Computer Networks
@CS.NYCU

Lab. 1: Network Emulation with Mininet

Instructor: Kate Lin

TA: 張祐誠、蘇名偉、翁瑞澤

Outline

- Objectives
- Background
- Tasks
- Submission
- Grading Policy
- References

Objectives

In this lab, we are going to write a Python program which can generate a network topology via *Mininet* and use *iPerf* to generate flows and measure the bandwidth in this topology

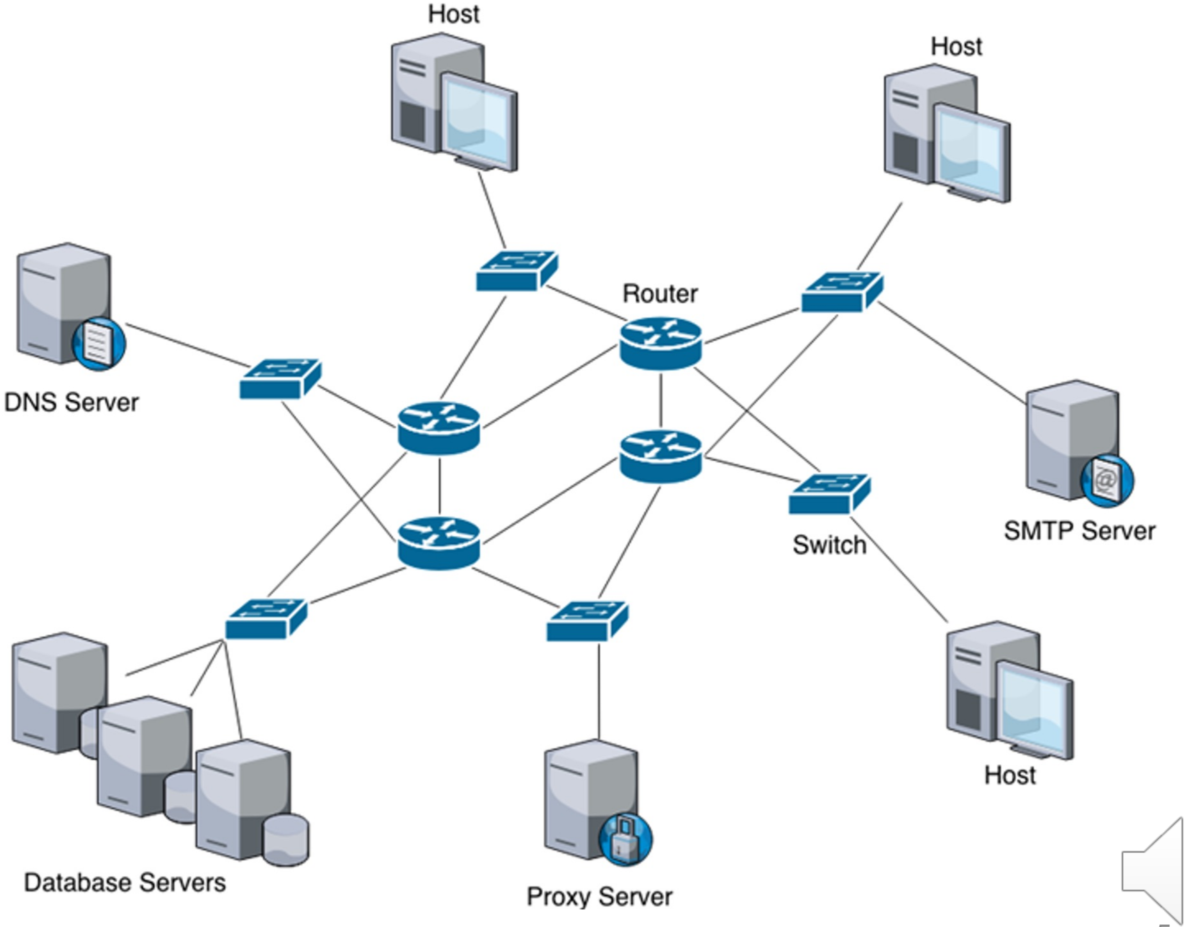
1. Learn how to create a network topology via **Mininet**
2. Learn how to generate flows by using **iPerf** in Mininet
3. Learn how to use **Wireshark** to filter packets and perform analysis

Background

- Network Topology
- Mininet
- iPerf
- Wireshark

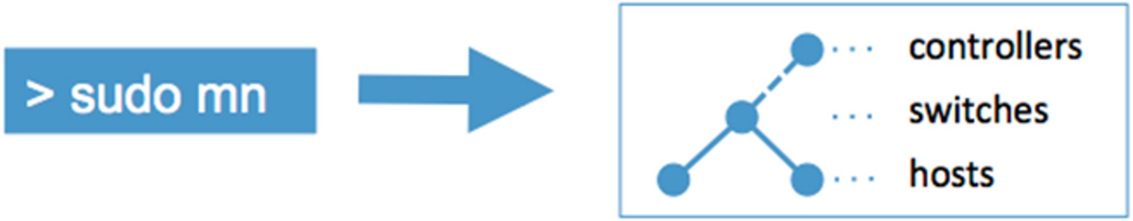
Network Topology

- Hosts
- Switches
- Links



Mininet

- Mininet is a network emulator
- Create a realistic **virtual** network, running real kernel, switch and application code, on a single machine (VM, cloud or native)
- Run a collection of end-hosts, switches, routers, and links on a single Linux kernel



```
> sudo mn
```

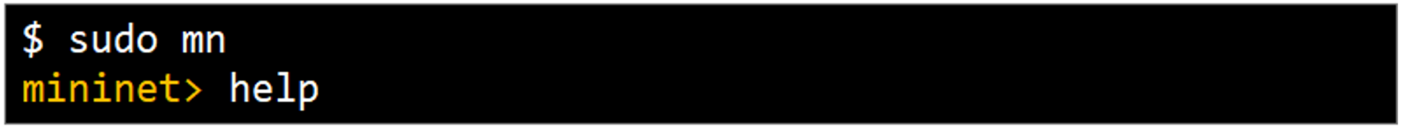
Notice: We have provided you a VM that has Mininet installed
(You don't have to install Mininet by yourself)

Why Mininet?

- Fast and easy to configure
- Create custom topologies
- Run real programs
- Customize packet forwarding
- Support OpenFlow and software-defined network (SDN)

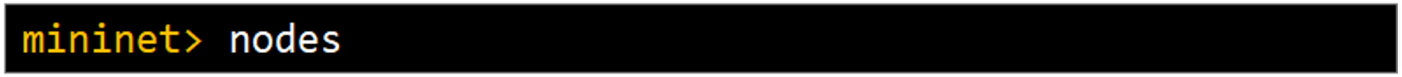
Mininet CLI (Command-Line Interface)

- Start a simple minimal topology and enter the CLI



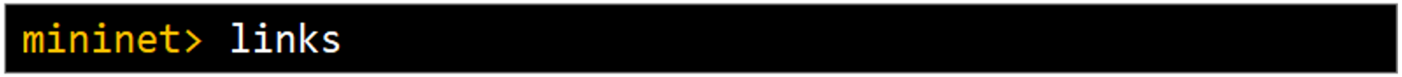
```
$ sudo mn
mininet> help
```

- Show the information of all the nodes



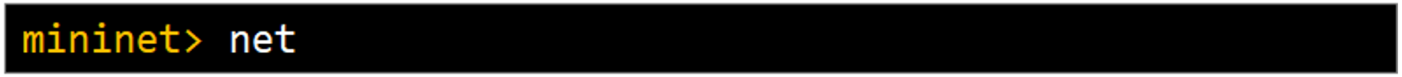
```
mininet> nodes
```

- Show all the links in the network



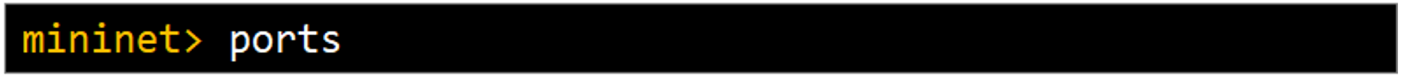
```
mininet> links
```

- Show the network topology



```
mininet> net
```

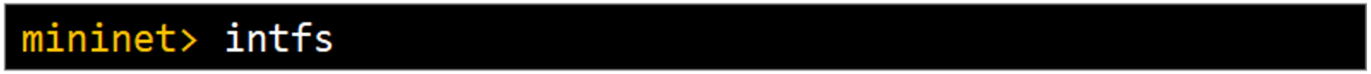
- Show all the ports on every switch



```
mininet> ports
```

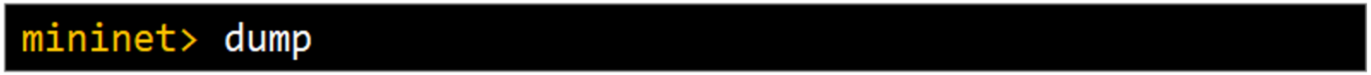
Mininet CLI (Command-Line Interface)

- Show all network interfaces



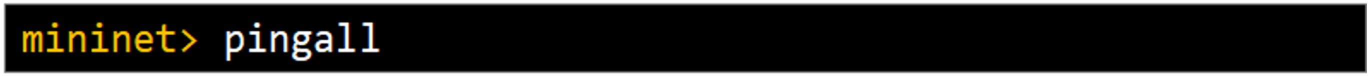
```
mininet> intfs
```

- Dump information about all the nodes



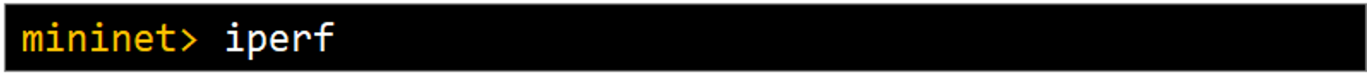
```
mininet> dump
```

- Test the connectivity of all the hosts



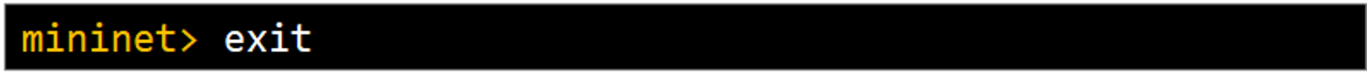
```
mininet> pingall
```

- Test TCP connection of two hosts with iPerf



```
mininet> iperf
```

- Leave the CLI mode



```
mininet> exit
```

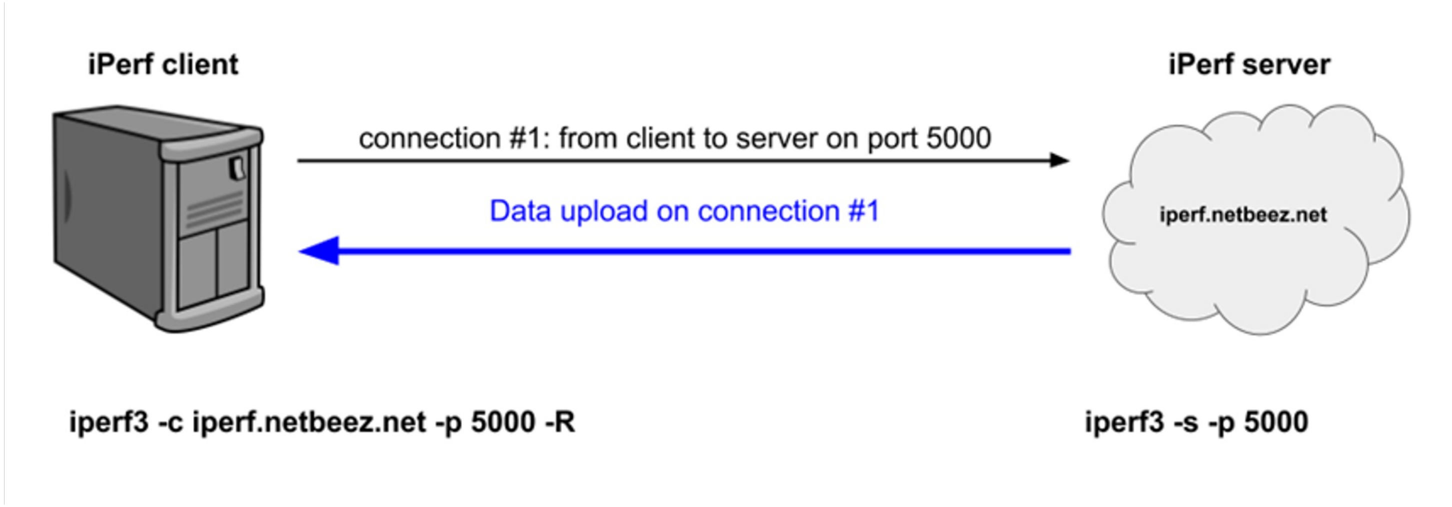
Notice: After exiting the mininet, use "sudo mn -c" to clean up the environment. Otherwise you may get some error, such as " File Exists Error".

Mininet References

- English
  - Mininet Walkthrough
  - Introduction to Mininet
  - Mininet Python API Reference Manual
  - A Beginner’s Guide to Mininet
- Chinese
  - GitHub/OSE-Lab - 熟悉如何使用 Mininet
  - 菸酒生的記事本 – Mininet 筆記
  - Hwchiu Learning Note – 手把手打造仿 mininet 網路
  - 阿寬的實驗室 – Mininet 指令介紹
  - Mininet 學習指南

iPerf

- iPerf is a tool for active measurements of the maximum achievable bandwidth in IP networks
- Support tuning of various parameters related to timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and IPv6)


iPerf client
iPerf server
connection #1: from client to server on port 5000
Data upload on connection #1
iperf.netbeez.net

```
iperf3 -c iperf.netbeez.net -p 5000 -R
```

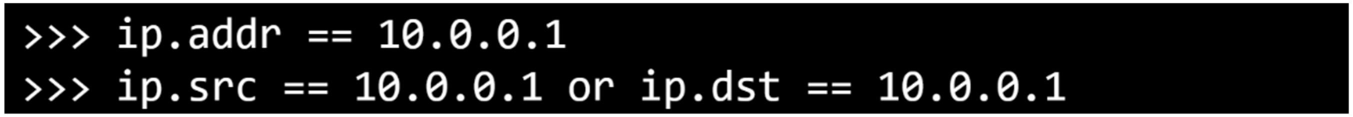
```
iperf3 -s -p 5000
```

iPerf

- iPerf Command line options
  - -s: (Server) Run iPerf in server mode
  - -c: (Client) Run iPerf in client mode, connecting to an iPerf server running on host
  - -i: (Interval) Sets the interval time in seconds between periodic bandwidth, jitter, and loss reports
  - -t: (Time) The time in seconds to transmit for
  - -p: (Port) The server port for the server to listen on and the client to connect to
  - -u: (UDP) Use UDP.
  - -b: (bandwidth) Set target bandwidth to n bits/sec (default 1 Mbit/sec for UDP, unlimited for TCP)
  - other

Wireshark

- Wireshark is a widely-used network protocol analyzer
  - Deep inspection of hundreds of protocols
  - Live capture and offline analysis
  - Most powerful display filter
  - Read/write many different capture file formats
- Examples of DisplayFilter
  - Load a PCAP file
  - Show any traffic to or from 10.0.0.1



```
>>> ip.addr == 10.0.0.1
>>> ip.src == 10.0.0.1 or ip.dst == 10.0.0.1
```

Wireshark Filtering Rules

- Filter the packets that match some conditions
  - For example, to find TCP packets with a port number of 80, you can use **tcp.port==80**
- For more filter instructions, please reference to:
  - DisplayFilters
- Frequently used:
  - ip.src, ip.dst, ip.addr, … (IP address)
  - tcp.port, tcp.srcport, tcp.dstport, … (port)
  - eth.src, eth.dst, eth.addr, … (MAC address)

Tasks

1. Environment Setup
2. Create a Topology
3. Generate Flows via Iperf
4. Compute Throughput
5. Check Your Answer
6. Report

Task 1. Environment Setup

- Step1. Install necessary tools on your computer
  - Wireshark
    - Windows / MacOS (Wireshark)
    - Ubuntu Linux

```
$ sudo apt install wireshark
```

- Step2. Join the **GitHub Classroom Lab1**
  - GitHub Classroom Lab1

Task 1. Environment Setup (cont.)

- Step3. Install Oracle VM VirtualBox
  - Oracle VM VirtualBox - Downloads
- Step4. Download TA's ova file and import it into your Oracle VM VirtualBox
  - Lab1.ova
  - Password: cn2023
  - How To Use OVA Files with VirtualBox (alphr.com)

Task 1. Environment Setup (cont.)

- Step5. Download required files from GitHub

```
$ git clone https://github.com/NYCU-CN2023/Lab1-
<GITHUB_ID>.git
```

- Step6. Get and set repository for global options

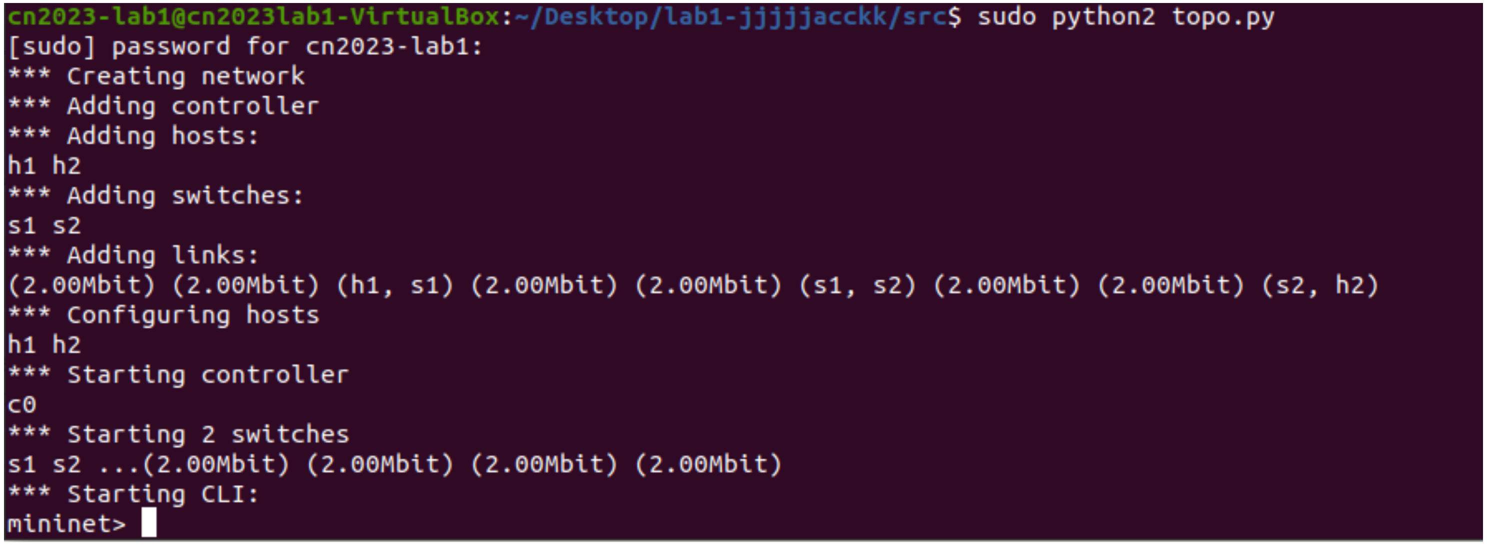
```
$ cd Lab1-<GITHUB_ID>
$ git config --global user.name "<NAME>"
$ git config --global user.email "<EMAIL>"
```

Task 2. Create a Topology

- Run the example code

```
# Notice: Mininet must run in python2
$ cd ./src/
$ sudo python2 topo.py
```

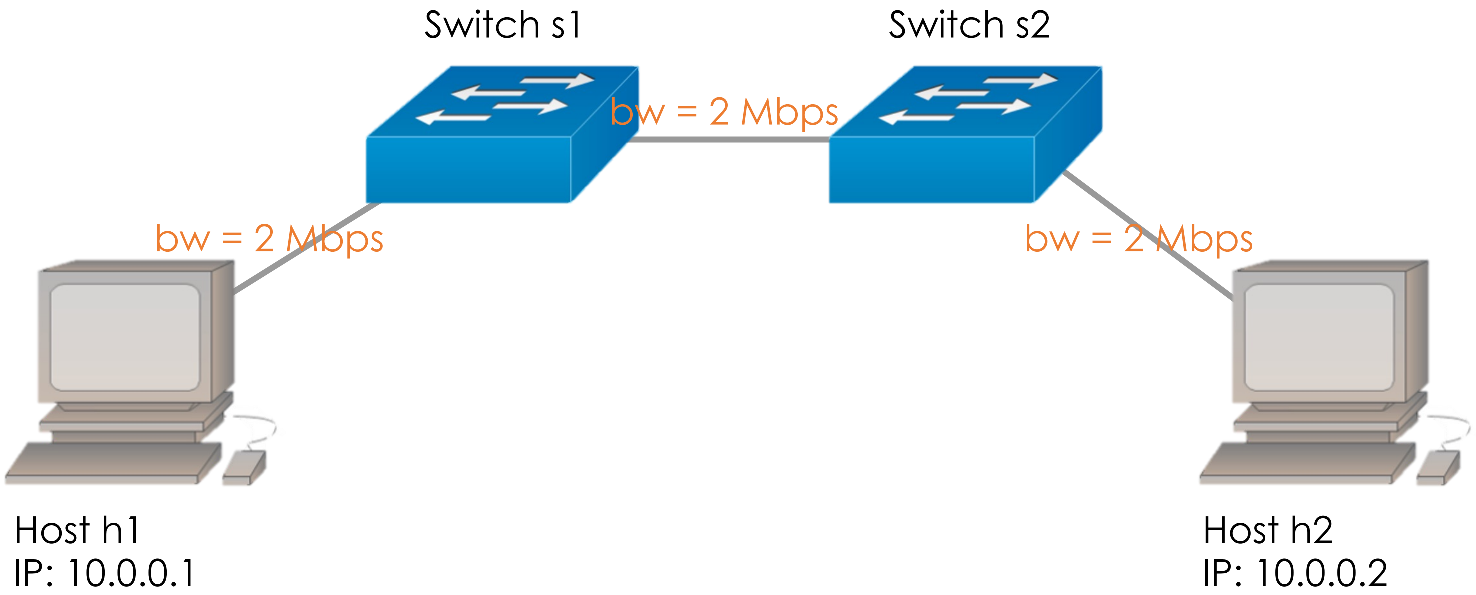
- Result



```
cn2023-lab1@cn2023lab1-VirtualBox:~/Desktop/lab1-jjjjacckk/src$ sudo python2 topo.py
[sudo] password for cn2023-lab1:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(2.00Mbit) (2.00Mbit) (h1, s1) (2.00Mbit) (2.00Mbit) (s1, s2) (2.00Mbit) (2.00Mbit) (s2, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(2.00Mbit) (2.00Mbit) (2.00Mbit) (2.00Mbit)
*** Starting CLI:
mininet> 
```

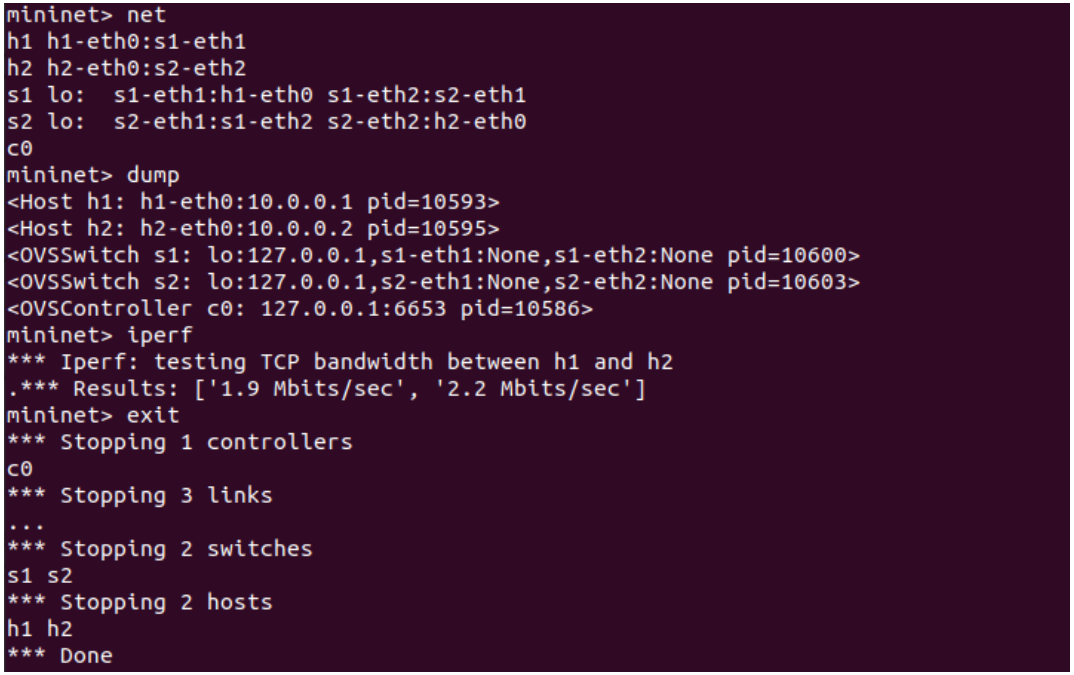
Task 2. Create a Topology (Cont.)

- Example network topology in **topo.py**



Task 2. Create a Topology (Cont.)

- You can try some command in page 8 and page 9, and use "exit" to terminate it

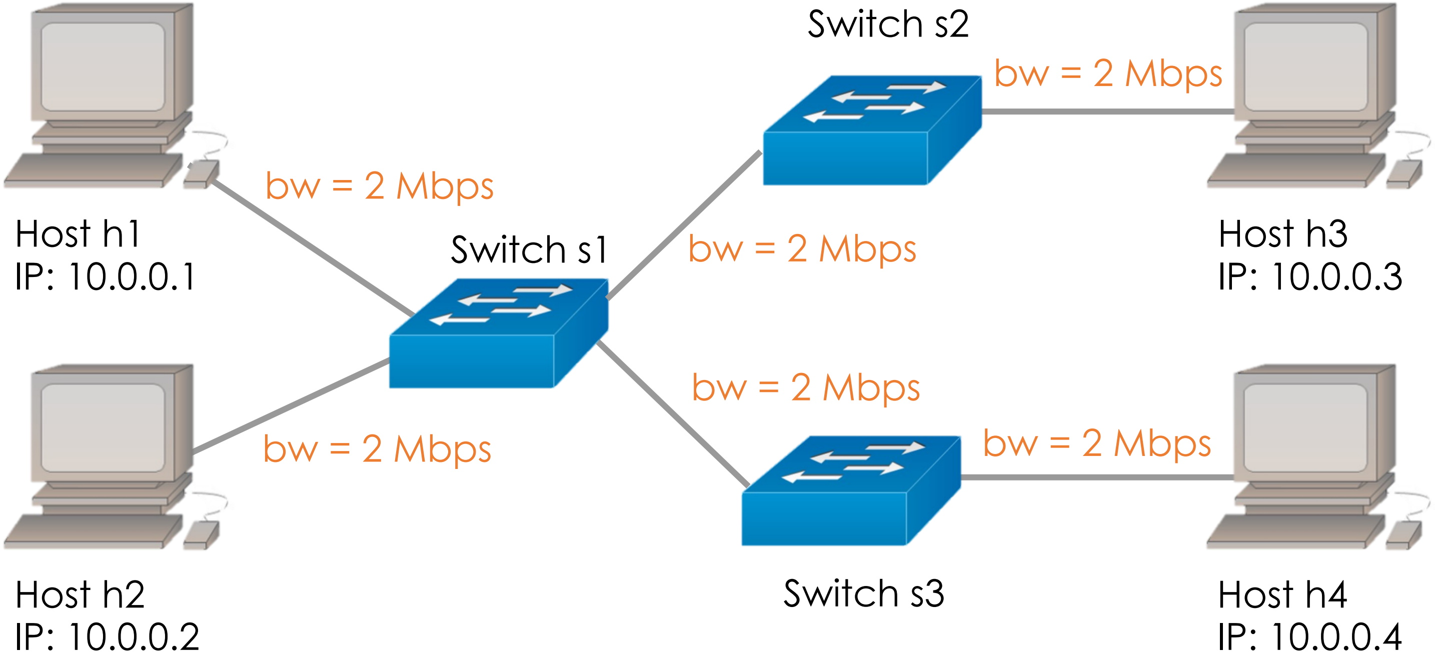


```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth1
s2 lo:  s2-eth1:s1-eth2 s2-eth2:h2-eth0
c0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=10593>
<Host h2: h2-eth0:10.0.0.2 pid=10595>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=10600>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None pid=10603>
<OVSController c0: 127.0.0.1:6653 pid=10586>
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['1.9 Mbits/sec', '2.2 Mbits/sec']
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

Notice: After exiting the mininet, use "`sudo mn -c`" to clean up the environment. Otherwise you may get some error, such as " File Exists Error".

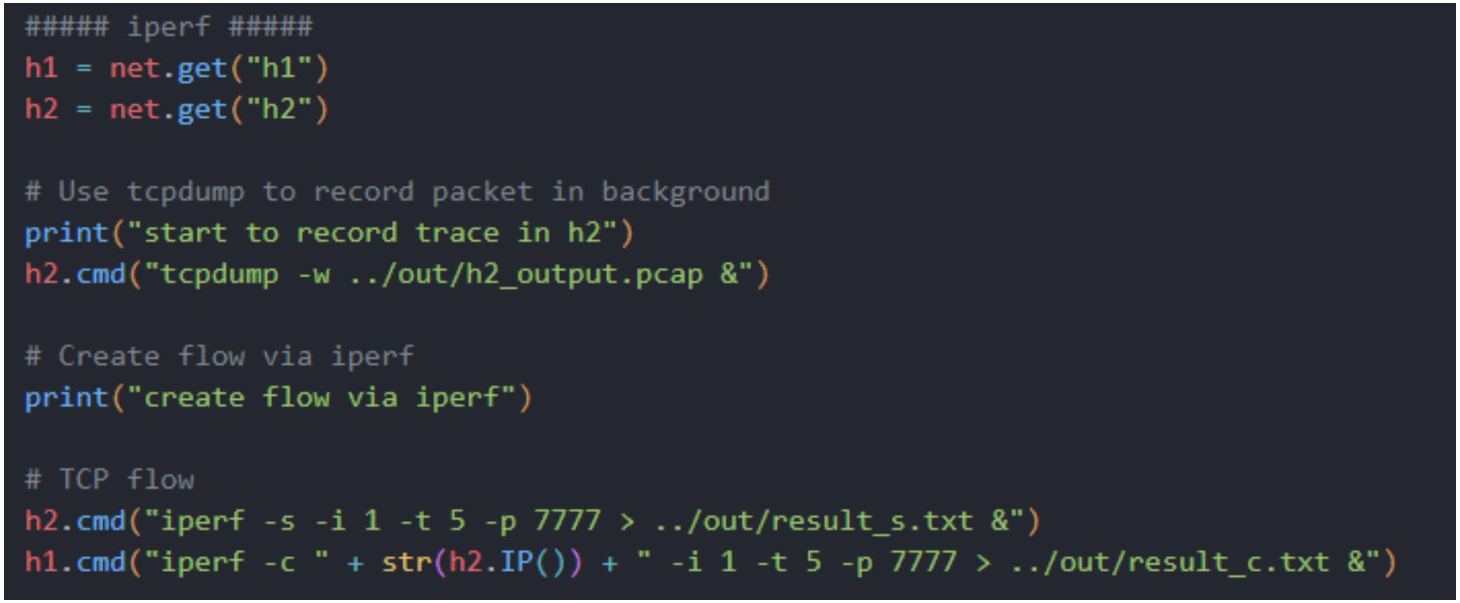
Task 2. Create a Topology (Cont.)

- Modify **topo.py** and create the following new network topology



Task 3. Generate Flows via iPerf

- Uncomment the iPerf code in **topo.py**



```
##### iperf #####
h1 = net.get("h1")
h2 = net.get("h2")

# Use tcpdump to record packet in background
print("start to record trace in h2")
h2.cmd("tcpdump -w ../out/h2_output.pcap &")

# Create flow via iperf
print("create flow via iperf")

# TCP flow
h2.cmd("iperf -s -i 1 -t 5 -p 7777 > ../out/result_s.txt &")
h1.cmd("iperf -c " + str(h2.IP()) + " -i 1 -t 5 -p 7777 > ../out/result_c.txt &")
```

  - It will generate a flow from h1 to h2 and record all packets in pcap file and iPerf data in txt file

Notice: Please wait for 5 seconds after you enter CLI mode to make sure flows are completed

Task 3. Generate Flows via iPerf (Cont.)

- Refer to **topo.py**, write another two Python programs with the same topology in Task 2:
  1. **topo_TCP.py**: generate two TCP flows from h1 to h3 and one TCP flow from h2 to h4. (three flows in total)
     - save the packet data into:

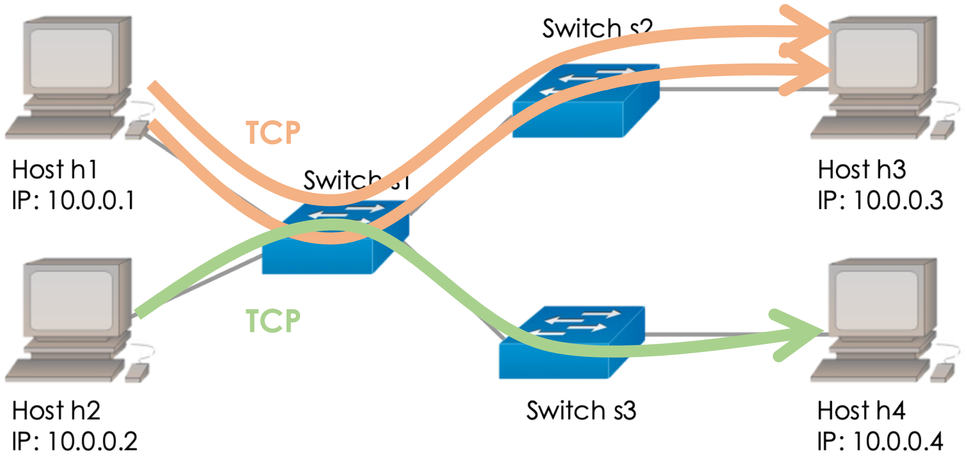
       “../out/TCP_h3.pcap" & "../out/TCP_h4.pcap“
     - save the iPerf data into:
       - "../out/TCP_c_h1_<n>.txt" & "../out/TCP_c_h2.txt"
       - "../out/TCP_s_h3_<n>.txt" & "../out/TCP_s_h4.txt"
  2. **topo_UDP.py**: generate two UDP flows from h1 to h3 and one UDP flow from h2 to h4. (three flows in total)
     - save the packet data into:
       - "../out/UDP_h3.pcap" & "../out/UDP_h4.pcap"
     - save the iPerf data into:
       - "../out/UDP_c_h1_<n>.txt" & "../out/UDP_c_h2.txt"
       - "../out/UDP_s_h3_<n>.txt" & "../out/UDP_s_h4.txt"

Task 3. Generate Flows via iPerf (Cont.)

- topo_TCP.py

output:
1. TCP_c_h1_1.txt
2. TCP_c_h1_2.txt

output:
1. TCP_c_h2.txt



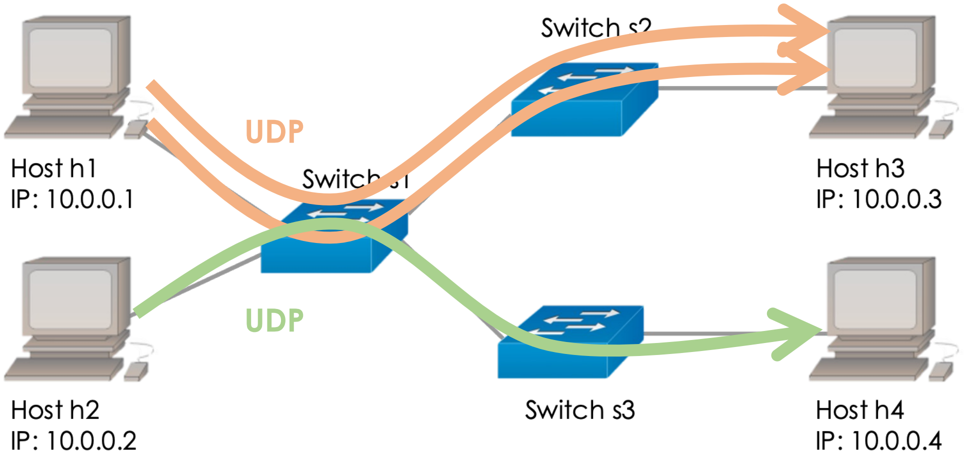
output:
1. TCP_h3.pcap
2. TCP_s_h3_1.txt
3. TCP_s_h3_2.txt

output:
1. TCP_h4.pcap
2. TCP_s_h4.txt

- topo_UDP.py

output:
1. UDP_c_h1_1.txt
2. UDP_c_h1_2.txt

output:
1. UDP_c_h2.txt



output:
1. UDP_h3.pcap
2. UDP_s_h3_1.txt
3. UDP_s_h3_2.txt

output:
1. UDP_h4.pcap
2. UDP_s_h4.txt

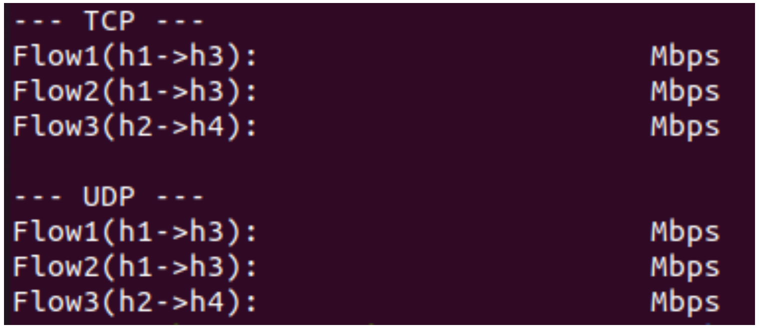
Task 4. Compute the Throughput

- Run parser.py

```
$ sudo python3 parser.py <pcap file path>
# (e.g.) sudo python3 parser.py ../out/h2_output.pcap
```

  - It will parse the pcap file and print some information of packets

- Refer to the parser.py and write a Python program named "**computeRate.py**" to compute throughput of each flow in Task 3
  - Save the screenshot of the result and insert to your report

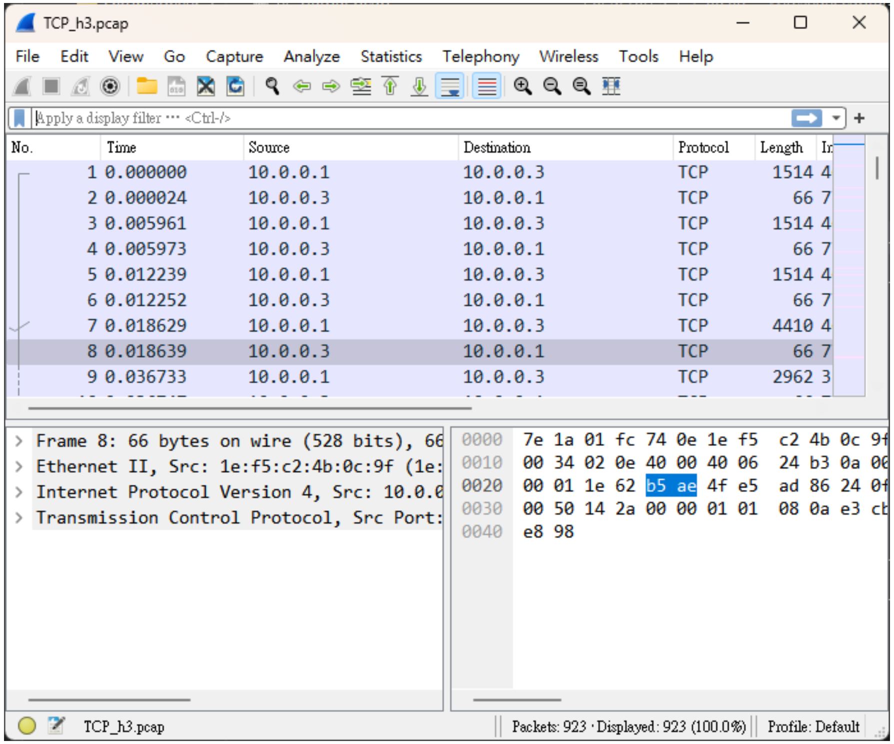


```
--- TCP ---
Flow1(h1->h3):                         Mbps
Flow2(h1->h3):                         Mbps
Flow3(h2->h4):                         Mbps

--- UDP ---
Flow1(h1->h3):                         Mbps
Flow2(h1->h3):                         Mbps
Flow3(h2->h4):                         Mbps
```

Task 5. Check Your Answer

- Step1. Push files to GitHub and close the VM
- Step2. Clone this repository from GitHub to your own computer
- Step3. Open the pcap file using Wireshark



No.	Time	Source	Destination	Protocol	Length	In
1	0.000000	10.0.0.1	10.0.0.3	TCP	1514	4
2	0.000024	10.0.0.3	10.0.0.1	TCP	66	7
3	0.005961	10.0.0.1	10.0.0.3	TCP	1514	4
4	0.005973	10.0.0.3	10.0.0.1	TCP	66	7
5	0.012239	10.0.0.1	10.0.0.3	TCP	1514	4
6	0.012252	10.0.0.3	10.0.0.1	TCP	66	7
7	0.018629	10.0.0.1	10.0.0.3	TCP	4410	4
8	0.018639	10.0.0.3	10.0.0.1	TCP	66	7
9	0.036733	10.0.0.1	10.0.0.3	TCP	2962	3

```
> Frame 8: 66 bytes on wire (528 bits), 66
> Ethernet II, Src: 1e:f5:c2:4b:0c:9f (1e:
> Internet Protocol Version 4, Src: 10.0.0
> Transmission Control Protocol, Src Port:
```

```
0000  7e 1a 01 fc 74 0e 1e f5  c2 4b 0c 9f
0010  00 34 02 0e 40 00 40 06  24 b3 0a 00
0020  00 01 1e 62 b5 ae 4f e5  ad 86 24 0f
0030  00 50 14 2a 00 00 01 01  08 0a e3 cb
0040  e8 98
```

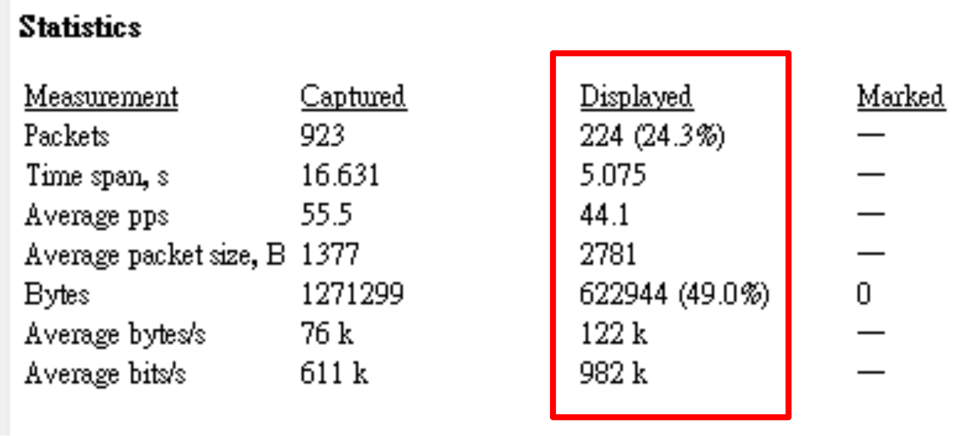
TCP_h3.pcap | Packets: 923 · Displayed: 923 (100.0%) | Profile: Default

Task 5. Check Your Answer (Cont.)

- Step4. Filter the packets
  - Enter filter command on DisplayFilters to filter 6 flows you generate in Task 3
  - Hint: use header info., e.g., IP address or/and port number

Task 5. Check Your Answer (Cont.)

- Step5. Statistic
  - After filtering the flows, click "Statistics" → "Capture File properties"
  - Save the screenshot of Statistics result



Statistics

Measurement	Captured	Displayed	Marked
Packets	923	224 (24.3%)	—
Time span, s	16.631	5.075	—
Average pps	55.5	44.1	—
Average packet size, B	1377	2781	—
Bytes	1271299	622944 (49.0%)	0
Average bytes/s	76 k	122 k	—
Average bits/s	611 k	982 k	—

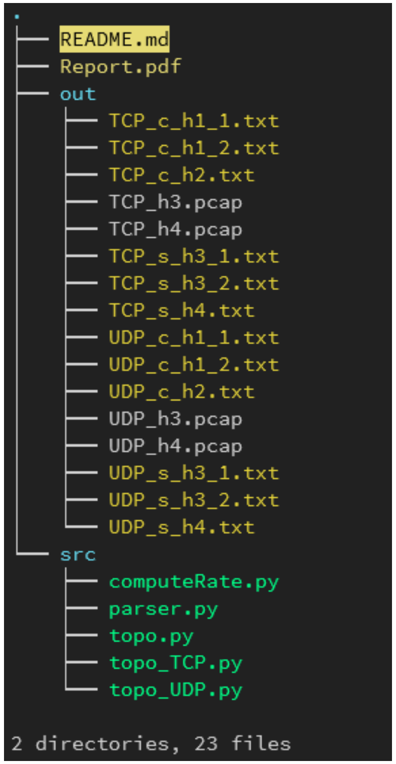
Notice: Insert these screenshots into your report (no need to output any files)

Task 6. Report

- A report in PDF format, contains:
  - Describe each step and how to run your program
  - Describe your observations from the results in this lab
  - Answer the following question in short:
    - What does each iPerf command you used mean?
    - What is your command to filter each flow in Wireshark?
    - Show the results of computeRate.py and statistics of Wireshark
    - Does the throughput match the bottleneck throughput of the path?
    - Do you observe the same throughput from TCP and UDP?
  - Bonus
    - What have you learned from this lab?
    - What difficulty have you met in this lab?

Submission

- **You should write your report in English**
- push all your files and report to your GitHub repository (NYCU-CN2023/Lab1-<GITHUB_ID>)
- Make sure the filename of each file is correct
- File Structure:



```
.
├── README.md
├── Report.pdf
├── out
│   ├── TCP_c_h1_1.txt
│   ├── TCP_c_h1_2.txt
│   ├── TCP_c_h2.txt
│   ├── TCP_h3.pcap
│   ├── TCP_h4.pcap
│   ├── TCP_s_h3_1.txt
│   ├── TCP_s_h3_2.txt
│   ├── TCP_s_h4.txt
│   ├── UDP_c_h1_1.txt
│   ├── UDP_c_h1_2.txt
│   ├── UDP_c_h2.txt
│   ├── UDP_h3.pcap
│   ├── UDP_h4.pcap
│   ├── UDP_s_h3_1.txt
│   ├── UDP_s_h3_2.txt
│   └── UDP_s_h4.txt
└── src
    ├── computeRate.py
    ├── parser.py
    ├── topo.py
    ├── topo_TCP.py
    └── topo_UDP.py

2 directories, 23 files
```

Notice: No need to submit to E3

Grading Policy

- Deadline – **2023.11.09 23:59**
- Grade
  - code correctness - 40%
  - Report - 60%
- Late Policy
  - (Your score) * 0.8^D, where D is the number of days over due
- Cheating Policy
  - Academic integrity: Homework must be your own – cheaters share the score
  - Both the cheaters and the students who aided the cheater equally share the score

Q&A

- If you have any question about Lab1:
  1. Post the question in Lab1 channel
  2. DM TAs for reservation (EC635)

     nycu-nc2023@googlegroups.com

     (Office hour: PM2:00 ~ PM4:00 Mon.)

References

- **Mininet**
  - English
    - Mininet Walkthrough
    - Introduction to Mininet
    - Mininet Python API Reference Manual
    - A Beginner’s Guide to Mininet
  - Chinese
    - GitHub/OSE-Lab - 熟悉如何使用 Mininet
    - 菸酒生的記事本 – Mininet 筆記
    - Hwchiu Learning Note – 手把手打造仿 mininet 網路
    - 阿寬的實驗室 – Mininet 指令介紹
    - Mininet 學習指南

References (Cont.)

- Python 2.7.15 Standard Library
- Python Tutorial - Tutorialspoint
- iPerf3 User Documentation
- Wireshark