an illustrative example. The entire structure is completely determined by the port-count of the switches available. This is limiting in at least two ways. First, it makes the design space very coarse: full bisection bandwidth fat-trees can only be built at sizes 3456, 8192, 27648, and 65536 corresponding to the commonly available port counts of 24, 32, 48, and 64[1]. Second, even if (for example) 50-port switches were available, the smallest "incremental" upgrade from the 48-port switch fat-tree would add 3,602 servers and would require replacing every switch.

There are, of course, some workarounds. One can replace a switch with one of larger port count or oversub-

A facebook datacenter



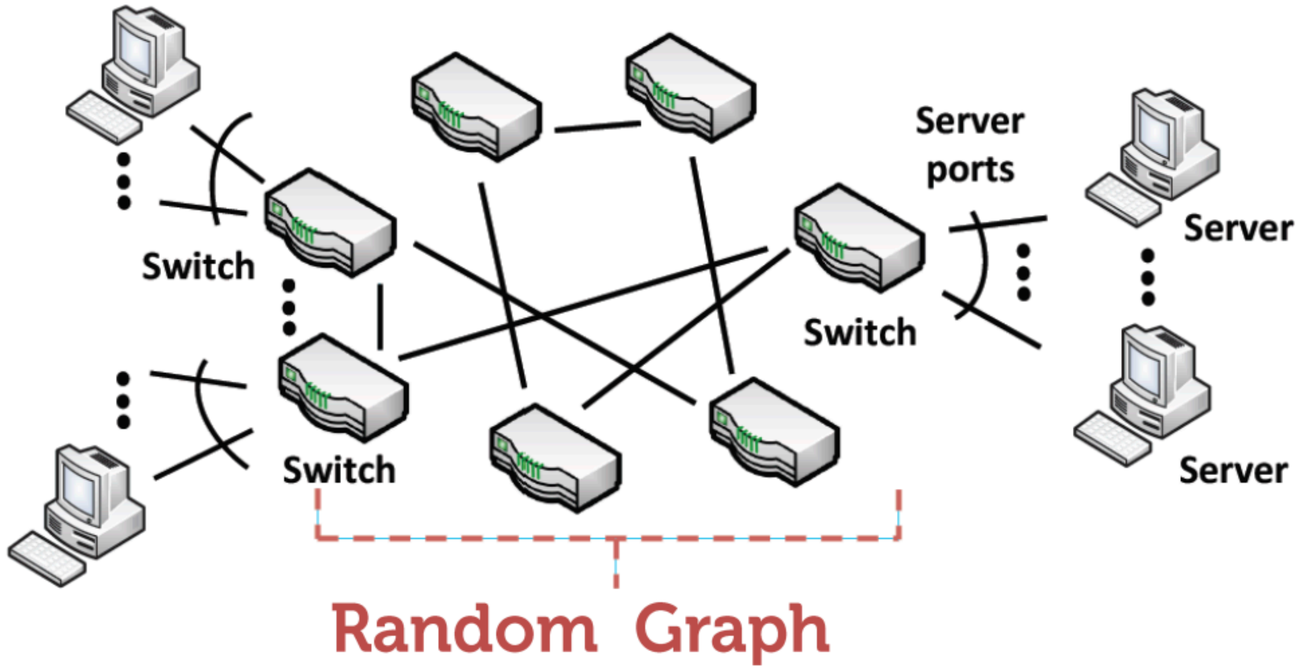
Conceptual datacenter



Paper goals

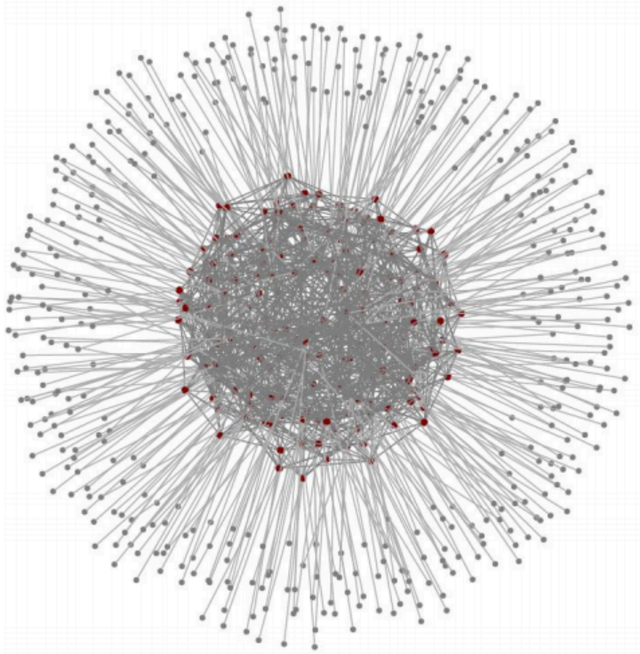
- High throughput
  - Allow VMs to live anywhere
- Incremental expansion
  - Easily add servers and switches

Jellyfish topology

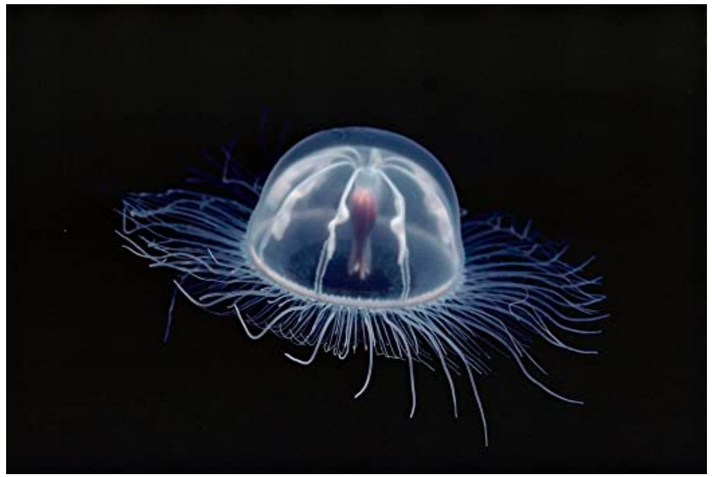


- Pick a random regular graph for core of network
- Each node in graph is a switch
- Hosts attach to each switch

A larger network



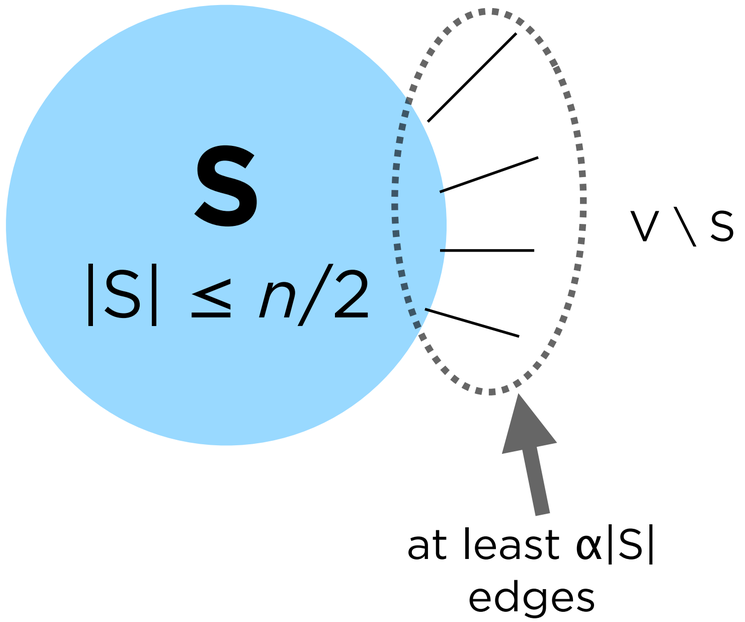
Jellyfish random graph
432 servers, 180 switches, degree 12



Jellyfish
Arctapodema

Larger context: graph expansion

def. G is an α-edge expander if all sets of vertices S of size at most n/2 nodes have a boundary of at least α|S| edges



Intuition: graph is *very* well connected

Graph expansion is cool!

- About a million theory applications
  - Derandomization, constructing error correcting codes, complexity
- Lots of nice properties (short paths, random walks mix quickly, robust to edge removal, etc…)
- Easy to find

More context: nobody does this

- Google has Jupiter architecture (will read later)
- Facebook has another, similar architecture
- etc…
- Something to keep in mind while you work on this: why is that?

Reproducing “Jellyfish: Networking Data Centers Randomly”

Due date: Monday, April 27 at 5pm

In this assignment, we will guide you through a reproduction of the paper “Jellyfish: Networking Data Centers Randomly” written by Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey which appeared at NSDI 2012. Our goal for this project is to introduce you to an interesting paper, demonstrate what it means to reproduce a research paper, and to prepare you for the final project.

What you’ll need to submit on Canvas

- A write-up (formatting/file type/etc… is up to you) which includes:
  - A paragraph or two summarizing what you found when reproducing the paper, and anything you’ve learned
  - Answers to the questions in part 1
  - Your reproduced figures from part 2 and 3, as well as descriptions of how you reproduced the figures
- Please commit any code you write to a private git repository, share the git repository with us (github: brucespang, bitbucket: bspang), and link to it from the top of your write-up. Github has free private repos you can use.
- An explanation of how to run your code in the README file in the repo.

Grading

- We’ll evaluate your writeup based on correctness, how well you reproduce the figures, and writing quality.
- We will run the code by starting with the provided virtual machine setup, cloning your repository, and following your README. We will evaluate how easy it is to recreate your results, and check the figures we get by following your instructions.
- As we’ll mention in more detail in Part 2, the goal here is to understand the Jellyfish paper and verify the claims of the paper. Your graphs may or may not look exactly like the paper. You will get full credit if you make a strong attempt to reproduce the figures, explain what you did, and justify the choices you made.

Starter code: We’ve provided starter code available on github. Instructions for running the code are in the README.

Note: This project will involve a lot of programming. We’ve provided some starter code which we encourage you to use, but you are welcome to use any language or library you prefer (provided we can replicate your figures). Googling things and looking at online resources is

Summary

- **Due date:** Monday, April 27, 5pm PST (w/o extension)
- **Requirements:**
  - Write-up
  - Source code in a private git repo
- Work alone
- Submit on canvas

What you'll need to do

- Answer short-answer questions about the paper
- Reproduce some key figures from the paper
- Extend the paper with Mininet

Paper reproduction

Goals

- Learn a lot about the details of the paper, and any limitations
- Get research experience with less risk
- Contribute to the networking community–tools, knowledge

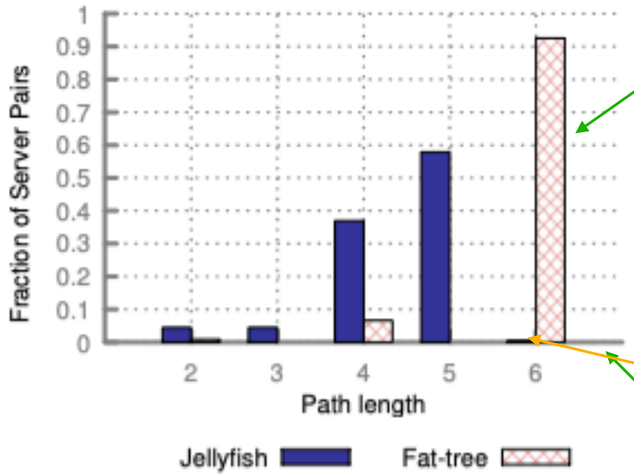
Paper reproduction

More goals

- **Don’t** need to get exactly the same graphs, but do your best
- We’d like you make good/pragmatic choices and justify them

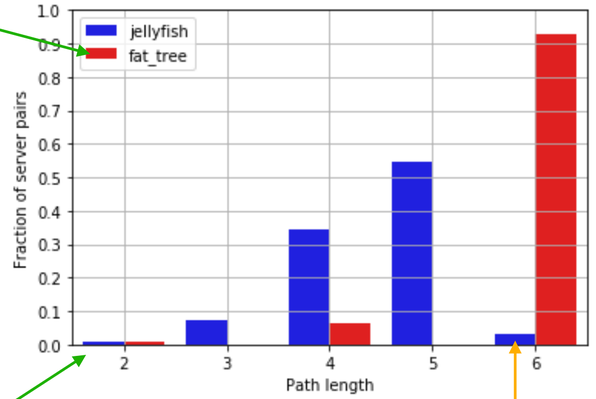
Example reproduced figure

Paper's figure 1c



Same colors

Our figure 1c



Same axes

Not exactly the same

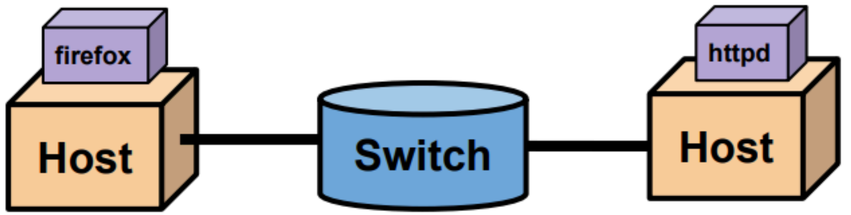
Part 3: implement in mininet

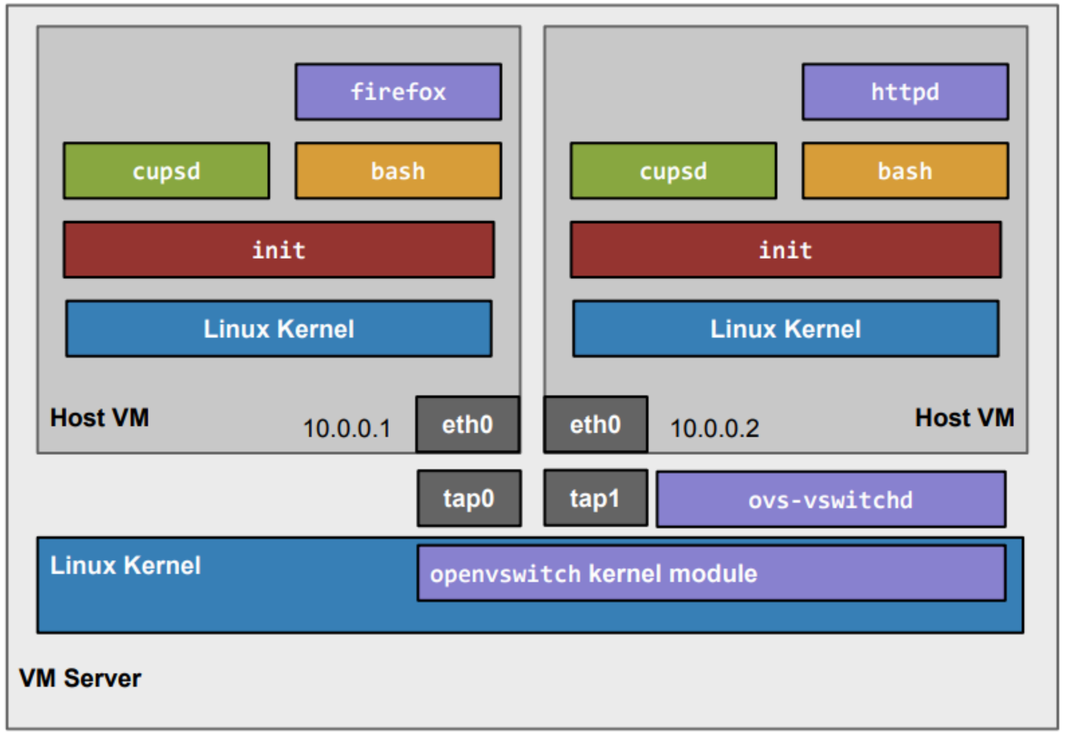
- Learn whether Jellyfish can work in a somewhat more realistic environment
- Learn how to use mininet!

Mininet

Mininet is a virtual network that you can run on your laptop

Mininet


firefox
Host
Switch
httpd
Host


firefox
cupsd
bash
init
Linux Kernel
Host VM
10.0.0.1
eth0
httpd
cupsd
bash
init
Linux Kernel
eth0
10.0.0.2
Host VM
tap0
tap1
ovs-vswitchd
Linux Kernel
openvswitch kernel module
VM Server

Mininet

Platform	Advantages	Disadvantages
Hardware testbed	Fast, accurate	Expensive, shared resource, hard to use
Simulator (e.g. ns2/ns3)	Cheap, easy to modify, sometimes faster than real life	Not real, maybe not believable
Emulator (e.g. mininet)	Cheap, reasonably fast/easy to modify/accurate	Usually slower than hardware, possibly inaccurate

Mininet example

```
vagrant@ubuntu-bionic:/vagrant$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

Summary

- **Due date:** Monday, April 27, 5pm PST (w/o extension)
- **Requirements:**
  - Write-up
  - Source code in a private git repo
- Work alone
- Submit on canvas

Questions?