Making Cloud-Provider Networks Better

Jeff Mogul
Google Platforms/Network Infrastructure
mogul@google.com
Agenda

- Some definitions
- How do cloud providers create virtual networks?
- How do we build a scalable, reliable data-center network?
- What problems are still unsolved?
  - Predicting the bandwidth requirements for cloud tenants
  - Inferring the latency requirements for cloud applications
Disclaimers & Credits

- Not all cloud providers do things exactly the same way
- There are a lot of things about Google that I can’t tell you
- I’ve borrowed a lot of material from other (mostly) Google talks:
  - **B4: Experience with a Globally-Deployed Software Defined WAN**
    - by lots of Googlers; talk by Amin Vahdat, at SIGCOMM 2013
  - **Cicada: Introducing Predictive Guarantees for Cloud Networks**
    - by Katrina LaCurts, J. Mogul, H. Balakrishnan, Y. Turner, at HotCloud 2014
  - a talk by Amin Vahdat at the **Open Networking Summit (ONS 2014)**
    - Recording is on [YouTube](https://www.youtube.com) (search for “Vahdat ONS 2014”)
  - **Inferring the Network Latency Requirements of Cloud Tenants**
    - by J. Mogul & R. Kompella, HotOS 2015
  - **Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network**
    - by Arjun Singh and many others, SIGCOMM 2015
  - **The Rise of Cloud Computing Systems**
    - Jeff Dean’s talk from the SOSP ‘15 **History Day Workshop**
Some definitions
There’s more than one kind of cloud

- **Infrastructure-as-a-Service (IaaS)**
  - Provider offers virtual computers, storage devices, and networks
  - Customer provides all the software, from the operating system to the applications
  - Examples: Amazon EC2, Google GCE

- **Platform-as-a-Service (PaaS)**
  - Provider manages high-level building blocks, makes them reliable and scalable
  - Customer writes code/scripts to glue these together (perhaps w/some IaaS)
  - Examples: Google Dataflow (big-data analytics-as-a-service)

- **Software-as-a-Service (SaaS)**
  - Provider creates and runs the applications
  - Users access applications via Web browser or apps
  - Examples: Salesforce.com (CRM), Gmail, Google Docs

- Many cloud “tenants” will use both IaaS and PaaS at the same time
SLIs, SLOs, SLAs, and nines

Are you getting what you paid for?

- **Service Level Indicator (SLI):** a carefully-defined measurement
  - e.g.: round-trip latency between two VMs, or service uptime
- **Service Level Objective (SLO):** a goal, based on one or more SLIs
  - e.g.: 99.9% of RPCs have a round-trip latency below 500 microseconds
  - e.g.: my storage service is available for use 99.95% of the time, over one month
- **Service Level Agreement (SLA):** an SLO with consequences
  - e.g.: if you don’t meet the latency SLO, you have to refund double what I paid you

Availability SLA is often stated in terms of “nines”

- For example, “5 nines” means that the SLA guarantees 99.999% uptime
- ... which is 5.26 minutes of downtime per year, or 864 milliseconds per day
What’s a “virtual network”?

A virtual network is to a real network as a virtual machine is to a real computer:

- In both cases, an abstraction that
  - preserves the important aspects from the user’s point of view
  - hides the boring details of the underlying “real” implementation
  - allows the provider to efficiently allocate resources among tenants
  - supports isolation (security and performance) between the tenants
- Typically, an IaaS tenant has a number of VMs connected by a virtual network
  - The provider maps this structure onto its underlying real network
  - This mapping is seldom 1:1
- PaaS tenants also usually have virtual networks, connecting to PaaS services
Implementing virtual networks
VMs and virtual networks in action
What do we need from virtual networks?

- Almost all IaaS and PaaS cloud tenants need to connect multiple things:
  - VMs  [or “containers”, but for simplicity, I will ignore that in this talk]
  - PaaS services
  - Internet users
  - On-premises systems

- Cloud tenants want:
  - Predictable, high performance and availability
  - Flexible scaling and re-arrangement of their virtual networks

- A cloud provider needs to:
  - Enforce isolation between tenants, and protect them and itself against attacks
  - Meet its SLAs for network availability and performance
  - Collect billing-related information
How do we build a cloud virtual network?

We need to take care of several different problems:

- Functional isolation: Stanford’s VMs cannot talk to Berkeley’s VMs
  - Isolation even within a tenant, student VMs cannot talk to grading-system VMs
- Performance isolation: Stanford doesn’t interfere with Berkeley’s throughput
- Connecting VMs to the Internet ... safely
- Connecting VMs to PaaS services

[plus others, but not enough time to discuss them all]
Building a virtual network: functional isolation

Basis of functional isolation:

- Virtualize the addresses: if I can’t address something, I can’t send it packets.
- Implementation:
  - Encapsulate tenant packets for transit through provider’s network
    - e.g.: VXLAN or GRE
  - Hypervisor maps tenant endpoint addresses onto proper encapsulation
Centralized control over isolation
A digression: Software-Defined Networking (SDN)

One-slide summary of SDN:

- **Separate traditional routers into “control plane” and “data plane”**
  - “data plane”: hardware (or software) that examines and/or updates every packet
  - “control plane”: software that configures the data plane
- **Well-defined protocol between control & data planes**
  - OpenFlow is an important standard, but not the only choice
- **Move control plane out of tiny switch CPUs and into real “controller” servers**
  - Logically centralized, but typically a distributed system (for scale & availability)

In cloud networks, we typically use a “distributed data plane”:

- relatively dumb hardware switches for aggregation of lots of traffic
- SW-based switching pushed into each server, in the hypervisor layer
  - So: switching resources scale along with computing resources
Functional isolation: summary

The provider achieves functional isolation by:

- Encapsulating/decapsulating tenant packets in the hypervisor layer
  - Tenants cannot talk to each other (except via explicit routers)
  - Each tenant can have its own IP address space(s)
  - Even within a tenant, VMs from different subnets are isolated
- Centralized controller manages the maps in each hypervisor
  - Yes, SDN is real
Building a virtual network: performance isolation

Network performance isolation means:

- Providing a network-performance guarantee to each tenant
- Ensuring that no tenant can undermine another’s guarantee

Two problems:

- How to define a guarantee?
- How to enforce isolation so that no tenant can harm another?
  - Also: so that no tenant can harm the provider

[These are both open research problems, to some extent]
How to define performance guarantees

We have a lot of choices to consider:

- per-tenant, per-VM, or per source->destination path?
- Cloud-internal, Cloud-to-Internet, or both?
- throughput, latency, or both?
- hard guarantee, or probabilistic? ("we promise 1Gb/sec, 99% of the seconds")
- "work-conserving" or not?

Shameless plug: a paper that discusses these options

*What we talk about when we talk about cloud network performance*

J. Mogul & Lucian Popa
SIGCOMM Computer Communication Review, October 2012
What performance guarantees can you get?

Typically, cloud providers today do not provide fancy guarantees; you could get:

- guaranteed-not-to-exceed (i.e., “no guarantee, but we will try to be nice”)
  - often, in proportion to the size of the VM you are renting
- pay a lot, and get reserved physical-network capacity
- probabilistic net throughput to the Internet
- or variations those

One problem: tenants often do not know what they need

- In theory, providers could help with that
  - [caution: my pet research interest … I’ll cover this later]
How to implement performance guarantees

Typically a provider does this in three steps:
1. Decide if I have enough resources to “admit” a tenant’s requested guarantee
2. Compute “rate limits” for each VM, so that all guarantees can be met
3. Enforce the rate limits, so that no VM sends faster than it should
(For simple guarantees, or “no guarantee”, the first 2 steps can be trivial)

Enforcing rate limits:
- Controller tells each hypervisor the transmit-rate limit for each VM
- Hypervisor interposes on packet-output path, to slow it down if necessary
  - Typically done with a “token bucket” algorithm
In theory, just connect a virtual network to an Internet-connected router.

In practice, the provider must handle many issues:

- Internet bandwidth is expensive (so: rate-limit it and bill for it)
- Avoid sending too much, too far (so: provide a CDN for “edge caching”)
- The Internet is full of bad things (so: firewall it and prevent DoS attacks)
  - Some of those bad things may be other tenants inside the provider’s cloud
  - + “Virtual Private Networks” (VPNs) for secure connections to on-prem systems
- Individual Internet links are unreliable (so: support adaptive re-routing)
- Someone has to tell users how to reach your VMs (so: advertise BGP routes)
- Users are all over the world (so: carefully choose egress points)

This is the kind of complexity that a cloud provider should hide from its tenants
- although you’ll probably pay extra for “premium” services
Implementing protected Internet connectivity

In “real networks”, we typically use “middleboxes” to add safety
- e.g., firewalls, intrusion-detection systems, specialized router features

But: middleboxes are expensive, inflexible, and hard to scale

Industry is moving towards “Network Function Virtualization” (NFV)
- NFV is to middleboxes as VMs are to real computers
  - In particular, we can push many of these functions into the hypervisors
  - This gives us easy scaling, and supports rapid evolution of complex NFV features
  - We can use SDN-style controllers to manage these distributed NFV functions
Adding NFV to the picture

The Internet

Internal Network

ToR 10.1.1/24
ToR 10.1.2/24
ToR 10.1.3/24
ToR 10.1.4/24

NVEN: 10.1.124

VNET: 192.168.32/24

VNET: 5.4/16

Host VMM

VM

VM

centralized controller

Load Balancing
DoS
ACLs
VPN

NFV

The Internet
Building a virtual network: PaaS connectivity

Tenant VMs usually need to connect with provider’s “platform services”

- storage; databases; big-data tools; language-translation services; etc.

This exposes many of the same issues as with Internet connectivity:

- We don’t want tenants breaking into the provider’s infrastructure
- We don’t want tenants making DoS attacks against PaaS services
- We want to be able to bill tenants for their use of these services
- We want this connectivity to be fast and reliable

We can use (more or less) the same mechanisms here as for Internet connectivity
The complete picture

VNET: 10.1.124
VNET: 192.168.32/24
VNET: 5.4/16

Load Balancing
DoS
ACLs
VPN

NFV

centralized controllers

Storage & PaaS Services

The Internet

Host VMM

VM
Software engineering for a virtual network

The previous diagram shows a need for building several kinds of software:

- Centralized controller(s)
- NFV functions
- Gateways to storage and PaaS services
- Data-plane functions inside the hypervisor

I’ll focus on the data-plane functions -- our implementation is called Andromeda

- in particular, a little about how we make it fast and efficient
Andromeda transmit datapath, with integrated programmable NFV

- This is a **logical** view of the host functionality
- In reality, we probably want to move some of these functions off of the per-packet path!
  - We really don’t want to do 4+ lookups per packet
- (The receive datapath is somewhat different)
Andromeda datapath goals & techniques

Our goals:
- near-native throughput and latency (uSecs matter!)
- high CPU efficiency

Lead to engineering decisions, such as:
- Datapath pipelined
- & replicated multiple times on one server, as VM resources scale out
- Place functionality in-network, when it’s efficient to do so

VM TX → Firewall / Security → Rate Limiting → Billing
Encap → Routing → Phys TX
Datapath throughput

TCP throughput

- **Gbps/sec**
- **1 stream**
- **200 streams**

- **Baseline**
- **Mar 2014**
- **Oct 2014**
- **Oct 2015**
Datapath throughput

We still have room for improvement
Data center networks
Our datacenters are big ...
Grand challenge for datacenter networks

• Tens of thousands of servers, interconnected in clusters
• 10 years ago: Islands of bandwidth were a bottleneck for Google
  ■ Engineers struggled to optimize for bandwidth locality
  ■ “Stranded” compute/memory resources
  ■ Hindered app scaling

1 Gbps / machine within rack

100 Mbps / machine within small cluster

1 Mbps / machine within datacenter
Grand challenge for datacenter networks

- **Challenge**: Flat bandwidth profile across all servers would
  - Simplify job scheduling, by removing the need for locality
  - Save significant resources, via better bin-packing
  - Allow better application scaling
Motivation

• **Traditional network architectures**
  - Cost prohibitive
  - Could not keep up with our bandwidth demands
  - Operational complexity of “box-centric” deployment

• **Opportunity: A datacenter is a single administrative domain**
  - One organization designs, deploys, controls, operates the n/w
  - ...And often also the servers
Three pillars that guided us

**Merchant silicon**: General purpose, commodity priced, off the shelf switching components

**Clos topologies**: Accommodate low radix switch chips to scale nearly arbitrarily by adding stages

**Centralized control / management**
Basic pattern for Google’s Clos networks

- Edge Aggregation Block 1
- Edge Aggregation Block 2
- Edge Aggregation Block 3
- Edge Aggregation Block 4
- ... Edge Aggregation Block N

- Spine Block 1
- Spine Block 2
- Spine Block 3
- Spine Block 4
- ... Spine Block M

Server racks with ToR switches
Evolution over five generations

- **Firehose 1.0**
- **Firehose 1.1**
- **Watchtower**
- **Saturn**
- **Jupiter (1.3P)**

Bisection b/w (bps)

- 1000T
- 100T
- 10T
- 1T

Year

- '04
- '05
- '06
- '07
- '08
- '09
- '10
- '11
- '12
- '13

(log scale)
Jupiter building blocks

40G to hosts; Scales out to 1.3 Pbps
Challenges we faced in building our own solution

- **Topology and deployment**
  - Introducing our network to production
  - Unmanageably high number of cables/fiber
  - Cluster-external burst b/w demand

- **Control and management**
  - Operating at huge scale
  - Routing scalability / routing with massive multipath
  - Interop with external vendor gear

- **Performance and reliability**
  - Small on-chip buffers
  - High availability from cheap/less reliable components
Challenges we faced in building our own solution

- **Topology and deployment**
  - Introducing our network to production
  - Unmanageably high number of cables/fiber
  - Cluster-external burst b/w demand

- **Control and management**
  - Operating at huge scale
  - Routing scalability / routing with massive multipath
  - Interop with external vendor gear

- **Performance and reliability**
  - Small on-chip buffers
  - High availability from cheap/less reliable components

I only have time for some of these -- see “Jupiter Rising” paper for others
Cable management

- Cable bundling saves 40% TCO
- 10x reduction in fiber runs to deploy
+ Connect externally via border routers
+ Massive external burst b/w
  + Enables (e.g.) cross-cluster MapReduce
+ No need to keep old “cluster routers”
SDN-based routing for a massive network

Replicated “Firepath” controller

Control Plane Network

Firepath Master

FMRP

Interface state update
Link State database
FMRP protocol
eBGP protocol (inband)

“clients” run in each Clos switch

Firepath Client 1
Firepath Client 2
Firepath Client N
(Border router) Firepath Client, BGP 1
(Border router) Firepath Client, BGP M
External BGP peers
Solutions to some other challenges

Interoperation with routers from other vendors:
- Integrate a BGP stack into the border routers

Small on-chip buffers in merchant-silicon switch chips:
- Use ECN on the switches, and DCTCP on the hosts

High reliability from cheap, low-reliability components:
- Exploit the redundancy in multi-path (Clos) network topologies
- Design topologies for diversity
  - e.g., don’t connect a “redundant” pair of links to the same two switch chips
- Implement only the features we need
- Learn, from our outages, how to build a reliable control plane
- Test the control plane in a virtualized testbed (at-scale testing is expensive!)
Research challenges
What’s still hard about cloud networking?

Network design and management
- How do we balance cost, throughput, latency, reliability, and manageability?
- How do tenants design and manage their virtual networks?

Cloud-network performance isolation
- How do we know what guarantees tenants actually need?
- How do we meet these guarantees while efficiently using network resources?
- How do we deal with short-term congestion?

Network capacity management
- How does a provider know how much capacity to install?

High reliability
- How do we design and manage networks with many “nines” of availability?

... and many other problems!
Two research projects on “what do tenants need?”

I’ll focus on my own research, since that’s what I know:

- How do we know what guarantees tenants actually need?

- Cicada: Introducing Predictive Guarantees for Cloud Networks
  - by Katrina LaCurts, J. Mogul, H. Balakrishnan, Y. Turner, at HotCloud 2014

- Inferring the Network Latency Requirements of Cloud Tenants
  - by J. Mogul & R. Kompella, HotOS 2015
Research on cloud network performance guarantees

A little background from MIT + HP Labs, then a new (speculative) Google project
CICADA

PREDICTIVE GUARANTEES FOR CLOUD NETWORKS

Katrina LaCurts
Jeff Mogul
Hari Balakrishnan
Yoshio Turner
MIT CSAIL, Google Inc., HP Labs
CICADA
PREDICTIVE GUARANTEES FOR CLOUD NETWORKS

(Jeff’s five-minute version of Katrina’s HotCloud 2014 talk)

https://www.usenix.org/conference/hotcloud14/workshop-program/presentation/lacurts
Bandwidth guarantees: a promise from the provider to the customer that its VMs will be able to communicate with each other at a particular rate

(informal definition)
POSSIBLE ARCHITECTURE

customer requests machines and guarantee

provider places customer’s VMs, enforces guarantee

what if some pairs of VMs use fewer than 1.7 Gb/s?

what if some pairs of VMs need more than 1.7 Gb/s?

over-provisioned network
increased cost to customer, wasted bandwidth within network

under-provisioned network
poor performance for customer’s application

Problem: how do customers know what guarantee they need?
CICADA’S ARCHITECTURE

cicada makes predictions about an application’s traffic to automatically generate a guarantee

customer makes initial request for machines

provider offers guarantee based on cicada’s prediction

hypervisors send measurements to cicada controller

provider places tenant VMs

provider updates placement based on cicada’s prediction

cicada predicts bandwidth for each tenant
TRAFFIC VARIABILITY

spatial variability: different pairs of VMs transfer different amounts of data

temporal variability: pairs of VMs transfer different amounts of data at different times

HOW DOES CICADA WORK?

Machine learning
Based on observed traffic patterns

(I’m leaving out a lot of this talk!)
DATASET

(in this talk) collected one traffic matrix per hour, for each application

each entry represents the number of bytes transferred between VM\textsubscript{j} and VM\textsubscript{k}

goal: use this dataset to quantify spatial and temporal variability
CICADA’S ALGORITHM

Instead of setting the weights beforehand, cicada learns the weights **online**; they are updated with every prediction based on errors made in previous predictions.

\[
\hat{M}_{n+1} = \mathcal{U}(M_1^1 \cdots M_n^1 w_1, \cdots, M_1^n \cdots M_n^n) + M_2^1 + \cdots + w_n \cdot M_i^n(n) \cdot M_n
\]

Cicada’s prediction algorithm draws inspiration from the following works:
RESULTS  
PREDICTING AVERAGE DEMAND

cicada’s predictions outperform VOC-style predictions (median error decreases by 90%) and require no customer input (71% for L2 error)

cicada occasionally under-predicts; VOC does not because we selected its parameters to avoid under-prediction

cicada’s predictions typically require only 1-2 hours of application history
SUMMARY

cloud applications exhibit variability that existing models don’t capture

![Graph 1]

![Graph 2]

![Graph 3]

cicada captures this variability, and provides guarantees that are

accurate

calculated quickly*

require little history

increase network utilization

improve relative error by 90%

< 10 milliseconds per application

1--2 hours for most applications

in some cases, inter-rack utilization can be doubled

*see paper for this result
Inferring the Network Latency Requirements of Cloud Tenants

Jeff Mogul (Google)
Ramana Rao Kompella (Google + Purdue)

HotOS XV
May 2015
Users care about latency

“Systems that respond to user actions [within 100ms] feel more fluid and natural to users”

-- J Dean & L Barroso, The Tail at Scale, CACM 56(2)

“[Amazon’s] services have stringent latency requirements … measured at the 99.9th percentile”

-- G DeCandia et al., Dynamo …, SOSP 2007

“Amazon found every 100ms of latency cost them 1% in sales”

-- (various non-original sources)
So, cloud tenants might care about intra-cloud latency
So, cloud tenants might care about intra-cloud latency

Cloud provider infrastructure

Internet

NiftySvc.com

VM
VM
VM
VM

Cloud database service

Cloud storage service

(the green arrows in this diagram)
How much do tenants care about intra-cloud latency?
How much do tenants care about intra-cloud latency?

- Providers don’t generally know
How much do tenants care about intra-cloud latency?

- Providers don’t generally know
- Tenants don’t generally know
So why should we care?
So why should we care?

- Our intuition: intra-cloud latency actually does matter to tenants
- So a cloud provider with better latency will have happier tenants
  - that is, more tenants, and/or tenants who are willing to pay more
- But building infrastructure to support low latency isn’t cheap
  - especially for low tail latency, which tends to require low utilization
Goal of our work: how much does latency matter?

How sensitive is a given cloud application (or VM) to the underlying intra-cloud network latency?

Specific focus of our work:
- Within-region vs. WAN latency
  - Intuition: local latency is easier to vary per-tenant
- Techniques requiring little or no help from the tenants
  - Intuition: tenant developers don’t want to be bothered
- Not on how much bandwidth an application uses
  - previous work has looked at inferring cloud bandwidth needs
    - Proteus (Xie et al., SIGCOMM ‘12); Cicada (LaCurts et al., HotCloud ‘14)
Why is inferring latency harder than inferring bandwidth?

Basic technique for inferring bandwidth needs:
- Temporarily turn off rate limiting
- Measure how much bandwidth the application (VM) uses
- Infer future needs from (measured) past behavior

It’s harder to apply this method to latency:
- How do you measure “how much latency the VM uses”? 
What could a provider do with this information?

Balance resource allocations between tenants:
- Use admission control, to avoid over-utilizing the network
- Place VMs to improve locality or reduce interference
  - as in Oktopus (Ballani et al. SIGCOMM ‘11) and Silo (Jang et al. ‘13)
- Rate-limit latency-insensitive tenant VMs
- Use DSCP settings to shift load between switch queues
- Plan infrastructure upgrades/expansions
- Adjust relative prices of VMs and guarantees for BW & latency
- Help tenants understand which provider better suits their needs
How bad is it?

We did a simple study to quantify latency variability

- “simple” means “WARNING: this is bad science”
- Do not attempt to compare providers using this data. Please.

Methodology:

- Buy two cheap VMs in each of several providers
- Run netperf TCP_RR for 60 seconds every 15 minutes
  - netperf reports latency histograms (in a weird way)
Latency results (see warnings on previous slide)

- Latencies can be quite large (at 99th %ile or 99.9th %ile)
- Latencies vary over both short and long time scales
- Latencies *seem* to vary between providers (WARNING: NOT ACTUAL SCIENCE!)
Our approach: Use correlation for latency inference

Goals:
● Infer a causal relationship between network latency and application-level latency (application-level SLO)
● Find threshold below which better network latency doesn't help
● Understand how well the application tolerates latency increases

Approach:
● Measure network latency variations
● Measure SLO effects
● Correlate! Statistics!
  ○ As in Cohen et al., “Correlating Instrumentation Data to System States”, OSDI ‘04
The hard parts:

- Measuring network latency variation
- Measuring SLO variation
- Doing this for a complex multi-VM application
- Doing this without (much) help from the tenants
Measuring network latency variation

Possible approaches:

● Exploit natural variation?
  ○ Only works if there is enough natural variation [maybe]
  ○ How can a VMM actually measure the latency seen by a VM?
    ■ add timestamps to packets? But what if there is no rapid response?
    ■ snoop on TCP headers? But what if no TCP? Or if VM uses IPSEC?

● Inject our own variation?
  ○ We can do it whenever we want (e.g., only for selected VMs)
  ○ No need to match up requests and responses
  ○ We control the frequency and amplitude
  ○ Relatively easy to do at the VMM layer
Measuring network latency variation

Possible approaches:

- **Exploit natural variation?**
  - Only works if there is enough natural variation [maybe]
  - How can a VMM actually measure the latency seen by a VM?
    - add timestamps to packets? But what if there is no rapid response?
    - snoop on TCP headers? But what if no TCP? Or if VM uses IPSEC?

- **Inject our own variation?** ✔
  - We can do it whenever we want (e.g., only for selected VMs)
  - No need to match up requests and responses
  - We control the frequency and amplitude
  - Relatively easy to do at the VMM layer
Measuring SLO variation (without help from tenants)

Options:

● Assume tenant uses provider-supplied load balancer
  ○ but: not all tenants use one, or they use Direct Server Return

● Measure Internet use; assume better results lead to more use
  ○ not always a good indicator

● VMM assumes HLT/MWAIT means VM is waiting for network
  ○ doesn’t work if there’s enough parallelism to keep cores busy

● Hook into cloud-monitoring tools (e.g., Tracelytics or AppDynamics)
  ○ not everyone uses these

We’re still trying to figure out which of these we can use (maybe several?)
How do we know which paths are latency-sensitive?

Load balancer where SLO is measured

Which green arrows are the ones whose network latency affects the overall SLO?
Insight: inject latency variation using pseudo-noise code

- Inject latency using time-varying pattern representing bit-sequence
  - added latency = 1
  - no added latency = 0
- Choose pattern using pseudo-noise (PN) codes
  - A set of PN codes can be chosen to be “highly orthogonal”
    - i.e., minimal correlation between pairs of PN codes
  - Assign one PN code to each latency path (green arrow)
- Correlate time-varying SLO measurements with *known* PN codes
  - This is what GPS receivers do (more or less)
- This should allow us to:
  - Understand which network paths actually matter
  - Separate effects of network latencies from those of service latencies
Cartoon version of PN codes in action

Load balancer where SLO is measured

NiftySvc.com

VM

VM

VM

Cloud database service

Cloud storage service

Correlate

Measured SLO variations

Injected PN codes
Cartoon version of PN codes in action

Load balancer where SLO is measured

NiftySvc.com

VM

VM

VM

Cloud database service

Cloud storage service

Correlate

The culprit!

Measured SLO variations

Injected PN codes
Implementation

Progress so far:
● [crickets]
System diagram

Load Balancer

Application

CPU Monitor
Latency Injector

Application-SLO Measurement
Latency PN Code generator

Correlator

Infrastructure

VM

Hypervisor

Data Center Network
Implementation/research challenges
Implementation/research challenges

Will we be able to measure SLOs efficiently and accurately, without much tenant help?
Implementation/research challenges

Will we be able to inject latency efficiently and accurately?
Implementation/research challenges

Will we be able to generate enough good (highly orthogonal) PN codes?
Implementation/research challenges

Will we be able to find correlations? How quickly?
Implementation/research challenges

With what PN codes (frequency, length, amplitude)? Will these really harm application performance?
Questions on this part?
A word from our sponsor ...
... we’re hiring!
Short links for jobs

Internships:
- MS/PhD SW Eng students: g.co/swegradintern - deadline is Feb 29
- Undergrad SW Eng: g.co/sweintern - deadline was Nov 30
- Hardware Eng: http://g.co/HardwareEngIntern - deadline was Jan 25
- Freshmen/soph “Engineering Practicum”: g.co/engpracticum - deadline was 11/30

Full-time jobs:
- PhDs - g.co/phdgrad
- Bachelors and Masters - g.co/swegrad
- google.com/careers/planet for our team specifically

If you apply: please let me know (mogul@google.com) so I can keep track of you

Questions? stanfordstudents@google.com
Interview Process (Full Time)

1-2 Phone Interviews

5 Onsite Interviews

Committee Review

Offer
Interview Process (Intern)

2-3 Phone Interviews

Engineering Review

Intern Host Matching

Offer
How to Apply

1. Find out what roles might interest you:
   - Visit google.com/careers/planet for our team
   - Visit google.com/careers for a broader search

2. Upload your CV and transcript (Unofficial is fine)

3. Drop me an email (mogul@google.com)
   - So we can track you -- sometimes people with specialized skills need to be steered towards us.