Network Emulation

Corso di Tecnologie di Infrastrutture di Reti

Martin Klapez

Department of Engineering *Enzo Ferrari*
University of Modena and Reggio Emilia

Modena, 23 May 2019

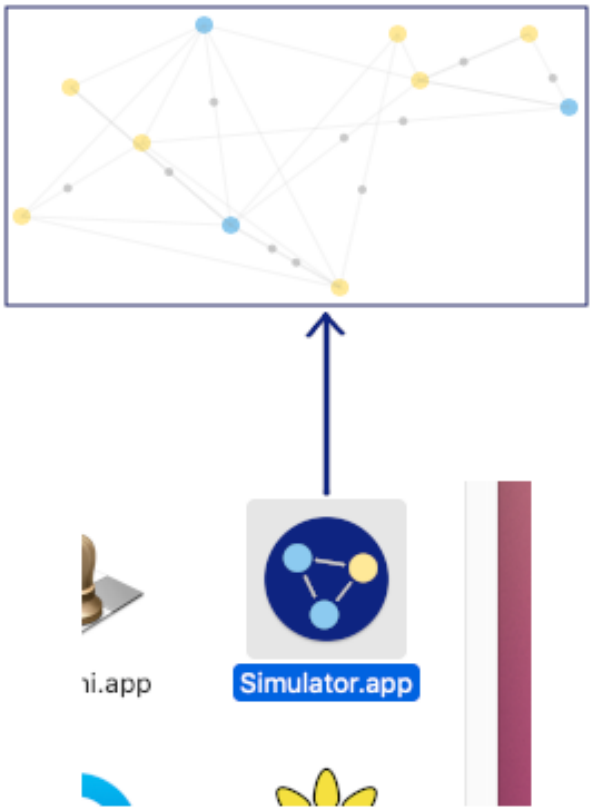
Overview

- Network Simulation vs The Real Thing vs Network Emulation
- Introduction to Mininet
- Experience Bufferbloat with Mininet
- Create an Emulated network with Mininet

Networks: Simulation vs The Real Thing vs Emulation

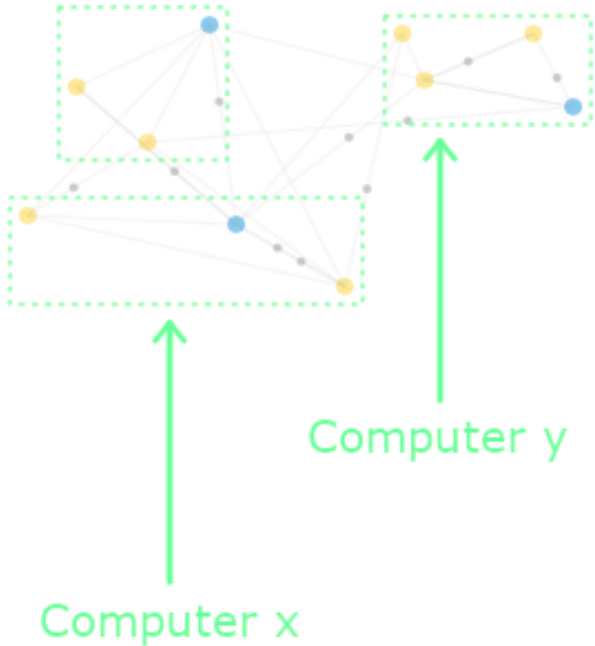
A graphical representation

Simulation



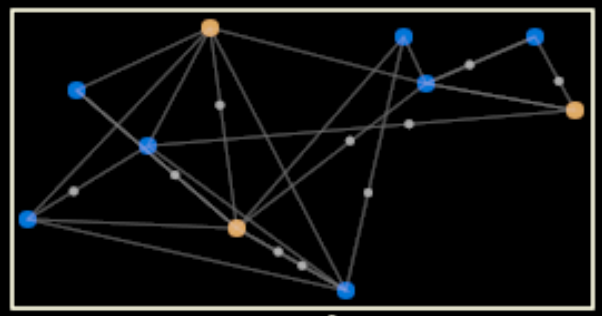
'Real'

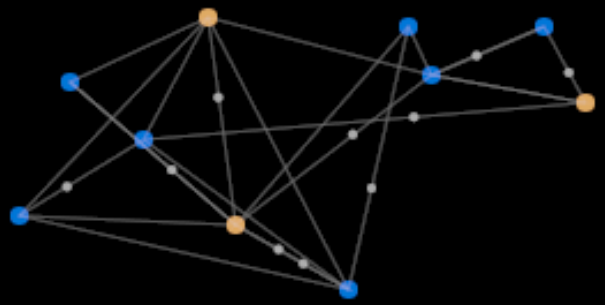
Emulation

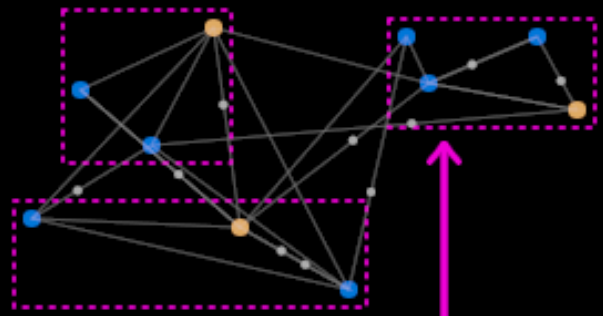


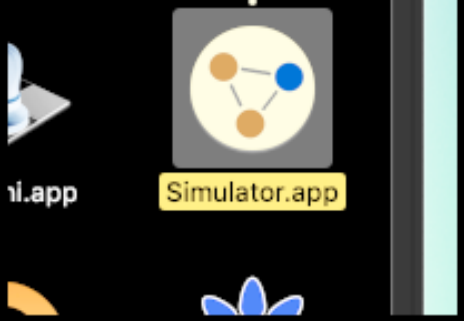
Networks: Simulation vs The Real Thing vs Emulation

A graphical representation









Computer y

Computer x

Networks: Simulation vs Emulation vs The Real Thing

A comparison

Network *Emulation*: why 👍?

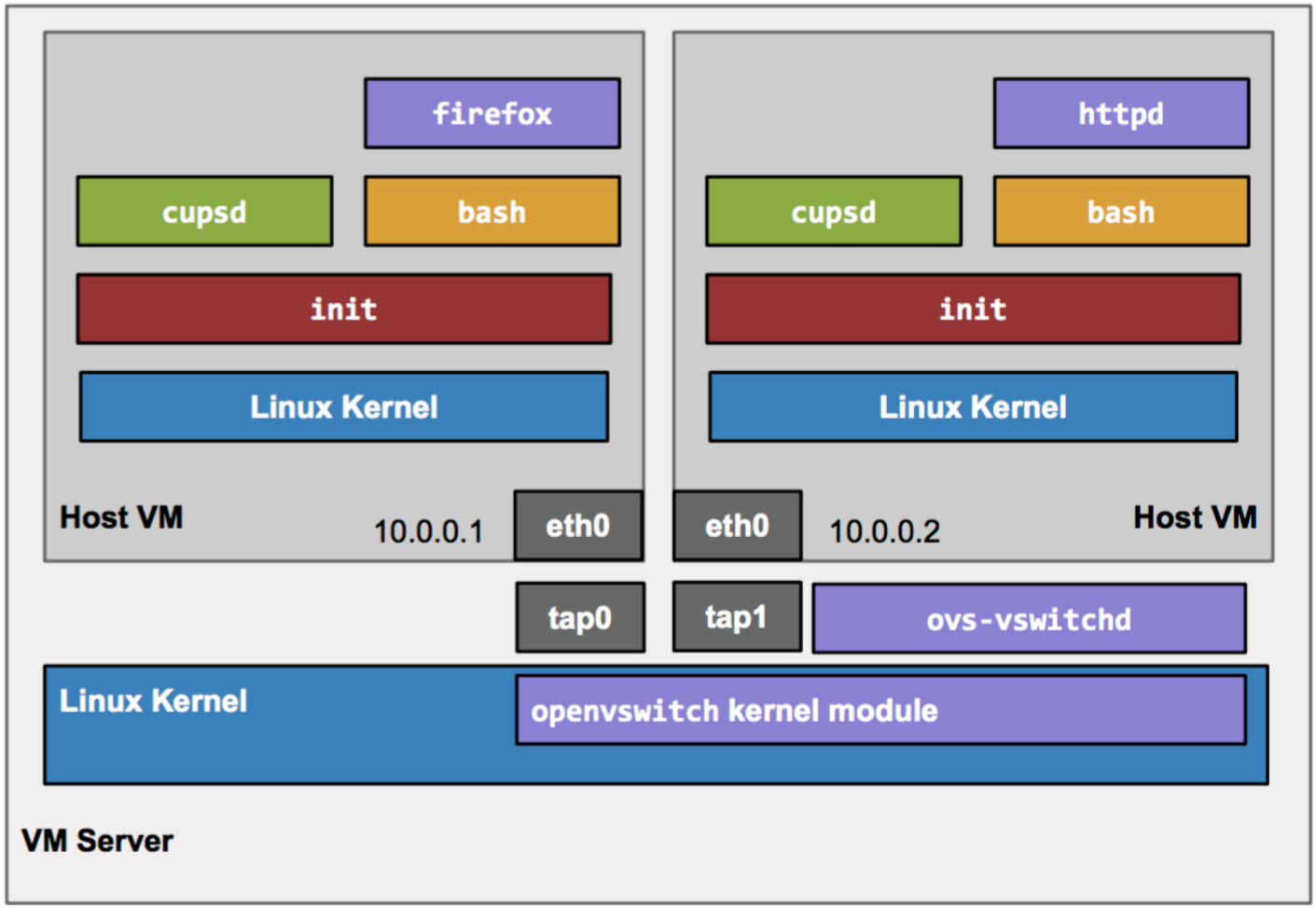
- Realism: the code created in the emulator is **the same code** that would be deployed in the real network
- Interactive
- Reasonably accurate
- Flexible
- Scalable
- Inexpensive

Network *Emulation*: why 👎?

- Slower than hardware
- Possible inaccuracies due to multiplexing of hardware resources

Mininet Introduction

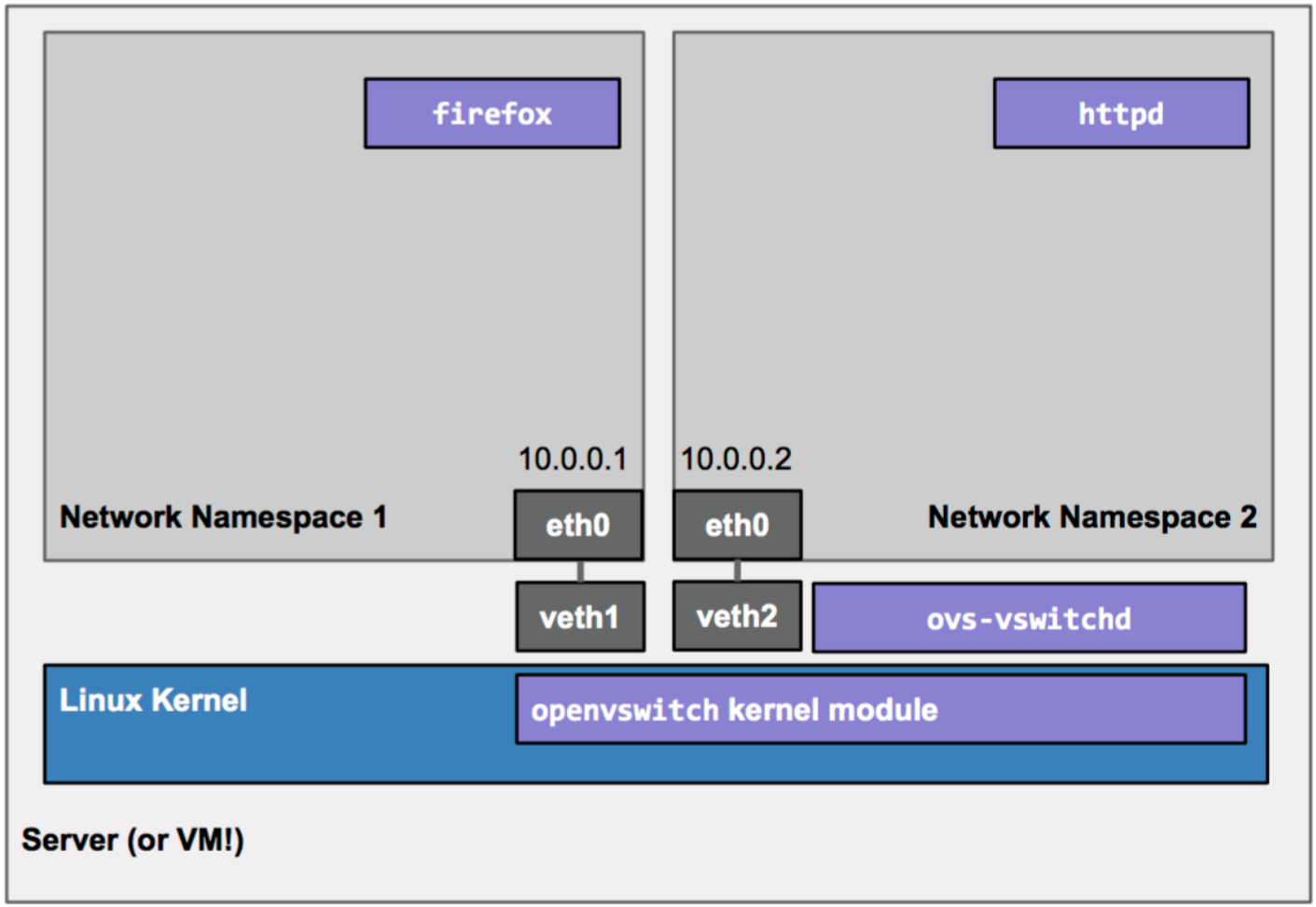
Network Emulator Architecture: Full System Virtualization



OK, but VMs are heavyweight: the memory overhead for each VM limits the scale to just a handful of switches and hosts.

Mininet Introduction

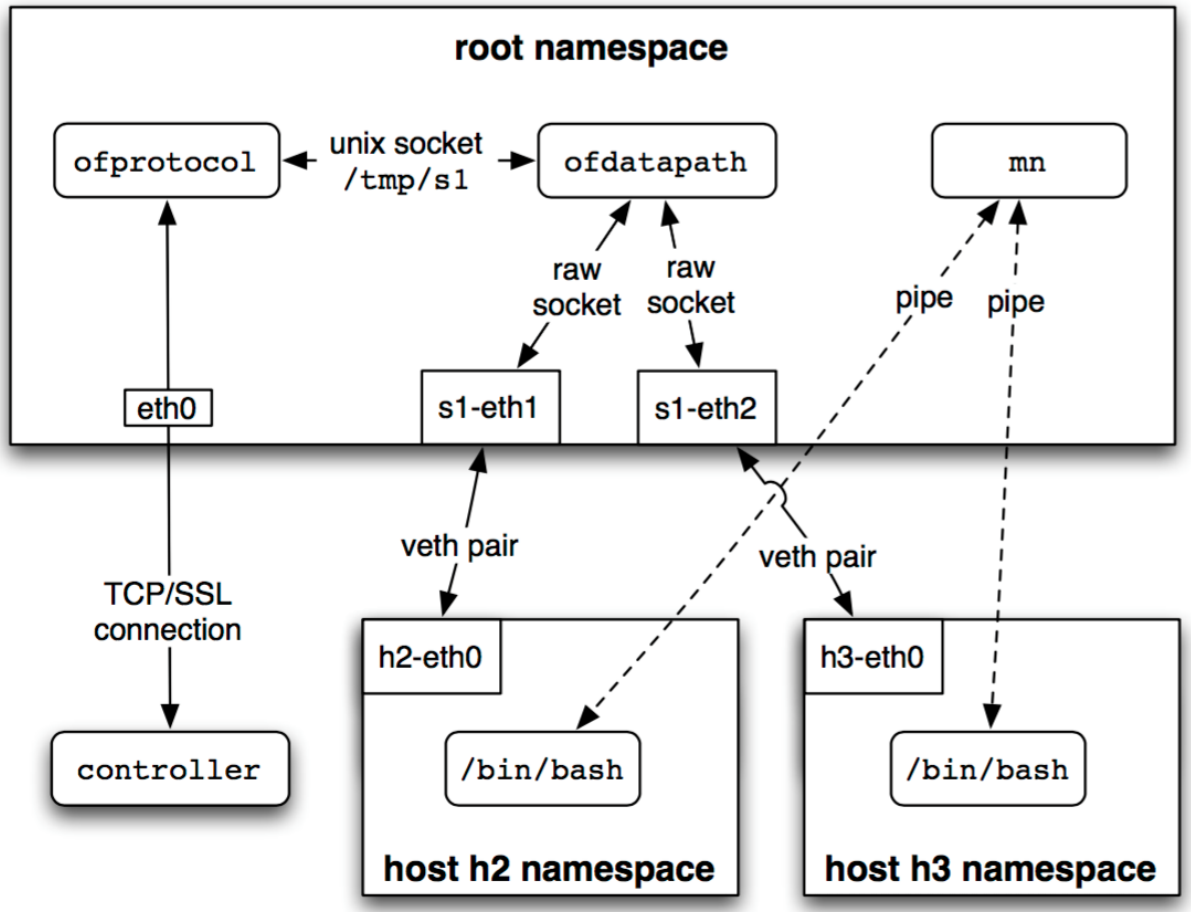
Network Emulator Architecture: Lightweight Virtualization (Mininet)



Mininet leverages Linux features (processes and virtual Ethernet pairs in network namespaces).

Mininet Introduction

Network Emulator Architecture: Namespaces (Mininet)



Mininet creates a virtual network by placing host processes in network namespaces and connecting them with virtual Ethernet (veth) pairs. In this example, they connect to a user-space OpenFlow switch.

- **Hosts**. *Network namespaces are containers for network state*. They provide processes (and groups of processes) with exclusive ownership of interfaces, ports, and routing tables (such as ARP and IP).
- **Links**. A virtual Ethernet pair, or veth pair, acts *like a wire* connecting two virtual interfaces; *packets sent through one interface are delivered to the other*, and each interface appears as a fully functional Ethernet port to all system and application software.
- **Switches**. The same packet delivery semantics that would be provided by a hardware switch.
- **Controllers**. An SDN controller could run inside a VM, natively on a host machine, or in the cloud.

- **Run commands on hosts**.
- Verify switch operations.
- Induce failures.
- Adjust link connectivity.
- Change the state of the network.
- Modify OpenFlow tables.
- ...

Lack of performance fidelity, *especially at high loads*.

CPU resources are multiplexed in real-time by the default Linux scheduler, which provides no guarantee that a host that is ready to send a packet will be scheduled promptly, or that all switches will forward at the same rate.

In addition, software forwarding may not match hardware. O(n) linear lookup for software tables cannot approach the O(1) lookup of a hardware-accelerated TCAM in a vendor switch, causing the packet forwarding rate to drop for large wildcard table sizes.

- Bob Lantz, Brandon Heller, Nick McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks", ACM, Hotnets '10, October 20-21, 2010, Monterey, CA, USA.
- Te-Yuan Huang, Vimalkumar Jeyakumar Bob Lantz, Brian O'Connor, Nick Feamster, Keith Winstein, Anirudh Sivar, "Teaching Computer Networking with Mininet", ACM, SIGCOMM 2014 Tutorial, August 18, 2014, Chicago, IL, USA.

Experience Bufferbloat with Mininet

Prepare the ground

Turn on the provided virtual machine (Mininet-VM).
user: mininet
pass: mininet

Ensure you have Internet access from the VM:

```
> ping 8.8.8.8
```

If you already have the folder below, delete it:

```
> sudo rm -r cs144_bufferbloat/
```

Configure your keyboard layout:

```
> sudo dpkg-reconfigure keyboard-configuration
```

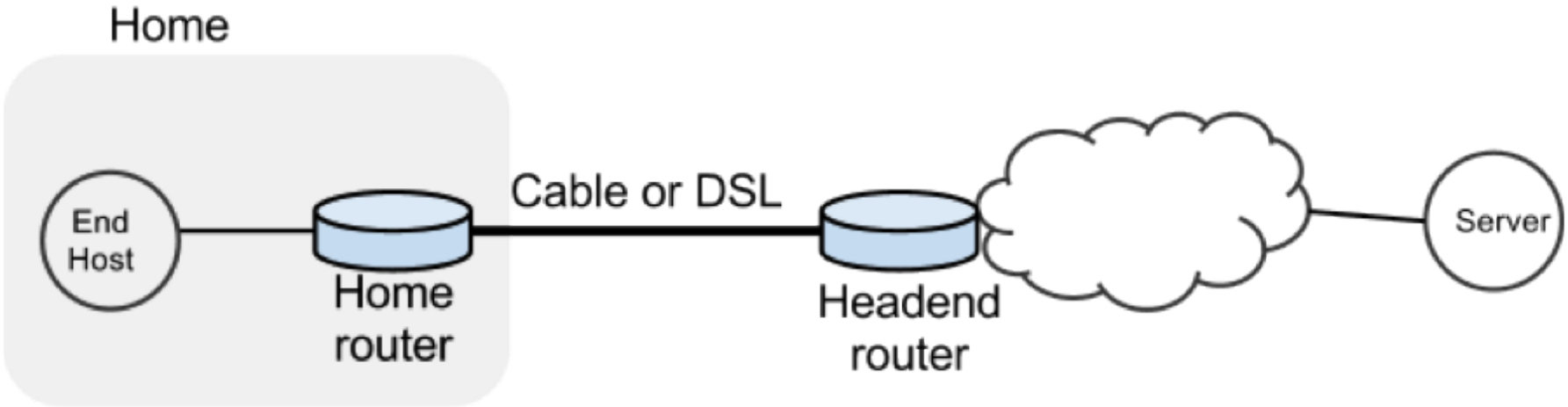
Experience Bufferbloat with Mininet

Introduction to the problem

Big buffers:  or  ?

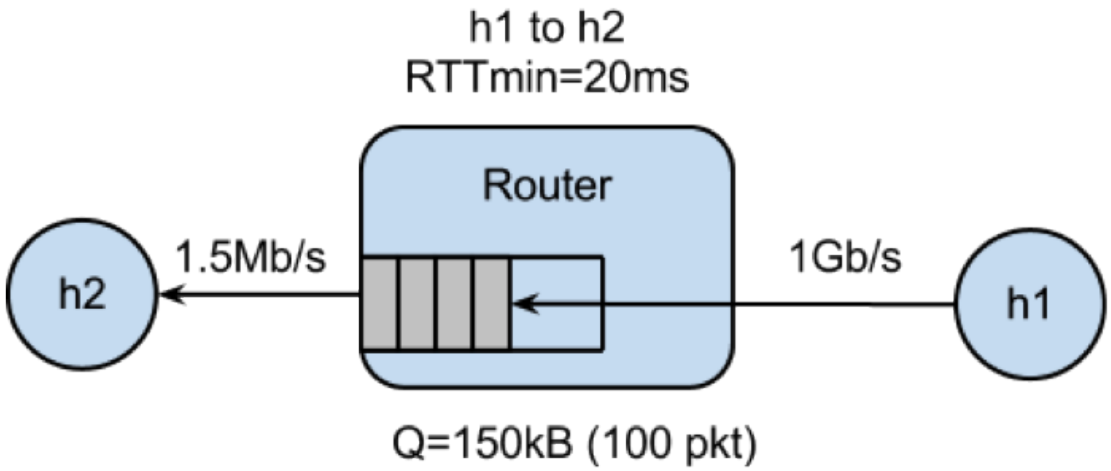
Big buffers deployed in the Internet may counterintuitively cause higher latencies, higher jitter, and lower throughput.

You lose less packets, but some of them may fall behind others for a very long time before being dispatched.



Experience Bufferbloat with Mininet

Get the topology



Get and run the Mininet network that emulates the topology:

```
> git clone https://bitbucket.org/huangty/cs144_bufferbloat.git
> cd cs144_bufferbloat/
> sudo ./run.sh
```

Experience Bufferbloat with Mininet

Commands

Measure the delay between the two hosts:

```
mininet > h1 ping -c10 h2
```

Measure how long it takes to download a web page from h1:

```
mininet > h2 wget http://10.0.0.1
```

Simulate a video streaming flow:

```
mininet > h1 ./iperf.sh
```

Experience Bufferbloat with Mininet

Video flow and Short flow: router buffer of 100 packets

1. Start a simulation of a video streaming flow using the provided iPerf script
2. While iPerf is running, see how the long-lived iPerf flow affects the web page download
3. Observe how the delay between the hosts evolve over time

Why it happens what it happens?

cwnd

buffers

Experience Bufferbloat with Mininet

Video flow and Short flow: router buffer of 100 packets

1. Start a simulation of a video streaming flow using the provided iPerf script
2. While iPerf is running, see how the long-lived iPerf flow affects the web page download
3. Observe how the delay between the hosts evolve over time

Why it happens what it happens?

cwnd

buffers

Experience Bufferbloat with Mininet

Video flow and Short flow: router buffer of 100 packets

1. Start a simulation of a video streaming flow using the provided iPerf script
2. While iPerf is running, see how the long-lived iPerf flow affects the web page download
3. Observe how the delay between the hosts evolve over time

Why it happens what it happens?

cwnd

buffers

Experience Bufferbloat with Mininet

Video flow and Short flow: router buffer of 20 packets

Restart Mininet with the smaller buffer:

```
mininet > exit
> sudo ./run-minq.sh
```

1. Start a simulation of a video streaming flow using the provided iPerf script
2. While iPerf is running, see how the long-lived iPerf flow affects the web page download
3. Observe how the delay between the hosts evolve over time

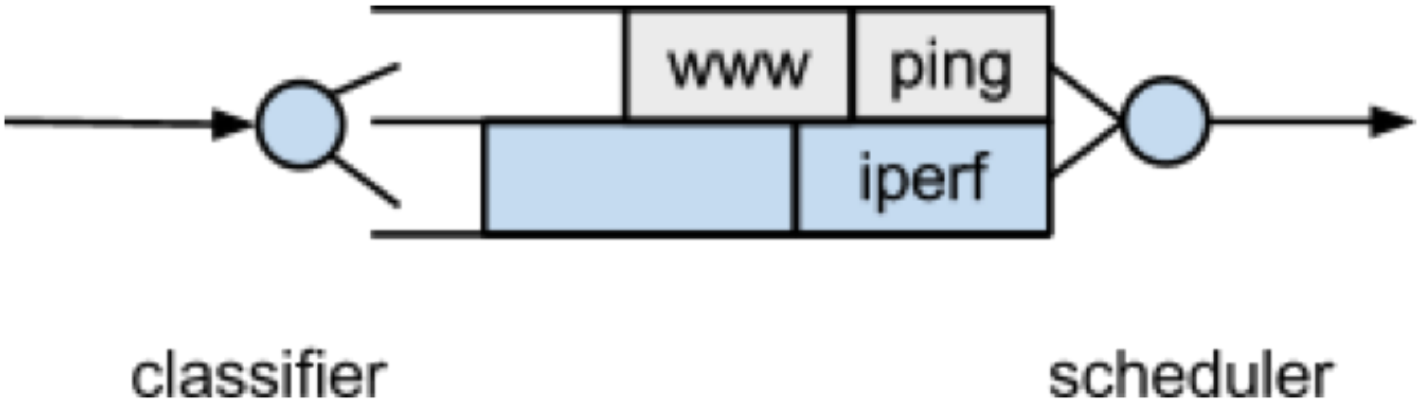
Worse? Better? Why?

Experience Bufferbloat with Mininet

Video flow and Short flow: different queues

The problem seems to be that packets from the short flow are stuck behind a lot of packets from the long flow.

What if we maintain a separate queue for each flow and then put iPerf and wget traffic into different queues?



In the experiment, the scheduler implements fair queueing so that when both queues are busy, each flow receives half of the bottleneck link rate.

Experience Bufferbloat with Mininet

Video flow and Short flow: different queues

Restart Mininet with the two queues on the Headend router:

```
mininet > exit
> sudo ./run-diff.sh
```

1. Start a simulation of a video streaming flow using the provided iPerf script
2. While iPerf is running, see how the long-lived iPerf flow affects the web page download
3. Observe how the delay between the hosts evolve over time

Worse? Better? Why?

Experience Bufferbloat with Mininet

Sources

- Stanford CS144, "An Introduction to Computer Networks, Bufferbloat Exercise", https://github.com/mininet/mininet/wiki/Bufferbloat.

Create Emulated Networks with Mininet

Assignment 1: Let's draw!

Your assignment is to draw the network topologies that Mininet provides as templates, exploring the networks by starting from a single command:

```
mininet > help
```

Good work!

1. > sudo mn --topo single
2. > sudo mn --topo linear
3. > sudo mn --topo linear,3
4. > sudo mn --topo tree
5. > sudo mn --topo tree,2
6. > sudo mn --topo tree,depth=2,fanout=3

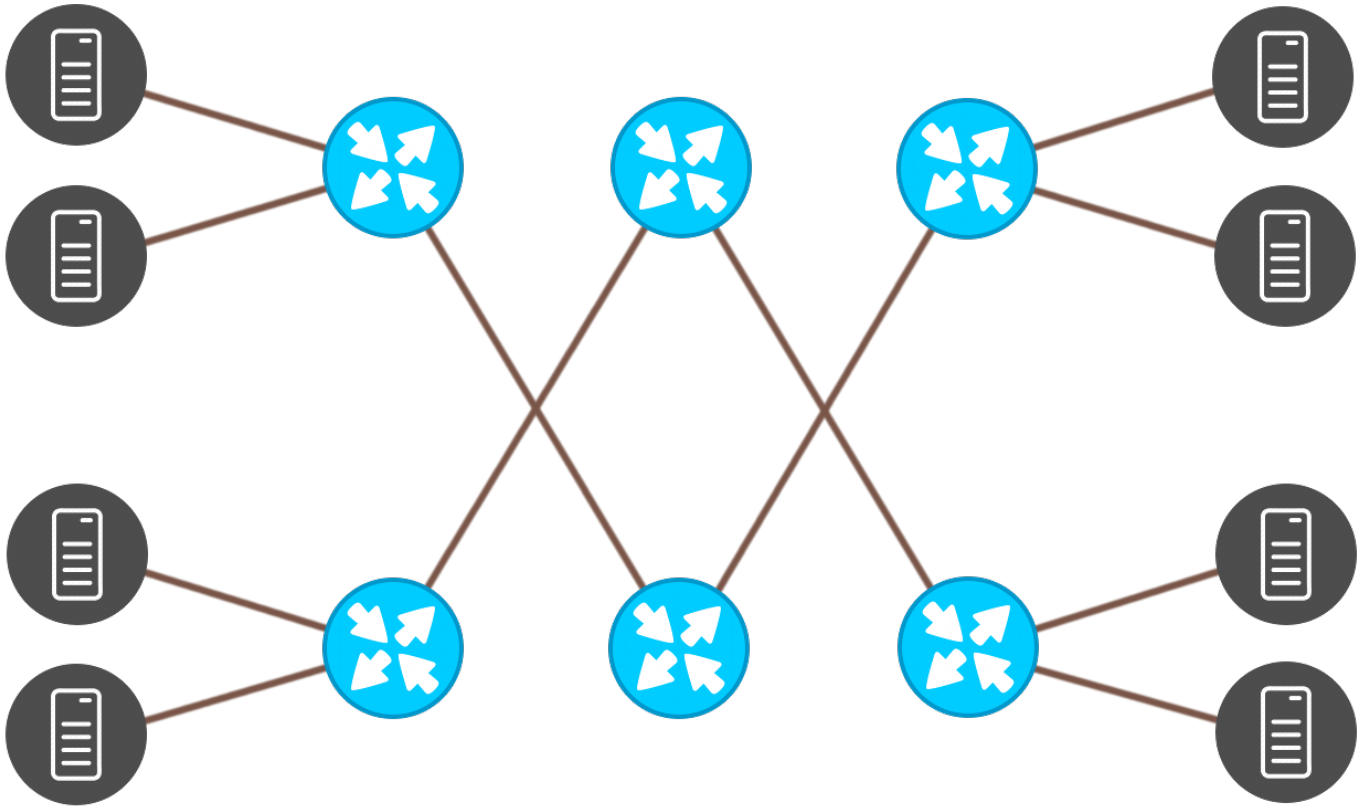
Your next assignment follows the inverse logic.

You should create with Mininet the custom topology that follows, getting inspiration from the example Mininet scripts, which you can find at:

- https://github.com/mininet/mininet/tree/master/examples
- In the provided virtual machine in the directory ~/mininet/mininet/examples

Create Emulated Networks with Mininet

Assignment 2: A custom topology



Good work!

Become familiar with basic OpenFlow concepts

Assignment 3: A custom SDN controller

In this assignment, you will modify an existing SDN controller.

— It is assumed that you already have an understanding of SDN —

Become familiar with basic OpenFlow concepts

Assignment 3: A custom SDN controller

SDN in a nutshell:

- One time, network hardware did both the decision-making and the actual work.
- It was not programmable or, if it was, you had to get your hands dirty by tinkering with vendor-specific APIs, directly on the hardware itself.
- So, a switch had to calculate where to forward incoming packets, and then had to actually send them. Direction & Execution.

Become familiar with basic OpenFlow concepts

Assignment 3: A custom SDN controller

SDN in a nutshell:

- Someone got tired by this lack of flexibility and saw the potential of making this differently.
- Abstractions are vastly applied in software. Why networking lacks them?
- The idea got executed through SDN (and NFV).
- SDN: "Now" (not really, the majority of the deployed network hardware still operates in the old way), decision-making is performed by software, wrote with modern programming languages, that run on general-purpose hardware (Control Plane), and execution is carried on by "dumb" but fast specialized hardware that just follow the rules sent to them (Data Plane).

L2 Hub.

A L2 Hub has the job of forwarding a packet to the destination.

It works this way:

- It receives a packet from a port x.
- It floods the packet on all the *other* ports.

Inefficient but simple.

L2 Learning Switch.

Same job, but it works this way:

- It has a table to keep track of [*mac&* : *port*] associations.
- It receives a packet from a port *x*.
- It puts in the table [*mac&* : *port*].
- It looks in the headers of the packet for the destination *mac&*.
- If it finds the destination *mac&* in the table, it forwards the packet to the corresponding port.
- Otherwise, it floods the packet as a hub, but as soon receives a packet from the destination *mac&* (a TCP ACK for instance), it tracks its port in the table.

Become familiar with basic OpenFlow concepts

Assignment 3: A custom SDN controller

Clone the POX repo.

```
> git clone https://github.com/noxrepo/pox.git
```

Start the default POX L2 Learning Switch.

```
> cd pox
> ./pox.py –unthreaded-sh forwarding.l2_learning &
```

Start Mininet with a basic topology (2 nodes, 1 switch) and connect the POX controller.

```
> sudo mn –controller remote
```

Become familiar with basic OpenFlow concepts

Assignment 3: A custom SDN controller

Your job is now to modify the default POX L2 Learning Switch.
Instead of installing rules on the switch, the controller has to check every packet that passes through and has to explicitly tell the switch to send it to the right port, every time.

Relevant documentation:

https://noxrepo.github.io/pox-doc/html/#openflow-messages

Useful commands:

```
mininet > h1 ping h2
```

```
> sudo pkill python (to shutdown the active POX controller)
```