

# Reproducible Network Research With High-Fidelity Emulation

Nikhil Handigol<sup>+</sup>, **Brandon Heller<sup>+</sup>**, Bob Lantz<sup>\*</sup>,  
Vimal Jeyakumar<sup>+</sup>, Nick McKeown<sup>+</sup>

<sup>+</sup>Stanford University, Palo Alto, USA

<sup>\*</sup>Open Networking Laboratory, Palo Alto, USA

The scientific method says:

experiments are only valid if they can be  
*reproduced.*

The norm in physics, medicine, etc..

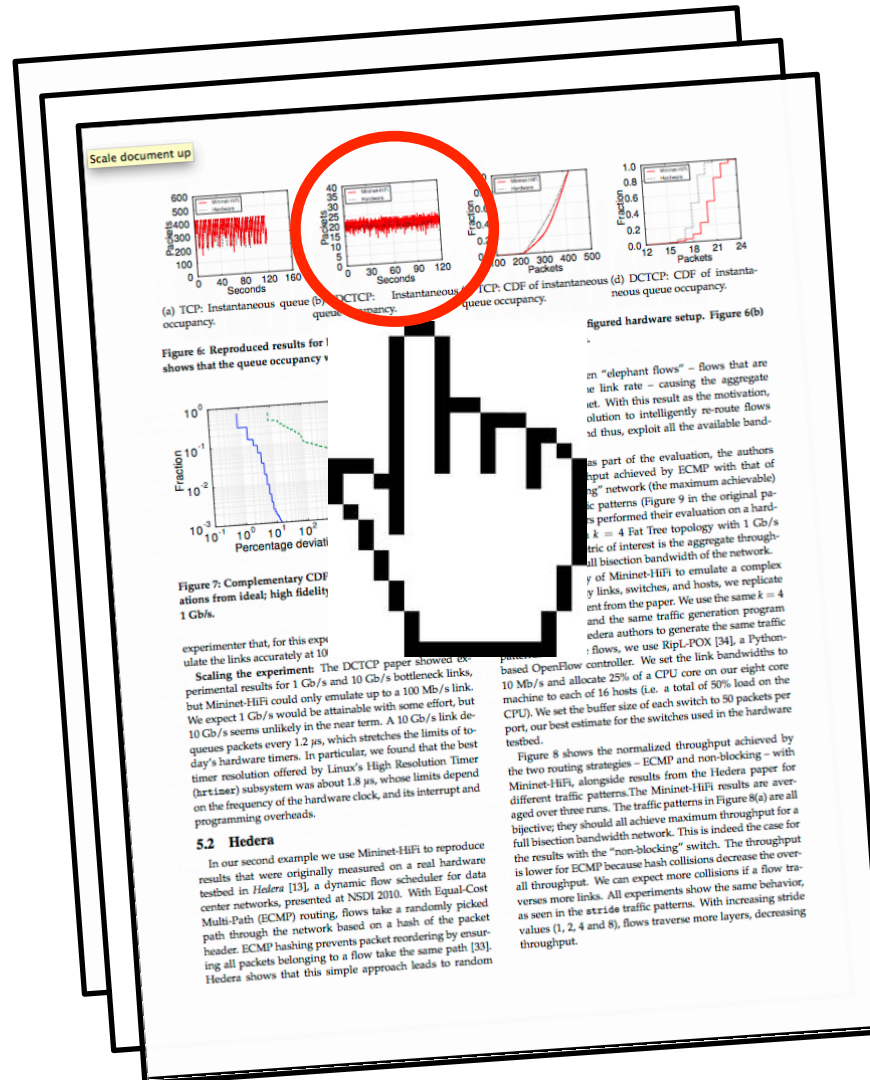
But what about the computational sciences?

D.L. Donoho, 1995:

*“An article about computational science is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.”*

This is what network research should be.  
It should be reproducible.  
Papers should be **runnable**.

# Click on a figure





# Brings up a webpage of instructions

## REPRODUCING NETWORK RESEARCH

using Mininet-HiFi to reproduce published networking experiments

[projects](#) / [about](#) / [contribute](#)

**Can network systems  
research papers be  
replicated?**

In Spring 2012, 37 Stanford CS244 students took on this challenge, using the Mininet-HiFi network emulator on [EC2](#) instances.

This blog details their stories, plus those from the class TAs and others who have been inspired to share their research.

For more details, check out the [Projects](#) gallery, the [About](#) page, or [Contribute](#).

**Tweet/post/send** them to your colleagues, comment at the bottom of each post, or even replicate each blog post using the provided instructions!

FOLLOW BLOG VIA  
EMAIL

### DCTCP

June 9, 2012 · by stanfordcs244 · [Bookmark the permalink.](#)

☆☆☆☆☆ [Rate This](#)

**Team:** Nikhil Handigol, Brandon Heller, Vimal Jeyakumar, and Bob Lantz.

**Key Result(s):** DCTCP consistently maintains a small queue occupancy while maintaining high throughput.

**Source(s):**

1. M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM, pages 63–74. ACM, 2010.
2. Dctcp patches. <http://www.stanford.edu/~alizade/Site/DCTCP.html>.
3. M. Alizadeh, A. Javanmard, and B. Prabhakar. Analysis of dctcp: stability, convergence, and fairness. In Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, pages 73–84. ACM, 2011.
4. K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification (ecn) to ip. 1999.

**Contacts:** Nikhil Handigol ([nikhilh@stanford.edu](mailto:nikhilh@stanford.edu)), Brandon Heller ([brandonh@stanford.edu](mailto:brandonh@stanford.edu)), Vimal Jeyakumar ([jvimal@stanford.edu](mailto:jvimal@stanford.edu)), Bob Lantz ([rlantz@cs.stanford.edu](mailto:rlantz@cs.stanford.edu))

Introduction

# Scroll to the bottom...

Emulating a 10Gb/s link would require full-sized packets to be dequeued every 1.2 $\mu$ s, which stretches the limits of today's hardware support for timers. In particular, we found that the best timer resolution offered by Linux's 'High Resolution Timers' (hrtimer) subsystem was about 1.8 $\mu$ s. (This value depends on the frequency of the hardware clock and overheads in programming them.)

## Instructions to Replicate This Experiment:

```
git clone https://bitbucket.org/nikhilh/mininet_tests.git
cd mininet_tests/dctcp
```

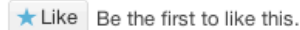
Follow the instructions in the README file there:

[https://bitbucket.org/nikhilh/mininet\\_tests/src/ad08368cf347/dctcp/README](https://bitbucket.org/nikhilh/mininet_tests/src/ad08368cf347/dctcp/README)

Share this:



Like this:



Tags: congestion control, data center, dctcp, ecn, red, tcp

# Launch an EC2 instance

The screenshot shows the AWS Management Console interface for the 'My Instances' page. The browser address bar indicates the URL is `https://console.aws.amazon.com/ec2/home?region=us-east-1#s=Instances`. The left sidebar contains a navigation menu with the following items:

- EC2 Dashboard
- Events
- INSTANCES
  - Instances (selected)
  - Spot Requests
  - Reserved Instances
- IMAGES
  - AMIs
  - Bundle Tasks
- ELASTIC BLOCK STORE
  - Volumes
  - Snapshots
- NETWORK & SECURITY
  - Security Groups
  - Elastic IPs
  - Placement Groups
  - Load Balancers
  - Key Pairs
  - Network Interfaces

The main content area, titled 'My Instances', features a 'Launch Instance' button and a table of instances. The table has the following columns: Name, Instance, AMI ID, Root Device, Type, State, Status Checks, Alarm Status, Monitoring, Security Groups, Key Pair Name, Virtualization, and Placement Group. A single instance is listed:

Name	Instance	AMI ID	Root Device	Type	State	Status Checks	Alarm Status	Monitoring	Security Groups	Key Pair Name	Virtualization	Placement Group
empty	i-807261f8	ami-d3248aba	ebs	c1.medium	stopped		none	basic	default	jvimal-east	paravirtual	

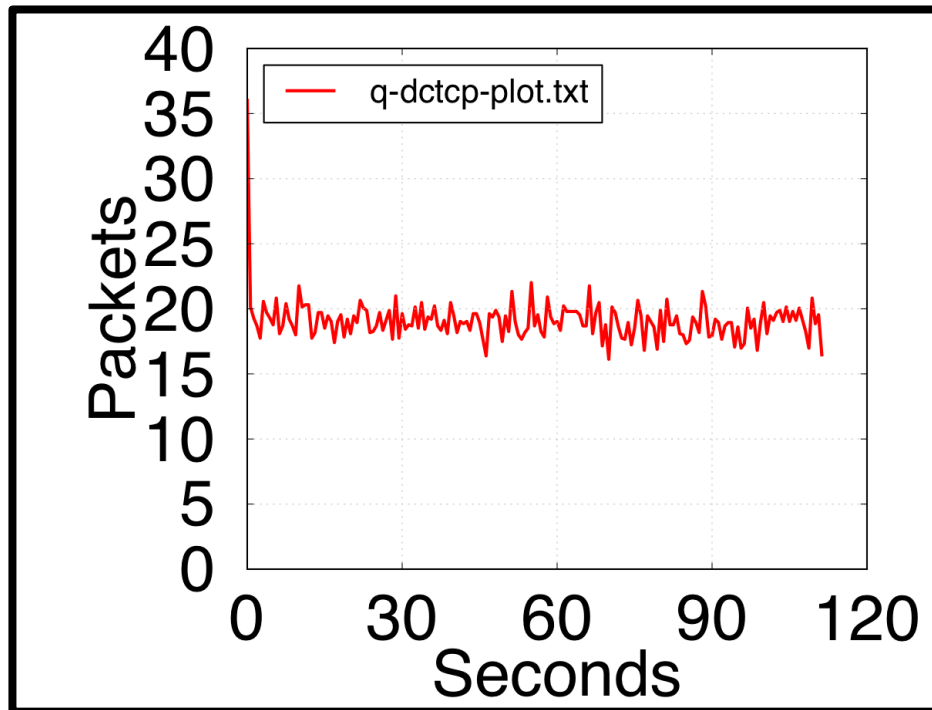
Below the table, a message states: 'No EC2 Instances selected. Select an instance above'.

Run a command in the terminal to  
generate results

```
> ~/mininet-tests/dctcp$ ./run-dctcp.sh
```

# 8 minutes & 8 cents after the click: a reproduced result

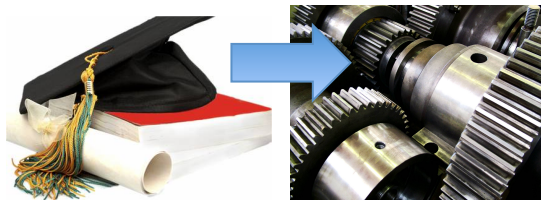
```
> ~/mininet-tests/dctcp$ ./run-dctcp.sh  
> ~/mininet-tests/dctcp/results$ xpdf dctcp.pdf
```



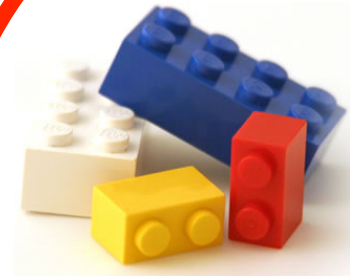
# If papers were runnable (easily reproducible):



easier to understand  
and evaluate papers



easier to transfer  
new ideas to industry

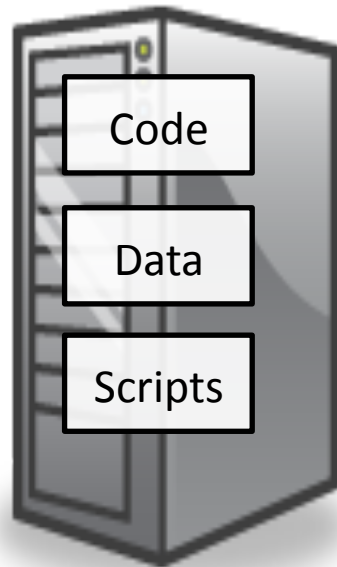


easier to build upon  
the work of others

Why aren't all  
networking research  
papers like this?

# Much of CS

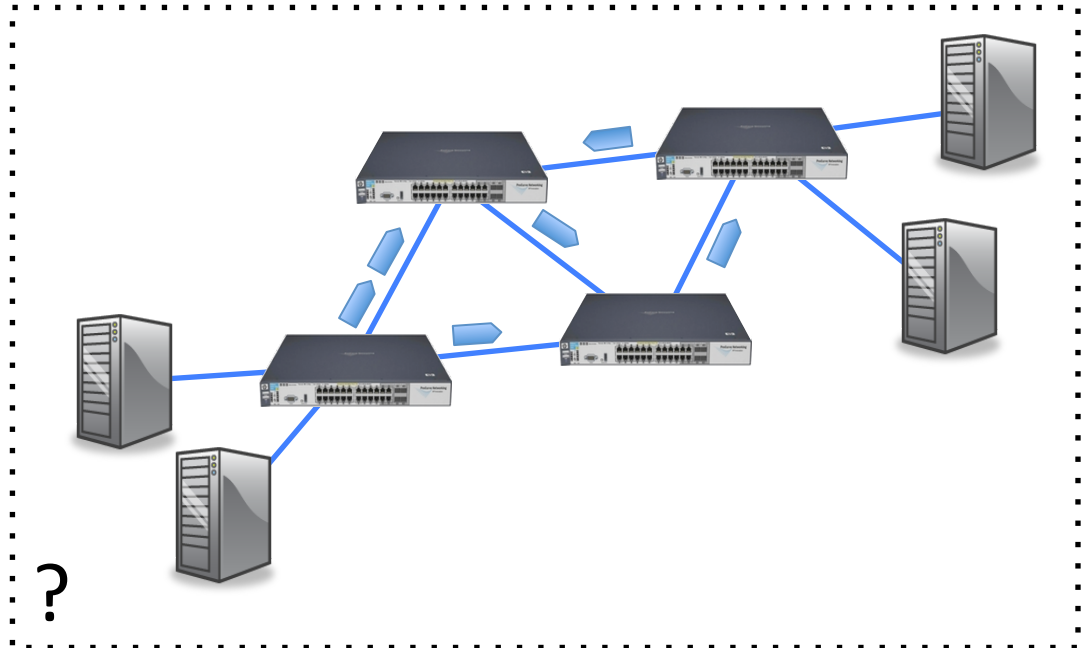
ML, PL, Security, ...



Use any commodity PC  
(or VM.)

# Network Systems

Congestion control, Routing, Topology, ...

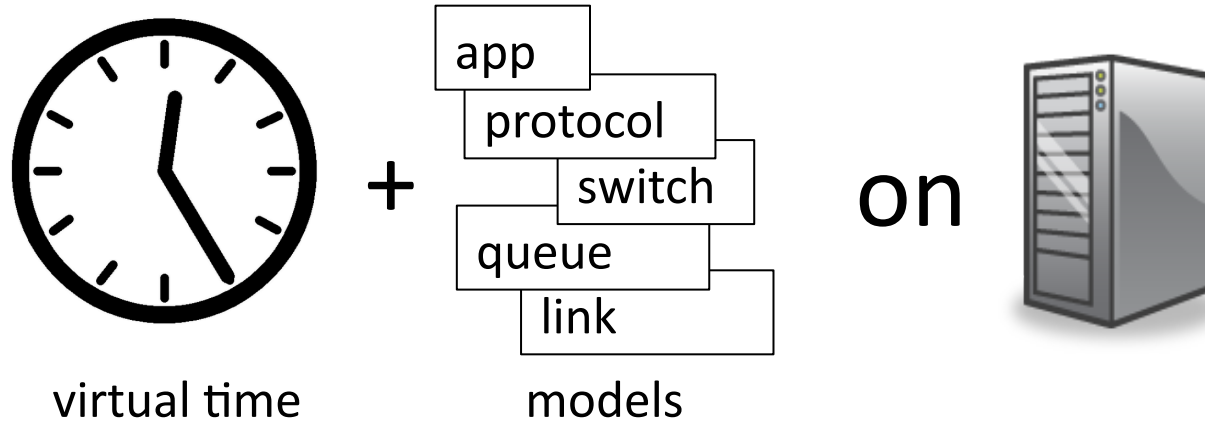


Must implement **multiple**  
servers, network elements, links –  
all running in **parallel**,  
all with **accurate timing**



# Discrete-Event Simulation

ns-2, ns-3, OPNET, ...



# But... we don't trust simulation.

Not *believable* unless validated:

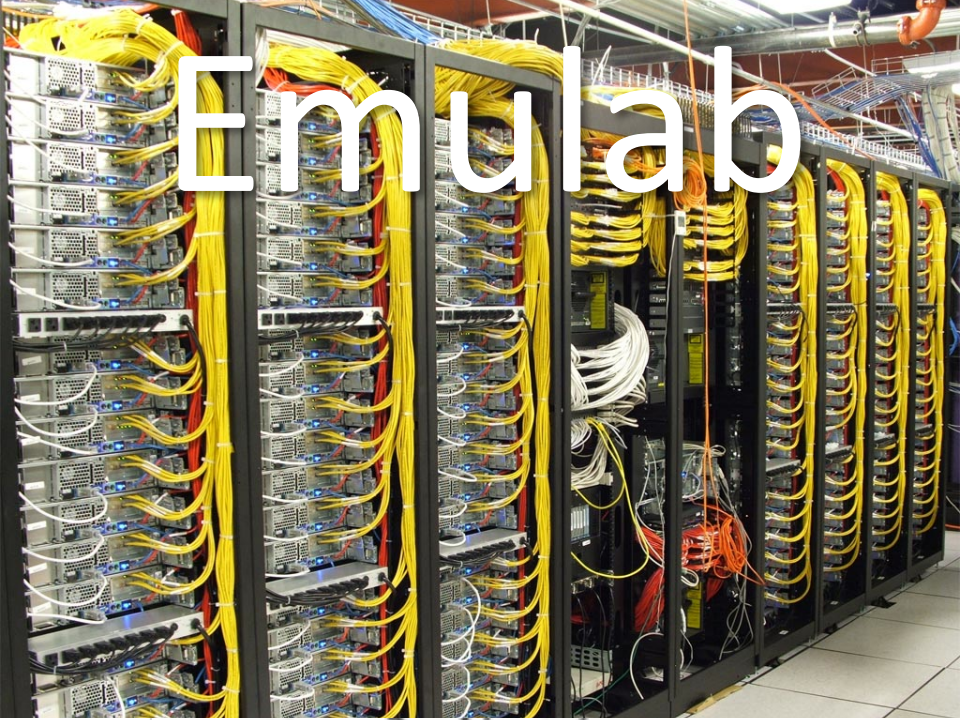
Modeled links == real links

Modeled switches == real switches

Modeled protocols == real protocols

Modeled applications == real applications

→ **Realism concerns.**



# Emulab



# GENI



# OFELIA

# Shared Testbeds

# Testbed results can be hard to (re)produce.

## Flexibility

- Topology restrictions
- May not be able to change firmware

## Resource availability

- before a deadline?
- one year later?

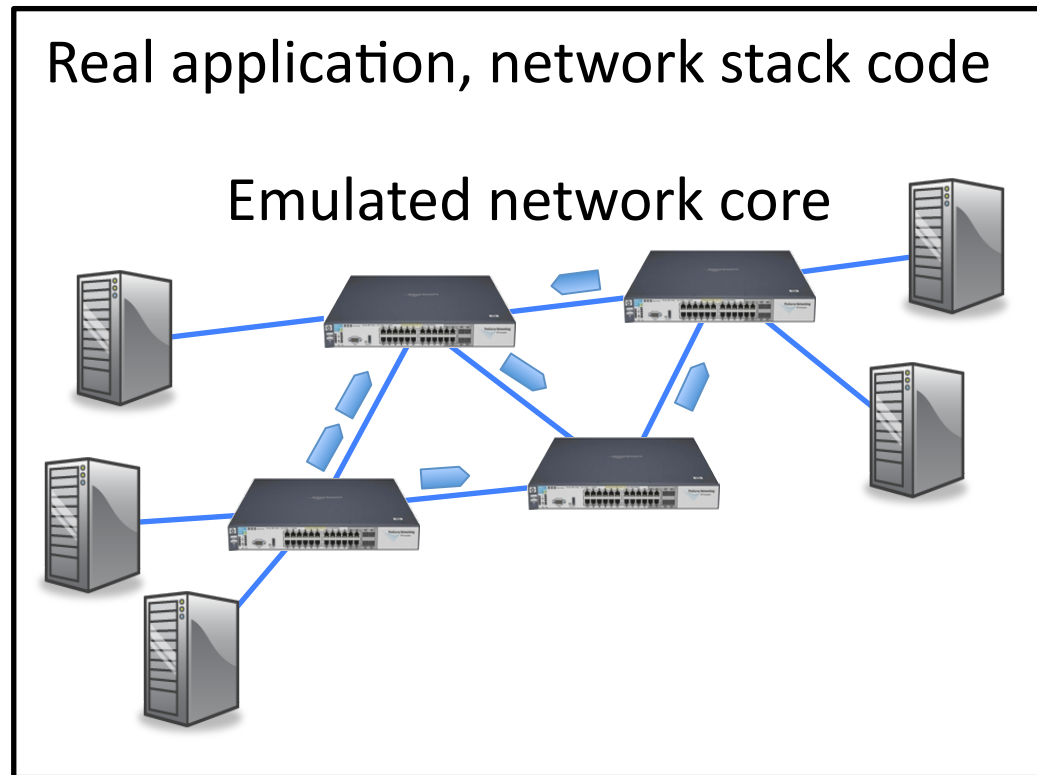
# Problem

Network research tends not to be both  
*easily reproducible* and *realistic*.

# Solution

Build an emulator whose results you  
can trust as well as verify.

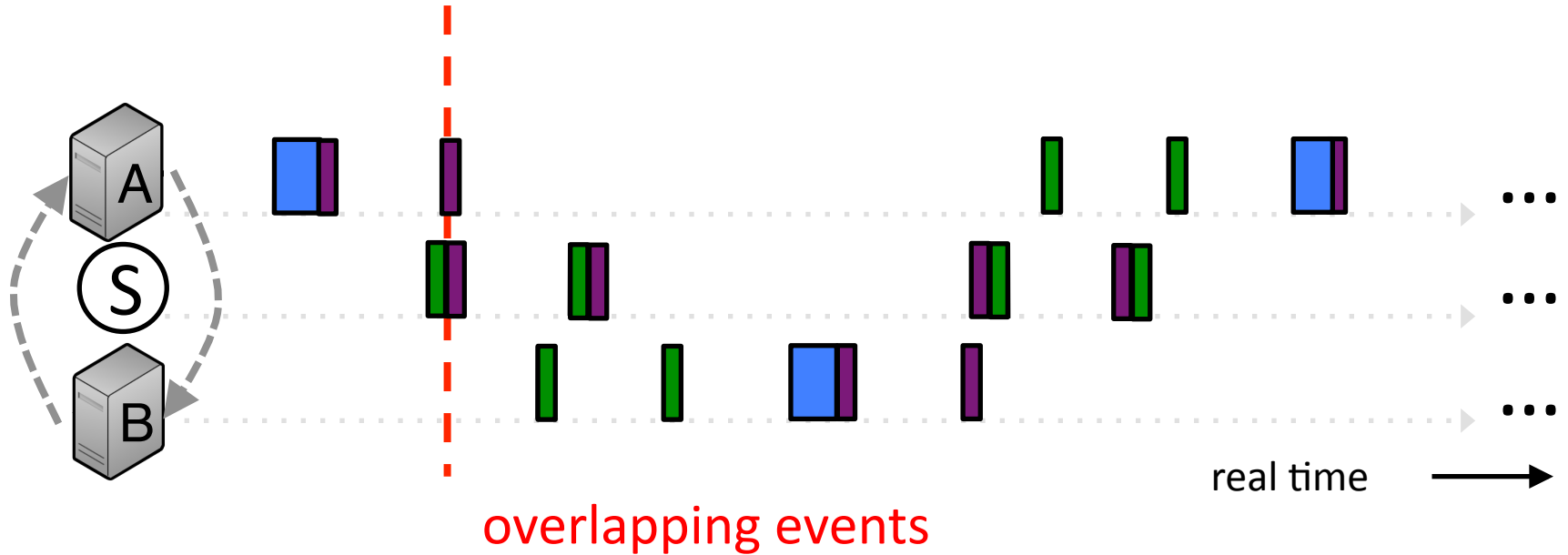
# What is a network emulator?



Matching the behavior of hardware:  
High Fidelity

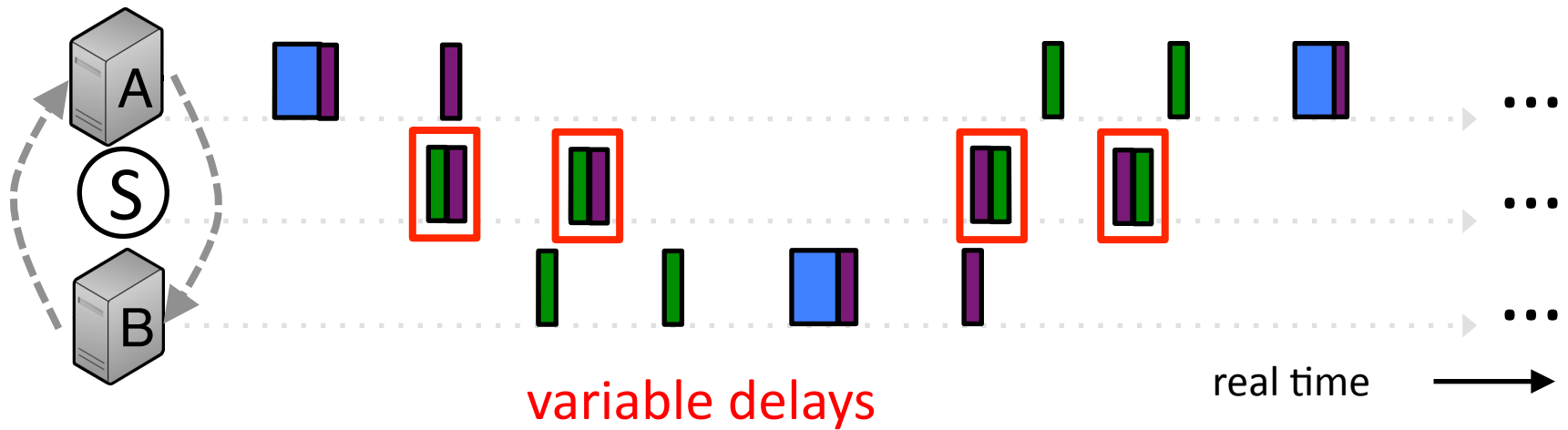
# Sources of Emulator Infidelity

## Event Overlap



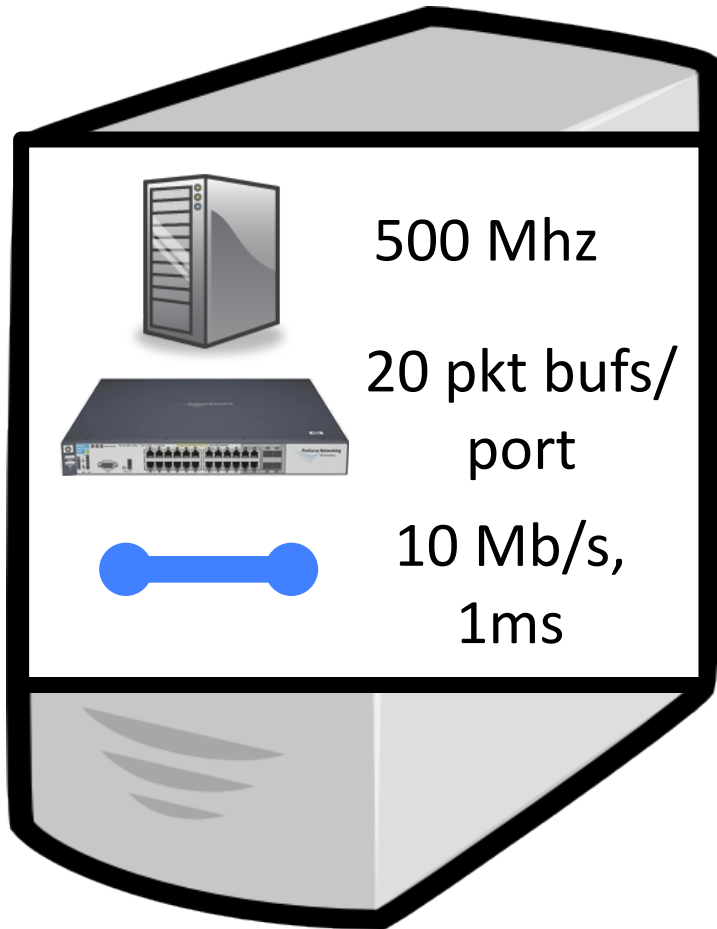
# Sources of Emulator Infidelity

## Software Forwarding





# Our Approach



+



Resource-Isolating  
Emulator (Mininet-HiFi)

Fidelity  
Monitor

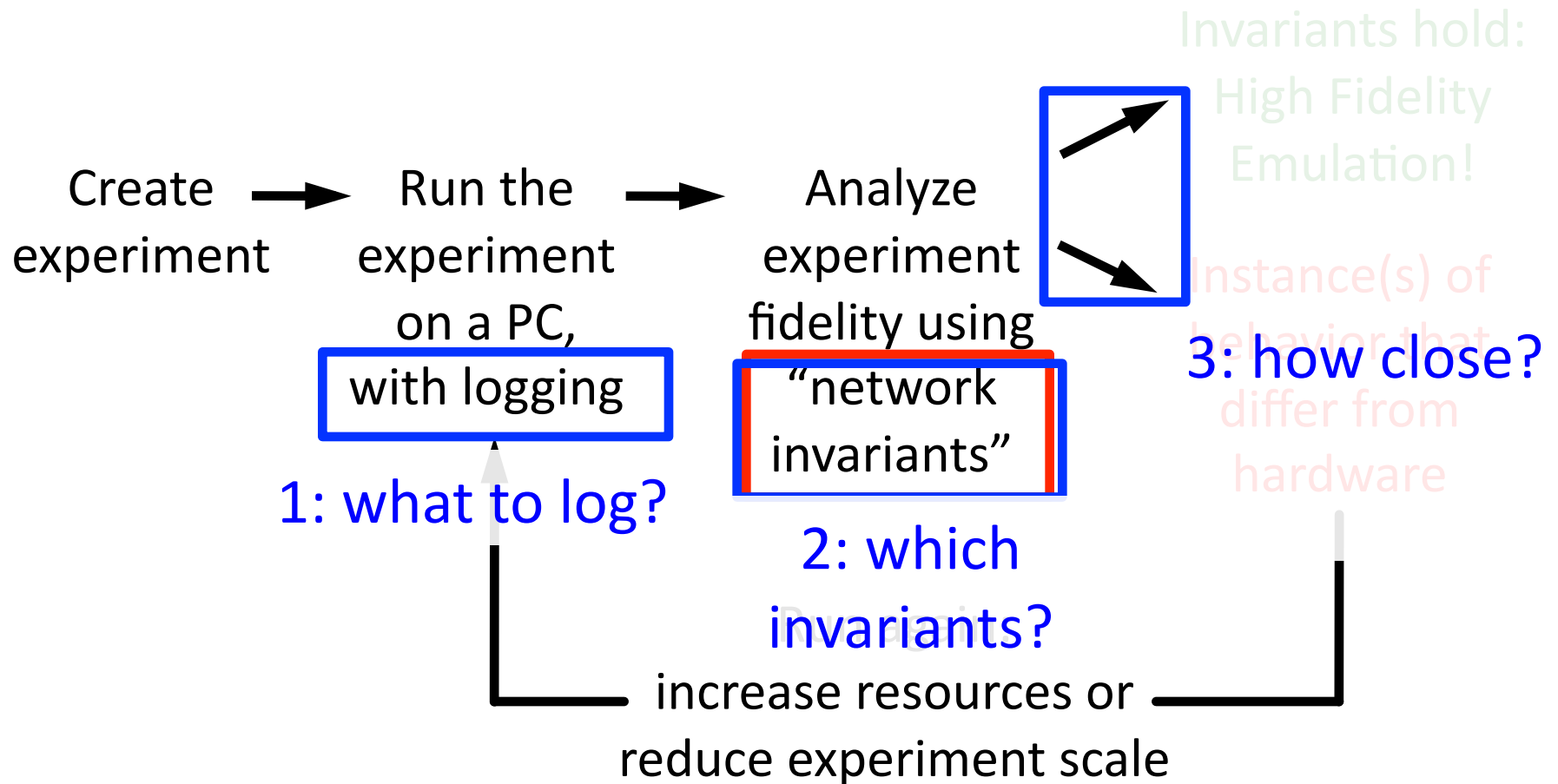
# Talk Outline

- ~~Motivation~~
- 1. Emulator Fidelity
- 2. Mininet-HiFi Architecture
- 3. Reproducing Research
- Related Work
- Progress Report

1.

# Emulator Fidelity

# A Workflow for High Fidelity Emulation

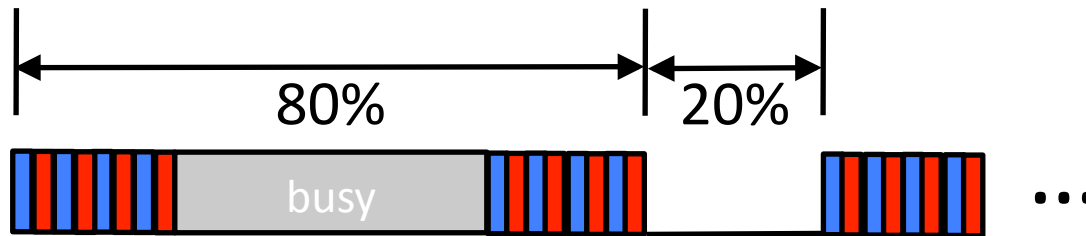


# What to log?

Consider logging utilization of the emulator CPU.

100% is bad.

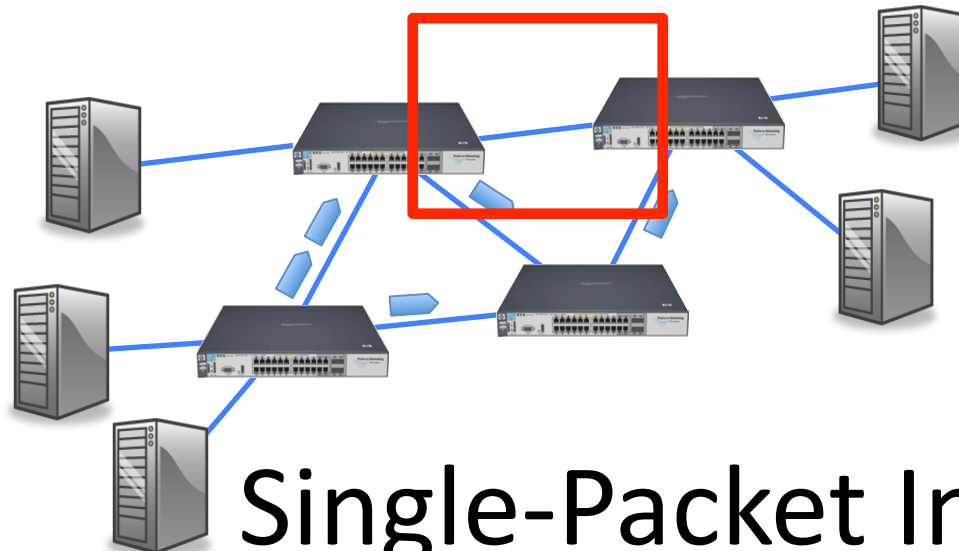
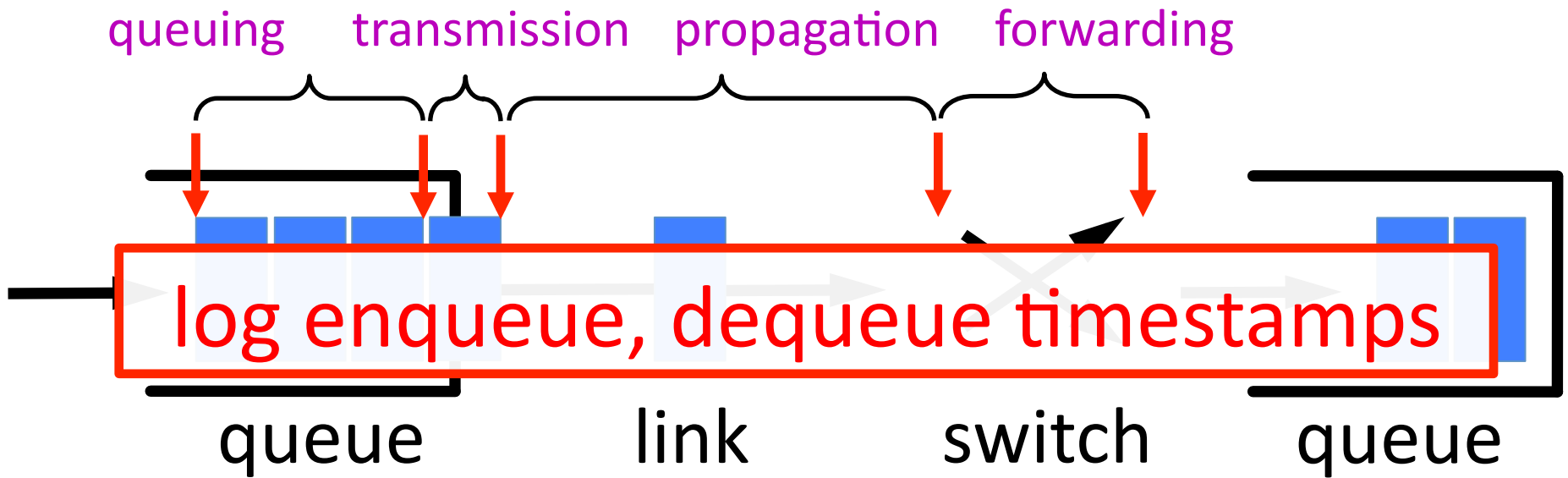
But is X% (say, 80%) necessarily good?



Can't get back "lost time" in an emulator.

→ CPU utilization is insufficient.

Need to consider **fine-grained event timings**.

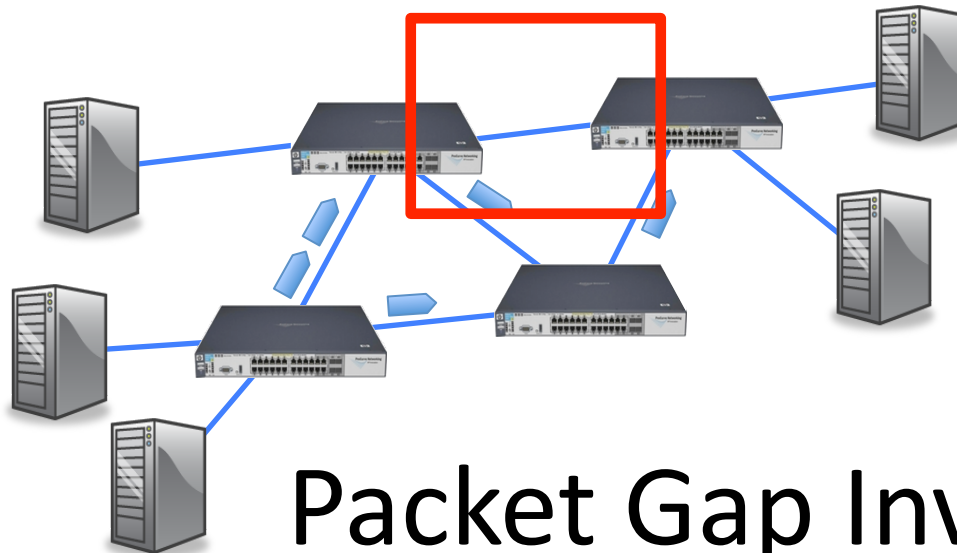
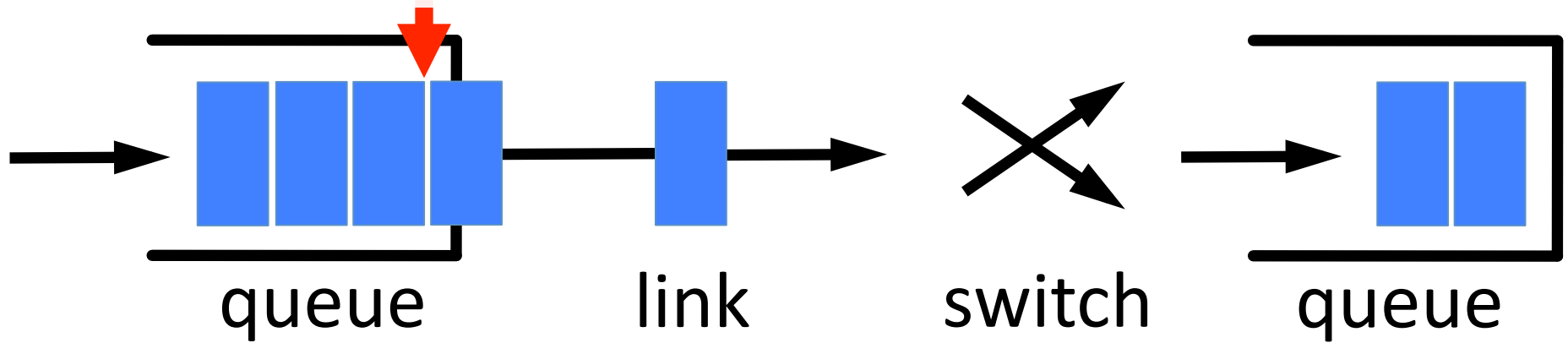


# Single-Packet Invariants

(when queue occupied)

packet spacing

Explored in the paper



# Packet Gap Invariants

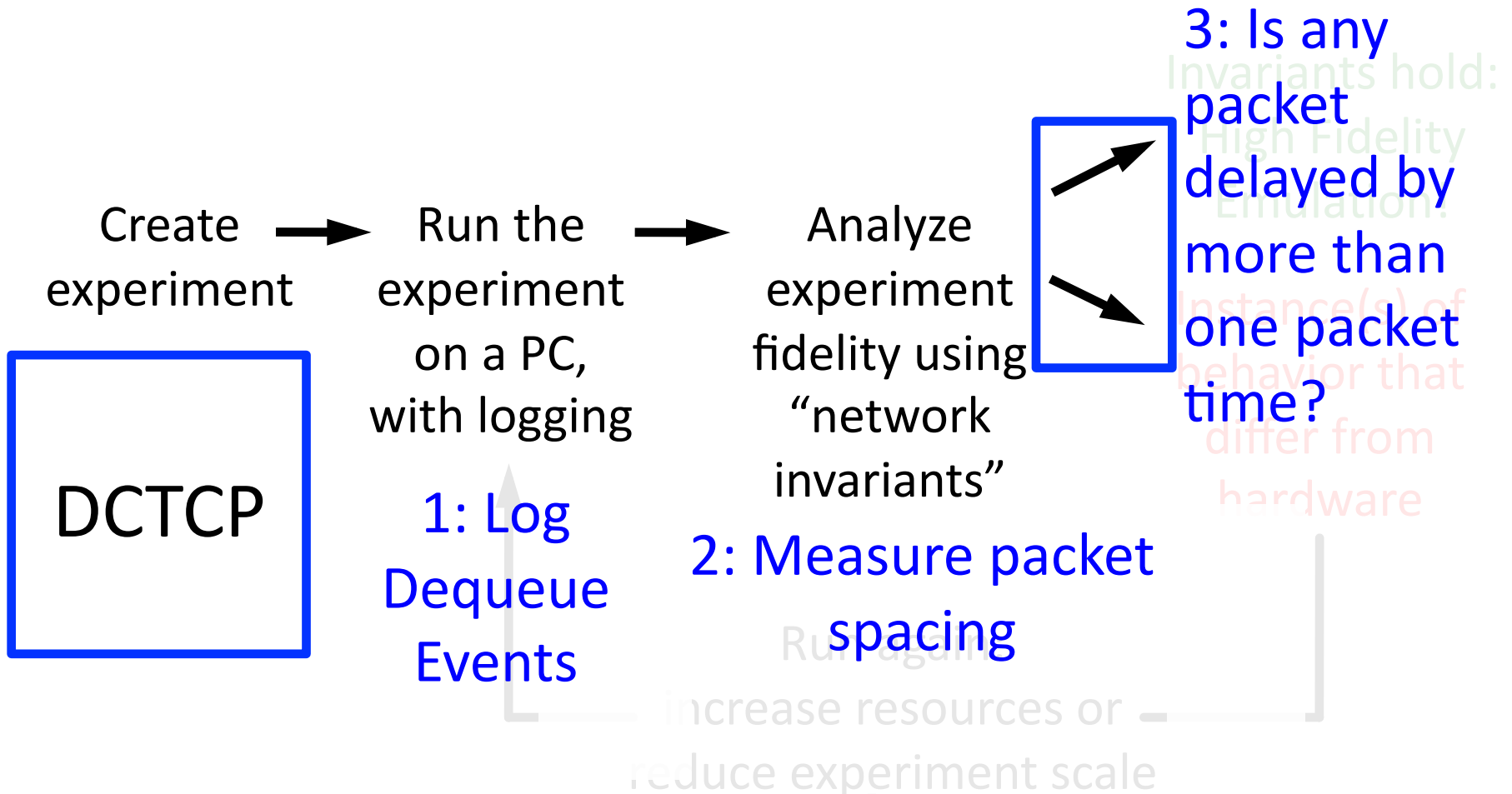
# How close?

**High Fidelity:** match hardware variability.

- Clock drift (**== one packet**)
- NIC to memory processing ( $\sim$  25 packets)
- Scheduler non-determinism ( $\sim$  milliseconds)

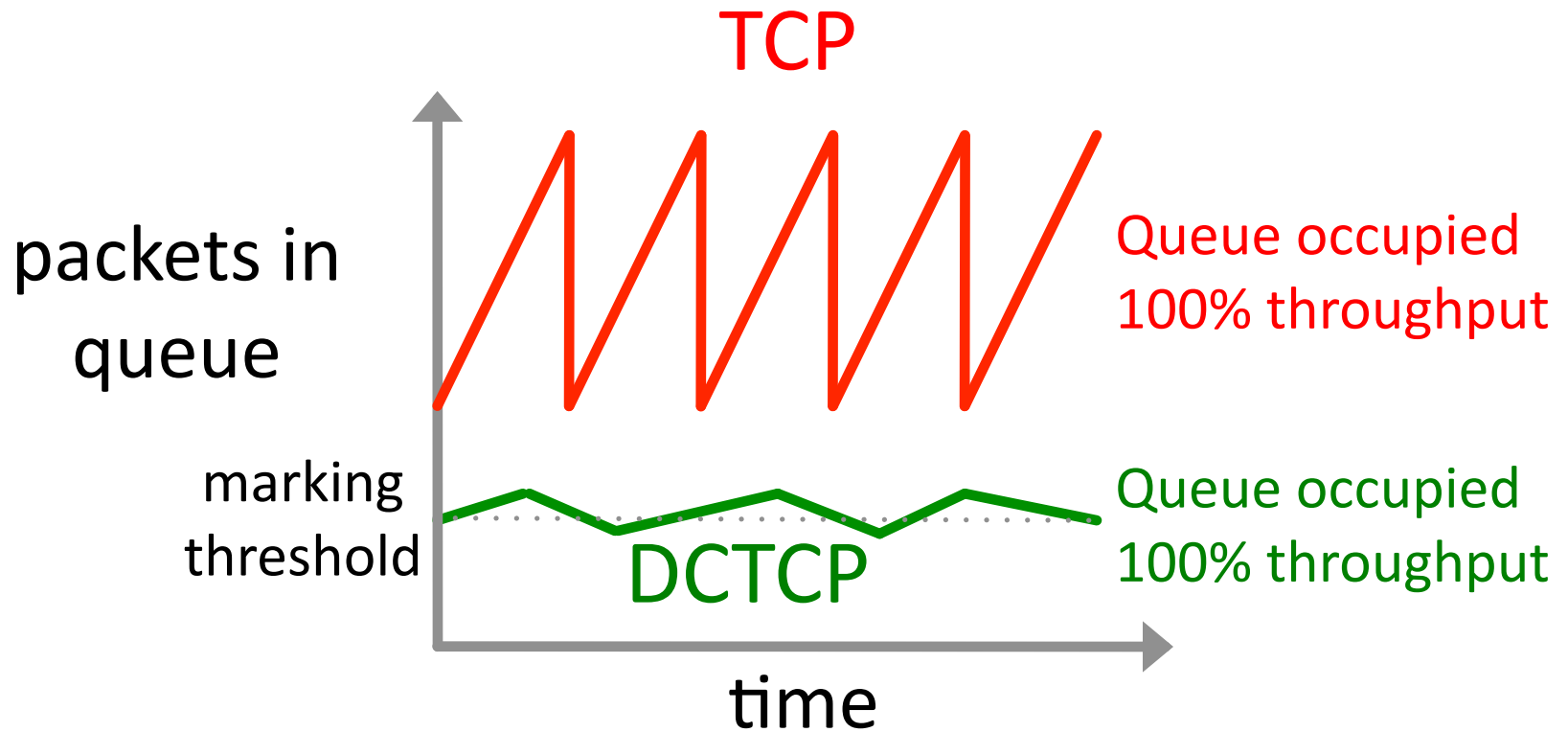


# Example Workflow for One Invariant

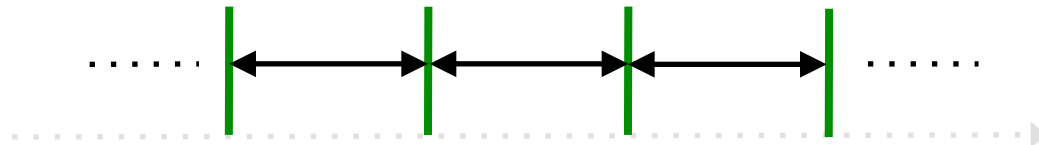


If this workflow is valid, “pass” → same result as hardware.

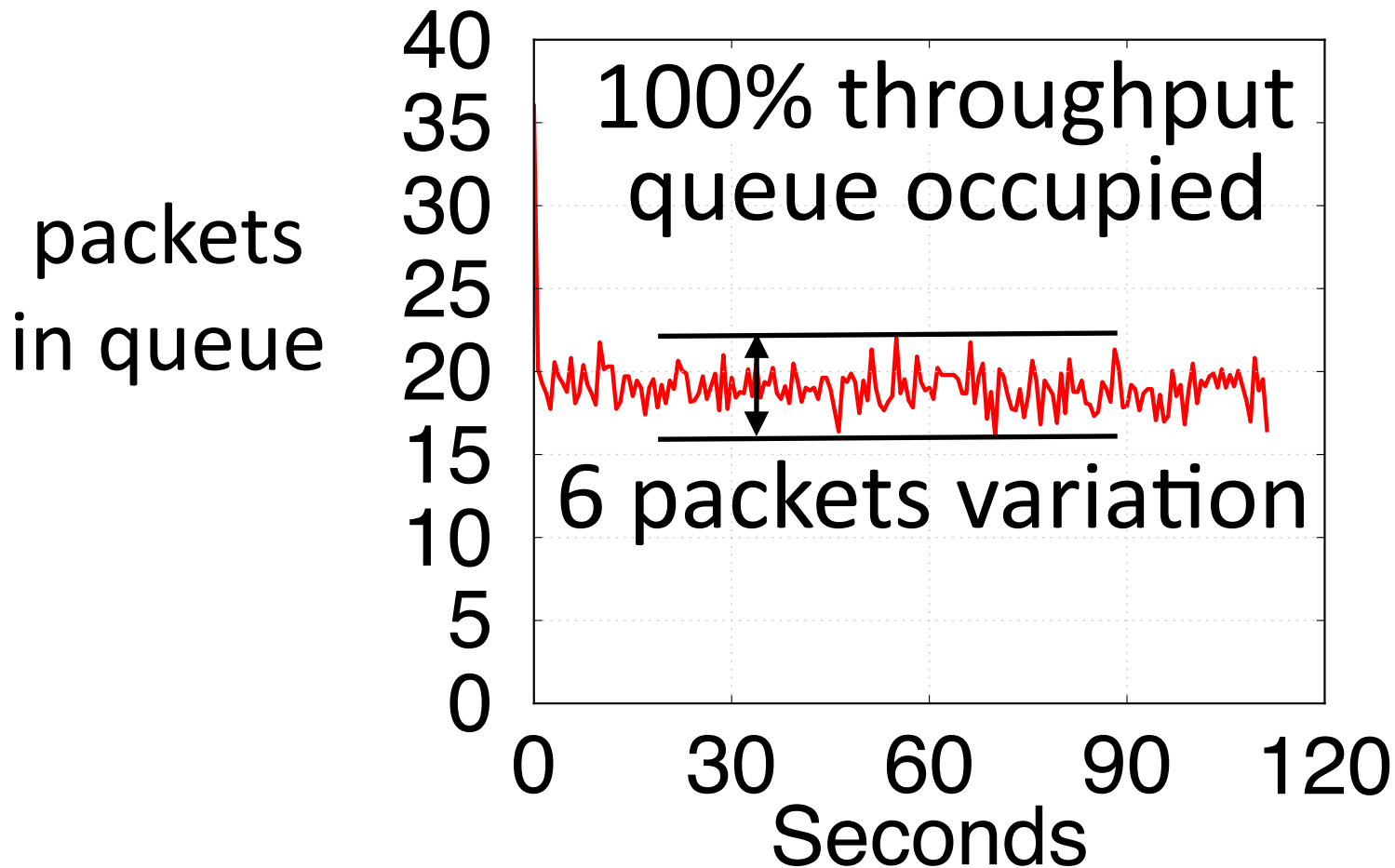
# Data Center TCP (DCTCP)



Packet spacing we should see:

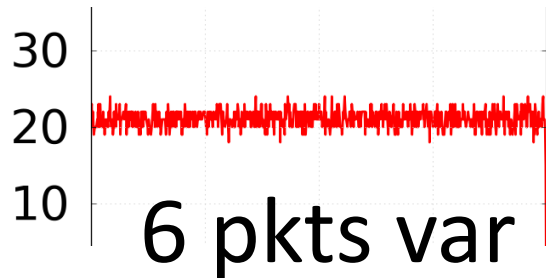
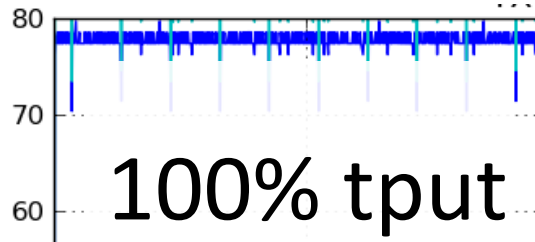


# Hardware Results, 100 Mb/s



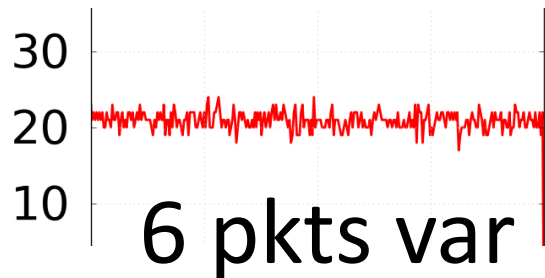
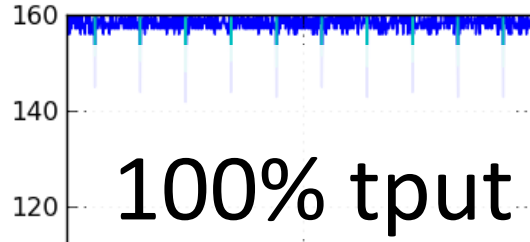
# Emulator Results

80 Mb/s



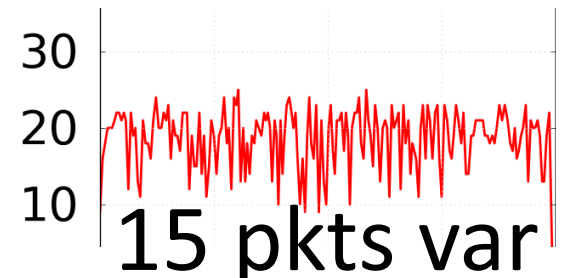
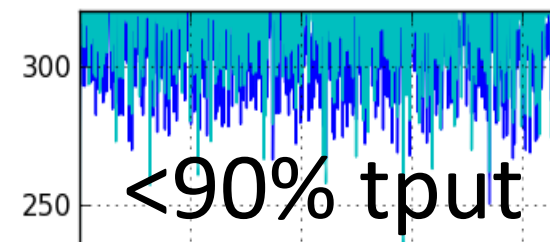
same result

160 Mb/s



same result

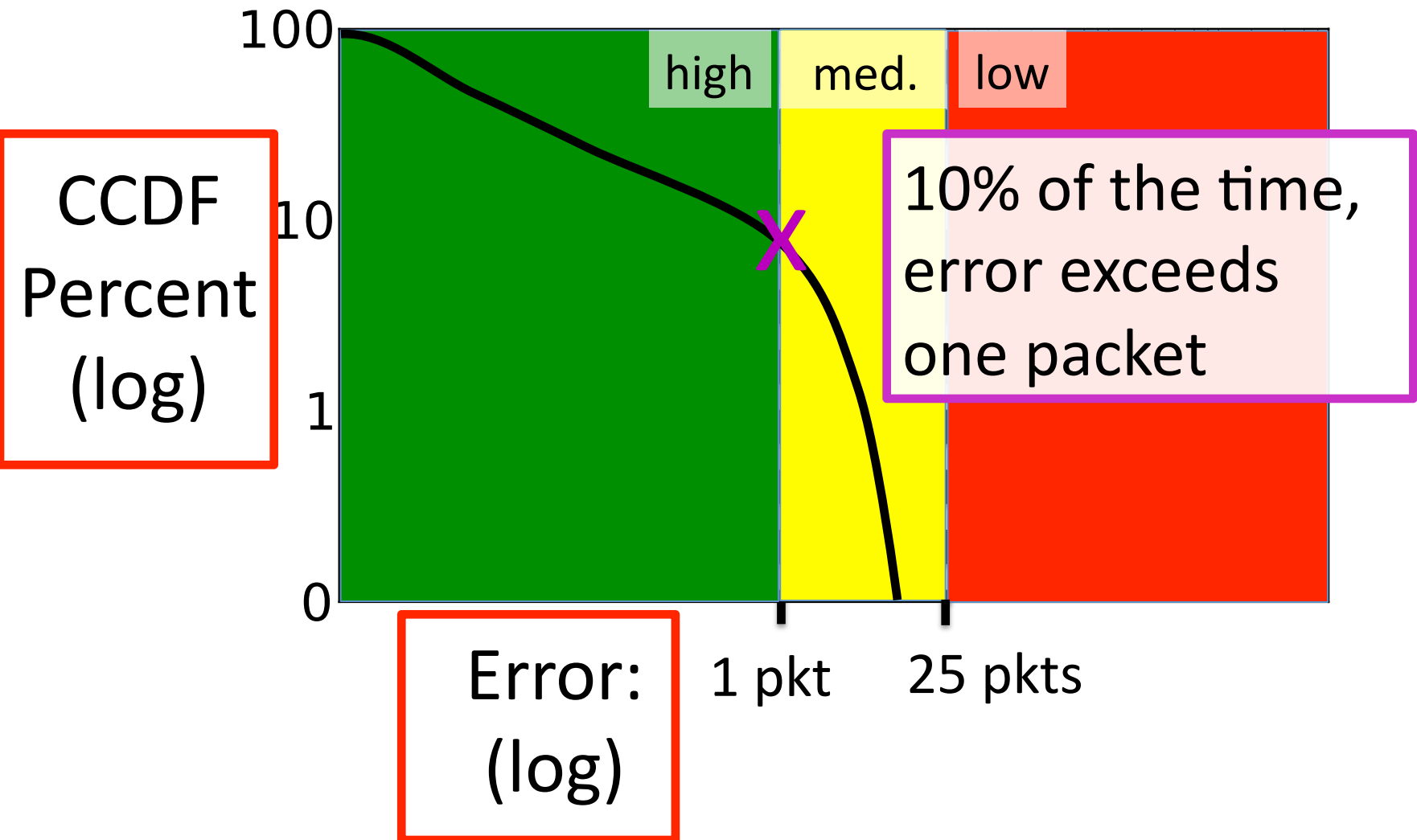
320 Mb/s



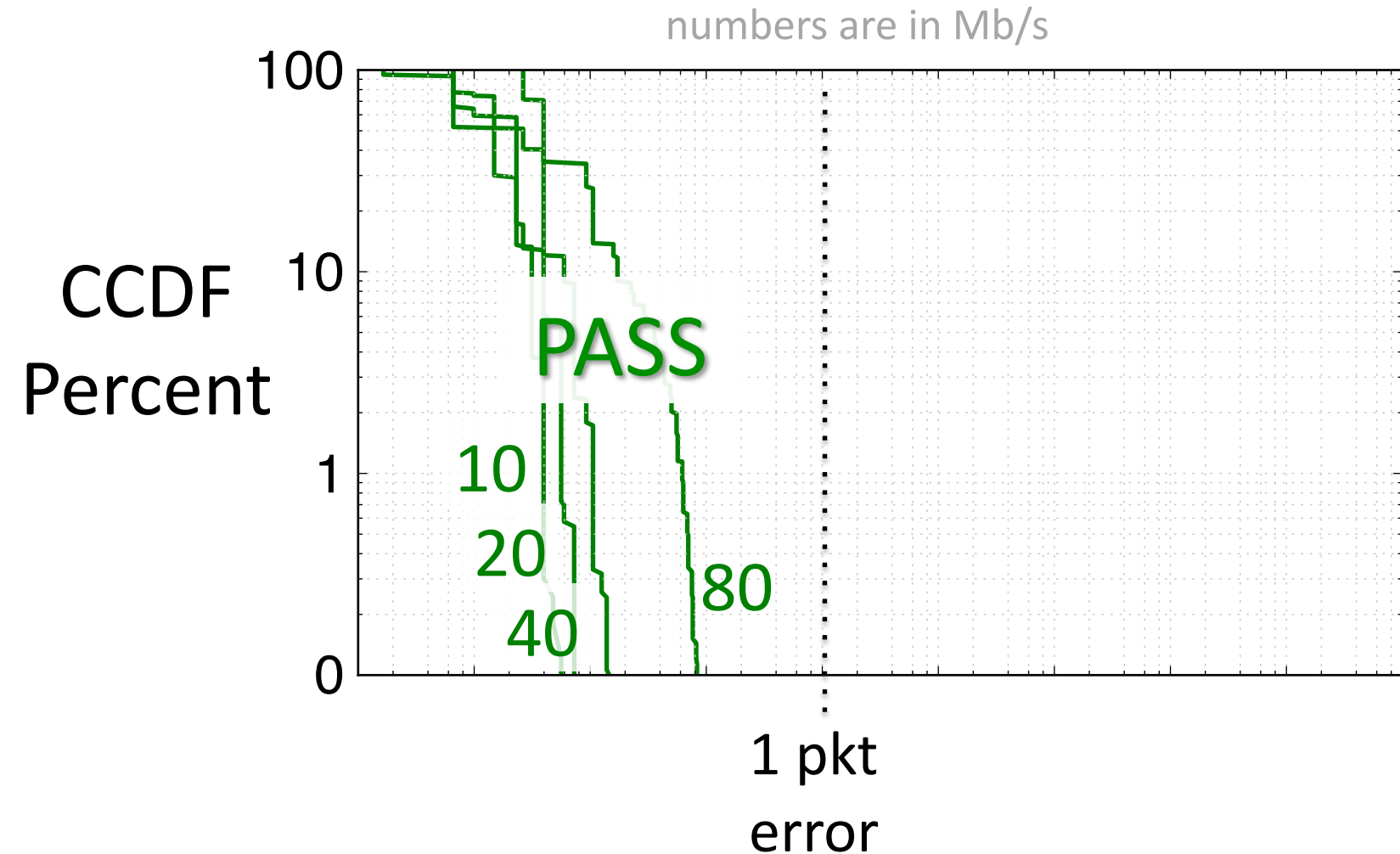
wrong;  
limits exceeded

Does checking an invariant (packet spacing) identify wrong results?

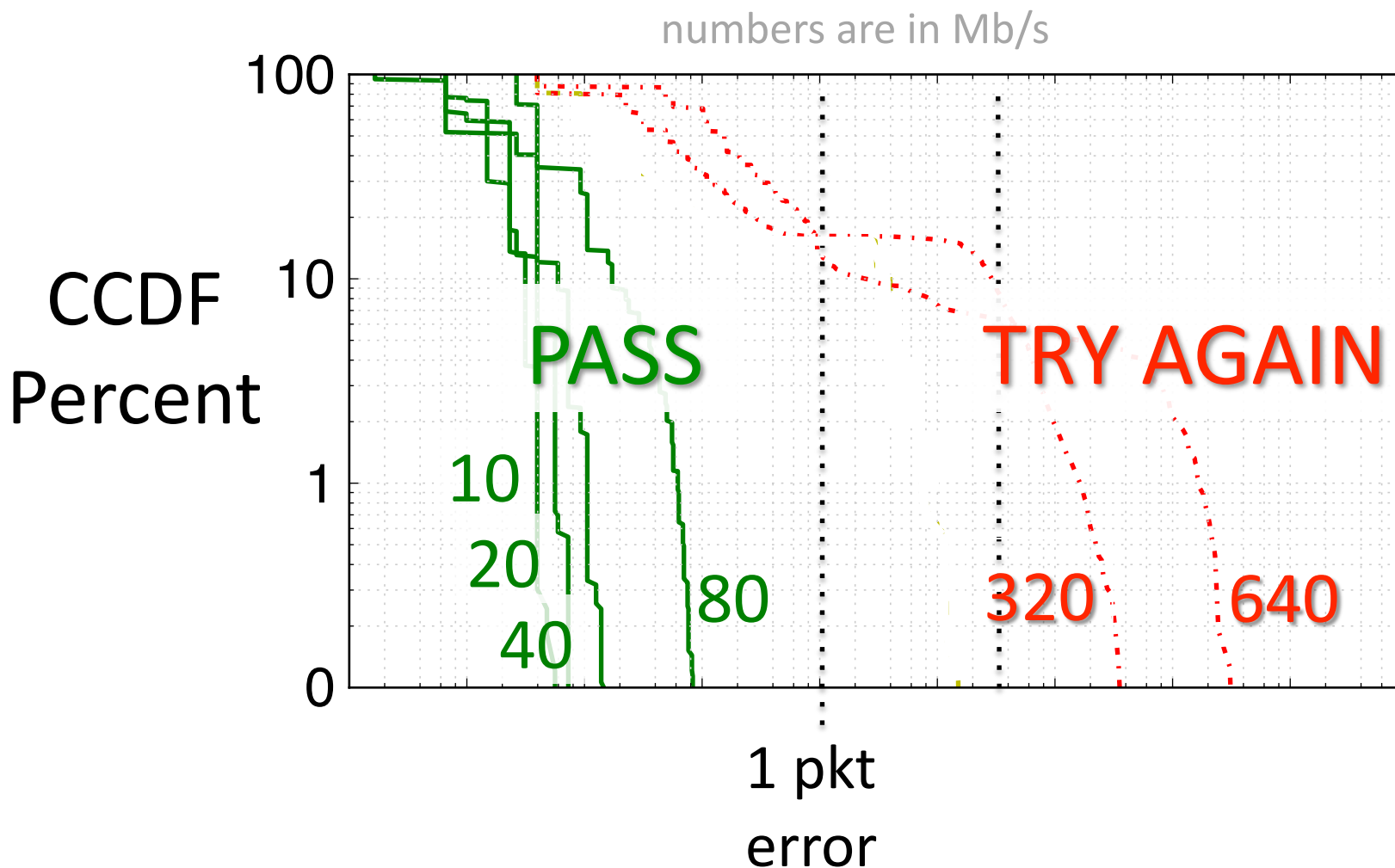
# Packet Spacing Invariant w/DCTCP



# Packet Spacing Invariant w/DCTCP

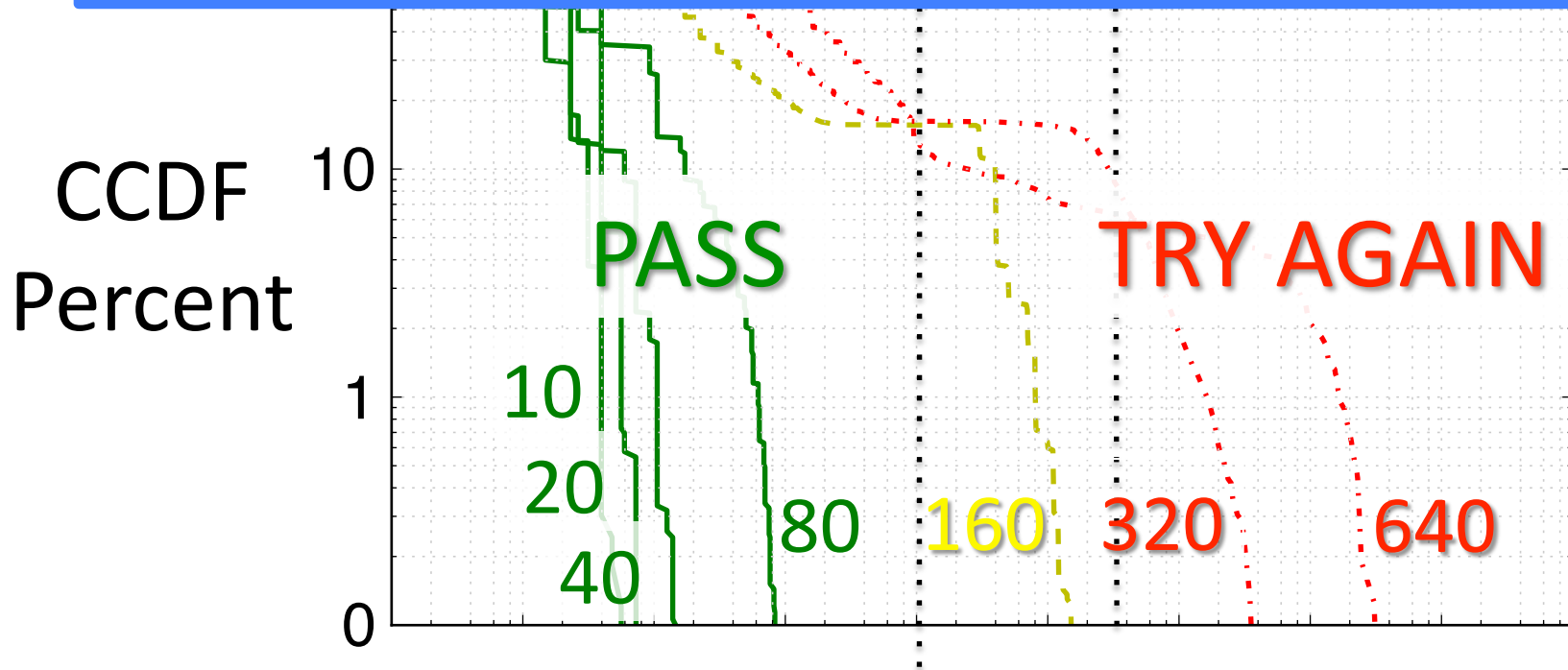


# Packet Spacing Invariant w/DCTCP



160 Mb/s: failed emulation?

Beauty of networks invariants is that it catches and quantifies the error in this run.



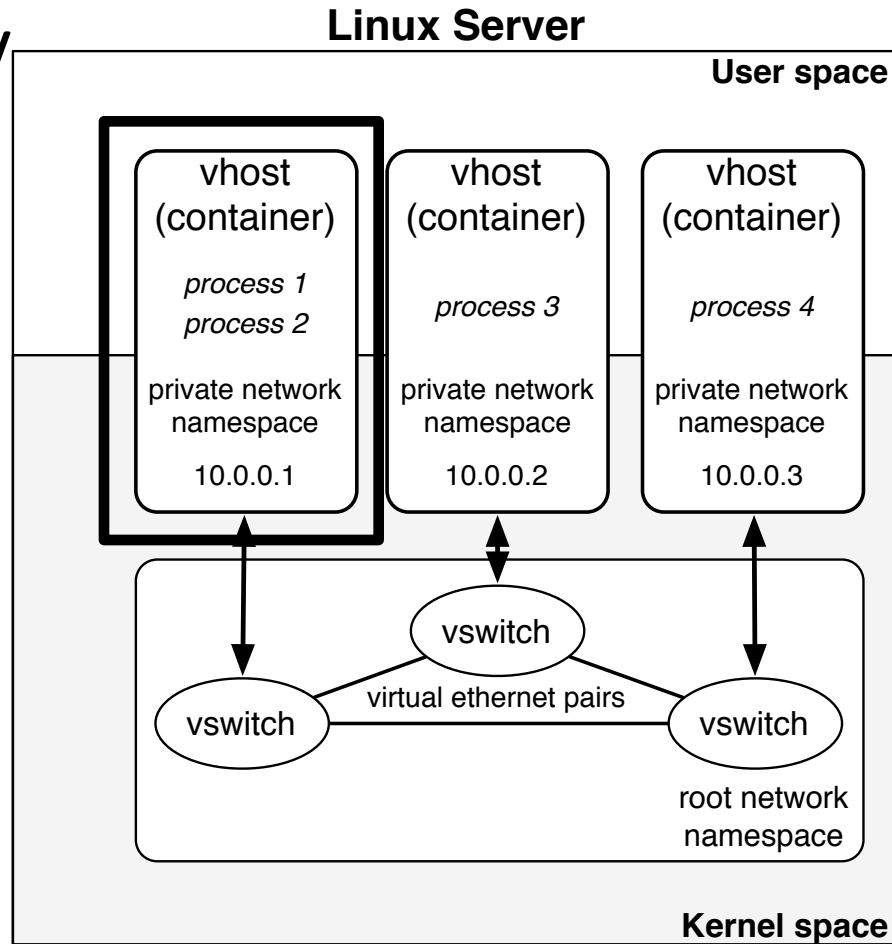


2:

# Mininet-HiFi Architecture

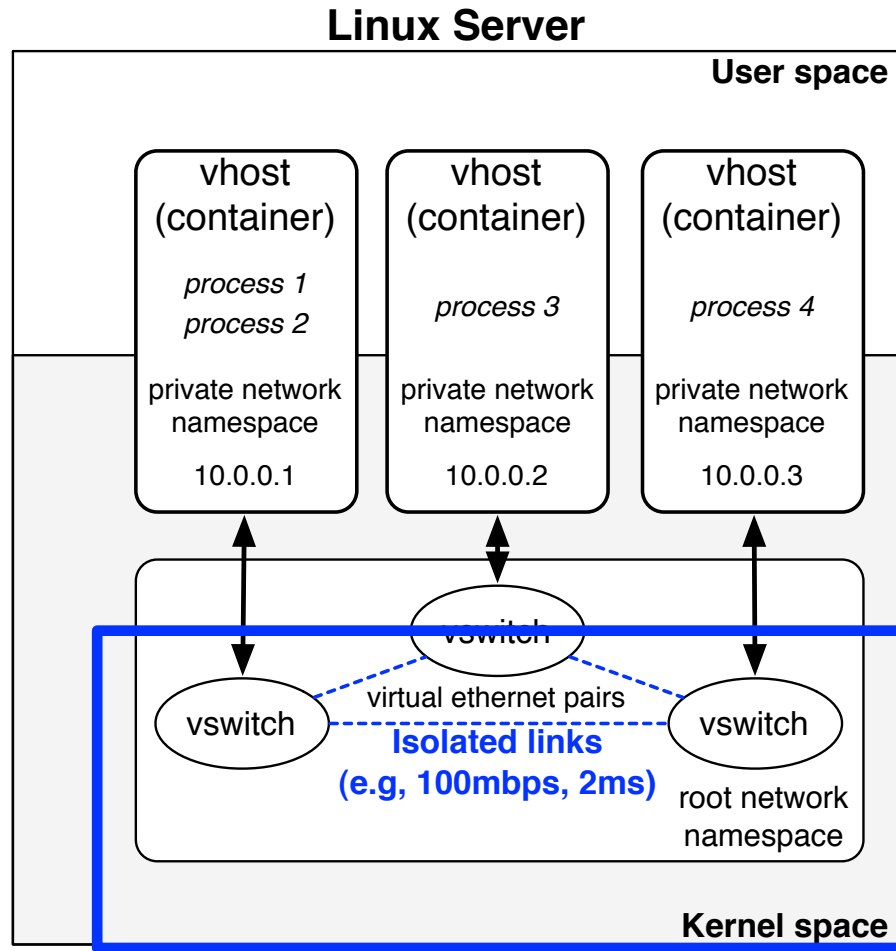
# Original Mininet

## Containers w/ Network Namespaces



Emulator

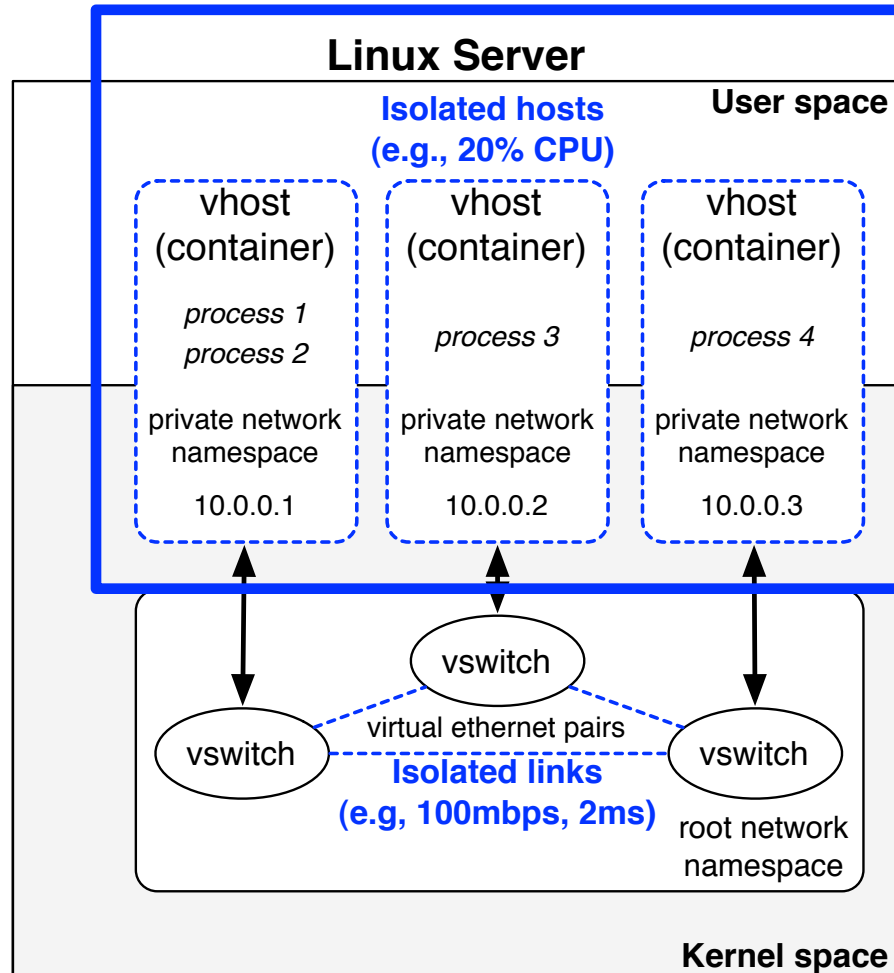
# Emulator + Performance Isolation



Linux  
packet  
schedulers  
HTB, HFSC

Emulator + Performance Isolation

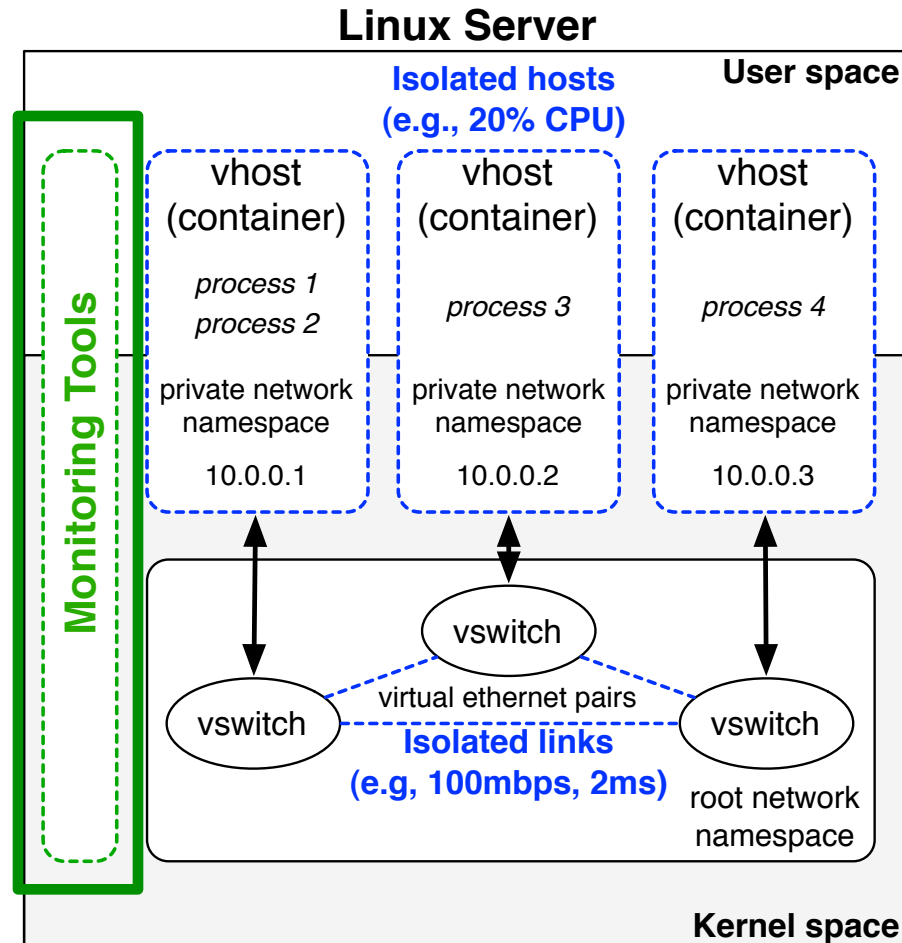
# Emulator + Performance Isolation



Linux  
process  
schedulers  
CFS+BWlimits,  
RT

# Emulator + Performance Isolation + Invariant Monitoring

Linux  
Kernel  
Tracing  
enqueue,  
dequeue, etc.



3:

# Reproducing Research

# Examples in the paper

- DCTCP [Alizadeh, SIGCOMM 2010]
- Router Buffer Sizing [Appenzeller, SIGCOMM 2004]
- Hedera ECMP [Al-Fares, NSDI 2010]

Able to replicate key results from 3 testbeds ...  
using an emulator.

How do you know it  
*really* works?

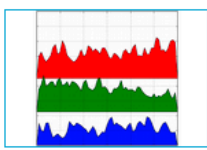
Test it. On ~~guinea pigs~~  
students.



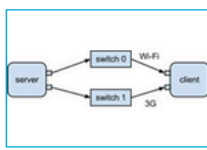


# Stanford CS244 Spring '12: Advanced Topics in Networking

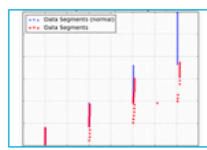
- Pick a paper.
- Reproduce a key result, or challenge it (with data).
- You have:
  - \$100 EC2 credit,
  - 3 weeks, and
  - must use Mininet-HiFi.



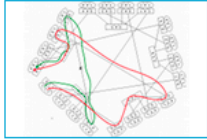
Exploring Outcast



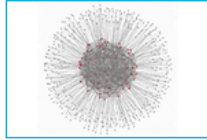
Multipath TCP over WiFi and 3G links



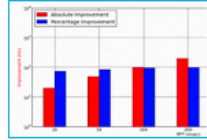
TCP Daytona: Congestion Control with a Misbehaving Receiver



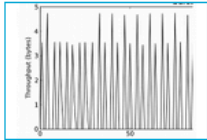
DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



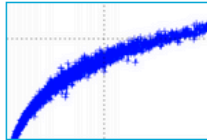
Jellyfish vs. Fat Tree



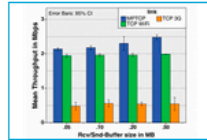
Choosing the Default Initial Congestion Window



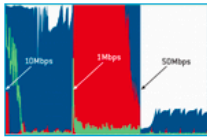
Seeing RED



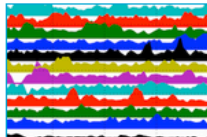
Why Flow-Completion Time is the Right Metric for Congestion Control



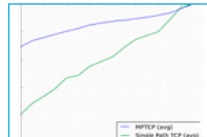
MPTCP Wireless Performance



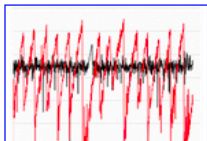
Solving Bufferbloat - The CoDel Way



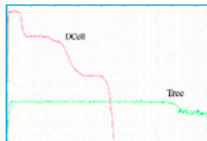
Life's not fair, neither is TCP (... under the following conditions)



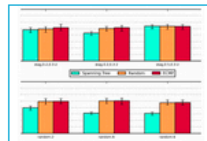
Fairness of Jellyfish vs. Fat-Tree



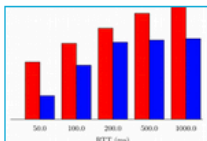
DCTCP and Queues



DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



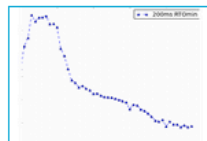
Hedera



Increasing TCP's Initial Congestion Window



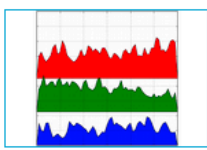
HULL: High Bandwidth, Ultra Low Latency



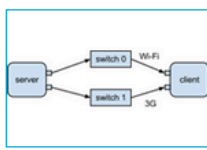
TCP Incast Collapse

CoDel  
HULL  
MPTCP  
Outcast  
Jellyfish  
DCTCP  
Incast  
Flow Completion Time  
Hedera  
DCell  
TCP Initial Congestion Window  
Misbehaving TCP Receivers  
RED

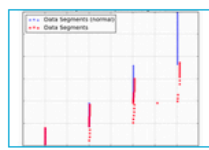
Project Topics:  
Transport,  
Data Center,  
Queuing



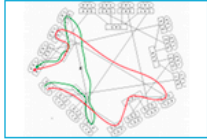
Exploring Outcast



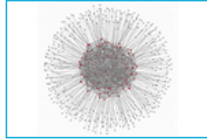
Multipath TCP over WiFi and 3G links



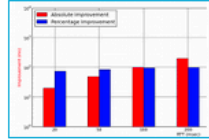
TCP Daytona: Congestion Control with a Misbehaving Receiver



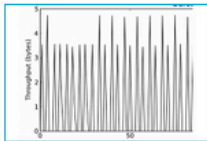
DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



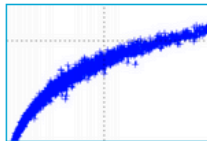
Jellyfish vs. Fat Tree



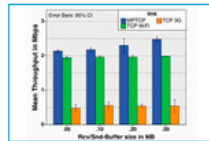
Choosing the Default Initial Congestion Window



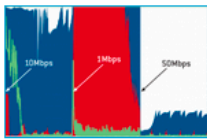
Seeing RED



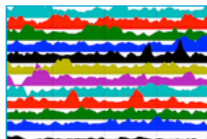
Why Flow-Completion Time is the Right Metric for Congestion Control



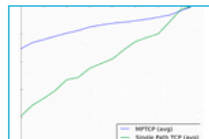
MPTCP Wireless Performance



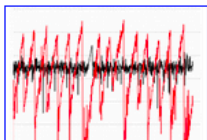
Solving Bufferbloat - The CoDel Way



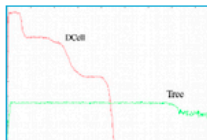
Life's not fair, neither is TCP (... under the following conditions)



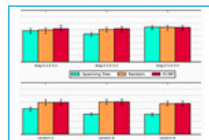
Fairness of Jellyfish vs. Fat-Tree



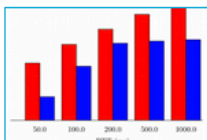
DCTCP and Queues



DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



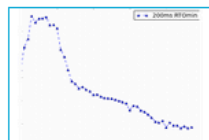
Hedera



Increasing TCP's Initial Congestion Window



HULL: High Bandwidth, Ultra Low Latency



TCP Incast Collapse

CoDel

HULL

MPTCP

Outcast

Jellyfish

DCTCP

Incast

Flow Completion Time

Hedera

DCell

TCP Initial Congestion

Window

Misbehaving TCP Receivers

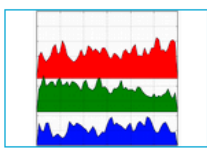
RED

37 students

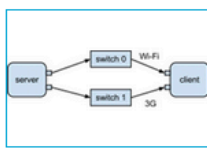
18 projects

16 replicated

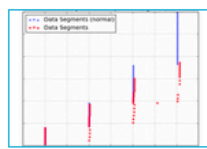




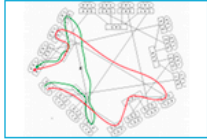
Exploring Outcast



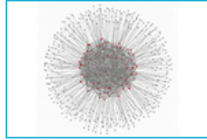
Multipath TCP over WiFi and 3G links



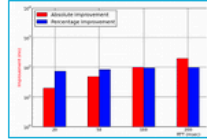
TCP Daytona: Congestion Control with a Misbehaving Receiver



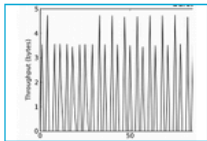
DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



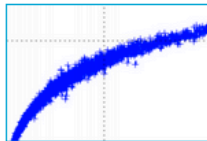
Jellyfish vs. Fat Tree



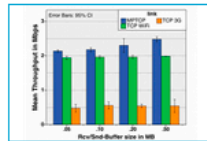
Choosing the Default Initial Congestion Window



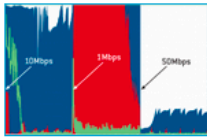
Seeing RED



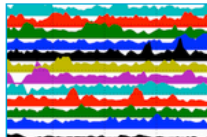
Why Flow-Completion Time is the Right Metric for Congestion Control



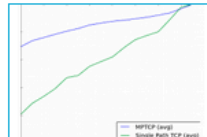
MPTCP Wireless Performance



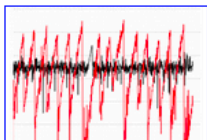
Solving Bufferbloat - The CoDel Way



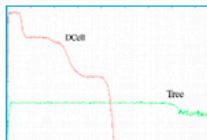
Life's not fair, neither is TCP (... under the following conditions)



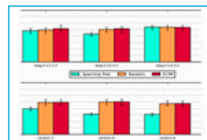
Fairness of Jellyfish vs. Fat-Tree



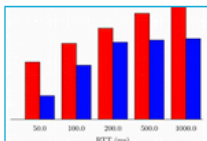
DCTCP and Queues



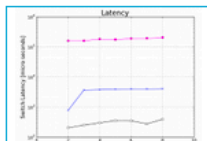
DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



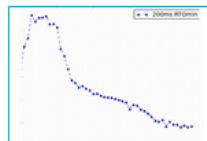
Hedera



Increasing TCP's Initial Congestion Window



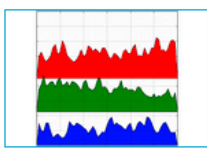
HULL: High Bandwidth, Ultra Low Latency



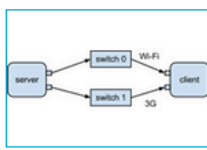
TCP Incast Collapse

CoDel  
HULL  
MPTCP  
Outcast  
Jellyfish  
DCTCP  
Incast  
Flow Completion Time  
Hedera  
DCell  
TCP Initial Congestion Window  
Misbehaving TCP Receivers  
RED

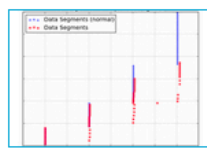
37 students  
18 projects  
16 replicated  
**4 beyond**



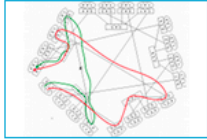
Exploring Outcast



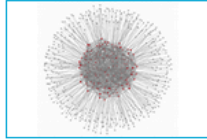
Multipath TCP over WiFi and 3G links



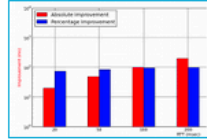
TCP Daytona: Congestion Control with a Misbehaving Receiver



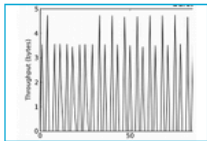
DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



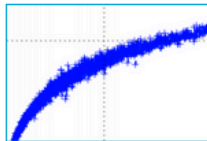
Jellyfish vs. Fat Tree



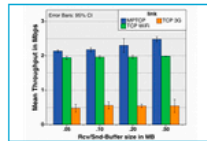
Choosing the Default Initial Congestion Window



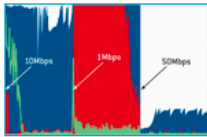
Seeing RED



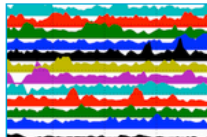
Why Flow-Completion Time is the Right Metric for Congestion Control



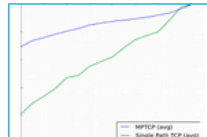
MPTCP Wireless Performance



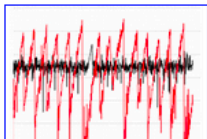
Solving Bufferbloat - The CoDel Way



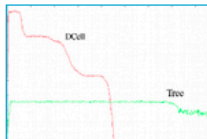
Life's not fair, neither is TCP (... under the following conditions)



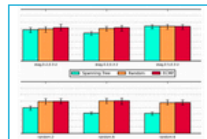
Fairness of Jellyfish vs. Fat-Tree



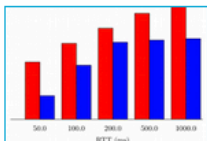
DCTCP and Queues



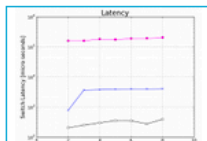
DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



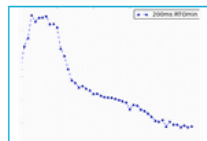
Hedera



Increasing TCP's Initial Congestion Window



HULL: High Bandwidth, Ultra Low Latency



TCP Incast Collapse

CoDel  
HULL  
MPTCP  
Outcast  
Jellyfish  
DCTCP

Incast

Flow Completion Time

Hedera

DCell

TCP Initial Congestion

Window

Misbehaving TCP Receivers

RED

37 students  
18 projects  
16 replicated  
**4 beyond**  
**2 not replicated**

# CoNEXT '12 runnable papers?

## 15/31 seem like candidates:

- MPTCP is not Pareto-optimal: Performance issues and a possible solution
- Architecting for Edge Diversity: Supporting Rich Services over an Unbundled Transport
- Tuning ECN for Data Center Networks
- Datacast: A Scalable and Efficient Reliable Group Data Delivery Service for Data Centers
- PAST: Scalable Ethernet for Data Centers
- Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE
- Towards Agile and Smooth Video Adaption in Dynamic HTTP Streaming
- Application-aware Request Splitting for Interactive Cloud Applications
- Automatic Test Packet Generation
- FindAll: A Local Search Engine for Mobile Phones
- A SOFT Way for OpenFlow Switch Interoperability Testing
- Defending against large-scale crawls in online social networks
- BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection
- Cachet: A Decentralized Architecture for Privacy Preserving Social Networking with Caching
- New Opportunities for Load Balancing in Network-Wide Intrusion Detection Systems

NOT: Wireless, Modeling, Hardware, Social Networking, Security

# Related Work



# Related Work

- vEmulab: scale-out emulation [ATC08]
- DieCast: time dilation [NSDI07]
- SliceTime: synchronized time slices [NSDI11]

All are complementary techniques that could be added to Mininet-HiFi.

None measure event fidelity (S1)

Last two use full-system virtualization (S2)

None evaluate reproducibility at scale (S3)

# Progress Report:

Making runnable  
the network-paper  
default.

# Runnable Paper Existence Proof

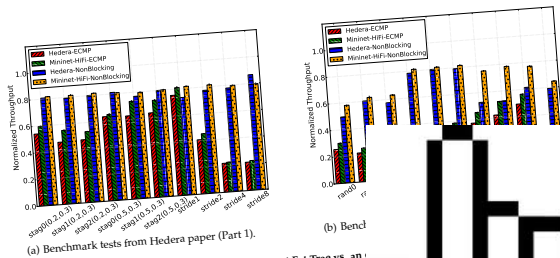


Figure 8: Effective throughput with ECMP routing on a  $k = 4$  Fat Tree vs. an  $8 \times 8$  hardware testbed [13].

values (1, 2, 4 and 8), flows traverse more layers, degrading throughput.

The Mininet-H11 results of the 20 traffic patterns they are hardware tested; in 16 of the 20 traffic patterns they are nearly identical. In the remaining four traffic patterns (randx2,3,4 and strides) the results in the paper have lower throughput because - as the authors explain - the commercial switch in their testbed is built from two switching chips, so the total buffering depends on the traffic pattern. To validate these results, we would need to know the mapping of hosts to switch ports, which is unavailable.

The main takeaway from these experiments is that HiFi reproduces the performance results for this set of data-center networking experiments. It appears possible to collect meaningful results in advance of (or possibly without) setting up a hardware testbed. If a testbed is built, the code and test scripts used in Mininet-HiFi can be reused without change.

Verifying fidelity: Guaranteeing that the system measurement depends on coarse-grained metrics such as aggregate throughput over a period of time. To ensure that no virtual host starved and that the system had enough capacity to sustain the network demand, we measured idle time during the experiment (as described in §3.4). In all runs, the system had at least 35% idle CPU time. The average idle time was 40% at all hosts, indicating that the OS was able to schedule network I/O and packet transmissions without starving other processes. The execution schedule on hardware was also verified by measuring the

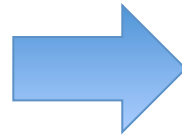
Lessons learned using Mininet. Mininet-based machines were equipped with 1 Gb/s network interfaces. We were unable to use Mininet-HiFi to replicate Hedera's results even with 100 Mb/s network links, as the virtual hosts did not have enough CPU capacity to saturate their network links, while Hedera's results do not qualita-

tively change  
to reproduce  
link/CPU t

### 5.3 Sizi

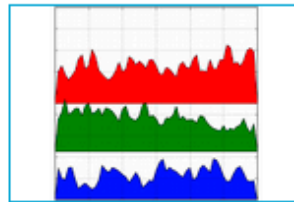
[illegible]

# and, click

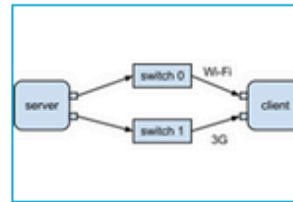
[illegible]

# Reproduced Research Examples

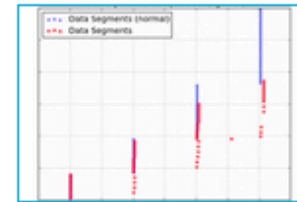
reproducingnetworkresearch.wordpress.com  
(or Google “reproducing network research”)



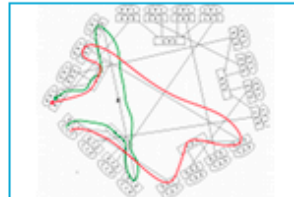
Exploring Outcast



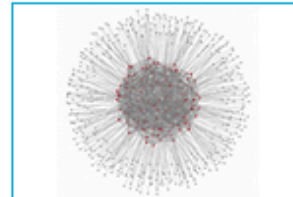
Multipath TCP over WiFi and 3G links



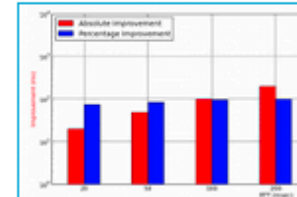
TCP Daytona: Congestion Control with a Misbehaving Receiver



DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



Jellyfish vs. Fat Tree

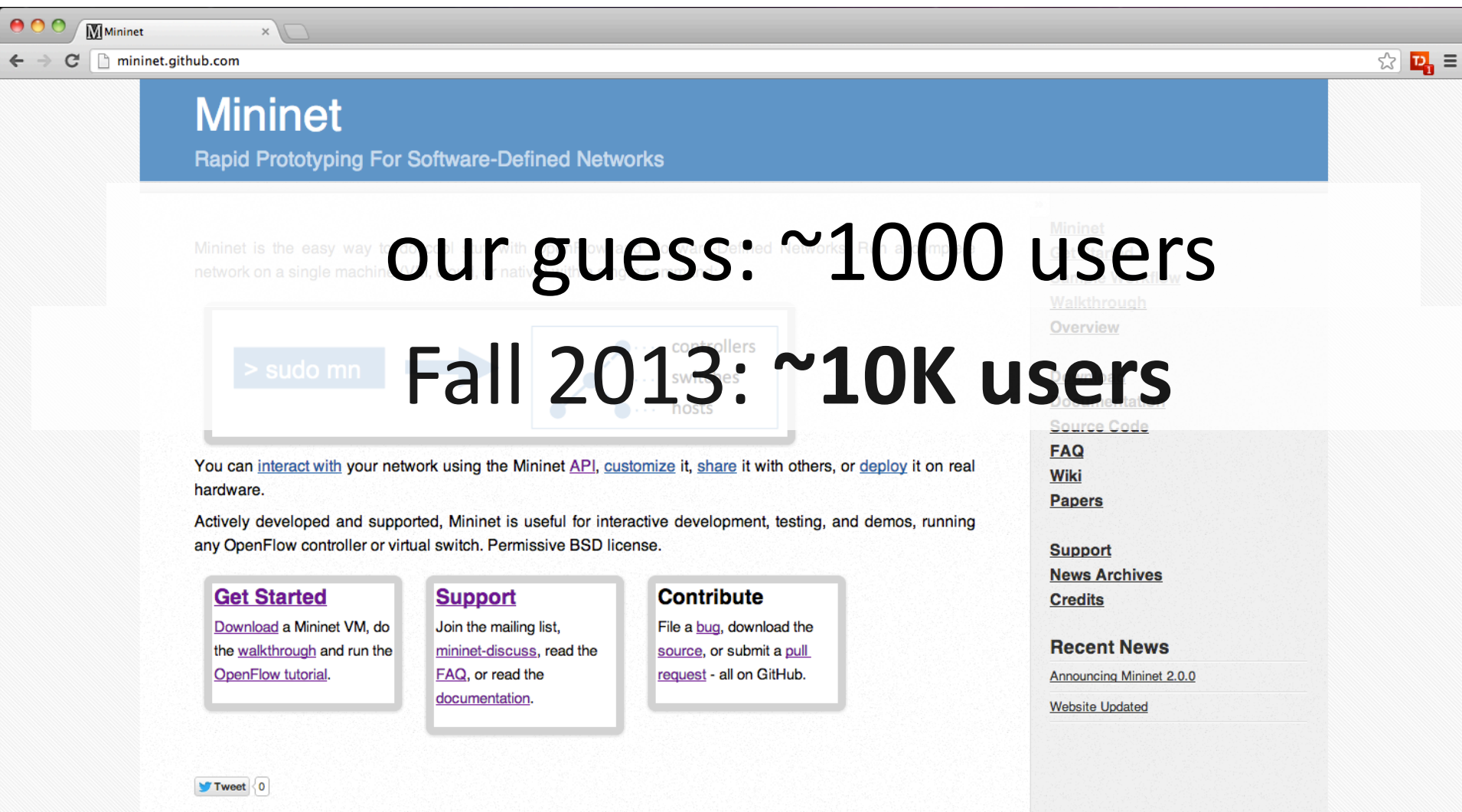


Choosing the Default Initial Congestion Window

## 20 and counting

# Open-Source System w/Active User Community

## mininet.github.com



The image is a screenshot of the mininet.github.com website. The browser window shows the URL 'mininet.github.com'. The website has a blue header with the 'Mininet' logo and the tagline 'Rapid Prototyping For Software-Defined Networks'. The main content area features a large text overlay: 'our guess: ~1000 users' and 'Fall 2013: ~10K users'. Below this, there is a terminal window showing the command '> sudo mn'. The website also includes sections for 'Get Started', 'Support', and 'Contribute', each with links to documentation and resources. A sidebar on the right contains links for 'FAQ', 'Wiki', 'Papers', 'Support', 'News Archives', 'Credits', and 'Recent News'.

Mininet  
Rapid Prototyping For Software-Defined Networks

Mininet is the easy way to build and test software-defined networks on a single machine. It is a native OpenFlow controller and switch, and it can be used to build a wide variety of network topologies.

our guess: ~1000 users  
Fall 2013: ~10K users

> sudo mn

You can [interact with](#) your network using the Mininet [API](#), [customize](#) it, [share](#) it with others, or [deploy](#) it on real hardware.

Actively developed and supported, Mininet is useful for interactive development, testing, and demos, running any OpenFlow controller or virtual switch. Permissive BSD license.

**Get Started**  
[Download](#) a Mininet VM, do the [walkthrough](#) and run the [OpenFlow tutorial](#).

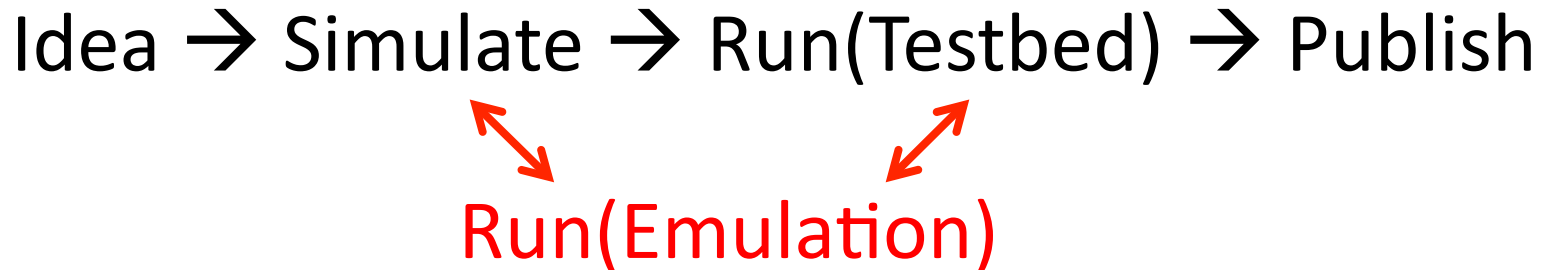
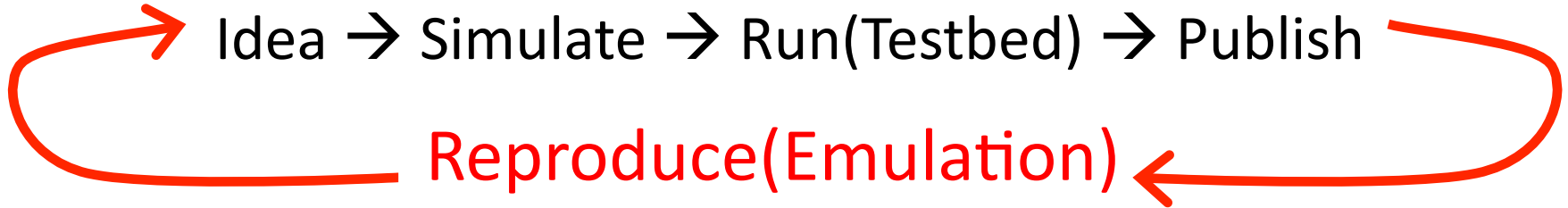
**Support**  
Join the mailing list, [mininet-discuss](#), read the [FAQ](#), or read the [documentation](#).

**Contribute**  
File a [bug](#), download the [source](#), or submit a [pull request](#) - all on GitHub.

[Mininet](#)  
[Walkthrough](#)  
[Overview](#)  
[Source Code](#)  
[FAQ](#)  
[Wiki](#)  
[Papers](#)  
  
[Support](#)  
[News Archives](#)  
[Credits](#)  
  
[Recent News](#)  
[Announcing Mininet 2.0.0](#)  
[Website Updated](#)

[Tweet](#) 0

# New Workflows



Idea → Run(Emulation) → Publish

# (aside) Why God doesn't have a Ph.D.

- 1) He had only one major publication.
- 2) It was in Hebrew.
- 3) It had no references.
- 4) It wasn't published in a refereed journal.
- 5) Some even doubt he wrote it by himself.
- 6) It may be true that he created the world, but what has he done since then?
- 7) His cooperative efforts have been quite limited.
- 8) The scientific community has had a hard time replicating his results.**
- 9) He never applied to the ethics board for permission to use human subjects.
- 10) When one experiment went awry he tried to cover it up by drowning his subjects.
- 11) When subjects didn't behave as predicted, he deleted them from the sample.
- 12) Some say he had his son teach the class.
- 13) He expelled his first two students for learning.
- 14) He rarely came to class, and he just told students to read the book.
- 15) Although there were only 10 requirements, most of his students failed his tests.
- 16) His office hours were infrequent and usually held on a mountaintop.

# Look for the shirt. Questions?



[mininet.github.com](https://mininet.github.com)

[reproducingnetworkresearch.wordpress.com](https://reproducingnetworkresearch.wordpress.com)



# Backup Slides

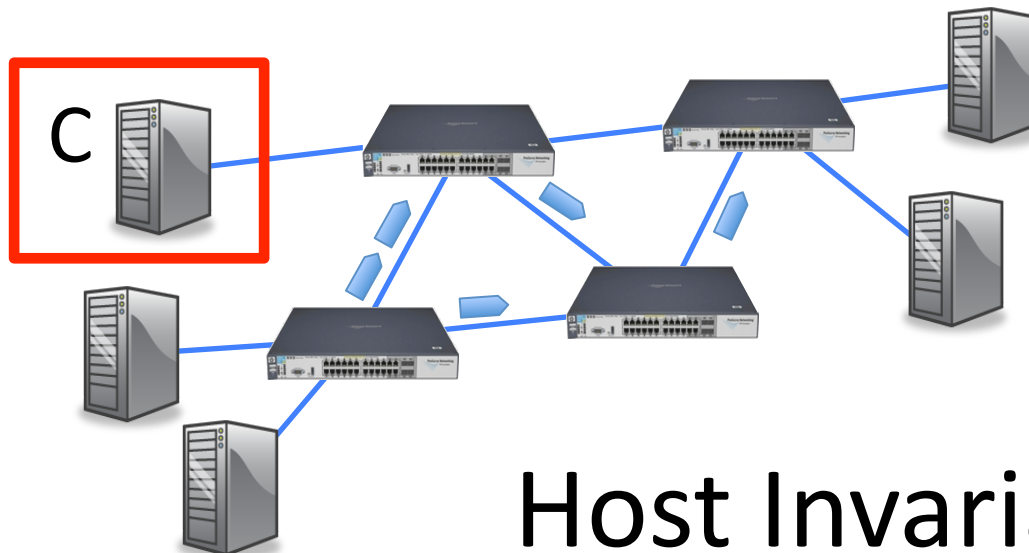
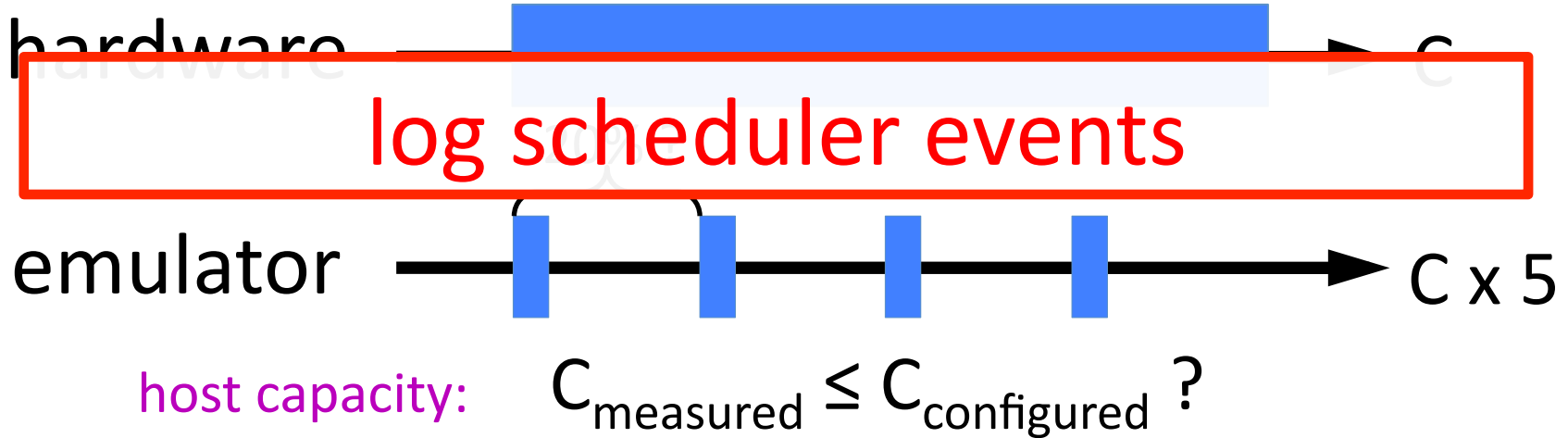
# Doesn't a simulator guarantee these invariants?

- Yes, exactly! A good one will.
- We're trying to get the network fidelity of an emulator to match a simulator with virtual time.

# What about an RTOS?

- Every process must be bounded-time.
- Requires kernel mods.
- Conservative provisioning make the resulting system too resource-limited to be useful.
- May needlessly limit resources when they could be used.

host with capacity C



Host Invariants

host with capacity  $C$

