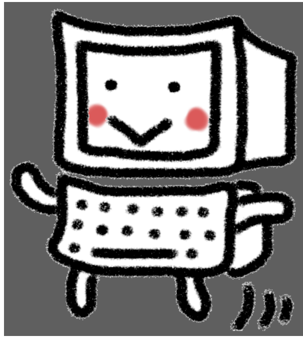


Lab 1

Layer 2 Forwarding &
MAC Learning

Date:	**2021/02/23**
Deadline:	2021/03/02 (Mon) 23:59 (Extended 2021/03/09)

Outline

- Objective
- Experiment Environment
- Mininet
- Packet analysis tools
- Lab requirements

Objective

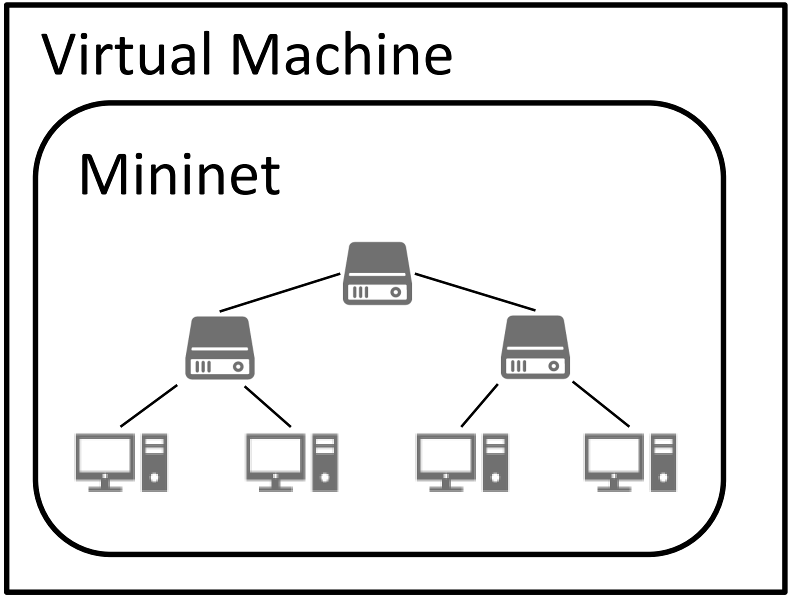
- Network Emulation Environment Setup
- Familiar with packet analysis tools
- Layer 2 Concept Recap
  - Packet Forwarding and MAC Learning

Experiment Environment

- VirtualBox: Open source cross-platform virtualization application (by Oracle)
  - A general-purpose full virtualizer for x86 hardware
- Ubuntu 18.04: Open source operating system
- Mininet: a *network emulator*
  - A virtual test bed and development environment for SDN
  - Can easily creates a network of virtual hosts, switches, controllers, and links
- Packet Analysis Tools: Wireshark, tcpdump

Experimental Environment

- A Virtual Machine with Ubuntu desktop 18.04 LTS
- Hardware requirements
  - 2 Cores (or 2 CPUs)
  - 4G RAM
  - 20G HDD
- Installation Guide:
  - **Environment_Setup.pdf**



Outline

- Objective
- Experiment Environment
- Mininet
  - Overview and installation
  - Basic Usage
- Network tools
- Lab requirements

Mininet Overview and Installation

- Mininet: Emulate a network on your computer
  - Provide Python API for building custom network
    - Nodes: switch, host and router
    - Links: edge between two nodes
  - Provide simple built-in topology with CLI commands
- Installation:

```
bash$ sudo apt install mininet
```

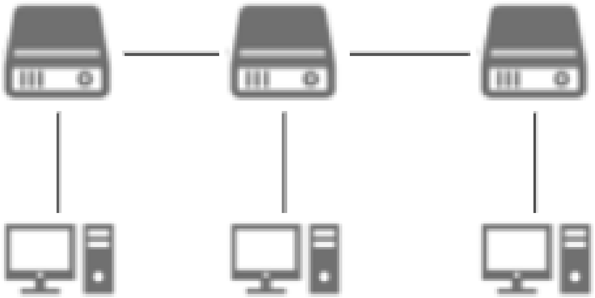
  - sudo: execute command as root permission
  - apt: advanced package tool to manage applications

Outline

- Objective
- Experiment Environment
- Mininet
  - Overview and installation
  - Basic Usage
    - Method 1: Built-in Topology
    - Method 2: Custom Topology
- Packet analysis tools
- Lab requirements

Method 1: Built-in Topology (1/2)

- Five Built-in topologies:
  - Minimal
  - Single
  - Linear
  - Torus
  - Tree

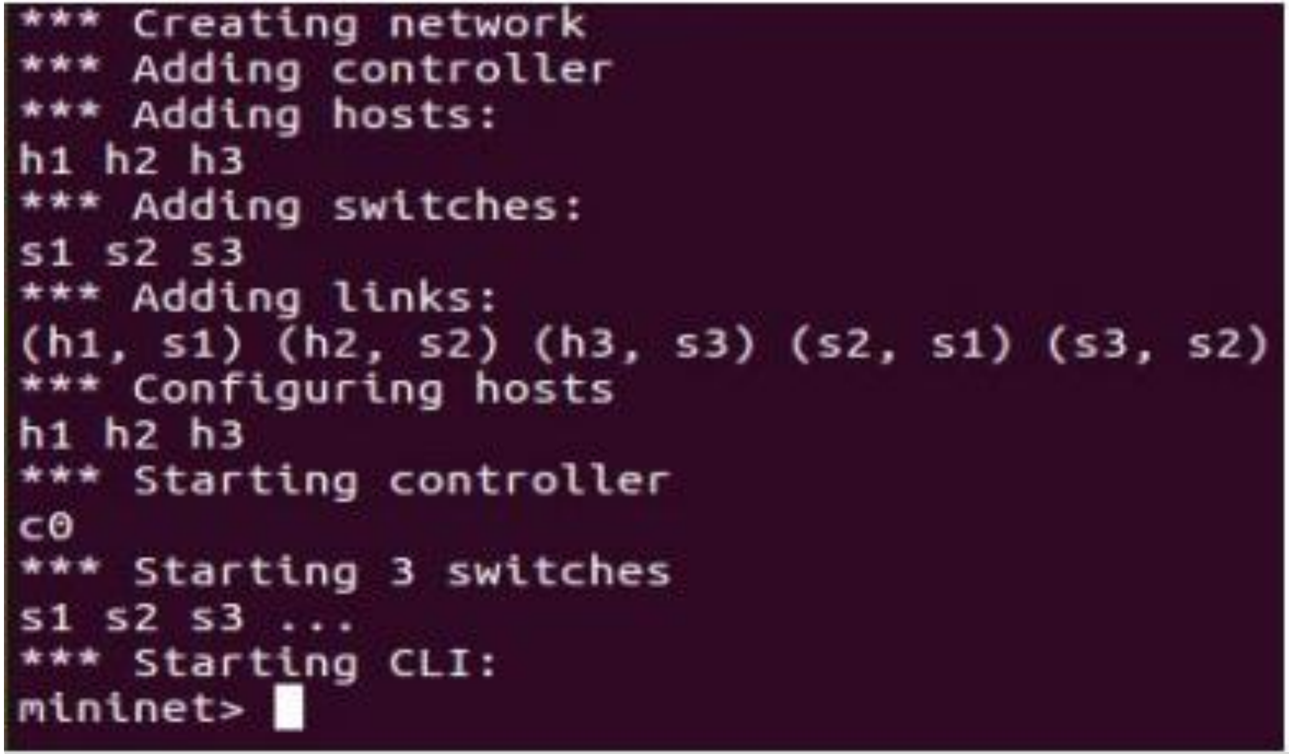


- Example: Create a linear topology with 3 switch

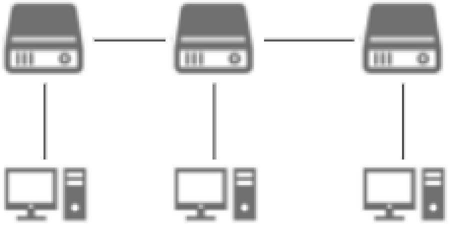
```
bash$ sudo mn --topo=linear,3
```

  - "--topo" specifies the topology
  - Each switch has a host

Method 1: Built-in Topology (2/2)



```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```



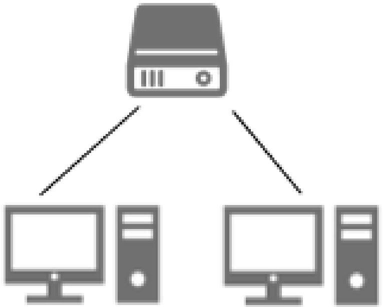
```
mininet> exit (exit mininet CLI)
```

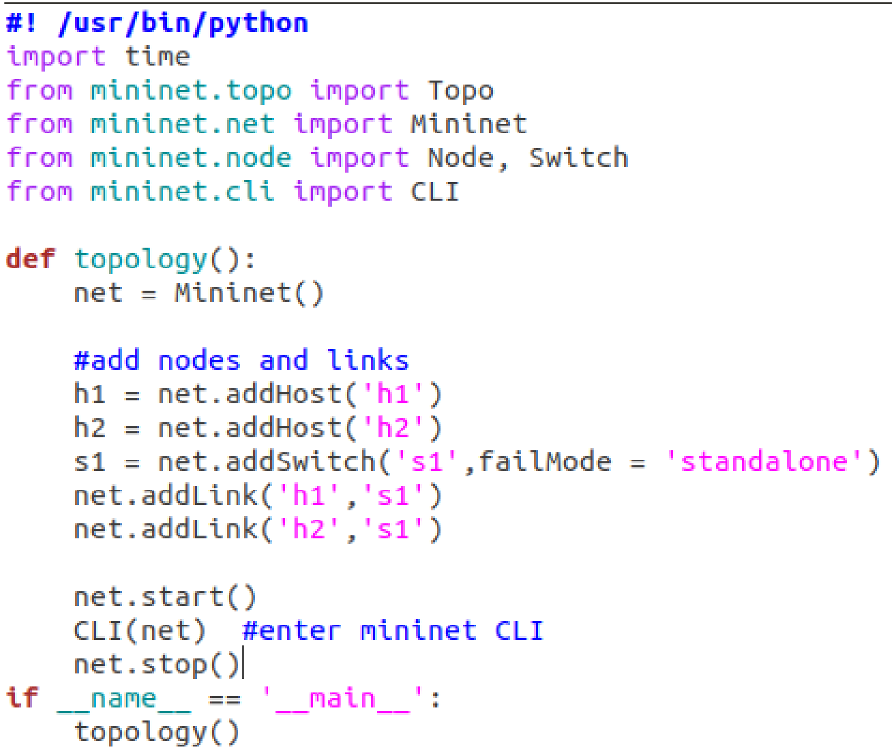
Outline

- Objective
- Experiment Environment
- Mininet
  - Overview and installation
  - Basic Usage
    - Method 1: Built-in Topology
    - Method 2: Custom Topology
- Network tools
- Lab requirements

Method 2: Custom Topology

- Write a Python script
  - Example: Tree of Depth 1





```
#! /usr/bin/python
import time
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node, Switch
from mininet.cli import CLI

def topology():
    net = Mininet()

    #add nodes and links
    h1 = net.addHost('h1')
    h2 = net.addHost('h2')
    s1 = net.addSwitch('s1',failMode = 'standalone')
    net.addLink('h1','s1')
    net.addLink('h2','s1')

    net.start()
    CLI(net)  #enter mininet CLI
    net.stop()
if __name__ == '__main__':
    topology()
```

- Execute your python script

```
bash$ sudo python <Python_Script>.py
```

Mininet Basic commands (1/2)

- Show all nodes (Hosts and Network Devices)

```
mininet> nodes
```

- Show all links between nodes

```
mininet> links
```

- Test the reachability of a pair of hosts (e.g. h1 and h2)

```
mininet> h1 ping h2
```

- Do an all-pairs “ping”

```
mininet> pingall
```

Mininet Basic commands (2/2)

- Start an xterm CLI panel of a node (e.g., h1 panel)

```
mininet> h1 xterm &
```

- Run command on a node

```
mininet> <node name> [command]
```

- Exit mininet

```
mininet> exit
```

- Always clear network topology before you start

```
bash$ sudo mn -c
```

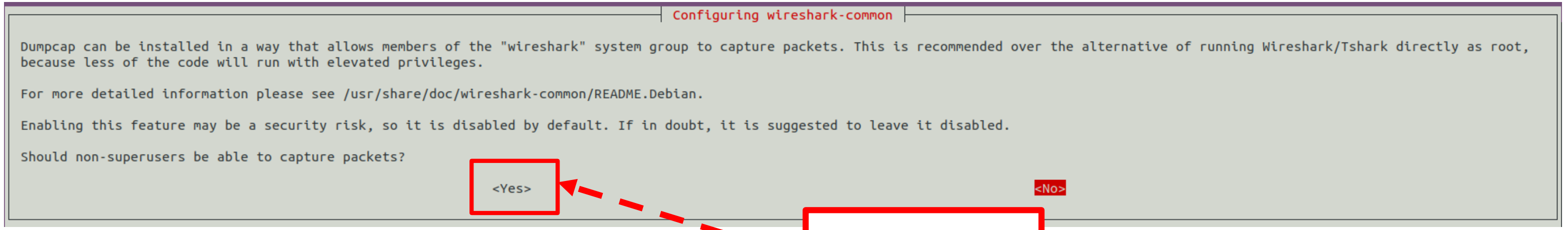
Outline

- Objective
- Experiment Environment
- Mininet
- Packet analysis tools
  - Wireshark
  - Tcpdump
- Lab requirements

Wireshark (1/6)

- A free and open-source GUI packet analyzer
  - Installation:

```
bash$ sudo apt install wireshark
```



```
Configuring wireshark-common

Dumpcap can be installed in a way that allows members of the "wireshark" system group to capture packets. This is recommended over the alternative of running Wireshark/Tshark directly as root,
because less of the code will run with elevated privileges.

For more detailed information please see /usr/share/doc/wireshark-common/README.Debian.

Enabling this feature may be a security risk, so it is disabled by default. If in doubt, it is suggested to leave it disabled.

Should non-superusers be able to capture packets?

                         <Yes>                                                  <No>
```

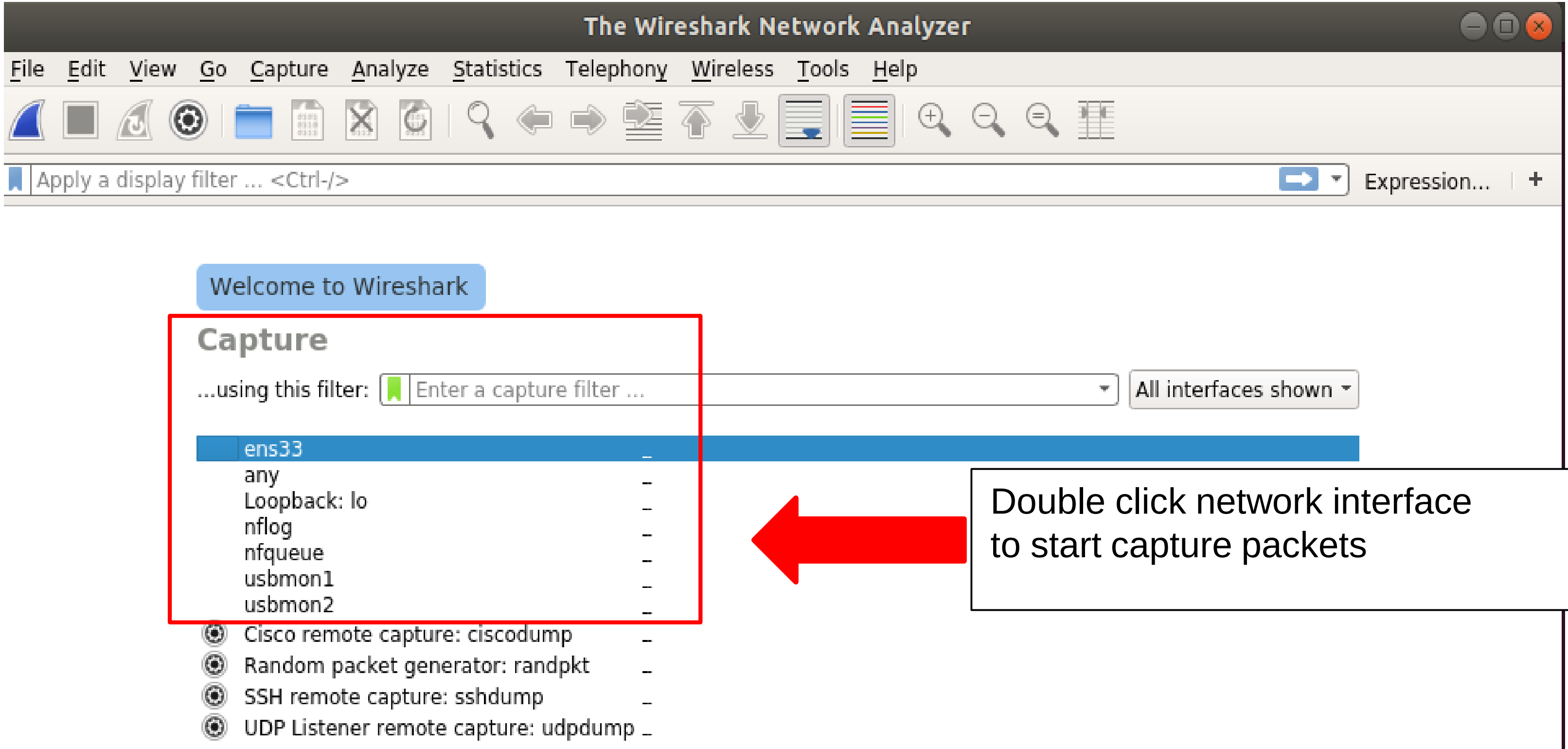
<Yes>

  - Run:

```
bash$ sudo wireshark
```

Wireshark (2/6)

- Wireshark GUI



Apply a display filter … <Ctrl-/>

Capture

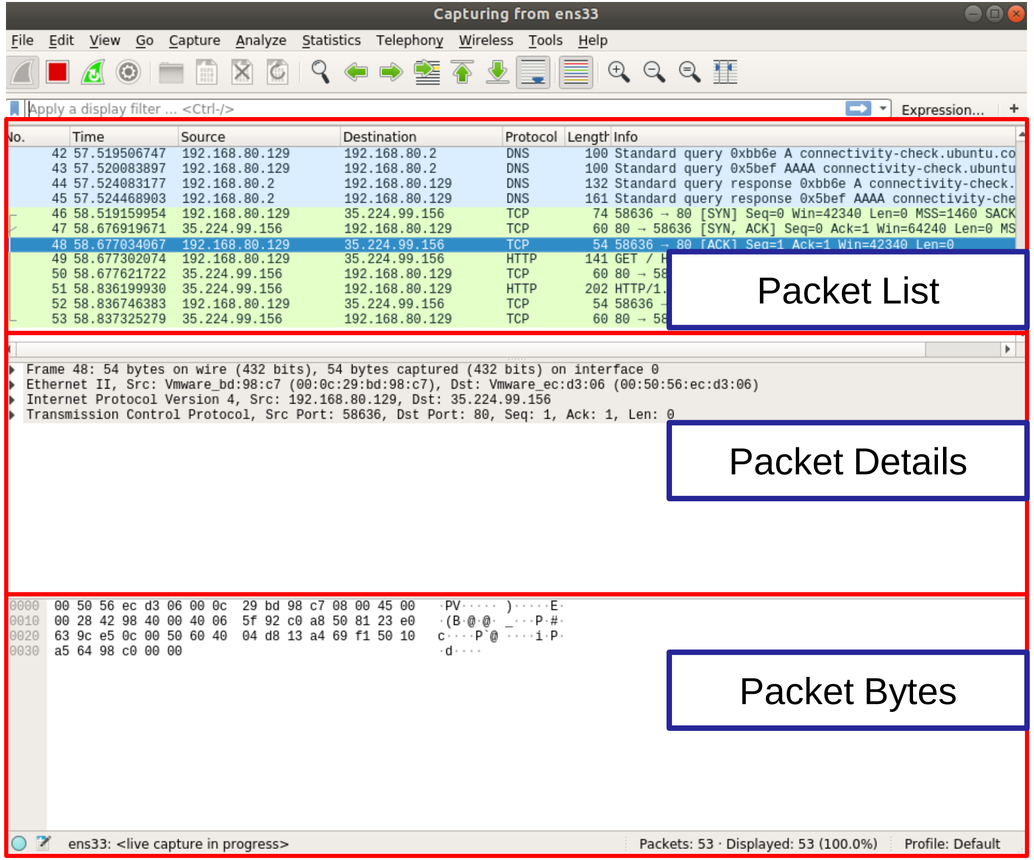
...using this filter: Enter a capture filter … All interfaces shown

ens33
any
Loopback: lo
nflog
nfqueue
usbmon1
usbmon2
Cisco remote capture: ciscodump
Random packet generator: randpkt
SSH remote capture: sshdump
UDP Listener remote capture: udpdump

Double click network interface to start capture packets

Wireshark (3/6)

- GUI panels


Capturing from ens33
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
+
Packet List
Packet Details
Packet Bytes

```
No.   Time          Source           Destination      Protocol Length Info
   42 57.519506747  192.168.80.129   192.168.80.2     DNS       100 Standard query 0xbb6e A connectivity-check.ubuntu.co
   43 57.520083897  192.168.80.129   192.168.80.2     DNS       100 Standard query 0x5bef AAAA connectivity-check.ubuntu
   44 57.524083177  192.168.80.2     192.168.80.129   DNS       132 Standard query response 0xbb6e A connectivity-check.
   45 57.524468903  192.168.80.2     192.168.80.129   DNS       161 Standard query response 0x5bef AAAA connectivity-che
   46 58.519159954  192.168.80.129   35.224.99.156    TCP        74 58636 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK
   47 58.676919671  35.224.99.156    192.168.80.129   TCP        60 80 → 58636 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MS
   48 58.677034067  192.168.80.129   35.224.99.156    TCP        54 58636 → 80 [ACK] Seq=1 Ack=1 Win=42340 Len=0
   49 58.677302074  192.168.80.129   35.224.99.156    HTTP      141 GET / H
   50 58.677621722  35.224.99.156    192.168.80.129   TCP        60 80 → 58
   51 58.836199930  35.224.99.156    192.168.80.129   HTTP      202 HTTP/1.
   52 58.836746383  192.168.80.129   35.224.99.156    TCP        54 58636 -
   53 58.837325279  35.224.99.156    192.168.80.129   TCP        60 80 → 58
```

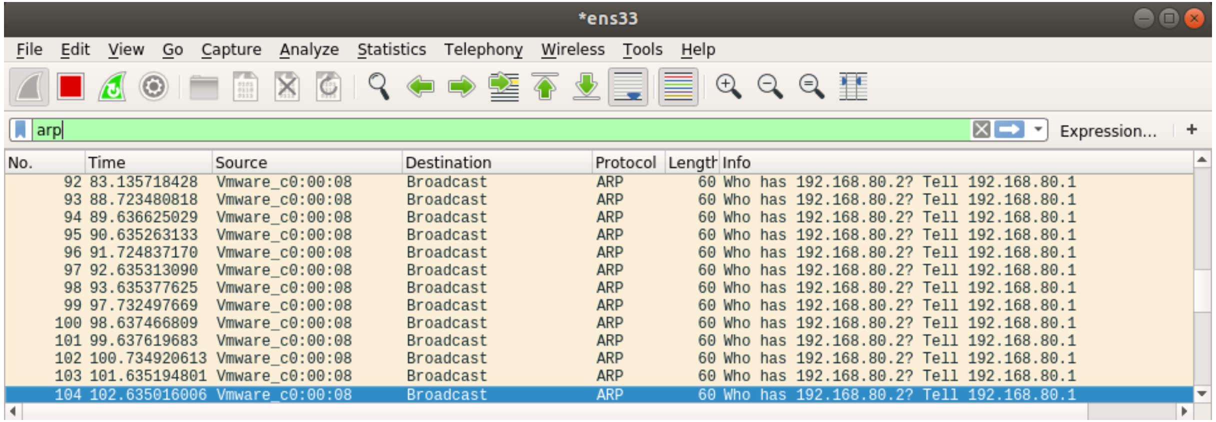
```
▸ Frame 48: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▸ Ethernet II, Src: Vmware_bd:98:c7 (00:0c:29:bd:98:c7), Dst: Vmware_ec:d3:06 (00:50:56:ec:d3:06)
▸ Internet Protocol Version 4, Src: 192.168.80.129, Dst: 35.224.99.156
▸ Transmission Control Protocol, Src Port: 58636, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
```

```
0000  00 50 56 ec d3 06 00 0c  29 bd 98 c7 08 00 45 00   ·PV·····  )·····E·
0010  00 28 42 98 40 00 40 06  5f 92 c0 a8 50 81 23 e0   ·(B·@·@·  _···P·#·
0020  63 9c e5 0c 00 50 60 40  04 d8 13 a4 69 f1 50 10   c····P`@  ····i·P·
0030  a5 64 98 c0 00 00                                  ·d····
```

ens33: <live capture in progress>   Packets: 53 · Displayed: 53 (100.0%)   Profile: Default

Wireshark (4/6)

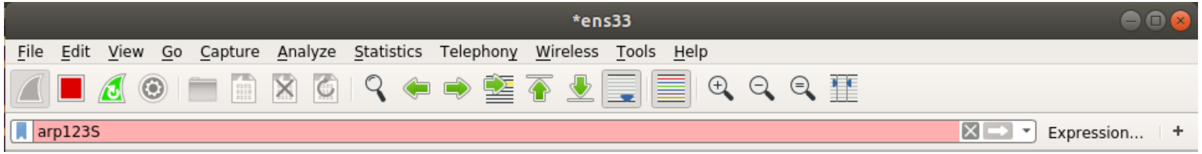
- Packet filter
  - Valid Filter


*ens33
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+

arp

No.	Time	Source	Destination	Protocol	Length	Info
92	83.135718428	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
93	88.723480818	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
94	89.636625029	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
95	90.635263133	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
96	91.724837170	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
97	92.635313090	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
98	93.635377625	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
99	97.732497669	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
100	98.637466809	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
101	99.637619683	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
102	100.734920613	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
103	101.635194801	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1
104	102.635016006	Vmware_c0:00:08	Broadcast	ARP	60	Who has 192.168.80.2? Tell 192.168.80.1

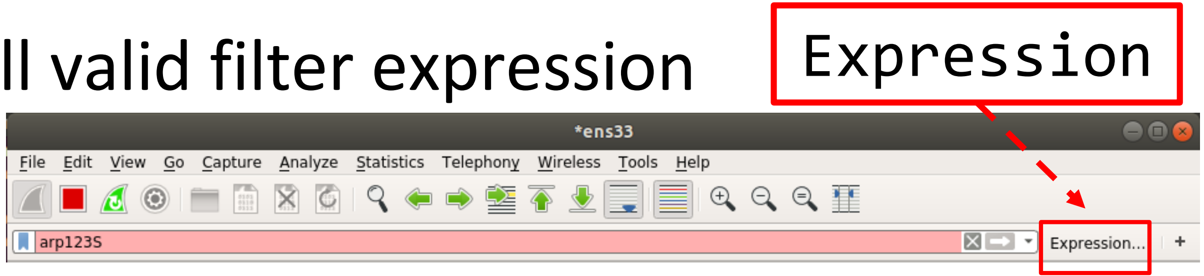
  - Invalid Filter


*ens33
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+

arp123S

Wireshark (5/6)

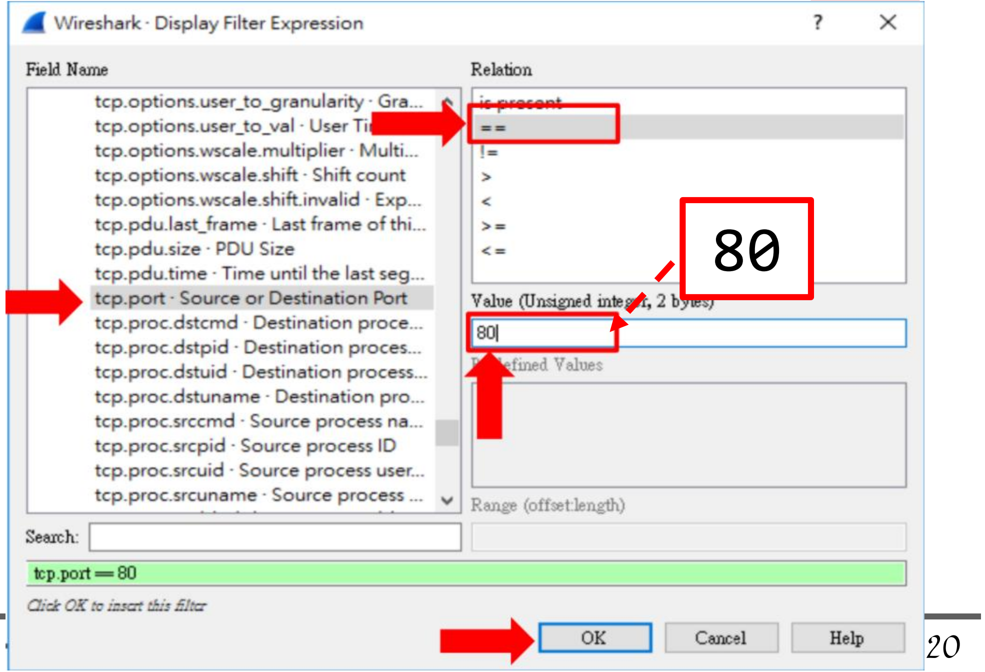
- How to obtain all valid filter expression
  - Click expression
  - Example:
    - Filter all tcp port 80
- Alternatives
  - Google for expressions



Expression

*ens33
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+

arp123S



Field Name:

- tcp.options.user_to_granularity · Gra...
- tcp.options.user_to_val · User Ti...
- tcp.options.wscale.multiplier · Multi...
- tcp.options.wscale.shift · Shift count
- tcp.options.wscale.shift.invalid · Exp...
- tcp.pdu.last_frame · Last frame of thi...
- tcp.pdu.size · PDU Size
- tcp.pdu.time · Time until the last seg...
- tcp.port · Source or Destination Port
- tcp.proc.dstcmd · Destination proce...
- tcp.proc.dstpid · Destination proces...
- tcp.proc.dstuid · Destination process...
- tcp.proc.dstuname · Destination pro...
- tcp.proc.srccmd · Source process na...
- tcp.proc.srcpid · Source process ID
- tcp.proc.srcuid · Source process user...
- tcp.proc.srcuname · Source process ...

Relation:

- is present
- ==
- !=
- >
- <
- >=
- <=

Value (Unsigned integer, 2 bytes)

80

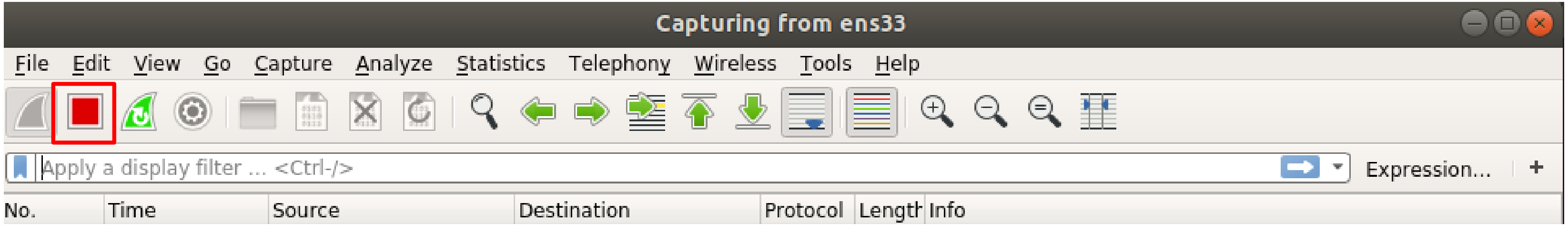
80

tcp.port == 80

Click OK to insert this filter

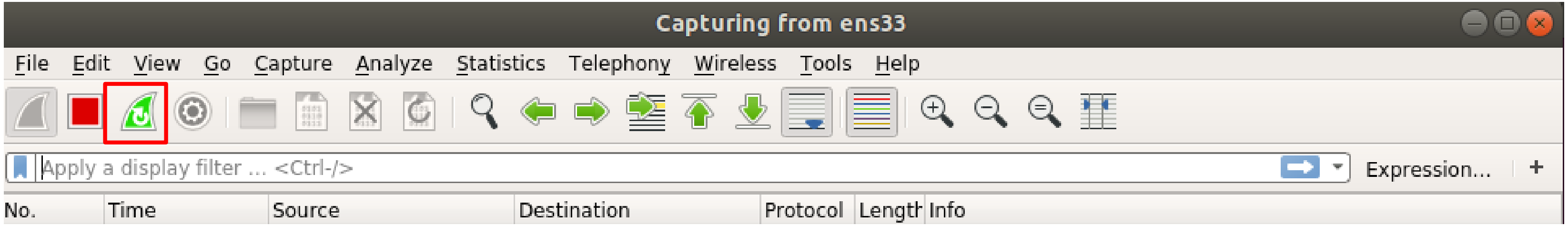
Wireshark (6/6)

- Stop capturing



Capturing from ens33
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
No. Time Source Destination Protocol Length Info

- Restart capturing



Capturing from ens33
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
No. Time Source Destination Protocol Length Info

Outline

- Objective
- Experiment Environment
- Mininet
- Packet analysis tools
  - Wireshark
  - tcpdump
- Lab requirements

Tcpdump (1/2)

- A packet analyzer runs under CLI
  - Works on most Unix-like operating systems
  - Use *libpcap.c* library to capture packets
- Installation

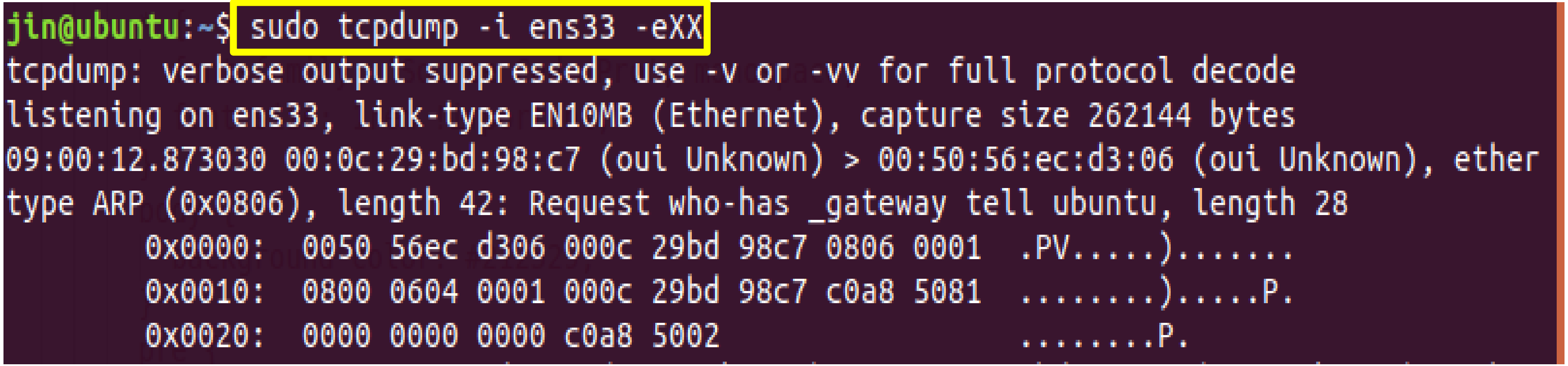
```
bash$ sudo apt install tcpdump -y
```

- Run

```
bash$ sudo tcpdump [option]
```

Tcpdump (2/2)

- Example



```
jin@ubuntu:~$ sudo tcpdump -i ens33 -eXX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
09:00:12.873030 00:0c:29:bd:98:c7 (oui Unknown) > 00:50:56:ec:d3:06 (oui Unknown), ether
type ARP (0x0806), length 42: Request who-has _gateway tell ubuntu, length 28
        0x0000:  0050 56ec d306 000c 29bd 98c7 0806 0001  .PV.....)......
        0x0010:  0800 0604 0001 000c 29bd 98c7 c0a8 5081  ........).....P.
        0x0020:  0000 0000 0000 c0a8 5002                 ........P.
```

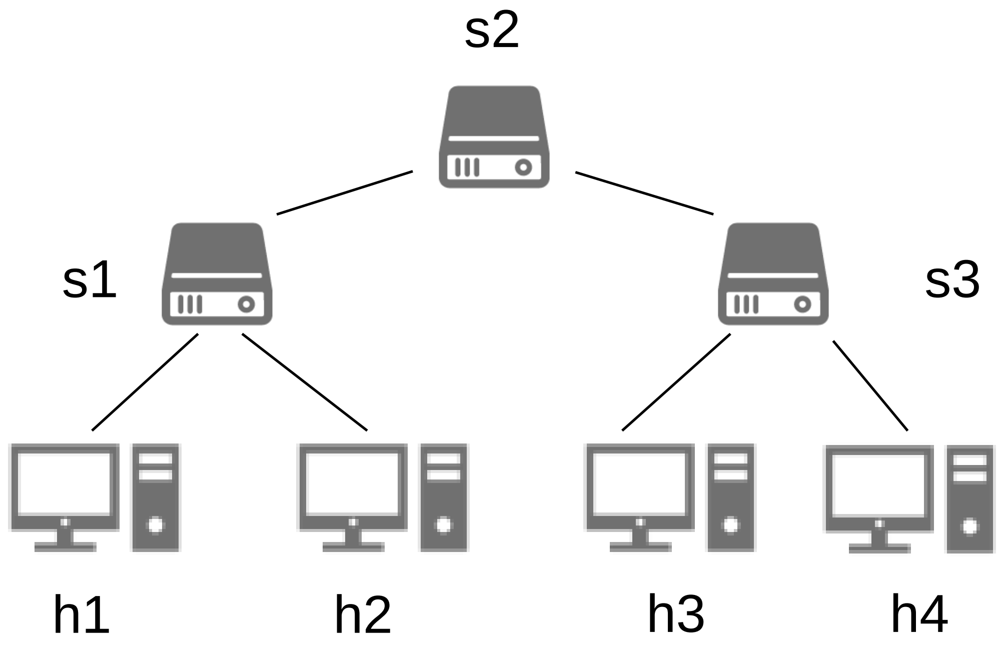
  - -i: choose an interface
  - -eXX: show packet byte value
- Man page of tcpdump
  - https://www.tcpdump.org/manpages/tcpdump.1.html

Outline

- Objective
- Experiment Environment
- Mininet
- Packet analysis tools
- Lab requirements
  - Part1: A Tree Topology
  - Part2: A Leaf-Spine Topology
  - About submit

Part1: A Tree Topology (1/4)

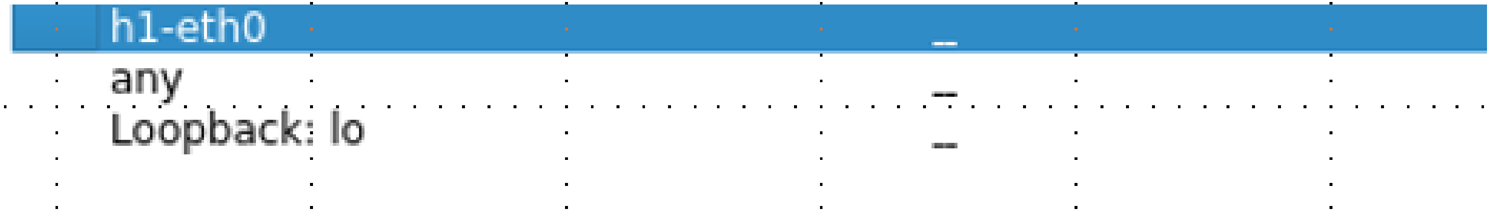
- Edit a Python script to build the following topology



Part1: A Tree Topology (2/4)

- Run part1 python script
- Run wireshark at node h1 and listen at h1-eth0

```
mininet> h1 wireshark &
```



```
h1-eth0
any
Loopback: lo
```

- Flush s1-s4 MAC learning table
  - Mininet switch may contain previous MAC information records
  - Invoke another terminal

```
bash$ sudo ovs-appctl fdb/flush s1
```

Part1: A Tree Topology (3/4)

- Check s1 MAC address table

```
bash$ sudo ovs-appctl fdb/show s1
```

- Do ping action

```
mininet> h1 ping h4 -c 5
```

  - -c: send given number ICMP packets
- Check s1 MAC address table again

Part1: A Tree Topology (4/4)

- Answer questions

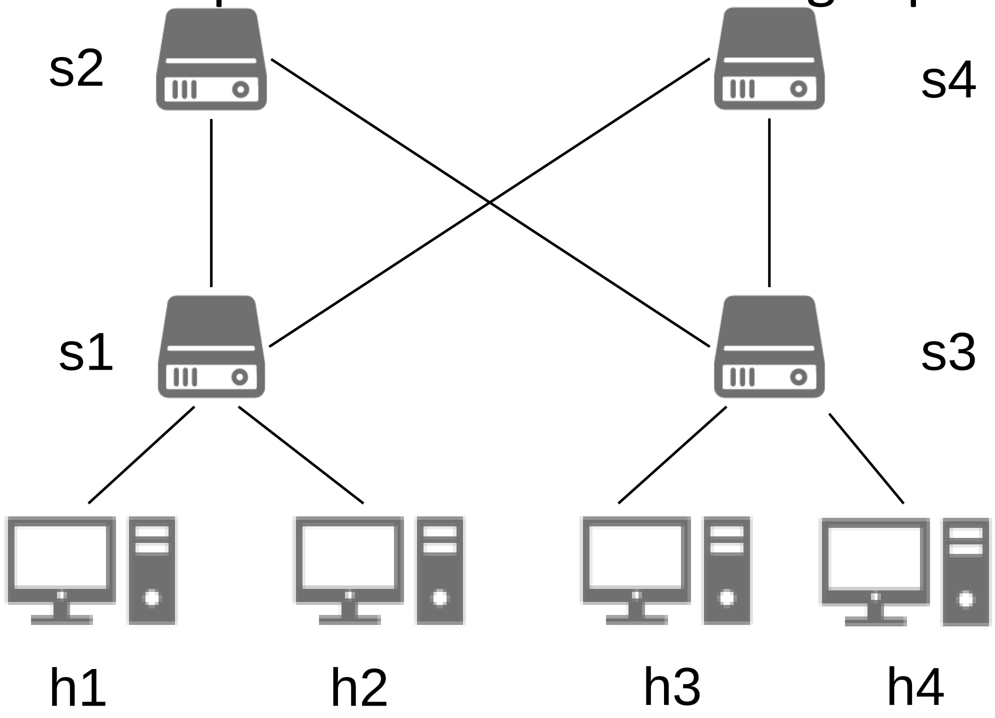
1. Flush all switch tables and take screenshots to show the switch tables of all switches.

- After h1 ping h4

2. How does h4 knows h1’s MAC address? Take screenshot on Wireshark to verify your answers.
3. How does h1 knows h4’s MAC address? Take screenshot on Wireshark to verify your answers.
4. Why does the first ping have a longer delay?
5. Show the switch tables and identify the entries that constitute the path of Ping.

Part2: A Leaf-Spine Topology (1/4)

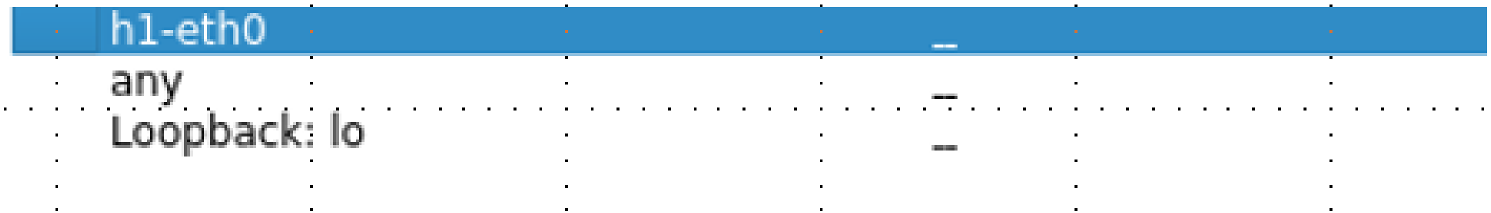
- Edit a Python script to build the following topology



Part2: A Leaf-Spine Topology (2/4)

- Run part2 python script
- Run wireshark at node h1 and listen at h1-eth0

```
mininet> h1 wireshark &
```



```
h1-eth0
any
Loopback: lo
```

- Flush s1-s4 MAC learning table
  - Mininet switch may contain previous MAC information records
  - Invoke another terminal

```
bash$ ovs-appctl fdb/flush s1
```

Part2: A Leaf-Spine Topology (2/4)

- Run ping on h1

```
mininet> h1 ping h4 -c 5
```

  - -c: send given number ICMP packets

Part2: A Leaf-spine Topology (3/4)

- Invoke another terminal
- Enable STP on all switches

```
bash$ sudo ovs-vsctl set bridge s1 stp-enable=true
bash$ sudo …
```

  - Commands may take few minutes.

- Run ping on h1

```
mininet> h1 ping h4 -c 5
```

Part2: A Leaf-spine Topology (4/4)

- Answer questions
  1. Can h1 ping h4 successfully before enabling STP?
  2. Can h1 ping h4 successfully after STP enabled?
  3. Show s1 MAC tables before and after enables STP and explain the differences.
  4. What have you observed and learned from this lab?

Report Submission

- Files
  - Python scripts:
    - lab1_part1_<stdudentID>.py (10%)
    - lab1_part2_<stdudentID>.py (10%)
  - A report: lab1_<studentID>.pdf (80%)
    - Part1, 2 Question Answers
- Submit
  - Zip Python scripts and the report into a zip file
    - Named: lab1_<studentID>.zip

References

1. Introduction to Mininet
   - https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
2. MIninet Python API
   - http://mininet.org/api/annotated.html

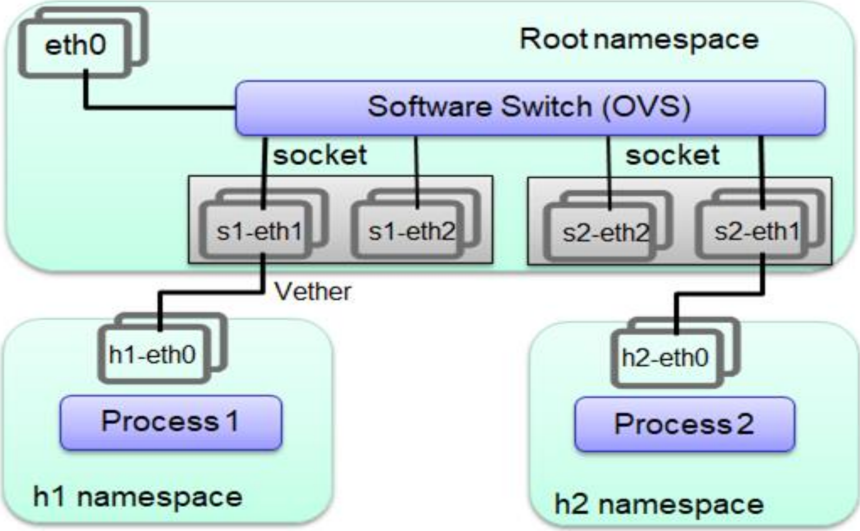
3. Manpage for Linux command
   - netstat
     - http://manpages.ubuntu.com/manpages/trusty/man8/netstat.8.html
   - mn
     - http://manpages.ubuntu.com/manpages/bionic/man1/mn.1.html

Q & A

Thank you

Appendix: Network Topology for Mininet Emulation

- ❑ Mininet employs lightweight virtualization features in the Linux kernel, including process groups, CPU bandwidth isolation, and network namespaces
- ❑ An emulated host in Mininet is a group of user-level processes moved into a network namespace


eth0
Root namespace
Software Switch (OVS)
socket
socket
s1-eth1
s1-eth2
s2-eth2
s2-eth1
Vether
h1-eth0
Process 1
h1 namespace
h2-eth0
Process 2
h2 namespace