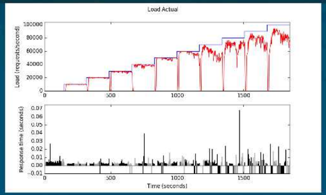
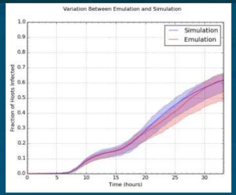
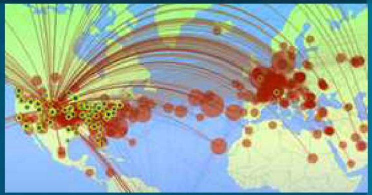
SAND2018-10092PE

Emulytics at Sandia: Emulation-based Modeling and Analysis of Computer Systems



FIREWHEEL





PRESENTED BY

Kasimir Gabert (5838)

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

1. Problem Space
2. Emulation-based Modeling – Tools and Case Studies
3. Model Snapshots
4. Verification and Validation Approaches and Challenges

Scope of Cybersecurity

- (Ambitious) Goal: **eliminate surprise** from our computers
- It is a large field, spanning the whole computer system space: from subtle software / hardware bugs to the motivation of cyber criminals to unintended radiation from physical devices
- Luckily, in order to be useful these systems are necessarily engineered to reduce surprise

Past (Surprising) Events

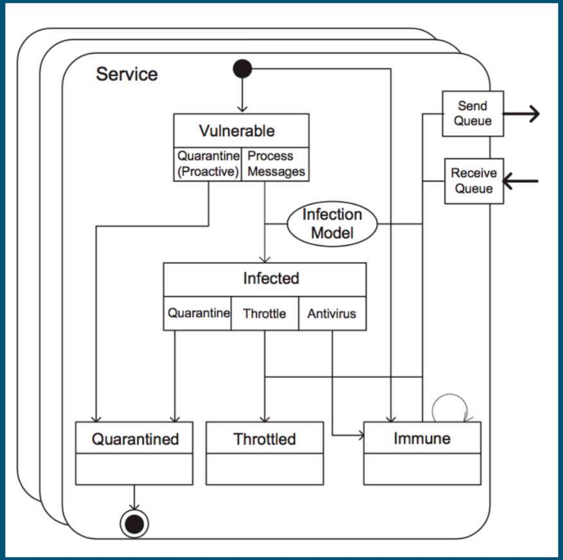
- January 2010 Operation Aurora
  - A series of advanced attacks first reported by Google and aimed at dozens of American companies
  - These attacks took advantage of previously unknown vulnerabilities in Internet Explorer and both exfiltrated intellectual property and accessed email of targeted users
- April 2011 Amazon Outage
  - A configuration error during an upgrade disconnected a large number of storage machines
  - Once the error was fixed the storage systems automatically tried to replicate, causing a re-mirroring storm, thread starvation, and a large system failure
- April 2014 Heartbleed
  - A vulnerability in the OpenSSL library was discovered allowing for arbitrary remote memory to be read, including private keys
  - This impacted a massive number of servers on the Internet.

Purpose of Cybersecurity Models

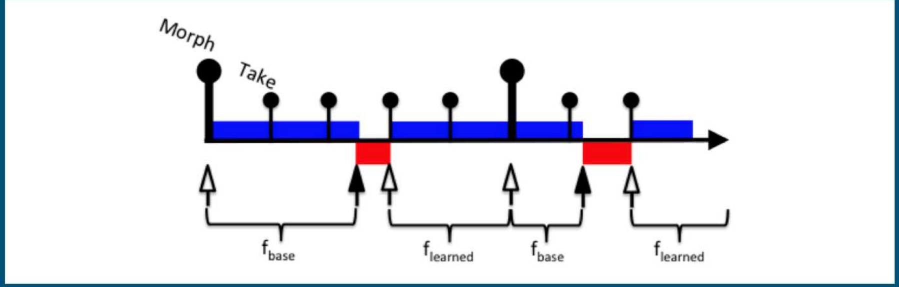
- Similar to cybersecurity: eliminate cyber surprise (using models!)
- We are asked to determine:
  - How a network or system design change might help or hurt (performance, security, usability)
  - How well a network can withstand or protect against an attack
  - Whether a new product will make a set of systems "more secure"
  - What an optimal deployment or design of a system change might be
- Some additional uses:
  - Help system developers prototype as they build
  - Help train individuals or teams
  - Dynamically explore a system with open-ended research questions

Differential Equation/Agent-based/Game Theoretic Models

- Critically, these tend to require a thorough understanding of assumptions and important parts of the system



"Agent-based modeling of malware dynamics in heterogeneous environments", Bose et al. 2011



"Evaluating Moving Target Defense with PLADD", Jones et al. 2015
PLADD: Probabilistic Learning Attacker, Dynamic Defender

$$\frac{dI(t)}{dt} = \beta I'(t) S'(t) - \frac{dR(t)}{dt}$$

"Modeling Botnet Propagation Using Time Zones", Dagon et al. 2006

Emulation-based Models

- We model computers using computers; the line between the "real world" and a "model" is blurry
  - A "real world" production virtual machine is a "model" element after being copied
- Emulation-based models ("Emulytics" at Sandia) take advantage of this blurry line by building models that consist mostly of virtual or physical machines running software pulled from the real world
- This approach can be augmented with simulation at a packet level or with the use of another (non-emulated) model

Model Assumptions / Parameters

- Numerous assumptions – known/unknown, implicit/explicit
  - For every device in the network (from computers to network cables):
    1. The operating system or firmware, from the version to every specific configuration option in every running service
    2. The hardware the device is running on, all the way from rough specifications (amount of memory, speed of processors) down to the specific motherboard, all of the integrated circuits on it, and their initial RAM states
    3. The software running on top of the system and all of its options and compilation / configuration settings
    4. The users' interactions with the device
  - Networks can have thousands of devices in them
  - Model-specific software, for example a tool that mimics a user or a change that artificially speeds up a download to save time
- Unlike other models, a single bit change in many cases can cause a state change

Running an Emulytics Model

- The models are run (at clock rate) on computer systems
- We use virtualization platforms and numerous physical computers, each with their own complete set of assumptions and parameters
- A model may have significantly different output if it is oversubscribing the underlying hardware, experiences contention with other devices, or simply is not scheduled well across the physical cluster
- This adds a degree of non-determinism that does not seem to exist with many other models

Outline

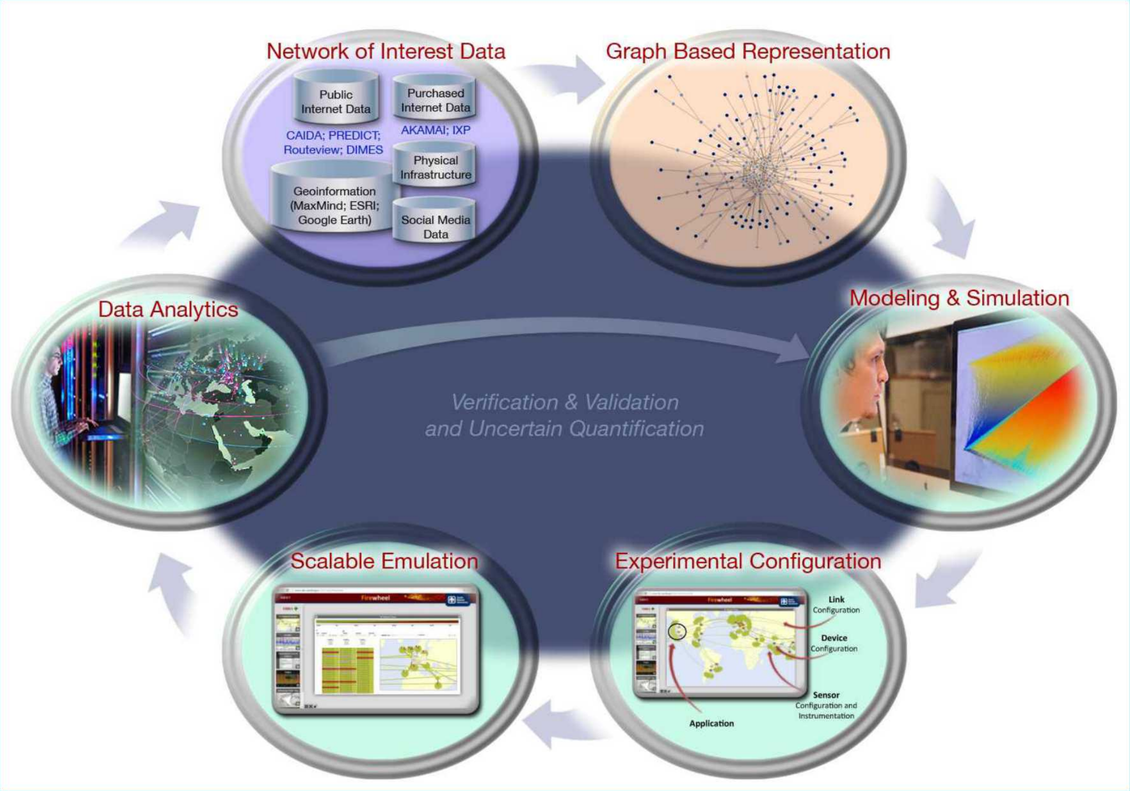
1. Problem Space
2. Emulation-based Modeling – Tools and Case Studies
3. Model Snapshots
4. Verification and Validation Approaches and Challenges

Emulation-based Modeling Tool: Firewheel

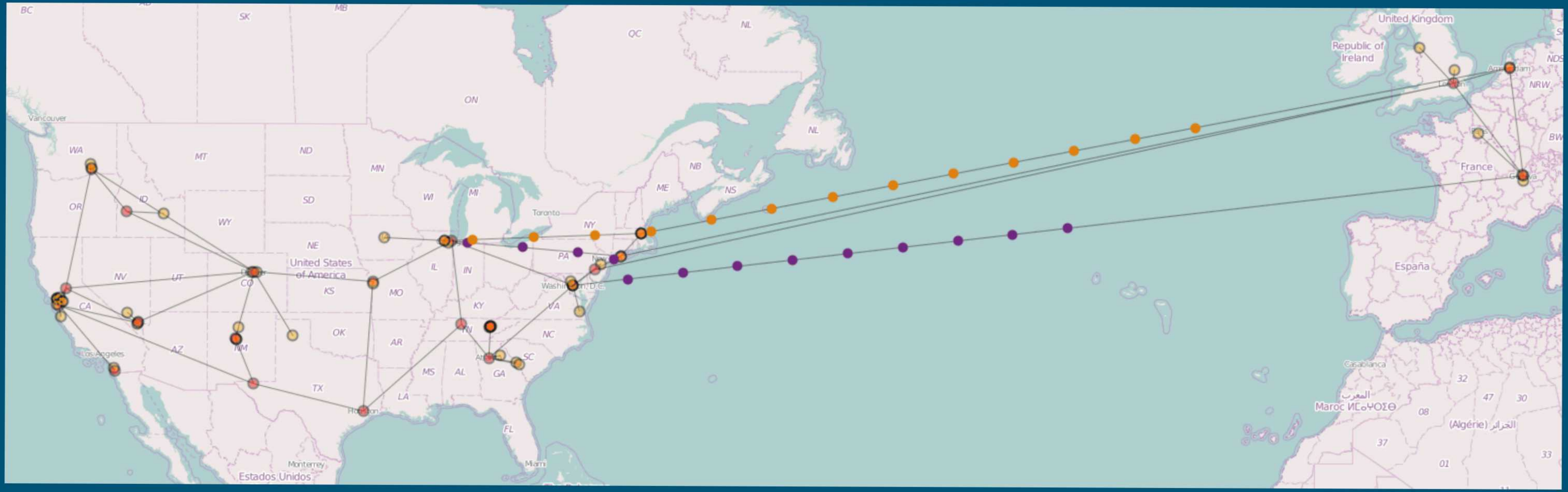
- A platform developed at Sandia that eases the process of building, running, and studying models
- DoD research organization had a need for a very large-scale virtual testbed of 100K+ virtual machines to emulate an Internet-like environment and perform large-scale analytics
- Original requirements
  - Scale up to 100K+ end points running Windows & Linux
  - Fast set up and boot time
  - Perform large-scale data analytics
- No national testbed met these requirements
- Firewheel has since been used on numerous projects for a variety of customers

Emulytics Process

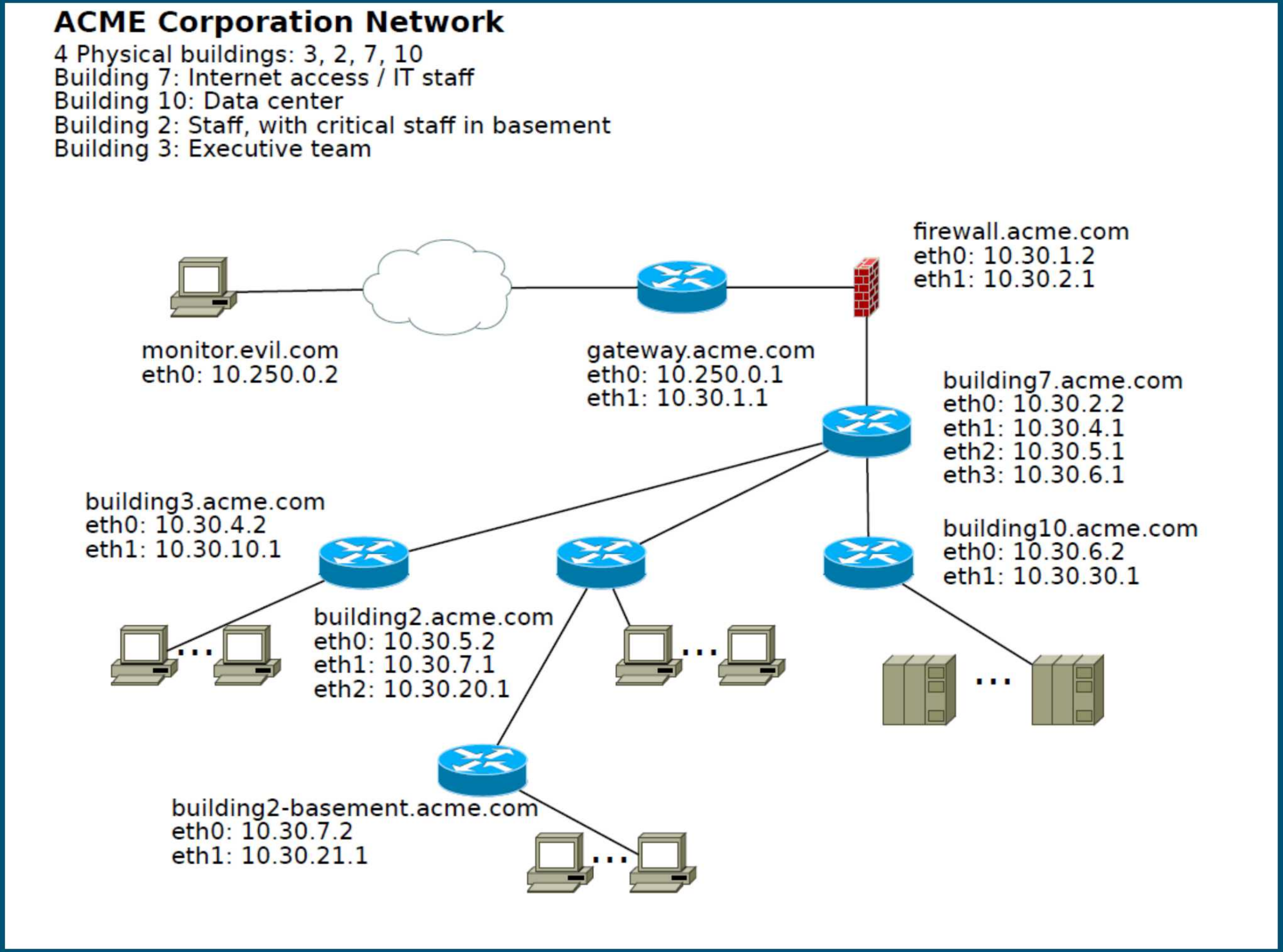
- Question / goal driven
  - A model is created to answer a specific question
- From the goal, topologies and applications are chosen or created
- The model is then deployed as virtual machines, networks, and software
  - This process will likely take many iterations before the deployment is close to what is desired
  - Instrumentation is added
- The collected data is used to answer the question or refine the model


Network of Interest Data
Public Internet Data
Purchased Internet Data
CAIDA; PREDICT; Routeview; DIMES
AKAMAI; IXP
Physical Infrastructure
Geoinformation (MaxMind; ESRI; Google Earth)
Social Media Data
Graph Based Representation
Modeling & Simulation
Experimental Configuration
Link Configuration
Device Configuration
Sensor Configuration and Instrumentation
Application
Scalable Emulation
Data Analytics
Verification & Validation and Uncertain Quantification

Example Topology: Energy Sciences Network



Example Topology: Small Notional Topology



ACME Corporation Network

4 Physical buildings: 3, 2, 7, 10
Building 7: Internet access / IT staff
Building 10: Data center
Building 2: Staff, with critical staff in basement
Building 3: Executive team

monitor.evil.com
eth0: 10.250.0.2

gateway.acme.com
eth0: 10.250.0.1
eth1: 10.30.1.1

firewall.acme.com
eth0: 10.30.1.2
eth1: 10.30.2.1

building7.acme.com
eth0: 10.30.2.2
eth1: 10.30.4.1
eth2: 10.30.5.1
eth3: 10.30.6.1

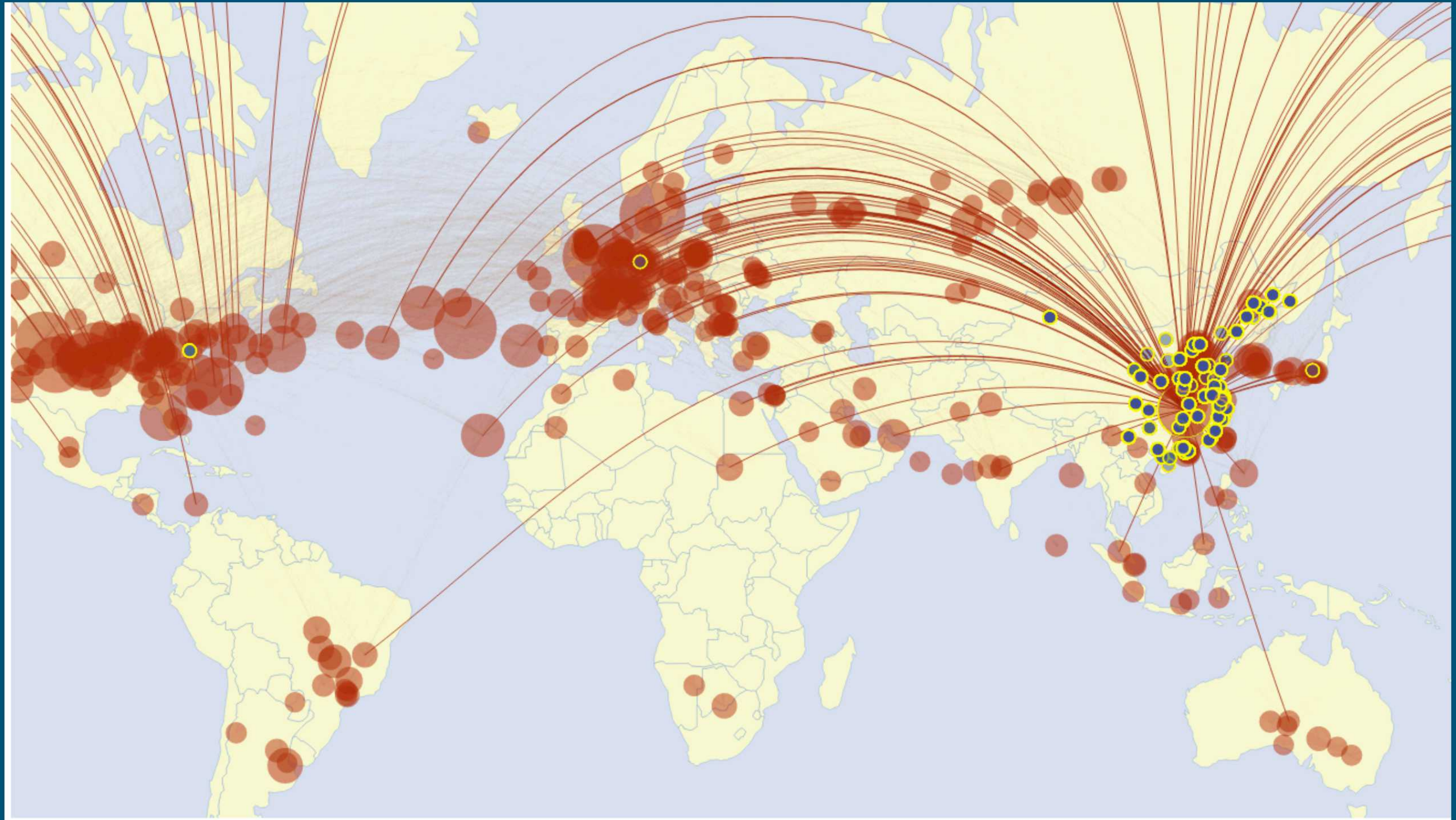
building3.acme.com
eth0: 10.30.4.2
eth1: 10.30.10.1

building2.acme.com
eth0: 10.30.5.2
eth1: 10.30.7.1
eth2: 10.30.20.1

building10.acme.com
eth0: 10.30.6.2
eth1: 10.30.30.1

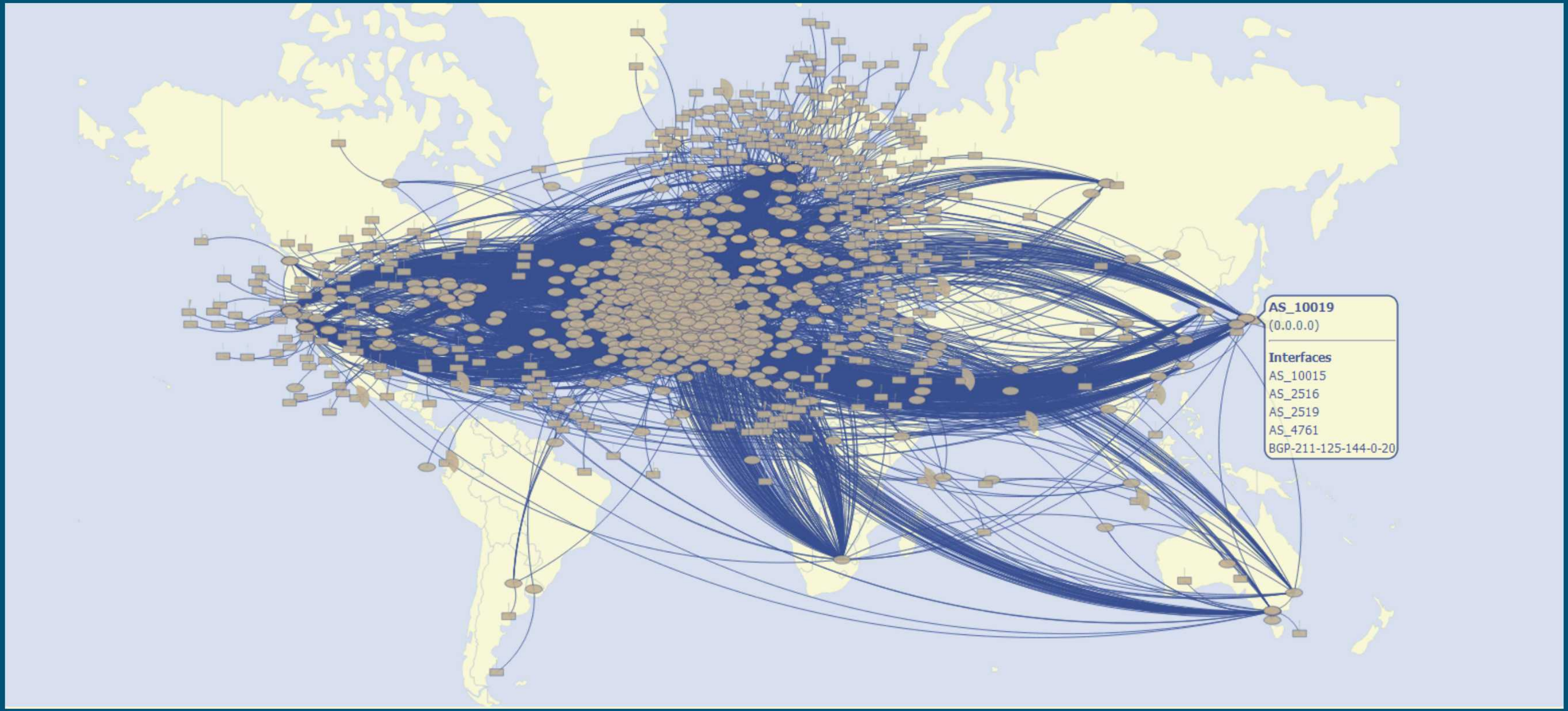
building2-basement.acme.com
eth0: 10.30.7.2
eth1: 10.30.21.1

Example Topology: Large AS and its Exit Routers



97.249.232.221.broad.wh.hb.dynamic.163data.com.cn (AS4134)

Example Topology:
Real-world Tor Network



High-Level Firewheel Components

Graph

- Graph Server (API)
- Scheduling
- Graph Analytics
- Plugins – 100s

Agents

- Agent Server (pluggable modules)
- Agent Development environments
- Agents – 100s

Core

- Network Virtualization
- Virtual Machine Launching / Control
- CLI – physical cluster management

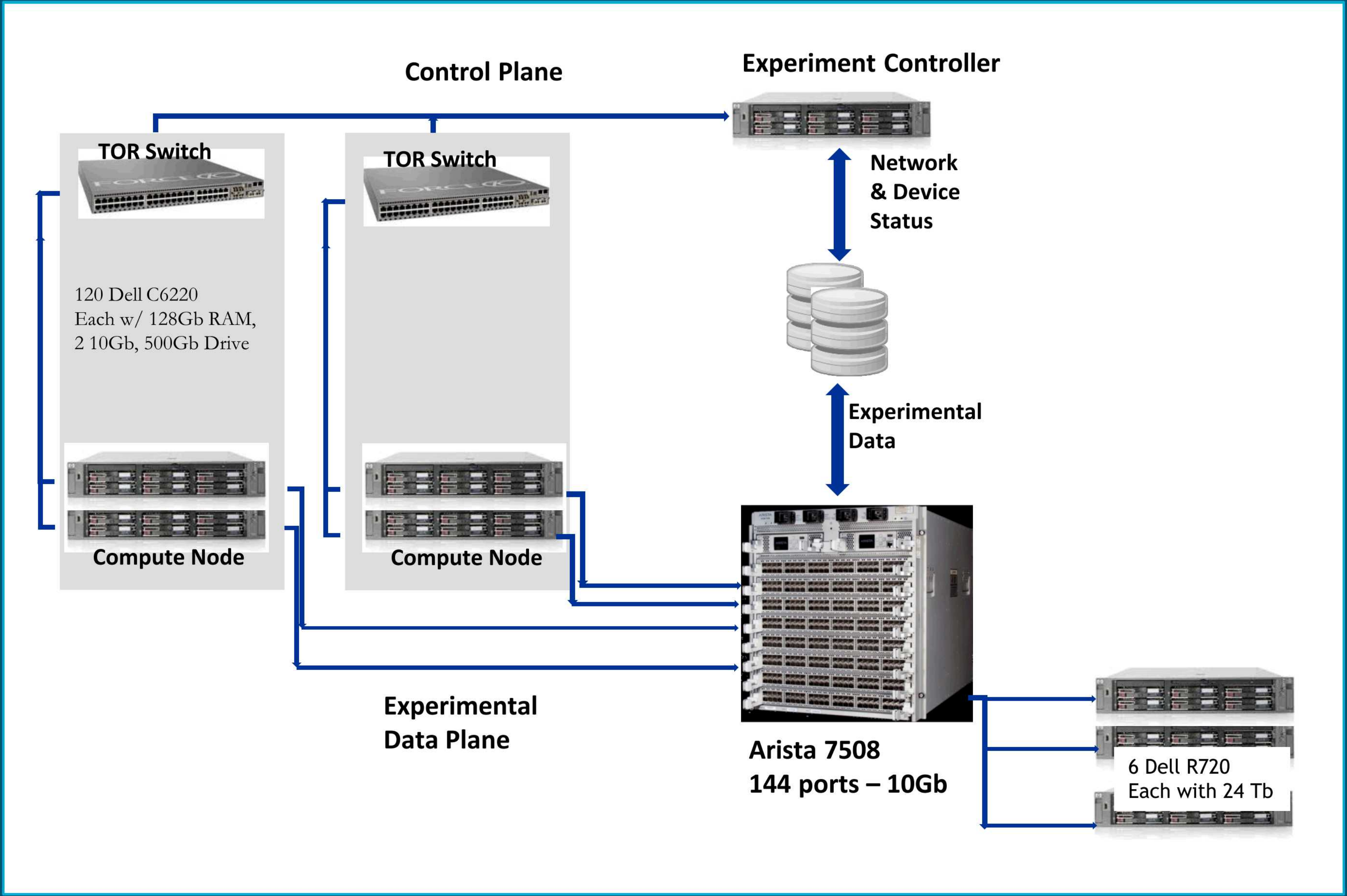
Topologies and Applications

- Windows services + domains
- Tor and Botnets
- Content Delivery Networks
- Programmatically generated topologies
- Etc.

Analytics

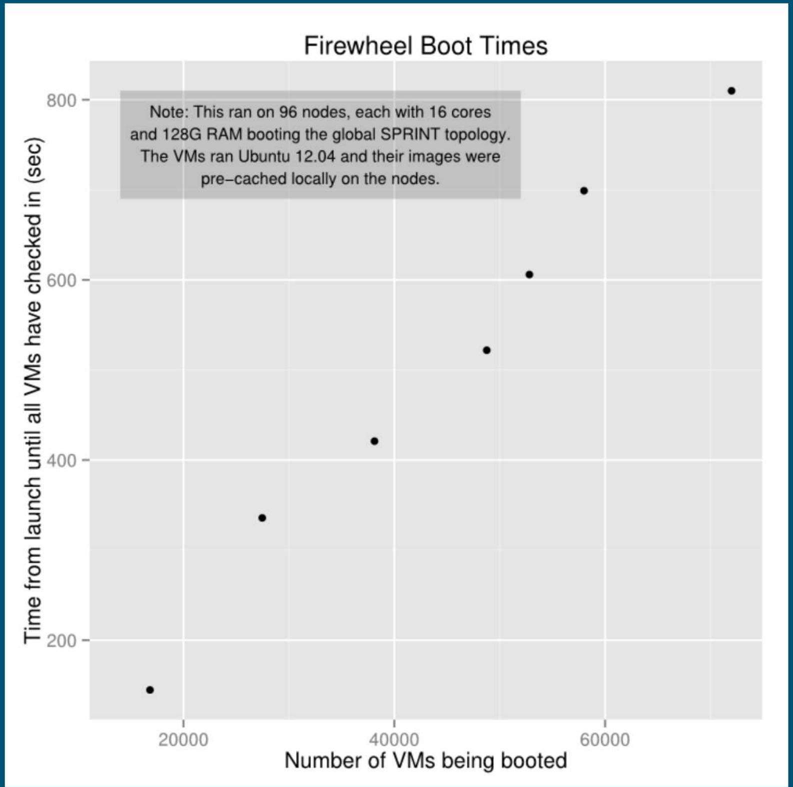
- Integrated Logstash / Kibana support
- NetFlow analysis, Windows events, BIND statistics, CPU / mem loads

Example Physical Testbed

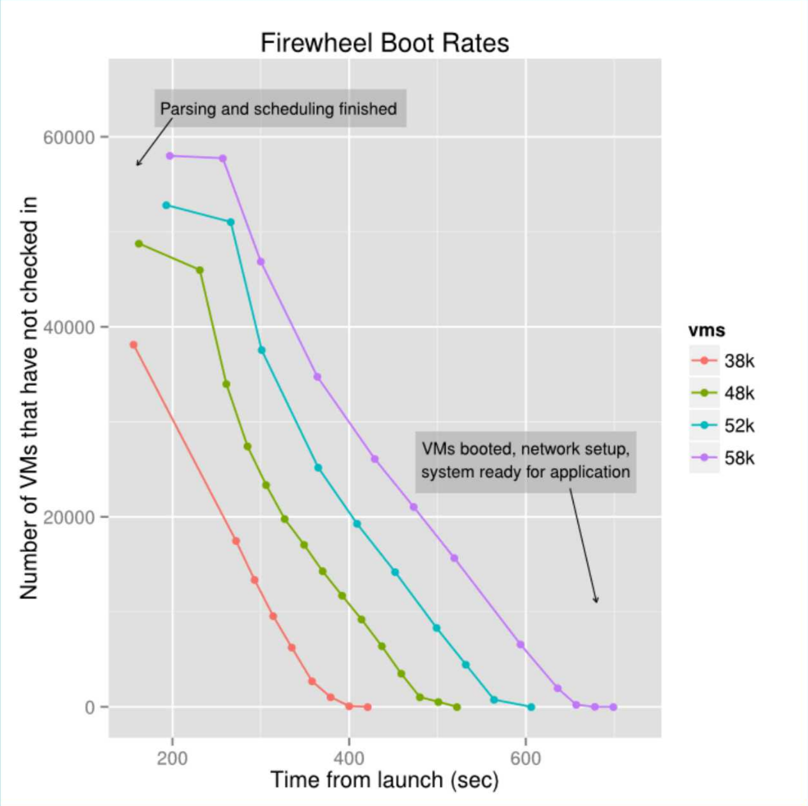


Architecture Efficiency

- Very high density of VMs per core, fast set up and near linear boot time
  - 750 Linux VMs per 16 physical core on Dell C6220
  - 15 minutes to boot 72K Ubuntu 12.04 VMs with full network convergence


Firewheel Boot Times

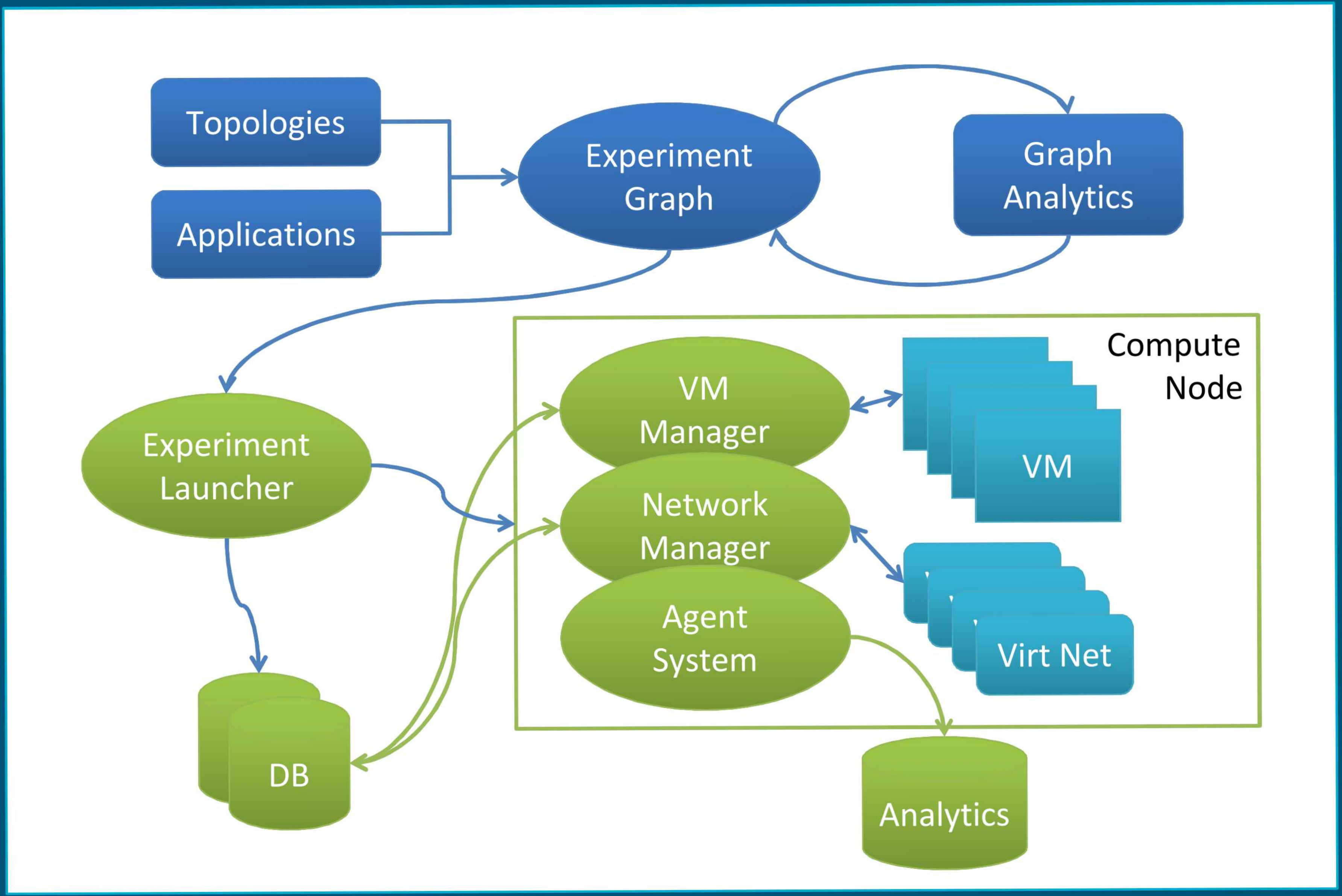
Note: This ran on 96 nodes, each with 16 cores and 128G RAM booting the global SPRINT topology. The VMs ran Ubuntu 12.04 and their images were pre-cached locally on the nodes.


Firewheel Boot Rates

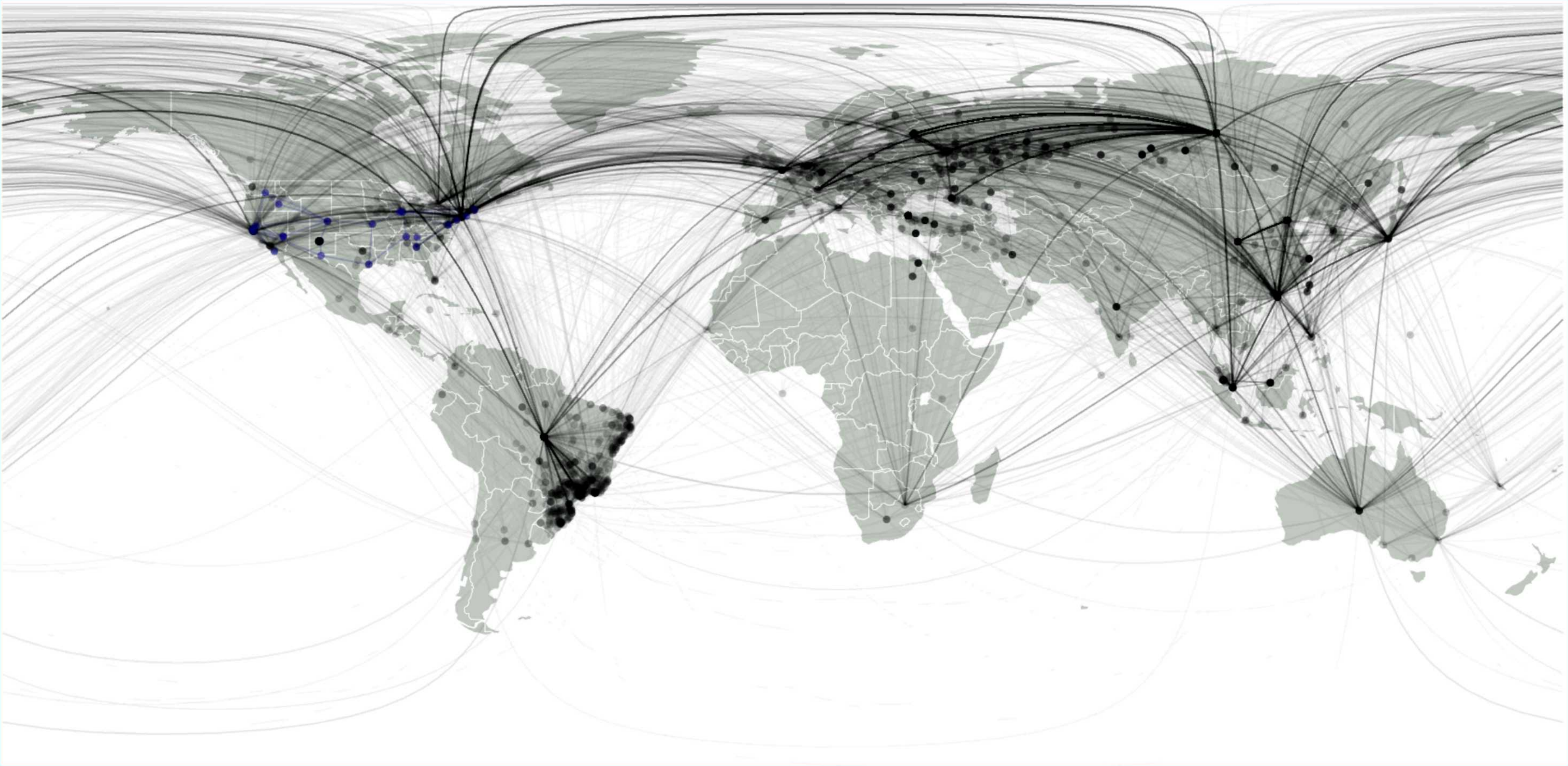
Parsing and scheduling finished

VMs booted, network setup, system ready for application

Architecture Design



Case Study 1: DNS Amplification Post-Event Analysis

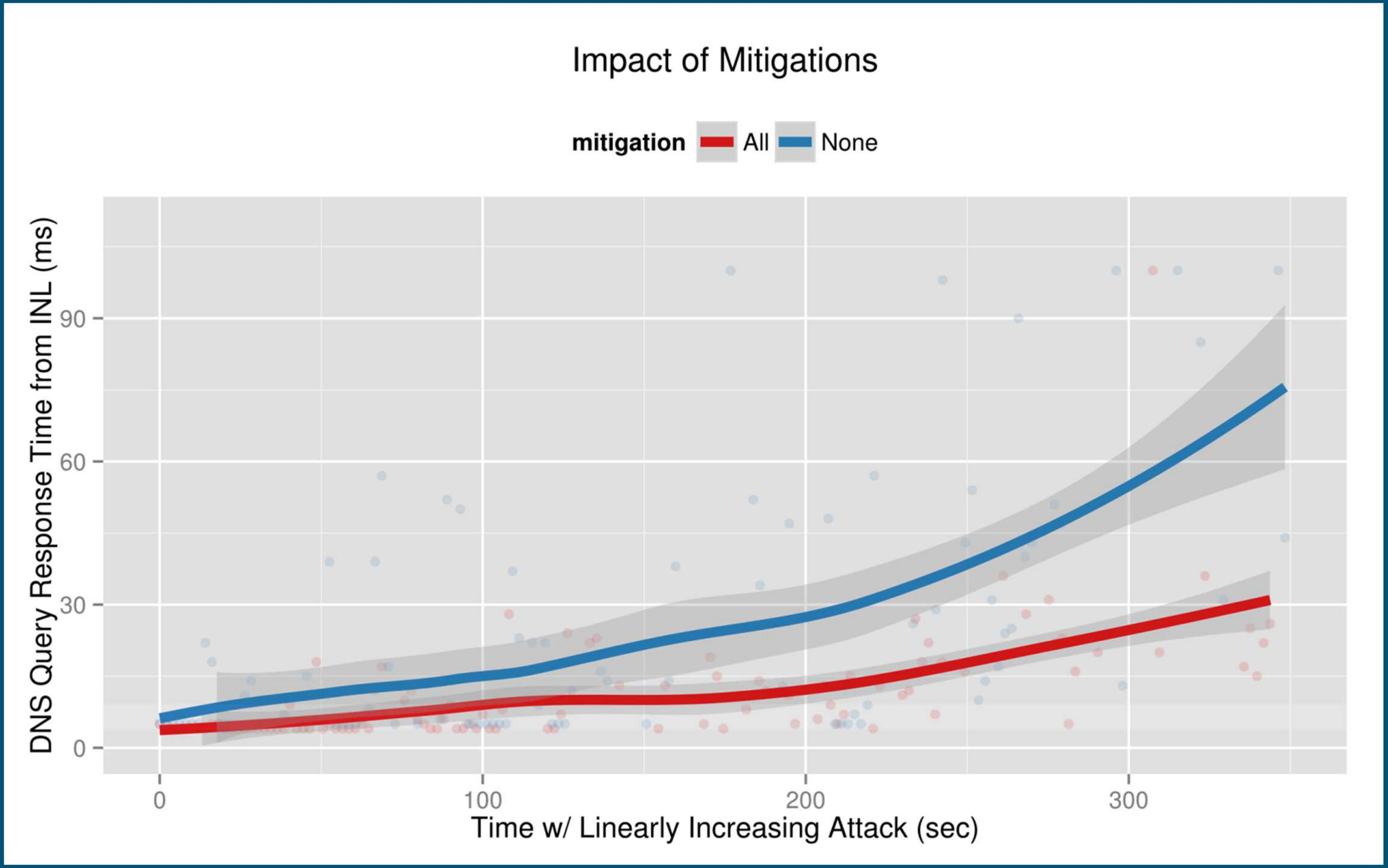


Mitigations

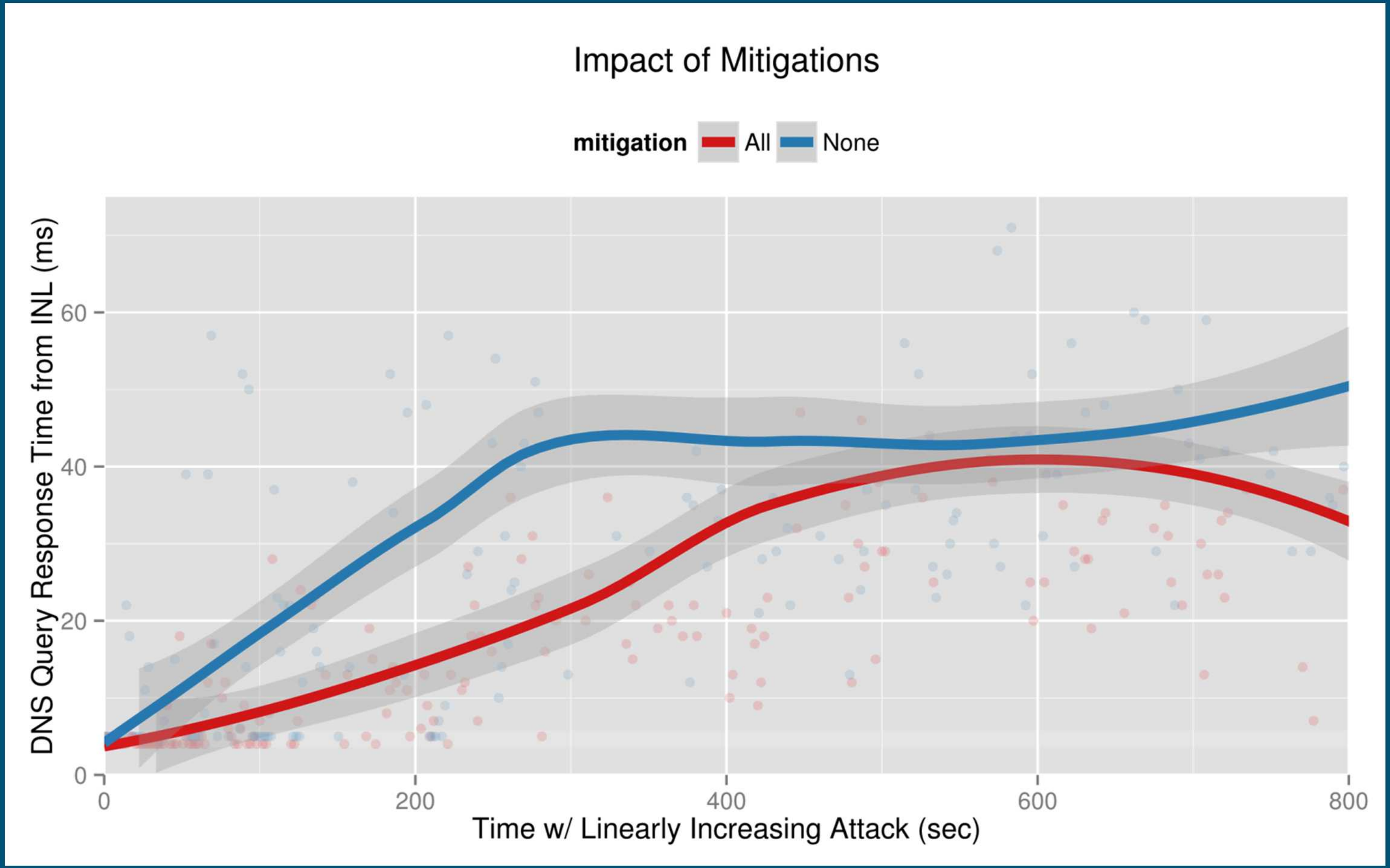
- After successfully mitigating the attack, some questions remained:
  - Can the mitigations protect against larger attacks?
  - Which mitigations had the most impact?
  - How do the mitigations affect other daily operations?
  - How might similar attacks impact surrounding networks?

- Mitigation 1: Disable query logging
- Mitigation 2: Enable rate-limiting
- Mitigation 3: Isolate the DNS servers in the topology

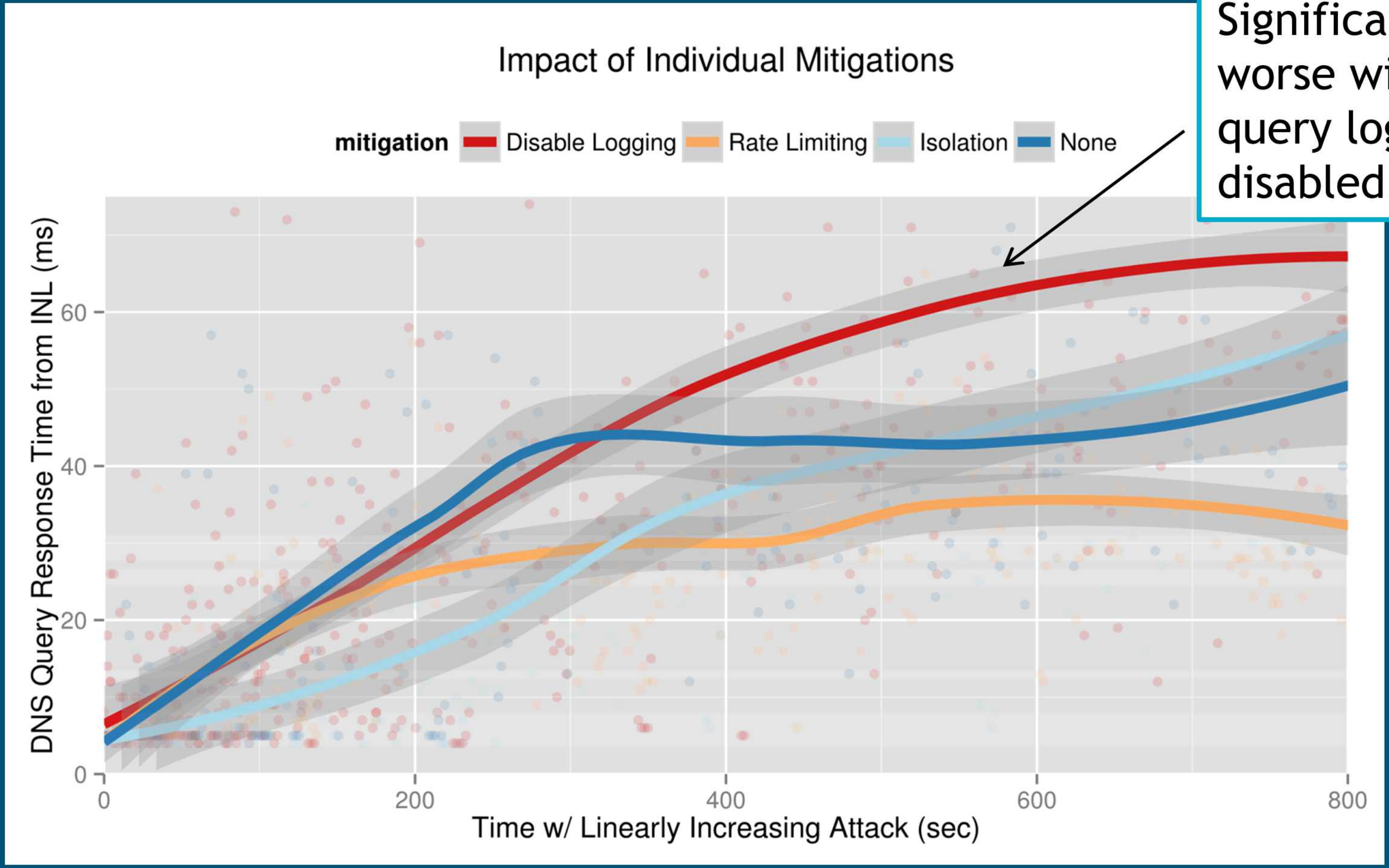
Mitigation Impact



What About Larger Attacks?

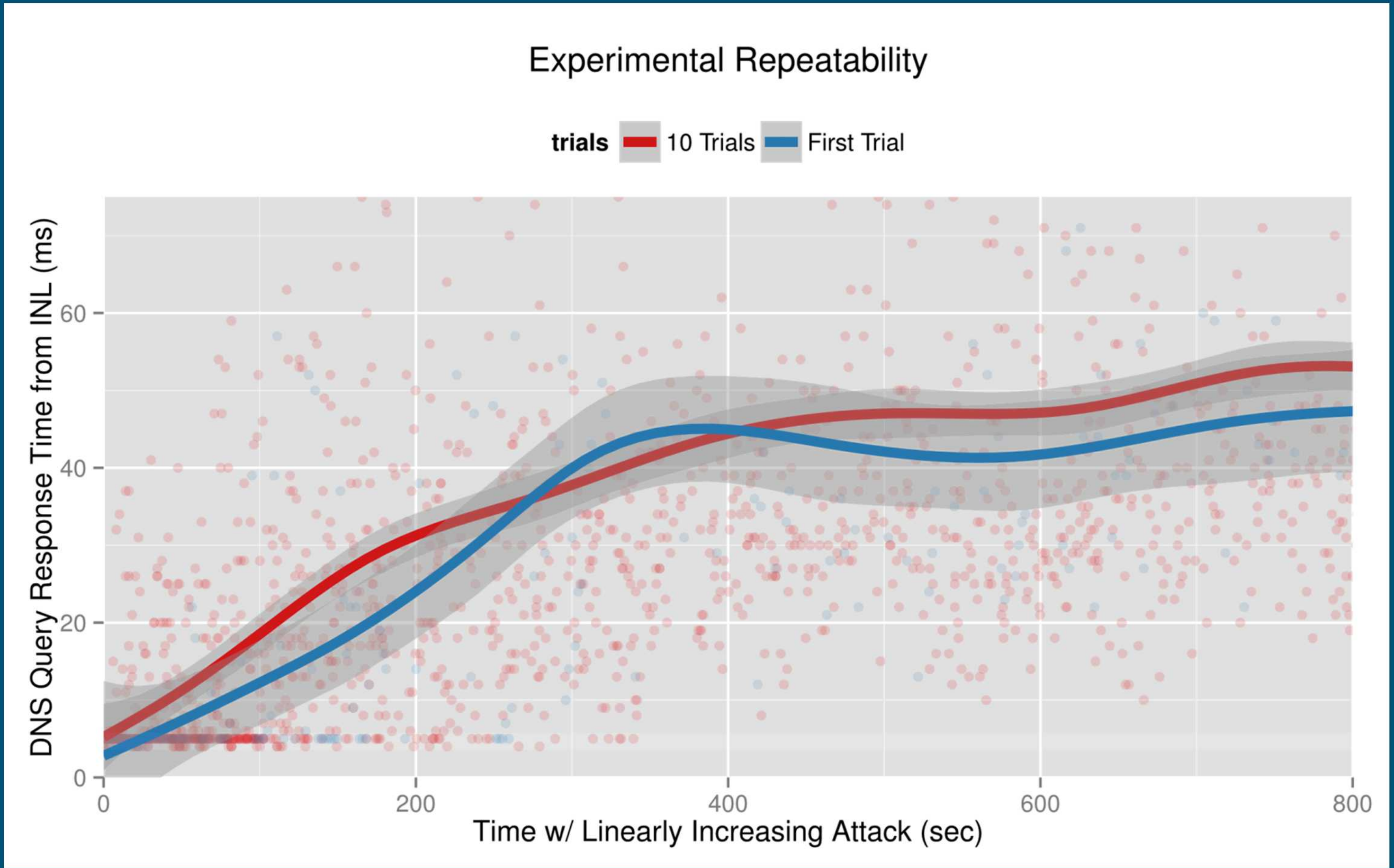


Impact of Each Mitigation



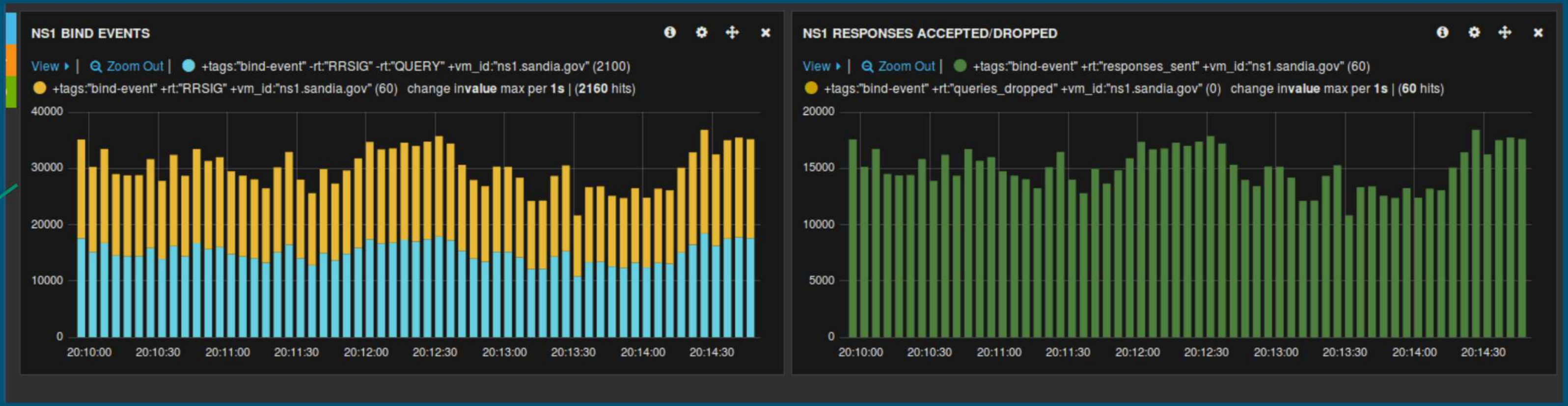
Significantly worse with query logging disabled!

Experiment Repeatability



Discovery!

30,000



NS1 BIND EVENTS

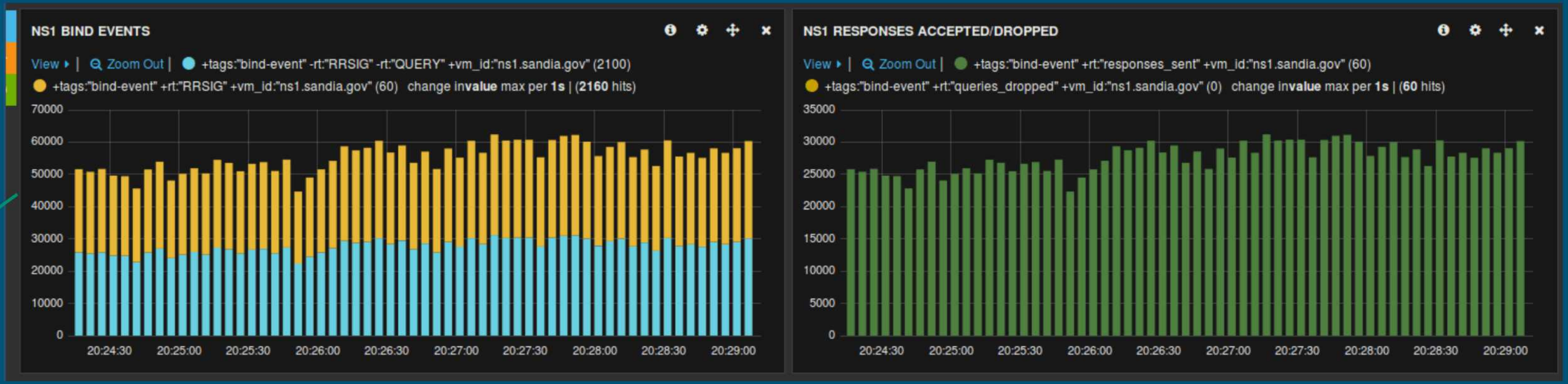
View ▸ | Zoom Out | ● +tags:"bind-event" -rt:"RRSIG" -rt:"QUERY" +vm_id:"ns1.sandia.gov" (2100)
● +tags:"bind-event" +rt:"RRSIG" +vm_id:"ns1.sandia.gov" (60) change in **value** max per **1s** | (**2160** hits)

NS1 RESPONSES ACCEPTED/DROPPED

View ▸ | Zoom Out | ● +tags:"bind-event" +rt:"responses_sent" +vm_id:"ns1.sandia.gov" (60)
● +tags:"bind-event" +rt:"queries_dropped" +vm_id:"ns1.sandia.gov" (0) change in **value** max per **1s** | (**60** hits)

No mitigations

50,000



NS1 BIND EVENTS

View ▸ | Zoom Out | ● +tags:"bind-event" -rt:"RRSIG" -rt:"QUERY" +vm_id:"ns1.sandia.gov" (2100)
● +tags:"bind-event" +rt:"RRSIG" +vm_id:"ns1.sandia.gov" (60) change in **value** max per **1s** | (**2160** hits)

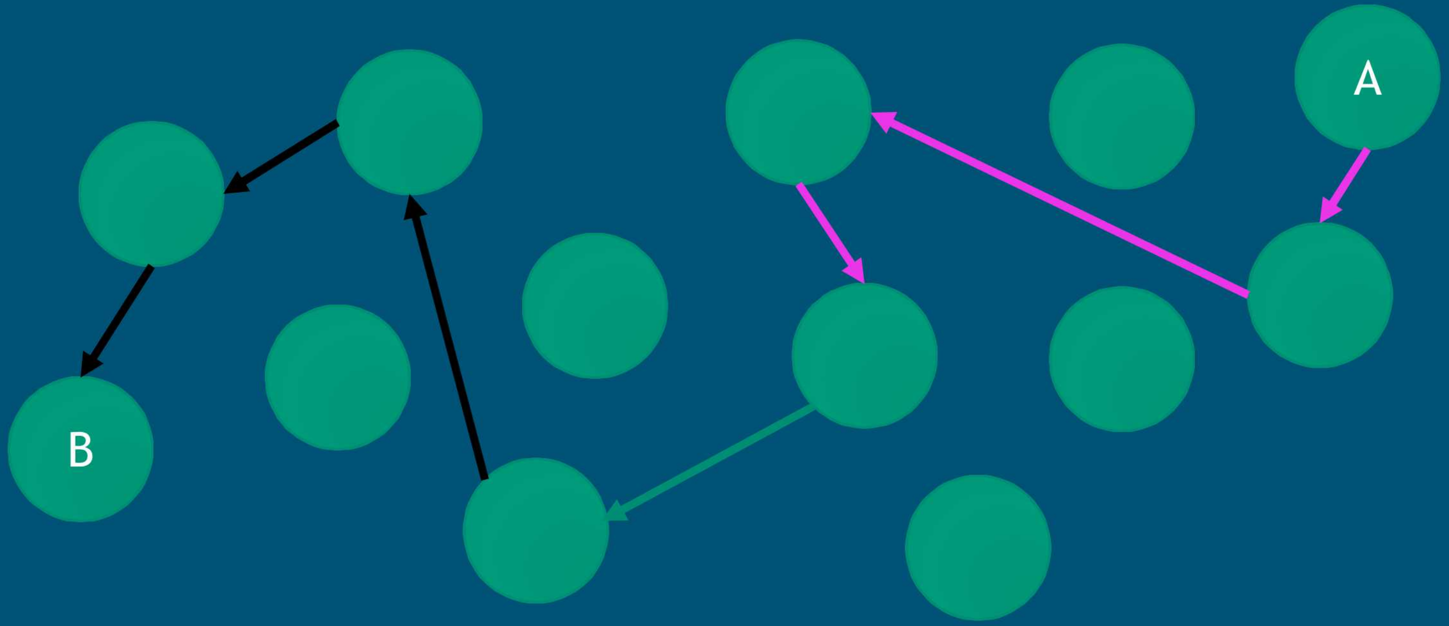
NS1 RESPONSES ACCEPTED/DROPPED

View ▸ | Zoom Out | ● +tags:"bind-event" +rt:"responses_sent" +vm_id:"ns1.sandia.gov" (60)
● +tags:"bind-event" +rt:"queries_dropped" +vm_id:"ns1.sandia.gov" (0) change in **value** max per **1s** | (**60** hits)

Mitigation 1 (Disable Query Logging)

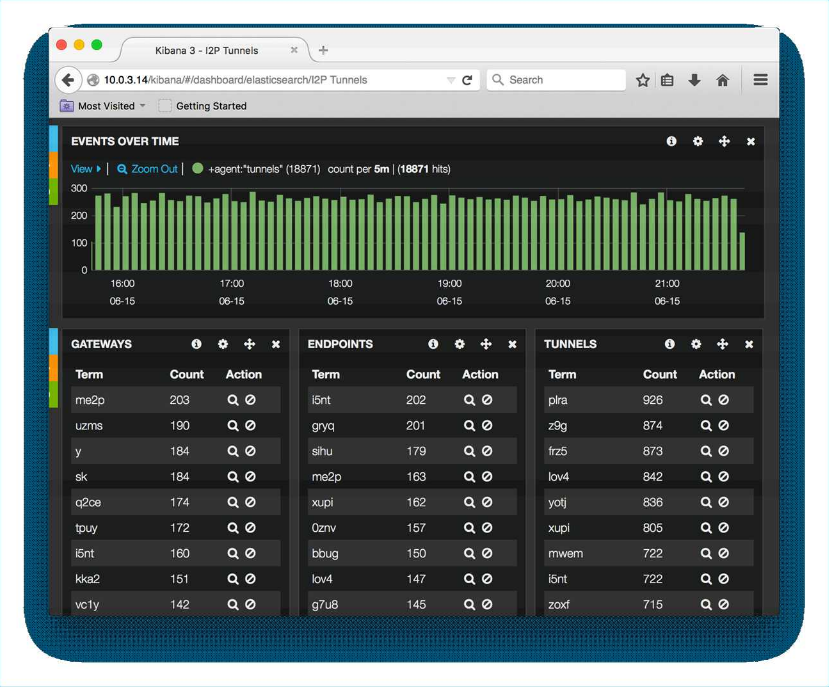
Case Study 2: I2P Study

- I2P (Invisible Internet Project) is a peer-to-peer overlay network designed to provide anonymous hosting
- Malware authors have moved communication servers into it—collaborating with Manos Antonakakis at Georgia Tech, we want to learn how to stop such malware
- I2P is a system defined by the currently running code and previously no system-wide non-trivial questions could be answered



Example I2P Model Goals

1. Suppose one can temporarily turn off I2P traffic to an autonomous system on the Internet. How much would this have to happen to increase the tunnel ratios?
2. The I2P developers may be willing to update the code to ban the malware. Will all users who update isolate themselves from I2P?
3. It appears that a poisoned router entry could break I2P. (Testing this in the model showed that it is not the case.)
4. Given a small testing infrastructure, can we perform a larger population estimation by varying unknown model parameters?



GATEWAYS

Term	Count	Action
me2p	203	🔍 ⊘
uzms	190	🔍 ⊘
y	184	🔍 ⊘
sk	184	🔍 ⊘
q2ce	174	🔍 ⊘
tpuy	172	🔍 ⊘
i5nt	160	🔍 ⊘
kka2	151	🔍 ⊘
vc1y	142	🔍 ⊘

ENDPOINTS

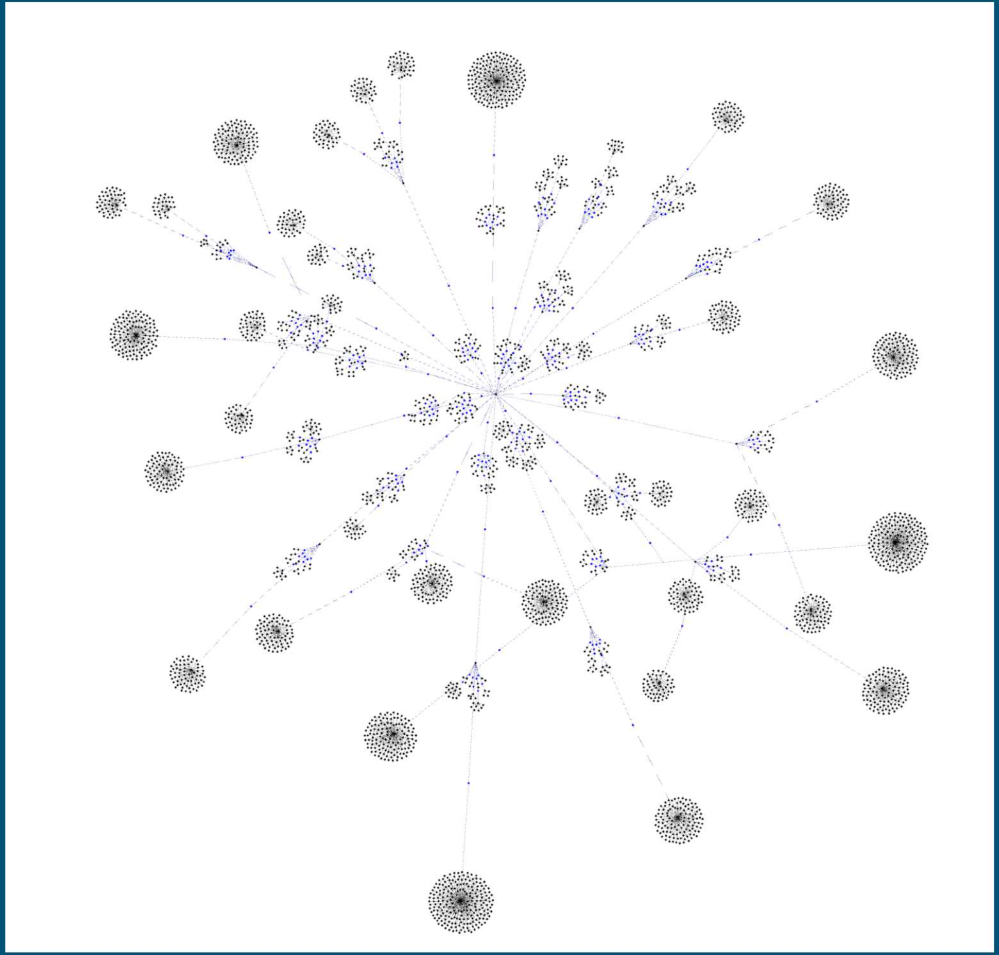
Term	Count	Action
i5nt	202	🔍 ⊘
gryq	201	🔍 ⊘
sihu	179	🔍 ⊘
me2p	163	🔍 ⊘
xupi	162	🔍 ⊘
0znv	157	🔍 ⊘
bbug	150	🔍 ⊘
lov4	147	🔍 ⊘
g7u8	145	🔍 ⊘

TUNNELS

Term	Count	Action
plra	926	🔍 ⊘
z9g	874	🔍 ⊘
frz5	873	🔍 ⊘
lov4	842	🔍 ⊘
yotj	836	🔍 ⊘
xupi	805	🔍 ⊘
mwem	722	🔍 ⊘
i5nt	722	🔍 ⊘
zoxf	715	🔍 ⊘

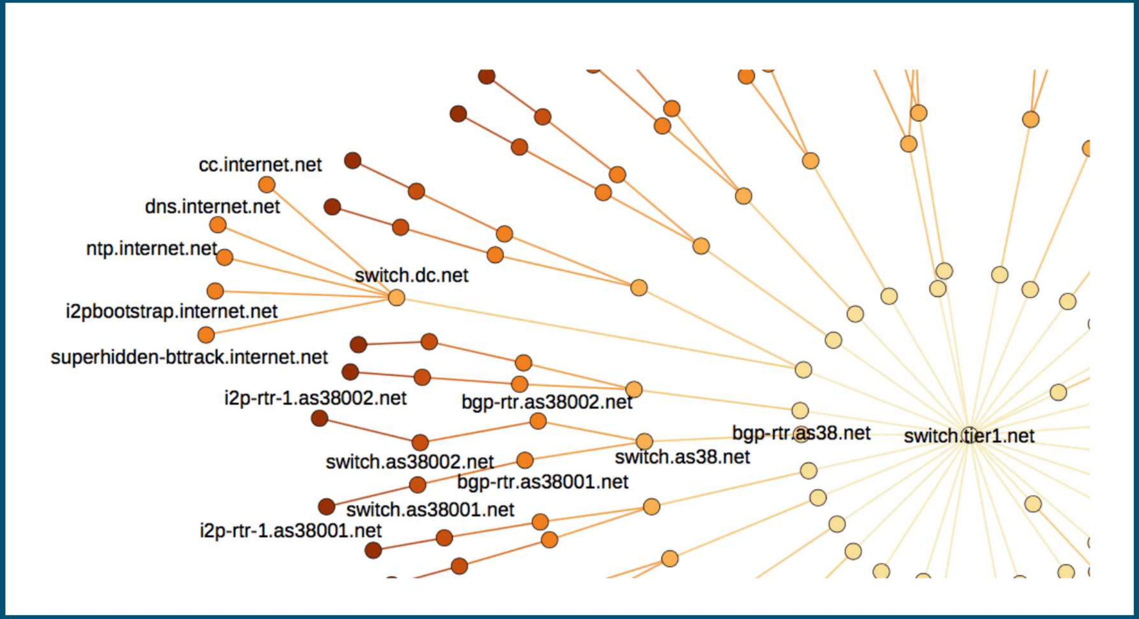
I2P Model

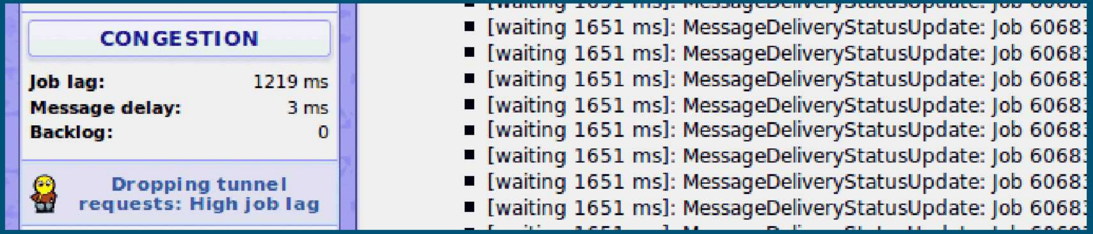
- I2P software is programmatically installed on Ubuntu 14.04 desktop clients
- I2P is bootstrapped (using our own keys)
- Everything is instrumented to report internal behavior
- Underlying network topology (emulated part representing the Internet) is built using histograms pulled from research papers
- User behavior: an infinite loop browsing one website and other simple activities



I2P Model Results and Considerations

- We have answered several questions at this point with the model, including identifying a likely root cause for a network anomaly last year and establishing population estimates
- There are numerous assumptions in this model (as with any model) and care needs to be taken to ensure that any conclusions drawn are not simply the result of a model-specific artifact





Job lag: 1219 ms
Message delay: 3 ms
Backlog: 0

Dropping tunnel requests: High job lag

```
■ [waiting 1651 ms]: MessageDeliveryStatusUpdate: Job 6068
■ [waiting 1651 ms]: MessageDeliveryStatusUpdate: Job 6068
■ [waiting 1651 ms]: MessageDeliveryStatusUpdate: Job 6068
■ [waiting 1651 ms]: MessageDeliveryStatusUpdate: Job 6068
■ [waiting 1651 ms]: MessageDeliveryStatusUpdate: Job 6068
■ [waiting 1651 ms]: MessageDeliveryStatusUpdate: Job 6068
■ [waiting 1651 ms]: MessageDeliveryStatusUpdate: Job 6068
■ [waiting 1651 ms]: MessageDeliveryStatusUpdate: Job 6068
```

Outline

1. Problem Space
2. Emulation-based Modeling – Tools and Case Studies
3. Model Snapshots
4. Verification and Validation Approaches and Challenges

Staghorn: System-wide Snapshots

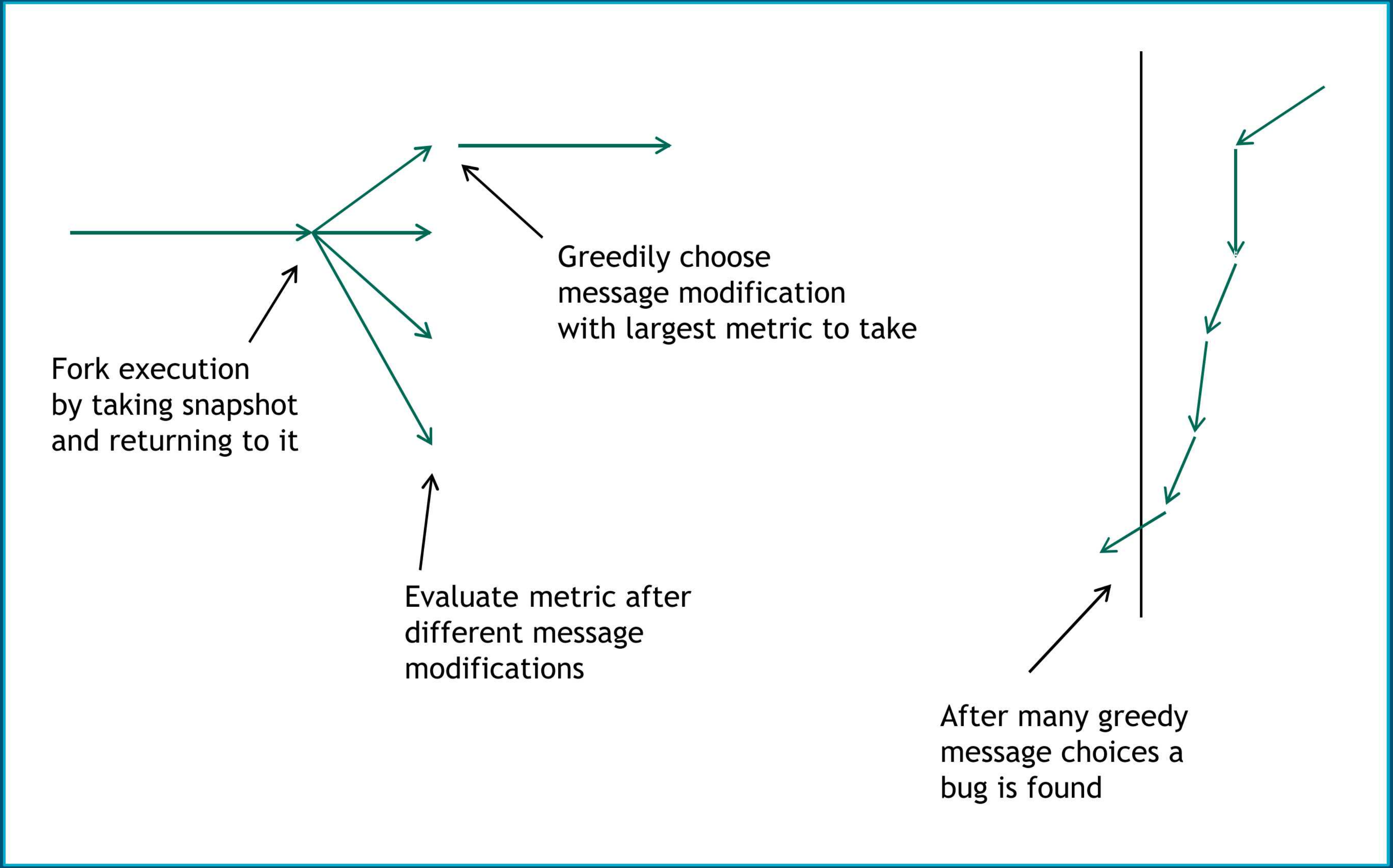
- With emulation, the modeling system **competes** with the model for CPU time
- We built a way to "stop time" within a model, opening the door to the larger world of offline model analysis

- The key contributions of this work are:
  - A full-system snapshot and restore capability which preserves network and I/O state
  - A network modification system that allows for modification of Ethernet frame contents and delivery, or the introduction and removal of frames, during a snapshot
  - The evaluation of this capability on real-world use-cases

Example Use Cases

- Vulnerability analysis
  - Manual or semi-automated state space exploration
  - Fuzzing at the system level
- Debugging systems
  - Stop the entire system on a failure, allowing for better fault isolation
  - Manual debugging can be drastically improved with periodic system checkpoints
- Optimizing tests
  - Test execution paths that share a common prefix can only diverge when necessary
- Experimental repeatability
  - The non-determinism at different stages in a model can be measured
- Training
  - Checkpoints can be made, and when a user corrupts state in a training environment it can be reset

Fuzzing Concept/Approach



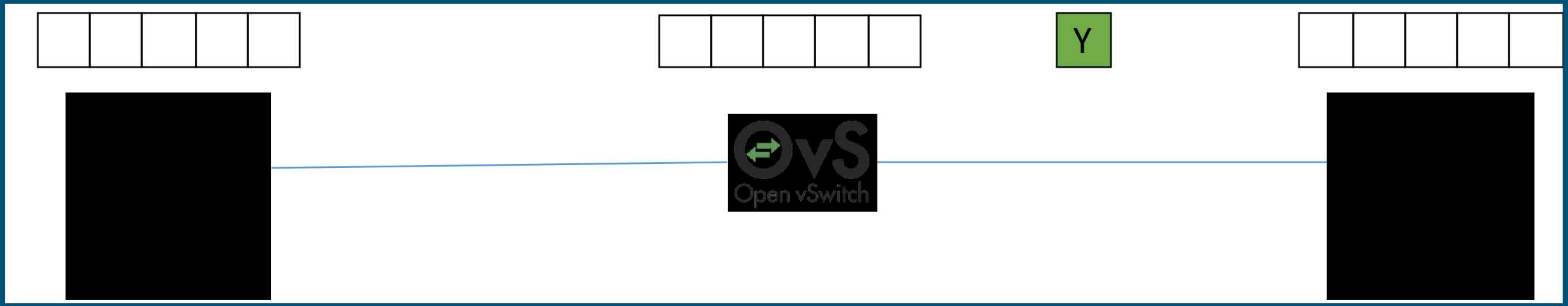
Greedily choose message modification with largest metric to take

Fork execution by taking snapshot and returning to it

Evaluate metric after different message modifications

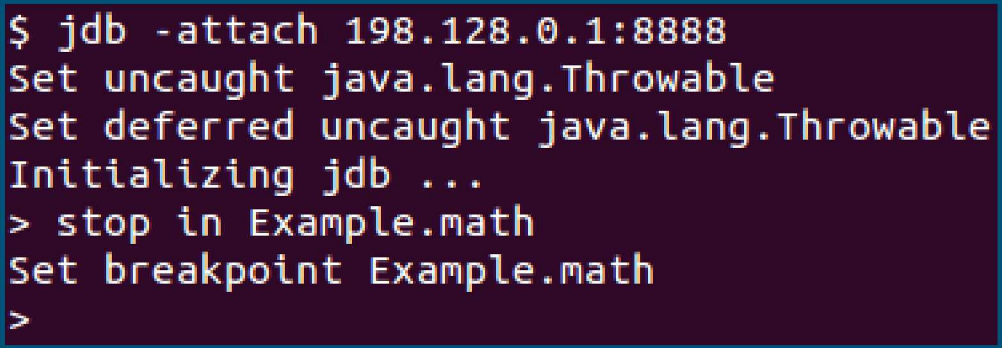
After many greedy message choices a bug is found

Fuzzing a Packet

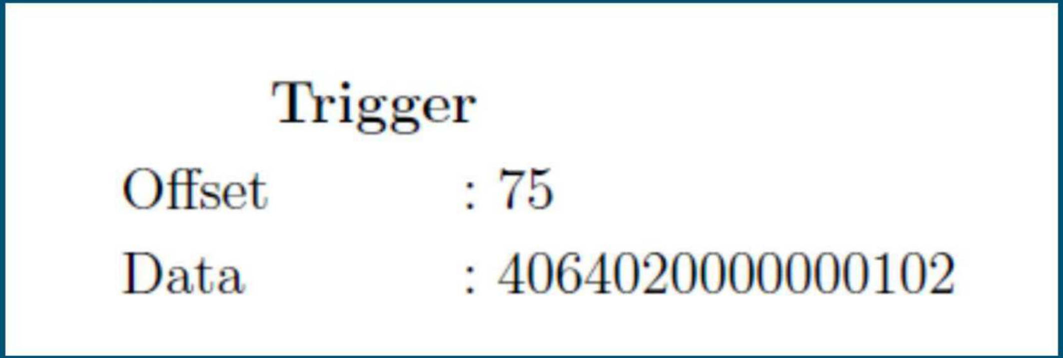


Debugging Systems

1. Set breakpoint
2. Install Staghorn Trigger
3. Staghorn will wait until the breakpoint is hit to snapshot the system.

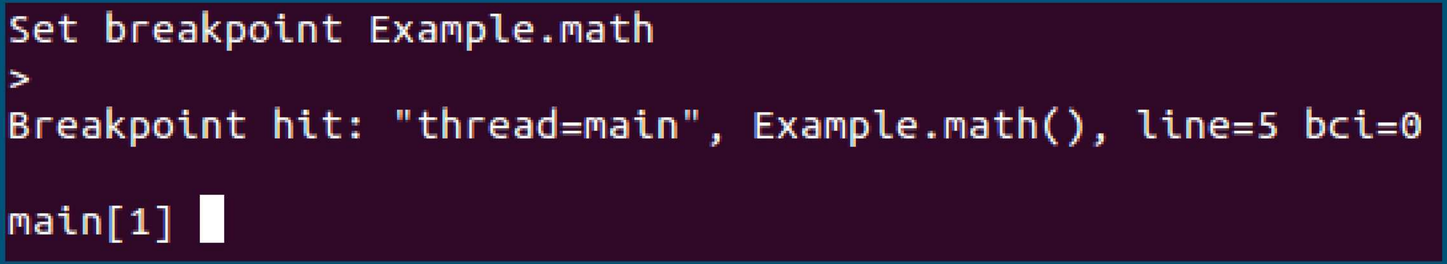


```
$ jdb -attach 198.128.0.1:8888
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
> stop in Example.math
Set breakpoint Example.math
>
```



Trigger

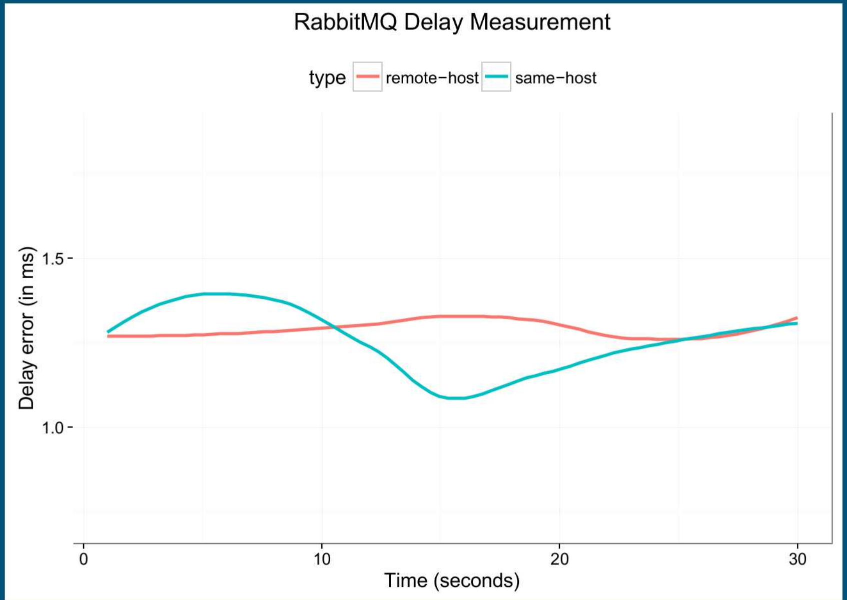
Offset	: 75
Data	: 4064020000000102

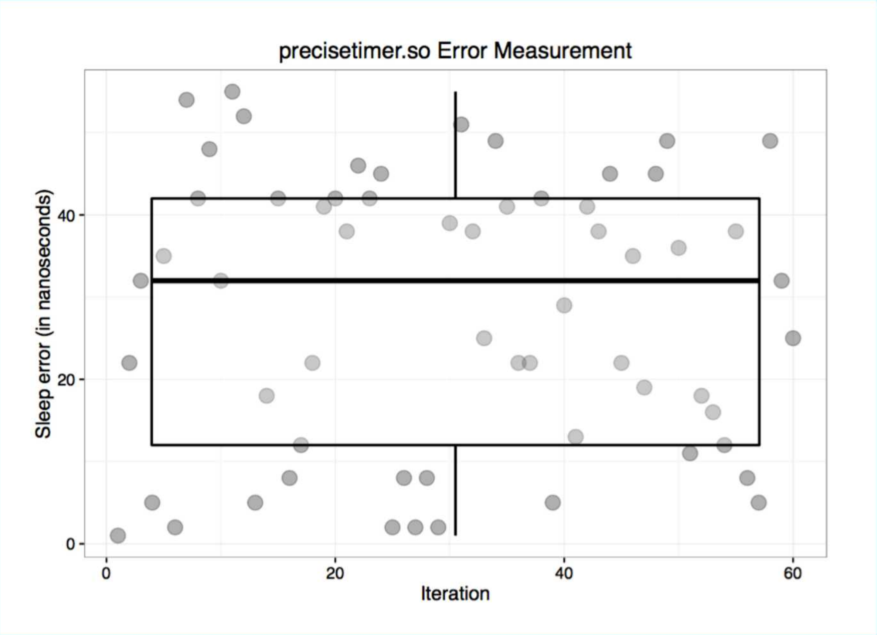


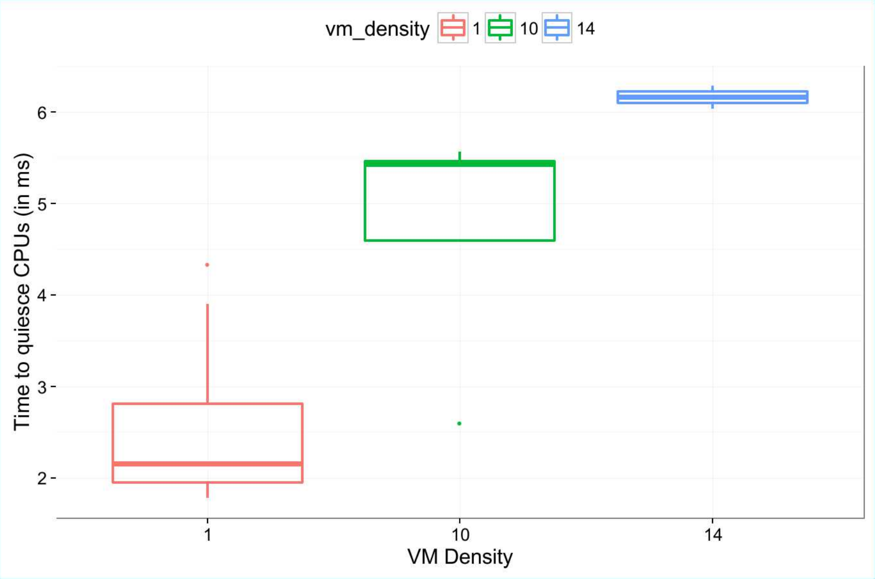
```
Set breakpoint Example.math
>
Breakpoint hit: "thread=main", Example.math(), line=5 bci=0

main[1] █
```

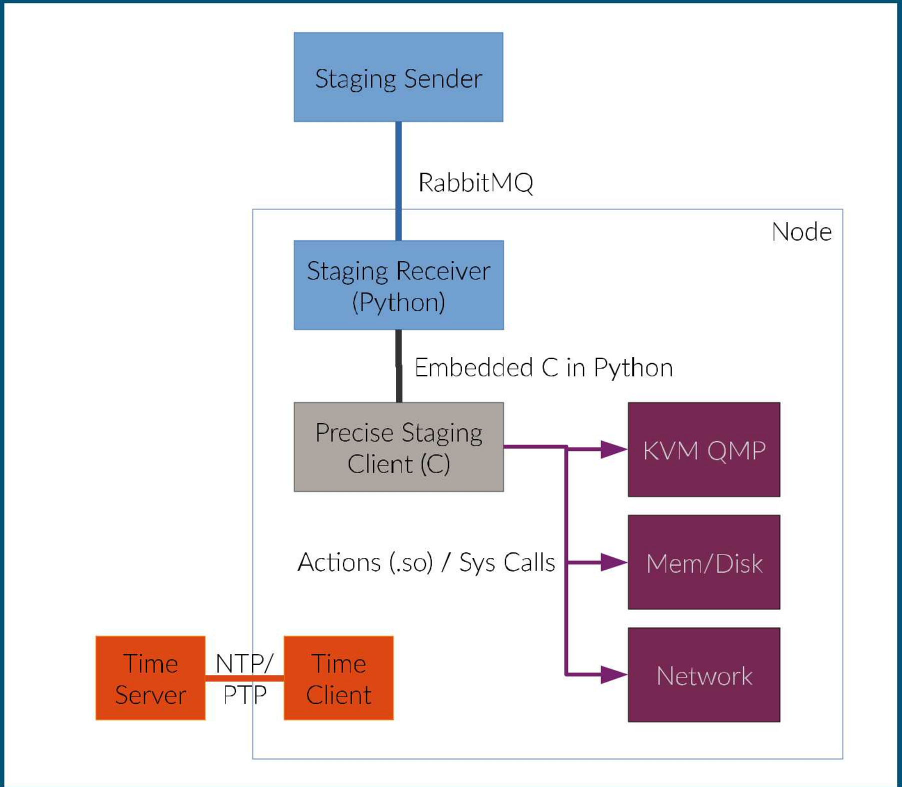
Staghorn Timing Results

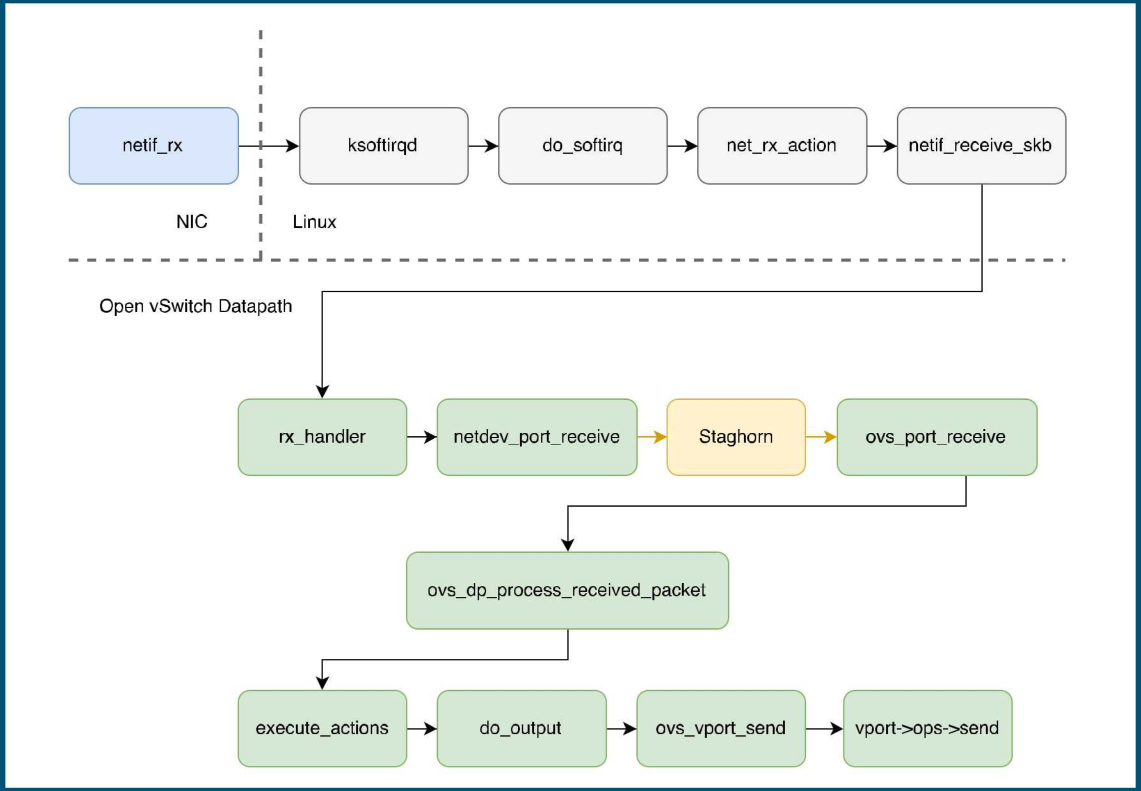






Staghorn Architecture

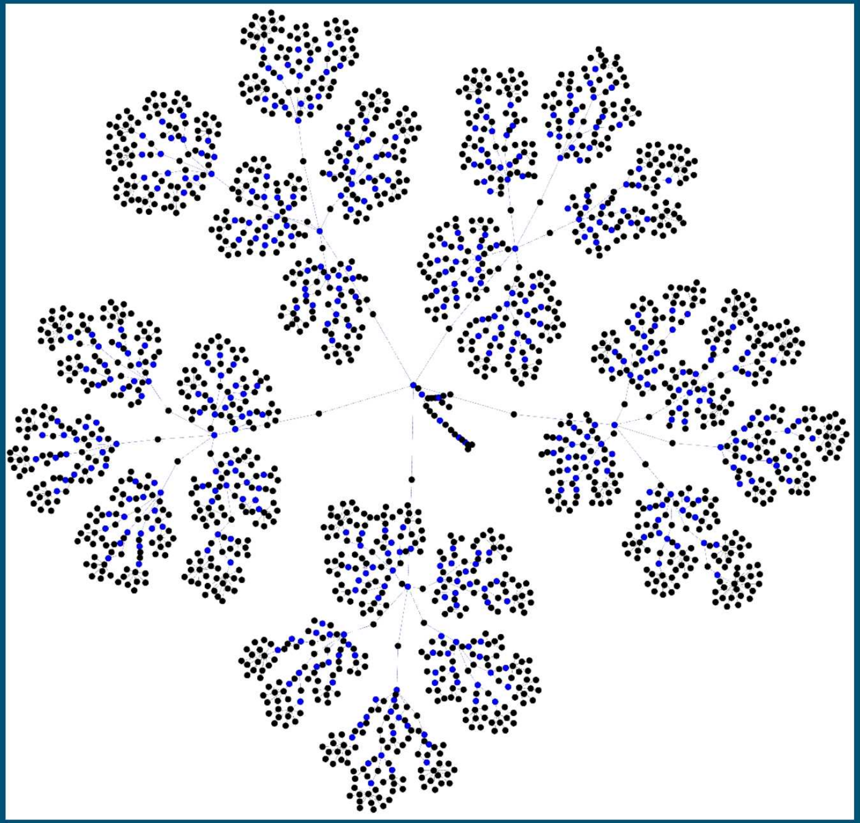


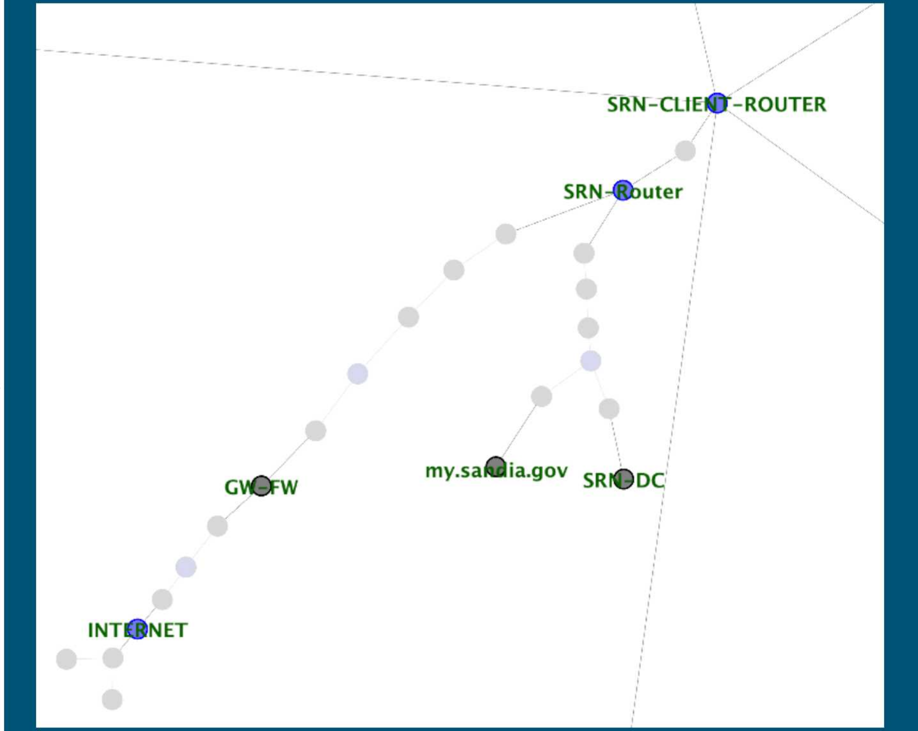


Outline

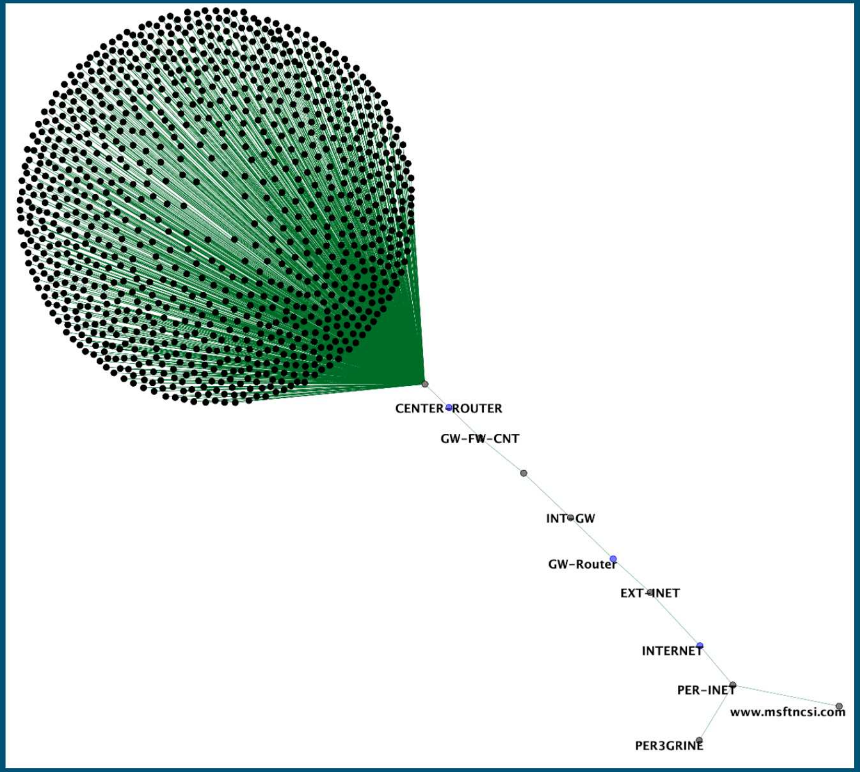
1. Problem Space
2. Emulation-based Modeling – Tools and Case Studies
3. Model Snapshots
4. Verification and Validation Approaches and Challenges

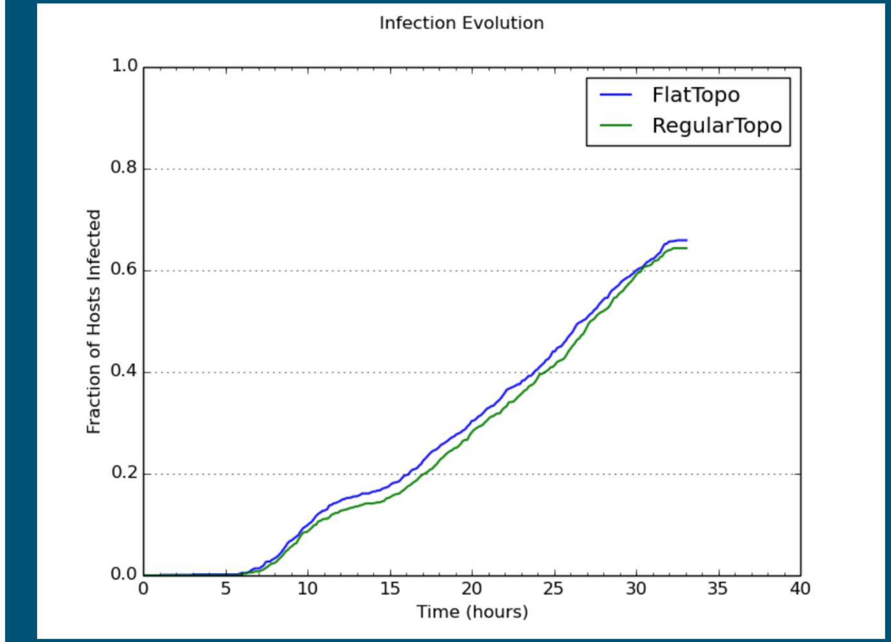
Topology Sensitivity



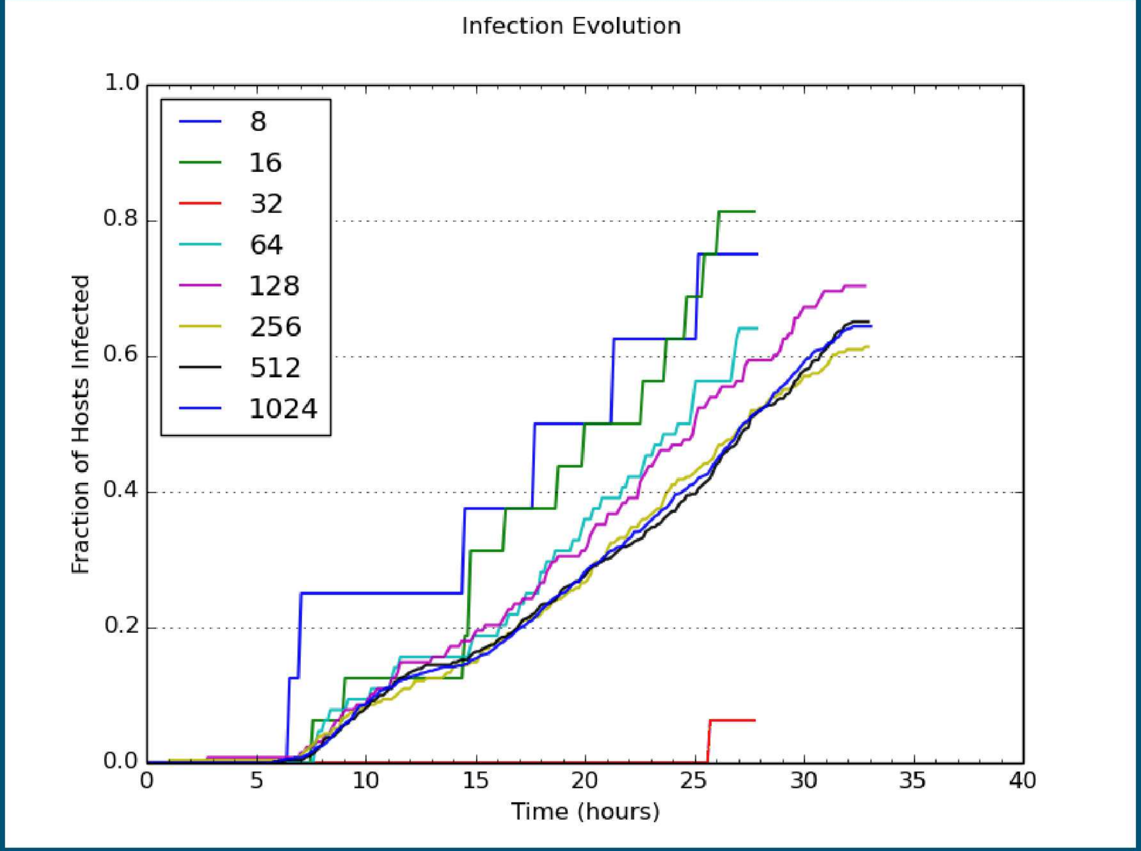


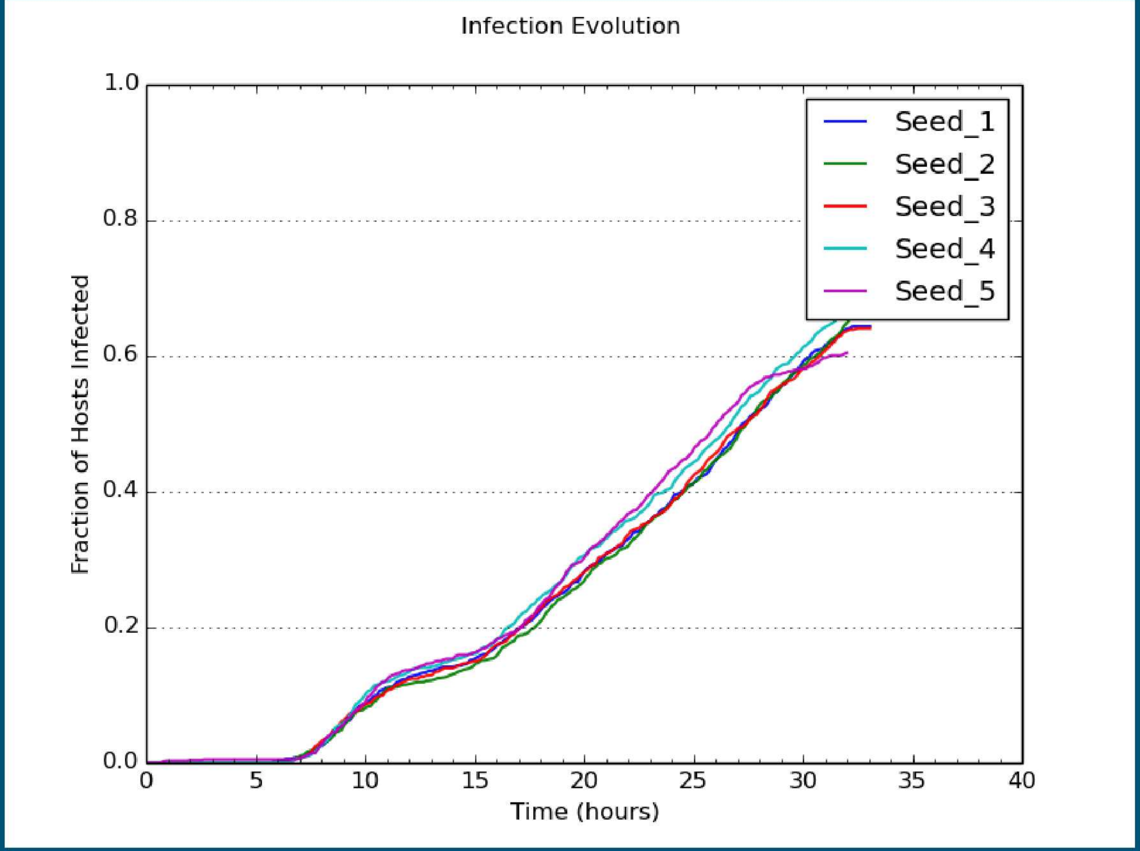
Topology Sensitivity





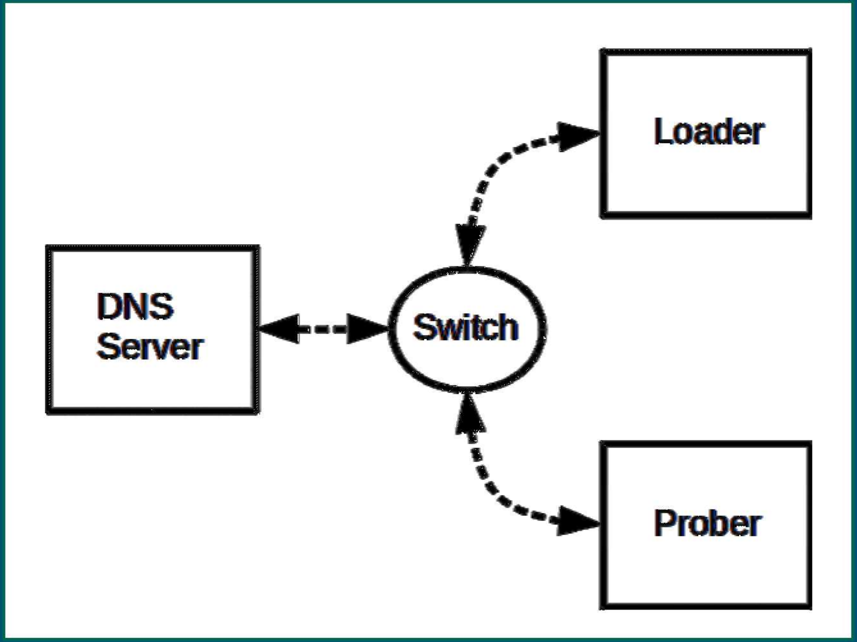
Topology Scale and Non-Determinism Sensitivity

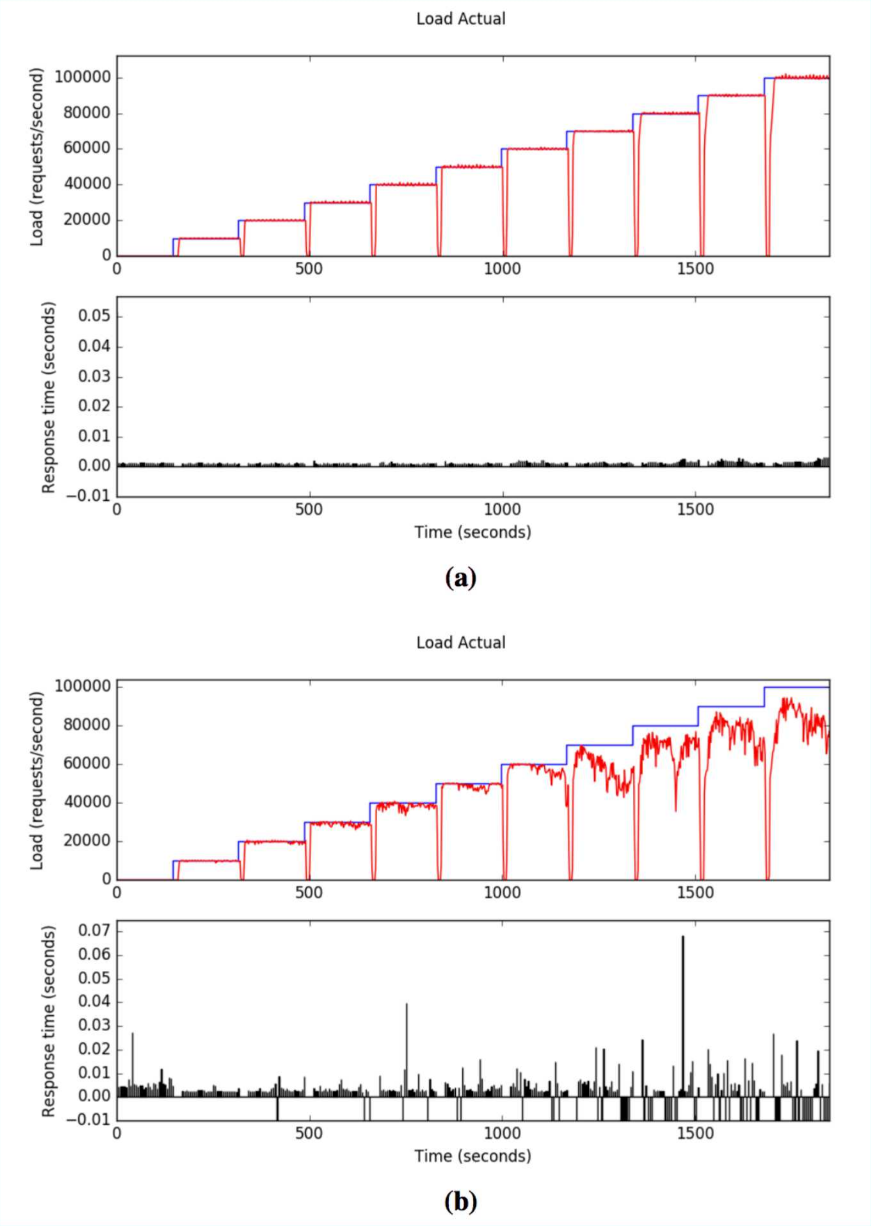




Verification and Validation

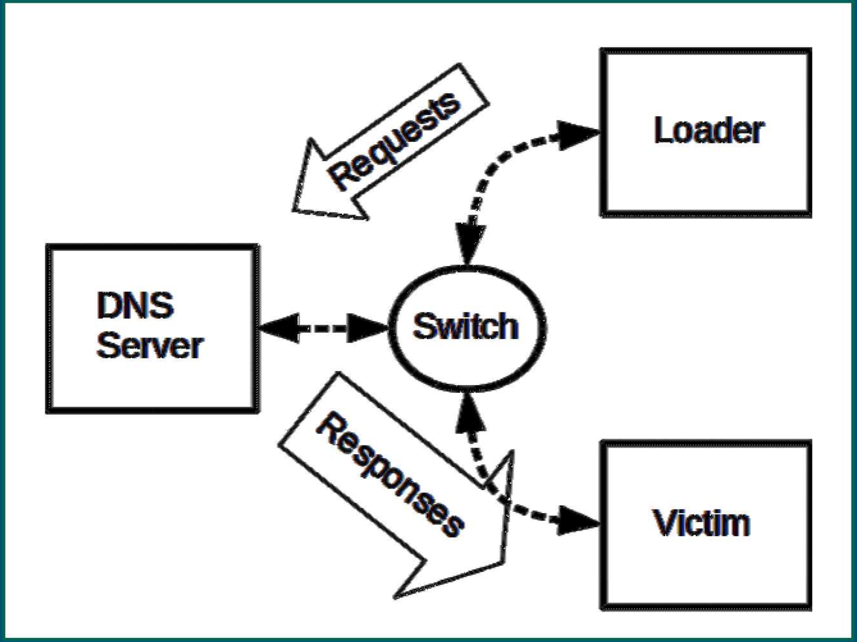
- Identified problems: massive number of parameters and difficult metrics / quantities of interest
- We are building small laboratory models of key components – hierarchical may work
- Great care needs to be taken!

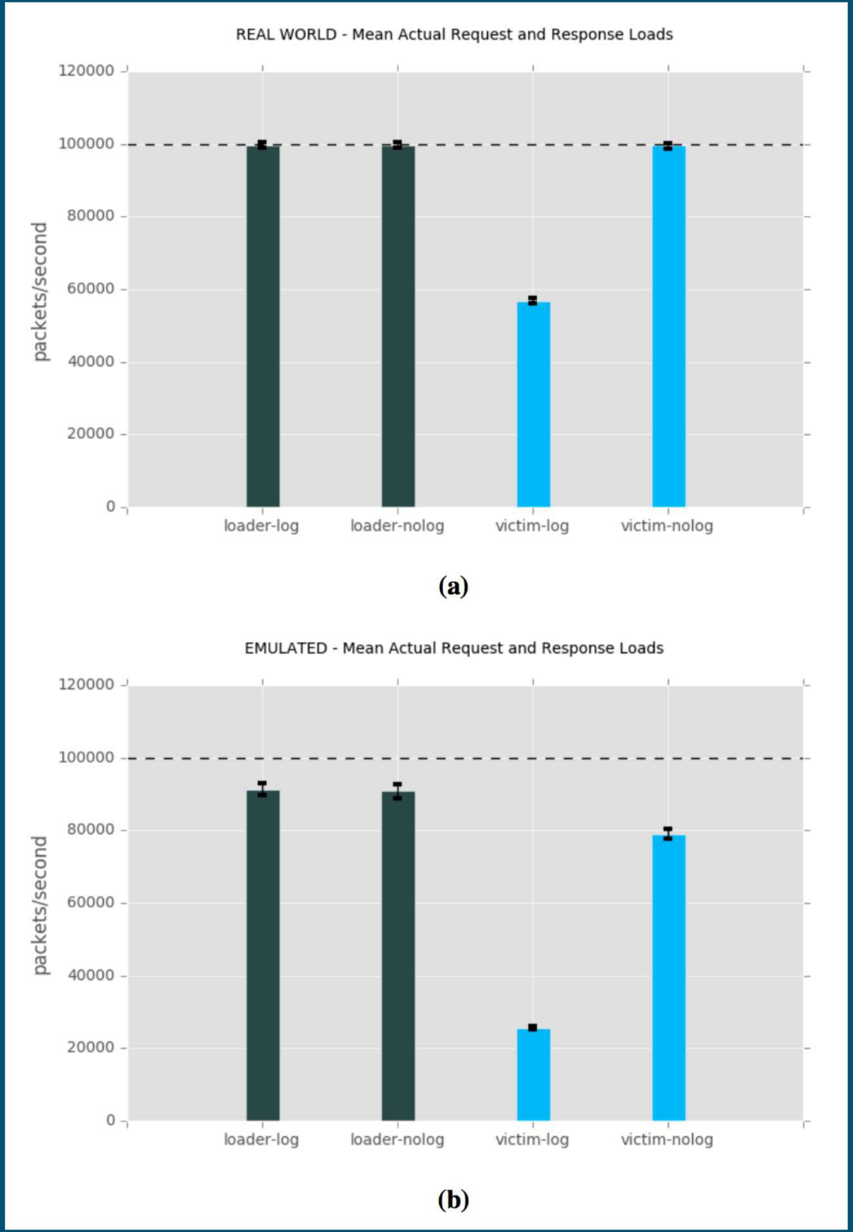




Verification and Validation

- Identified problems: massive number of parameters and difficult metrics / quantities of interest
- We are building small laboratory models of key components – hierarchical may work
- Great care needs to be taken!





(a)

(b)

Possible Validation Distinguisher

- As a general validation metric, use a Turing machine as a distinguisher between real and emulated
- D is a probabilistic Turing machine *distinguisher*, E is an oracle for the emulator, and R is an oracle for the real world
- Difficulty is in choosing the environment for D (the probability space and allowed accesses to the oracles)
- $Adv(D) = |\Pr[D(E) = 1] - \Pr[D(R) = 1]|$

Validation Process

1. Understand the model's purpose
2. Prioritize important model features
3. Select observables of the model to compare, known as Quantities of Interest (QoI)
4. Identify the model parameters and value ranges
5. Select a real world referent to which to compare your model
6. Design validation experiments
7. Perform the validation experiments and collect data
8. Compare the QoIs using a pre-defined validation metric
9. Compare the metric results to validity thresholds
10. Collect a description of the full validation process and the metric results into a credibility case that can be used to judge whether a model is suitable for its purpose

Conclusion

- We are at a point where emulation-based models of computer systems are scalable and useful
- Our modeling tools and environments are currently being used to help study real systems
- Great care needs to be taken in designing and running experiments and important results that are trusted need to come hand-in-hand with a validation study

- Questions?