



# Lab 1

## *Layer 2 Forwarding & MAC Learning*

**Deadline: 2022/03/01 (Tue) 23:59**



# Outline

- Objective
- Experiment Environment
- Mininet
- Packet Analysis Tools
- Lab Requirements



# 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



# Outline

---

- Objective
- Experiment Environment
- Mininet
- Packet Analysis Tools
- Lab Requirements



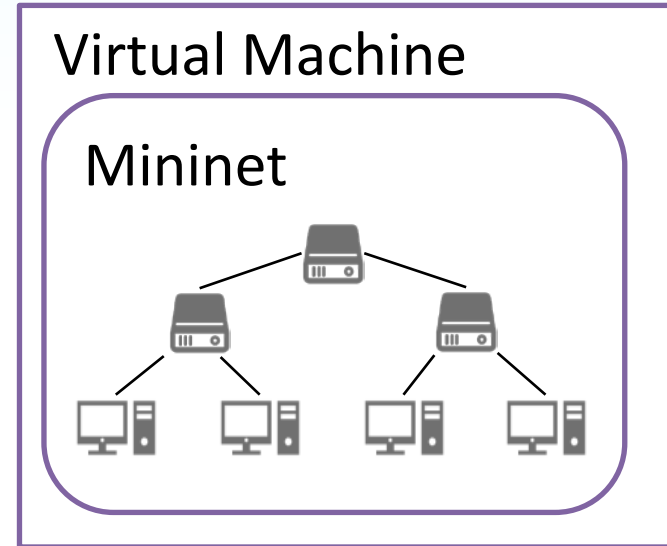
# Experiment Environment

- Oracle VM VirtualBox:
  - Developed by Oracle Corporation
  - A free and open-source hosted hypervisor
- 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, and links
- Packet Analysis Tools: Wireshark, tcpdump



# Experiment Environment

- A Virtual Machine with **Ubuntu** desktop **18.04** LTS
- **Min** Hardware requirements
  - 2 Cores (or 2 CPUs)
  - 4G RAM
  - 20G HDD
- For more installation detail, please refer to:
  - **Environment\_Setup.pdf**





# Outline

- Objective
- Experiment Environment
- Mininet
  - Overview and Installation
  - Basic Usage
- Packet Analysis Tools
- Lab Requirements





# Mininet Overview and Installation

- Mininet: Emulate a network on your computer
  - Provide Python API for building custom network
  - Provide simple built-in topology with CLI commands

- Installation:

```
$ sudo apt install mininet
```

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



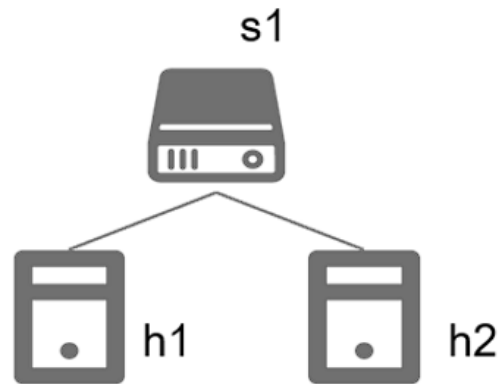
# Outline

- Objective
- Experiment Environment
- Mininet
  - Overview and Installation
  - Basic Usage
    - Method 1: Built-in Topology
    - Method 2: Custom Topology
- Packet Analysis Tools
- Lab Requirements

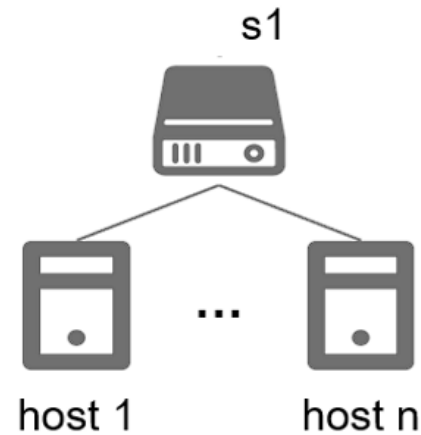


# Method 1: Built-in Topology (1/2)

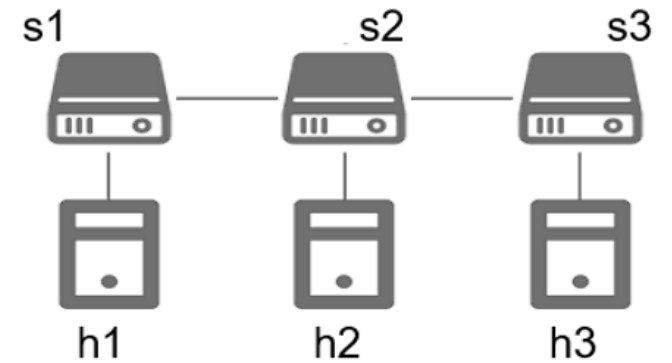
- Five built-in topologies
  - Minimal (Default)
  - Single
  - Linear
  - Torus
  - Tree



▲ Minimal



▲ Single



▲ Linear

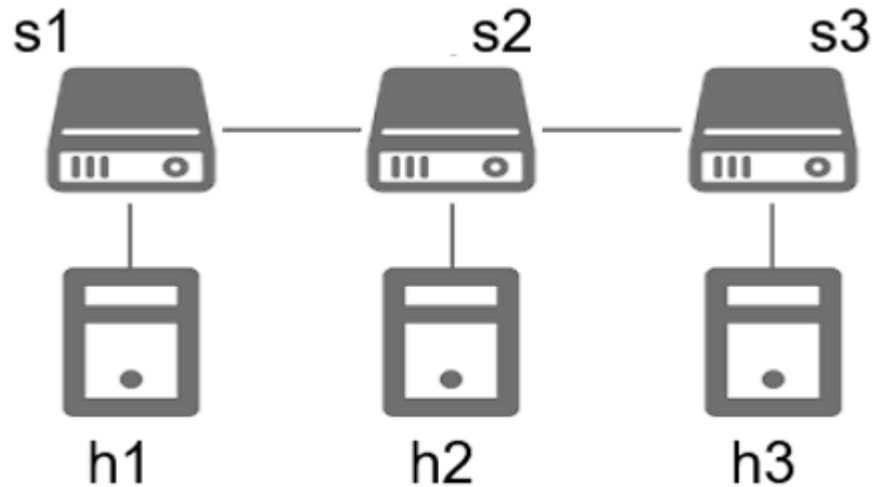


## Method 1: Built-in Topology (2/2)

- Example: Create a linear topology with 3 switch

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

- “--topo” specifies the topology
- By default, each switch has a host



```
*** 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)
```



# Clear Your Experiment Environment

- Note:

- Make sure to clean up the environment of Mininet after every time you exit Mininet CLI

```
$ sudo mn -c          # clean and exit
```

- A "cleanup" command to get rid of junk (interfaces, processes, files in /tmp, etc.) which might be left around by Mininet or Linux



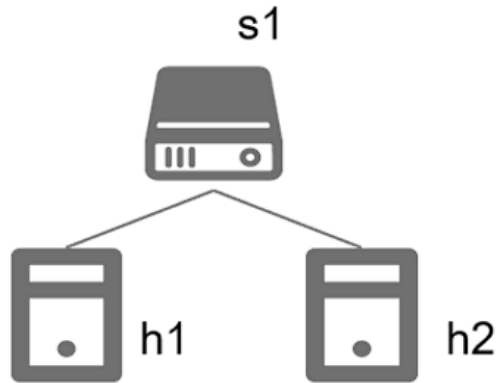
# Outline

- Objective
- Experiment Environment
- Mininet
  - Overview and Installation
  - Basic Usage
    - Method 1: Build-in Topology
    - Method 2: Custom Topology
- Packet Analysis Tools
- Lab Requirements



## Method 2: Custom Topology

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



- Execute your python script

```
$ sudo python sample.py
```

```
#!/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()
```

sample.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)

- Run command on a node

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

- Show network interface configuration of a node

```
mininet> h1 ifconfig
```

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

```
mininet> h1 xterm &
```

- Exit mininet

```
mininet> exit
```

- Always clear network topology before you start

```
$ sudo mn -c
```



# Outline

---

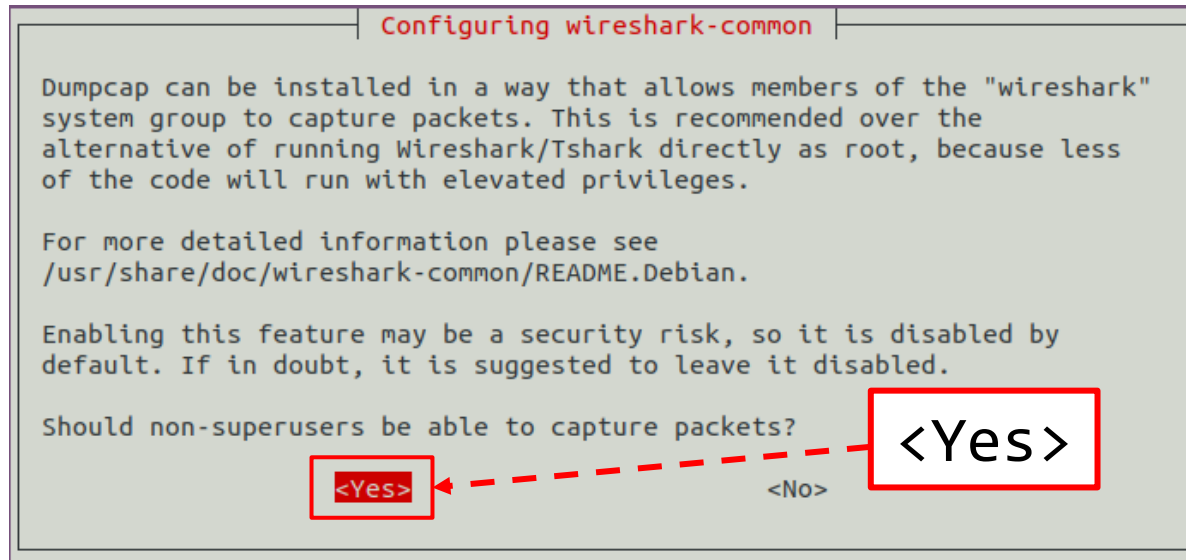
- Objective
- Experiment Environment
- Mininet
- Packet Analysis Tools
  - Wireshark
  - tcpdump
- Lab Requirements



# Wireshark (1/6)

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

```
$ sudo apt install wireshark
```



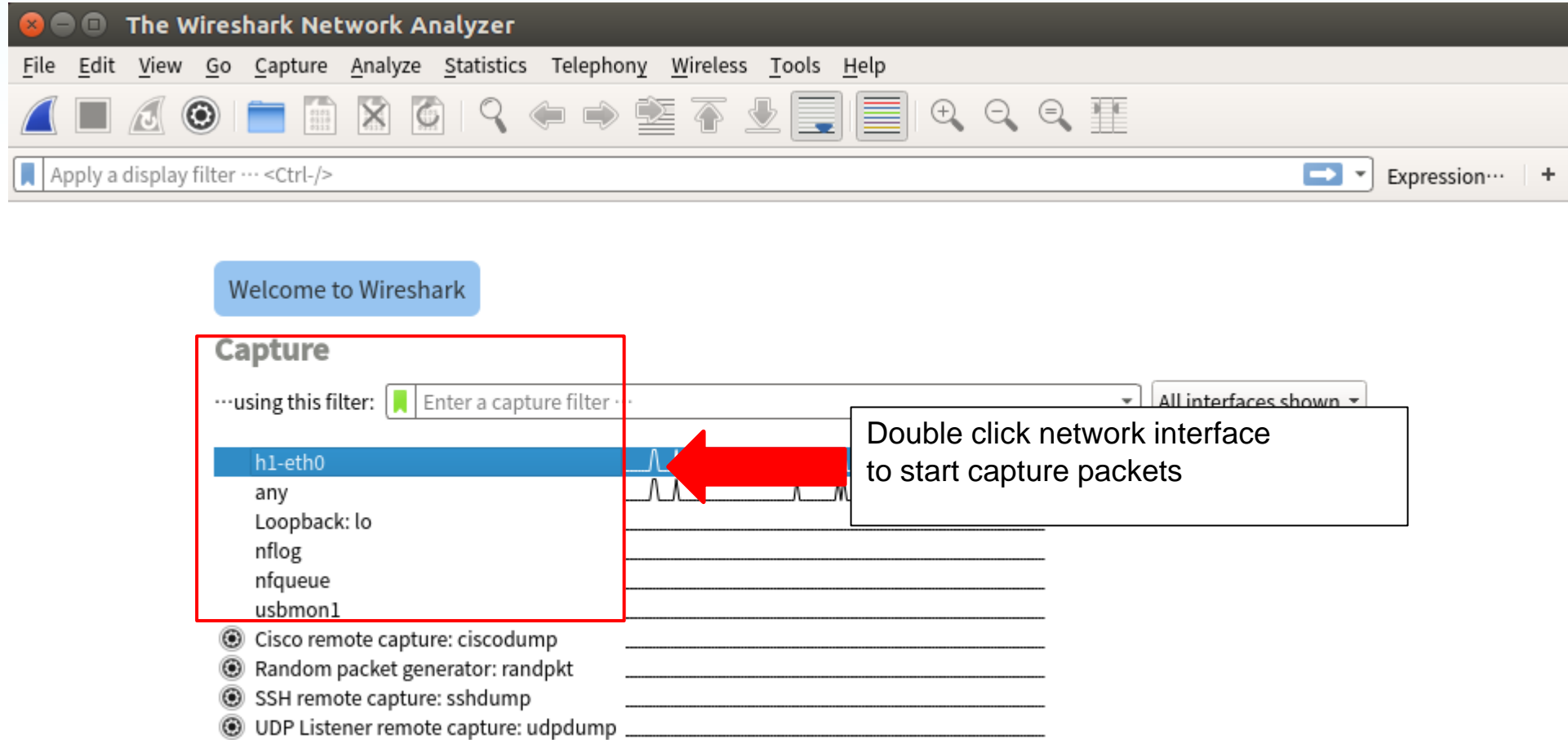
- Run:

```
$ sudo wireshark
```



# Wireshark (2/6)

- Wireshark GUI





# Wireshark (3/6)

- GUI panels

The screenshot displays the Wireshark interface with the following components:

- Packet List:** A table showing captured packets. The selected packet is number 11, an ICMP Echo (ping) reply from 10.0.0.1 to 10.0.0.2.
- Packet Details:** A hierarchical tree view showing the structure of the selected packet. It includes Ethernet II, Internet Protocol Version 4, and Internet Control Message Protocol (ICMP) fields.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::bc85:2ff...	ff02::2	ICMPv6	70	Router Solicitation fro...
2	1.223609612	d2:d0:85:d9:45...	Broadcast	ARP	42	Who has 10.0.0.1? Tell ...
3	1.223619323	be:85:2f:f1:54...	d2:d0:85:d9:45:1d	ARP	42	10.0.0.1 is at be:85:2f...
4	1.224041840	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...
5	1.224048790	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id...
6	2.223572682	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...
7	2.223582782	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id...
8	3.231450711	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...
9	3.231460011	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id...
10	4.255600693	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...
11	4.255619114	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id...
12	5.279746062	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...
13	5.279754713	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id...
14	6.400427000	be:85:2f:f1:54...	d2:d0:85:d9:45:1d	ARP	42	Who has 10.0.0.2? Tell ...
15	6.401049607	d2:d0:85:d9:45...	be:85:2f:f1:54:98	ARP	42	10.0.0.2 is at d2:d0:85...
16	7.409923618	fe80::1c0e:69f...	ff02::fb	MDNS	107	Standard query 0x0000 P...
17	11.002132083	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...
18	11.002140313	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id...
19	12.031867227	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...
20	12.031876468	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id...
21	13.055944965	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...
22	13.055953465	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id...
23	14.099729595	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id...

**Packet Details:**

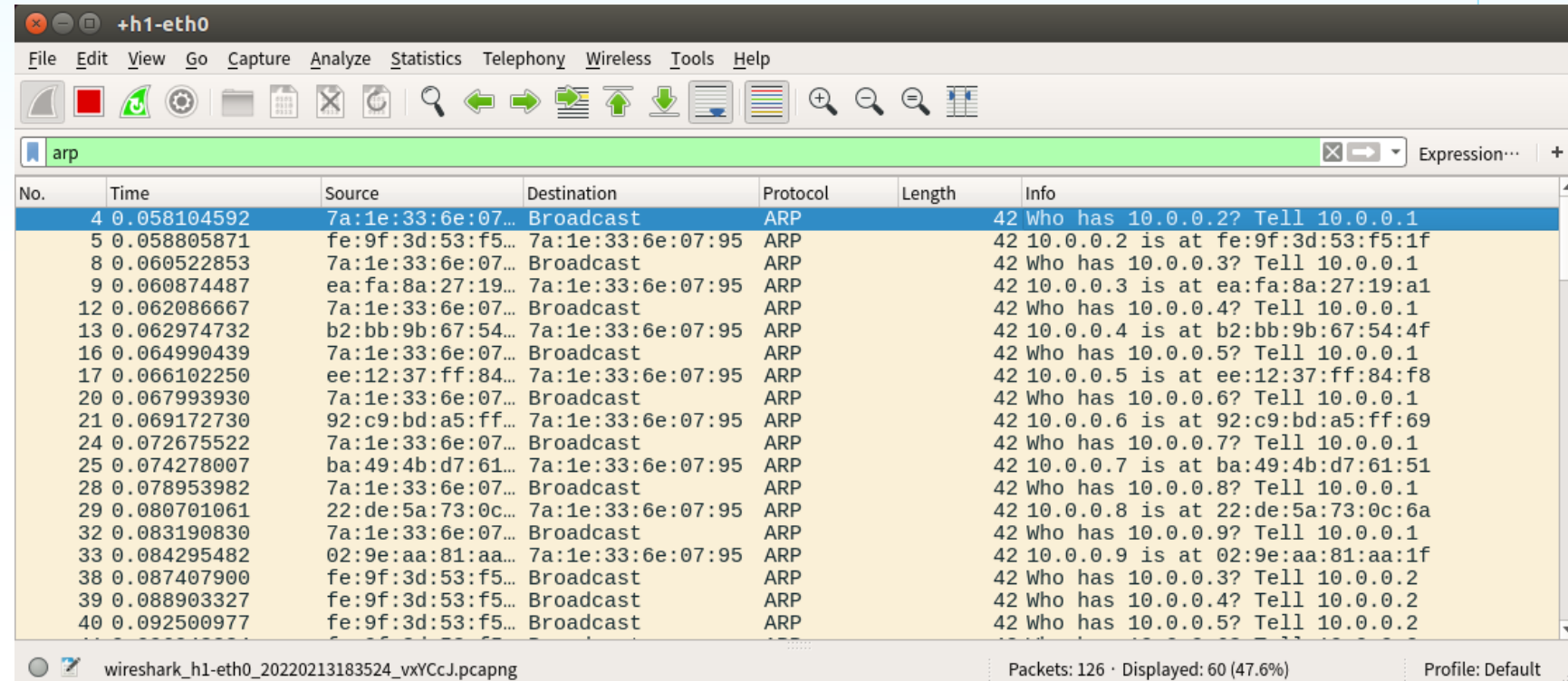
- Frame 11: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
- Ethernet II, Src: be:85:2f:f1:54:98 (be:85:2f:f1:54:98), Dst: d2:d0:85:d9:45:1d (d2:d0:85:d9:45:1d)
- Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
  - 0100 .... = Version: 4
  - .... 0101 = Header Length: 20 bytes (5)
  - Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  - Total Length: 84
  - Identification: 0x9612 (38418)
  - Flags: 0x0000
  - Time to live: 64
  - Protocol: ICMP (1)
  - Header checksum: 0xd094 [validation disabled]
  - [Header checksum status: Unverified]
  - Source: 10.0.0.1
  - Destination: 10.0.0.2
- Internet Control Message Protocol



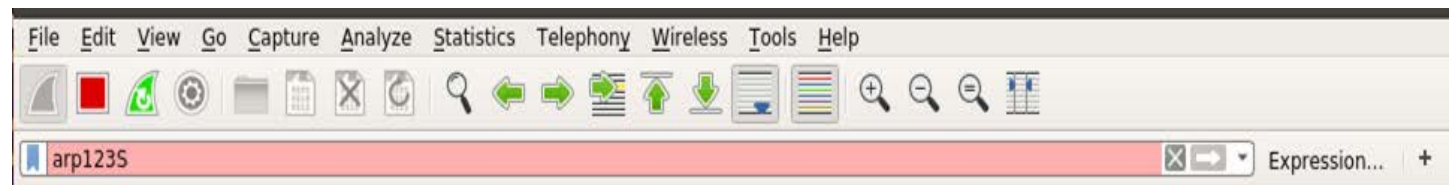
# Wireshark (4/6)

- Packet filter

- Valid Filter →



- Invalid Filter →



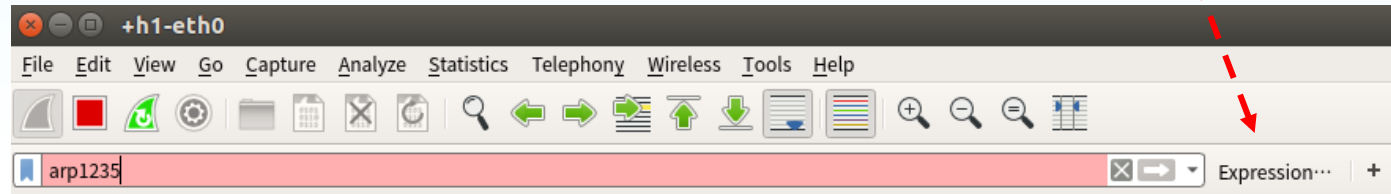


# Wireshark (5/6)

- How to obtain all valid filter expression

- Click expression

Expression



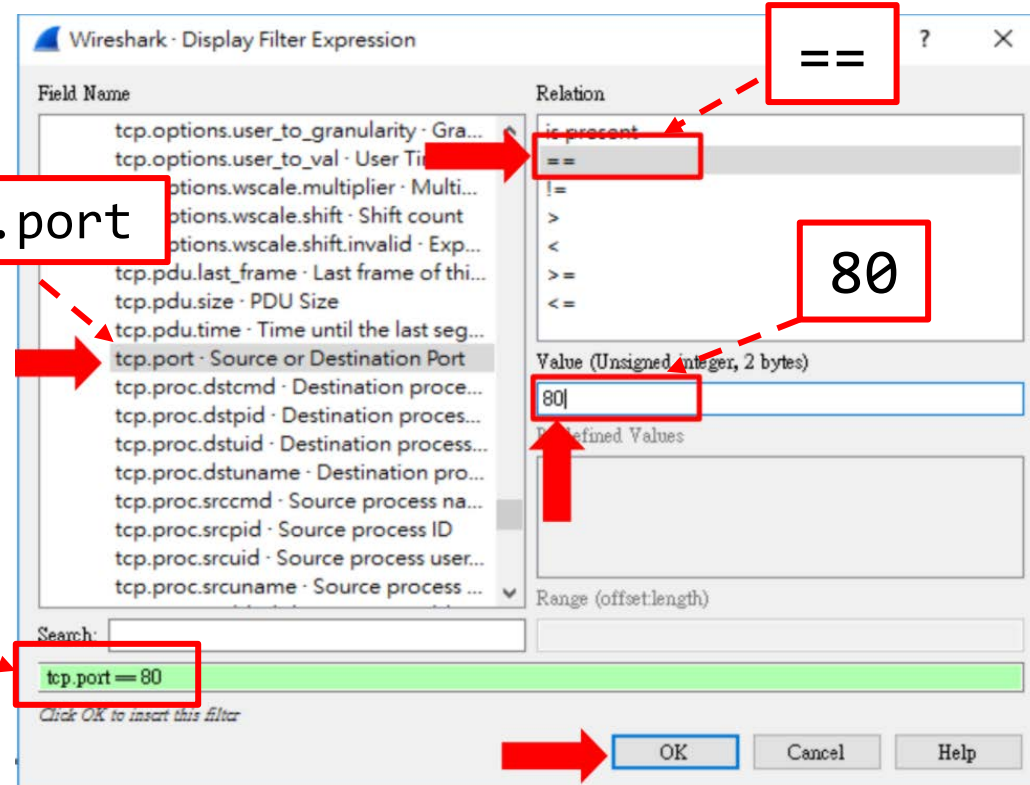
- Example:

- Filter all tcp port 80

tcp.port

80

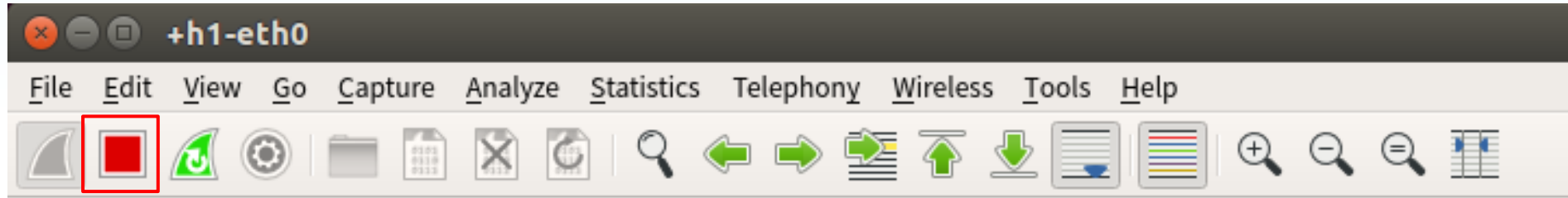
tcp.port == 80



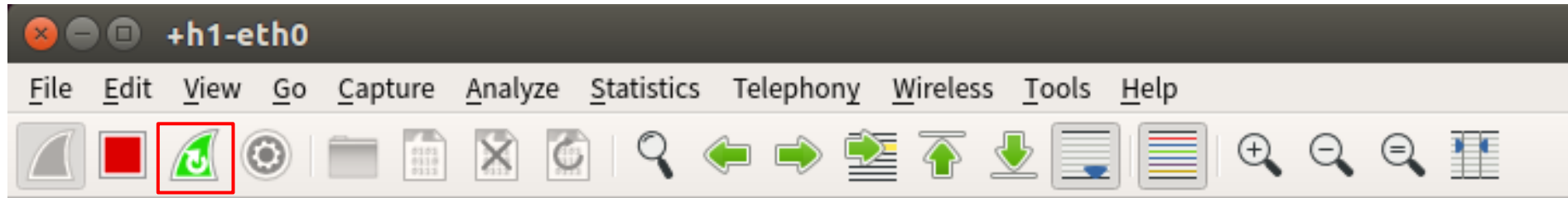


# Wireshark (6/6)

- Stop capturing



- Restart capturing







# Outline

---

- Objective
- Experiment Environment
- Mininet
- Packet Analysis Tools
  - Wireshark
  - tcpdump
- Lab Requirements



# tcpdump (1/3)

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

- Installation

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

- Run

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



## tcpdump (2/3)

- Example

```
winlab@server172:~$ sudo tcpdump -i ens11f0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens11f0, link-type EN10MB (Ethernet), capture size 262144 bytes
08:40:02.873698 02:eb:6b:52:03:fb (oui Unknown) > Broadcast, ethertype Unknown (
0x8942), length 135:
    0x0000:  0207 0418 0f76 fb12 6a04 0202 3106 0200  ....v..j...1...
    0x0010:  78fe 12a4 2305 014f 4e4f 5320 4469 7363  x...#..ONOS.Disc
    0x0020:  6f76 6572 79fe 17a4 2305 026f 663a 3030  overy...#..of:00
```

- -i: choose an interface
- Man page of tcpdump
  - <https://www.tcpdump.org/manpages/tcpdump.1.html>



## tcpdump (3/3)

- How to know what interfaces are on the host

```
winlab@server172: ~  
winlab@server172:~$ ifconfig  
enp8s0    Link encap:Ethernet  HWaddr 8c:ea:1b:30:da:01  
          inet addr:140.113.131.172  Bcast:140.113.131.191  Mask:255.255.255.192  
          inet6 addr: fe30::8eea:1bff:fe30:da01/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:75055400 errors:0 dropped:28 overruns:0 frame:0  
          TX packets:2803394 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:5310400469 (5.3 GB)  TX bytes:181265732 (181.2 MB)  
          Memory:c7500000-c75fffff  
  
ens11f0   Link encap:Ethernet  HWaddr 8c:ea:1b:30:da:6f  
          inet addr:192.168.168.2   Bcast:192.168.168.255  Mask:255.255.255.0  
          inet6 addr: fe80::8eea:1bff:fe30:da6f/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:1758006 errors:0 dropped:1757674 overruns:0 frame:0  
          TX packets:4541 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:237316142 (237.3 MB)  TX bytes:307630 (307.6 KB)
```



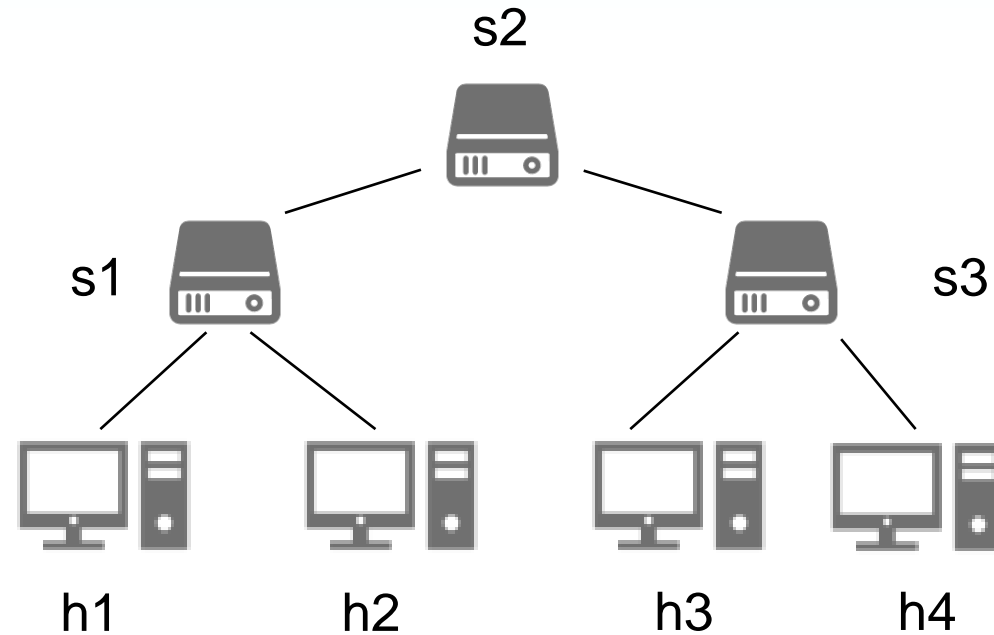
# Outline

- Objective
- Experiment Environment
- Mininet
- Packet Analysis Tools
- Lab Requirements
  - Part1: A Tree Topology
  - Part2: A Leaf-Spine Topology
  - About Submission



## 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

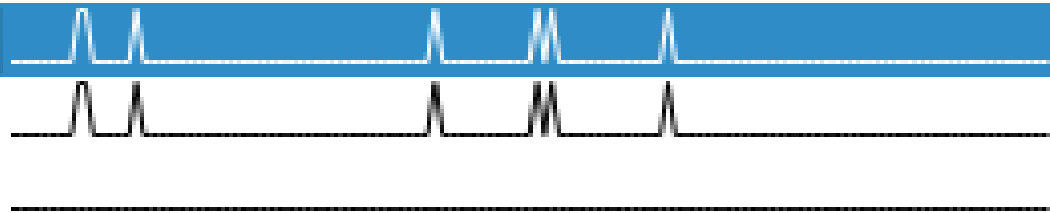
```
mininet> h1 wireshark &
```

- Capture packets on h1-eth0

h1-eth0

any

Loopback: lo





## Part1: A Tree Topology (3/4)

- Invoke another terminal
- Flush s1~s3 MAC address table
  - Mininet switch may contain previous MAC information records

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

- Observe s1 MAC address table

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

- Do ping action

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

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

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





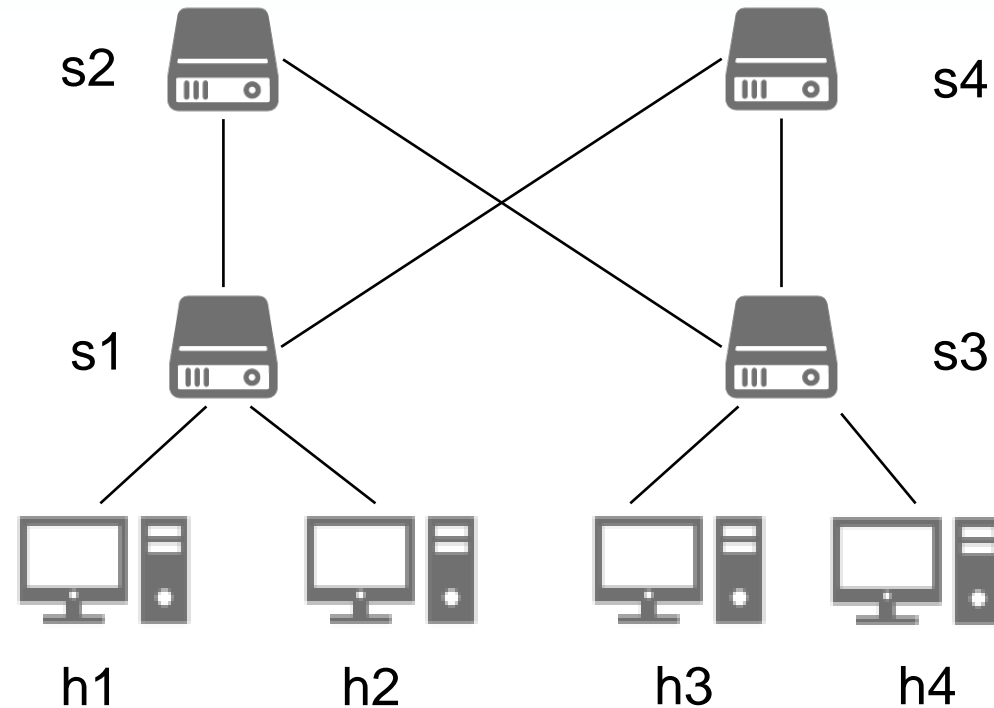
## Part1: A Tree Topology (4/4)

- Answer questions
  1. Flush all switch tables and take screenshots to show the switch tables of all switches. (5%)
    - After h1 ping h4
  2. How does h4 knows h1's MAC address? Take screenshot on Wireshark to verify your answers. (10%)
  3. How does h1 knows h4's MAC address? Take screenshot on Wireshark to verify your answers. (10%)
  4. Why does the first ping have a longer delay? (10%)
  5. Show the switch tables and identify the entries that constitute the path of Ping. (10%)



## Part2: A Leaf-Spine Topology (1/5)

- Edit a Python script to build the following topology





## Part2: A Leaf-Spine Topology (2/5)

- Run part2 python script
- Run wireshark at node h1

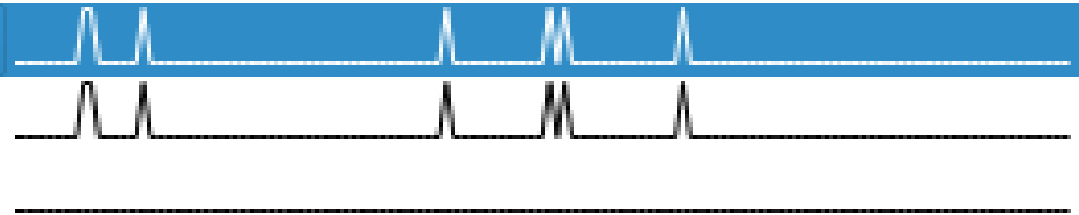
```
mininet> h1 wireshark &
```

- Capture packets on h1-eth0

h1-eth0

any

Loopback: lo





## Part2: A Leaf-Spine Topology (3/5)

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

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

- Run ping on h1

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

- -c: send given number ICMP packets



## Part2: A Leaf-Spine Topology (4/5)

- Enable STP on all switches (s1~s4)

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

- Commands may take few minutes.

- Run ping on h1

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



## Part2: A Leaf-Spine Topology (5/5)

- Answer questions
  1. Can h1 ping h4 successfully before enabling STP? Take screenshots to justify your answer. (10%)
  2. Can h1 ping h4 successfully after STP enabled? Take screenshots to justify your answer. (10%)
  3. Show s1 MAC tables before and after enables STP and explain the differences. (10%)
  4. What have you observed and learned from this lab? (5%)



# About Submission

- Files

- Two Python scripts:
  - lab1\_part1\_<studentID>.py (10%)
  - lab1\_part2\_<studentID>.py (10%)
- A report: lab1\_<studentID>.pdf (80%)
  - Part1 and Part2 Question Answers

- Submission

- Zip Python scripts and the report into a zip file
  - Named: lab1\_<studentID>.zip
- Wrong file name or format subjects to 10 points deduction



# References

- Introduction to Mininet
  - <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>
- Mininet Python API
  - <http://mininet.org/api/annotated.html>
- 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