Computer Networks
@CS.NCTU

Lab. 1: Network Emulation with Mininet

Instructor: Kate Lin

Deadline: 2020.10.24 0:00 (by Fri. night)

Agenda

- 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 for *Mininet* and use *iPerf* to generate flows and measure the bandwidth in this topology

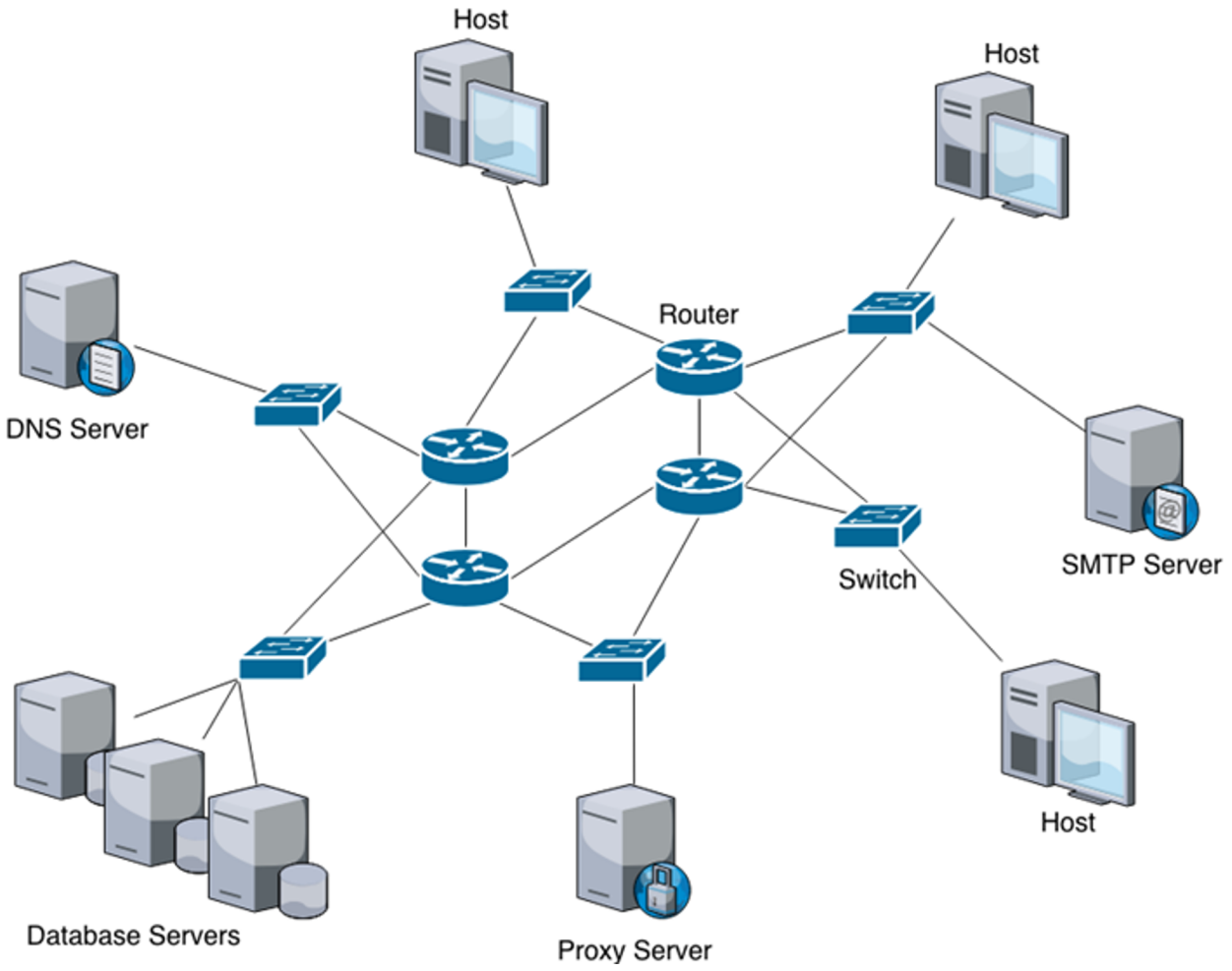
1. Learn how to create a network topology for **Mininet**
2. Learn how to generate flows by using **iPerf** in Mininet
3. Learn how to use **Wireshark** to filter packets and perform analysis

Background

- Network Topology
- Mininet
- iPerf
- Wireshark

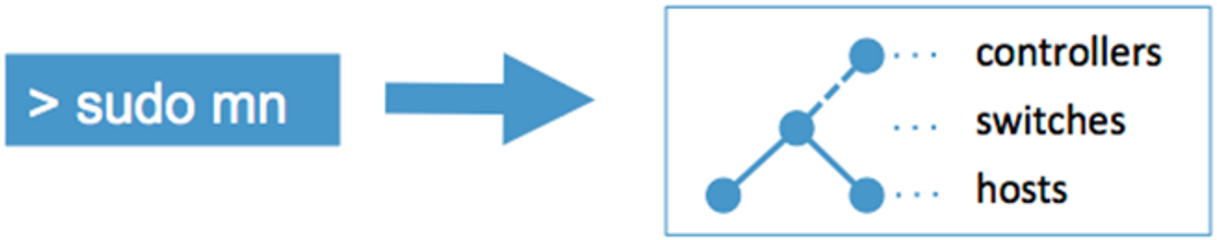
Network Topology

- 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



Notice: We have provided you a container that has installed Mininet (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

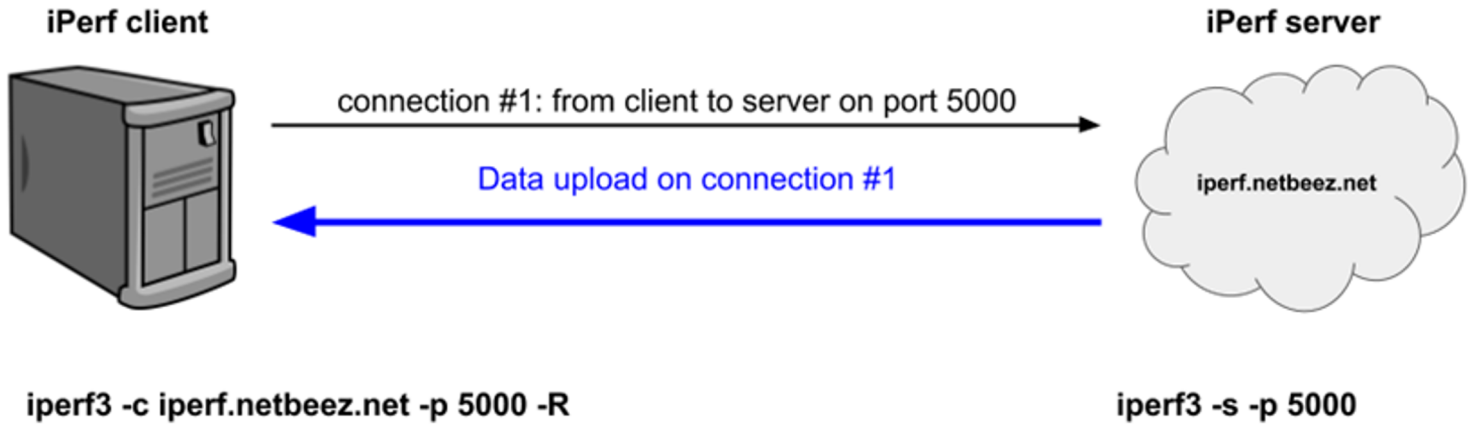
```
mininet> exit
```

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 on IP networks
- Support tuning of various parameters related to timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and IPv6)



iPerf

- iPerf Command line option
  - -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 3.2.7)
    - Ubuntu Linux

      ```
      $ sudo apt-get install wireshark
      ```
  - MobaXterm
    - Windows only
- Step2. Join the **Github Classroom Lab1**
  - Github Classroom Lab1

Task 1. Environment Setup (cont.)

- Step3. Login to your **Docker** container using **SSH**
  - IP address: 140.113.195.69
  - Port: port list
  - Login: root
  - Password: cn2020
- For Windows
  - Open **MobaXterm** -> Session -> SSH
- For MacOS/Linux
  - Open Terminal

```
$ ssh root@140.113.195.69 -p <port>
```

Task 1. Environment Setup (cont.)

- Step4. Download required files from Github

```
$ git clone https://github.com/nctucn/lab1-<GITHUB_ID>.git
```

- Step5. Get and set repository for global options

```
$ cd lab1-<GITHUB_ID>/
$ git config --global user.name "<NAME>"
$ git config --global user.email "<EMAIL>"
```

Task 1. Environment Setup (cont.)

- Step6. Run Mininet for testing
  - After logging to your container, you may meet the following error when running Mininet

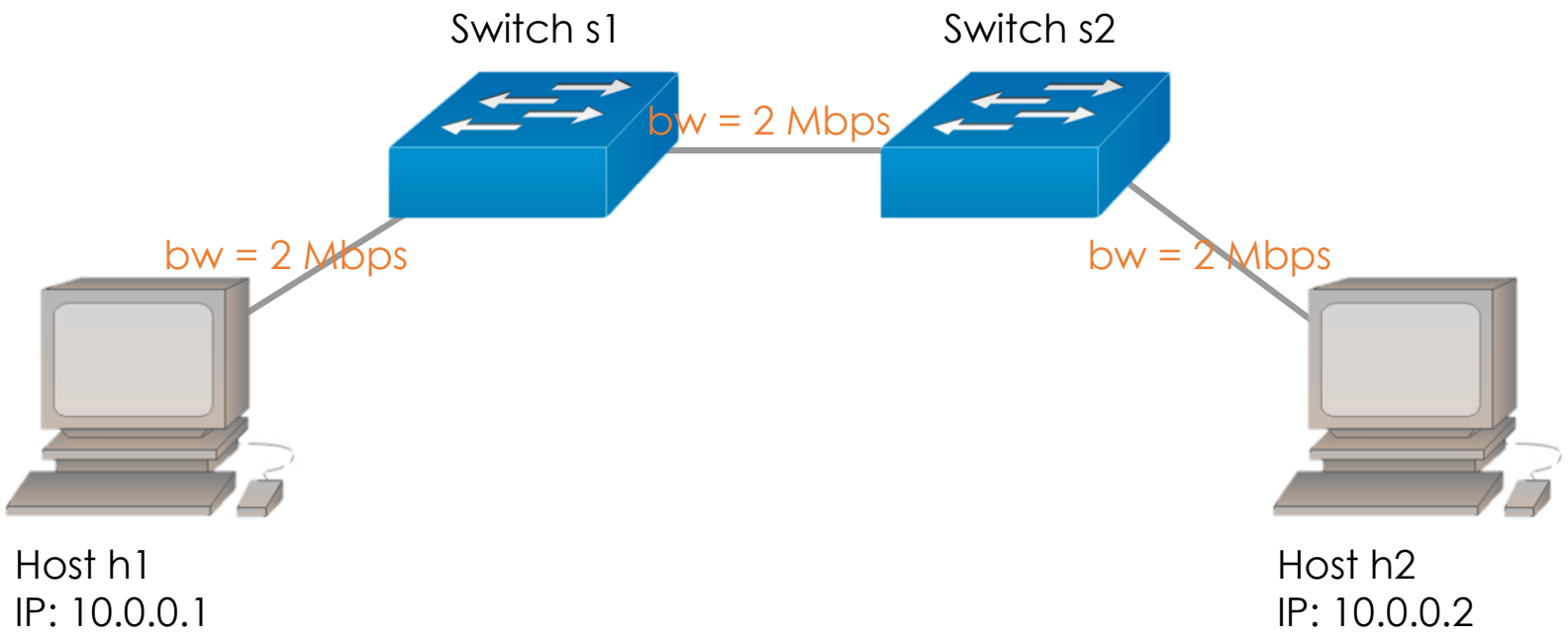
```
# Run Mininet for testing
$ [sudo] mn
......
*** Error connecting to ovs-db with ovs-vsctl
Make sure that Open vSwitch is installed, that ovsdb-
server is running, and that
"ovs-vsctl show" works correctly.
You may wish to try "service openvswitch-switch start".
```

  - Solution

```
$ service openvswitch-switch start
```

Task 2. Create a Topology

- Example network topology in *topo.py*



Task 2. Create a Topology (Cont.)

- Run the example code

```
$ cd ./src/
$ sudo python topo.py
```

Notice: Mininet must run in python2

- Result

```
root@33ceaa30e015:~/Lab1_test/src# sudo python topo.py
*** Error setting resource limits. Mininet's performance may be affected.
*** 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, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(2.00Mbit) (2.00Mbit) (2.00Mbit) (2.00Mbit)
start to record trace in h2
create flow via iperf
*** Starting CLI:
mininet>
```

Task 2. Create a Topology (Cont.)

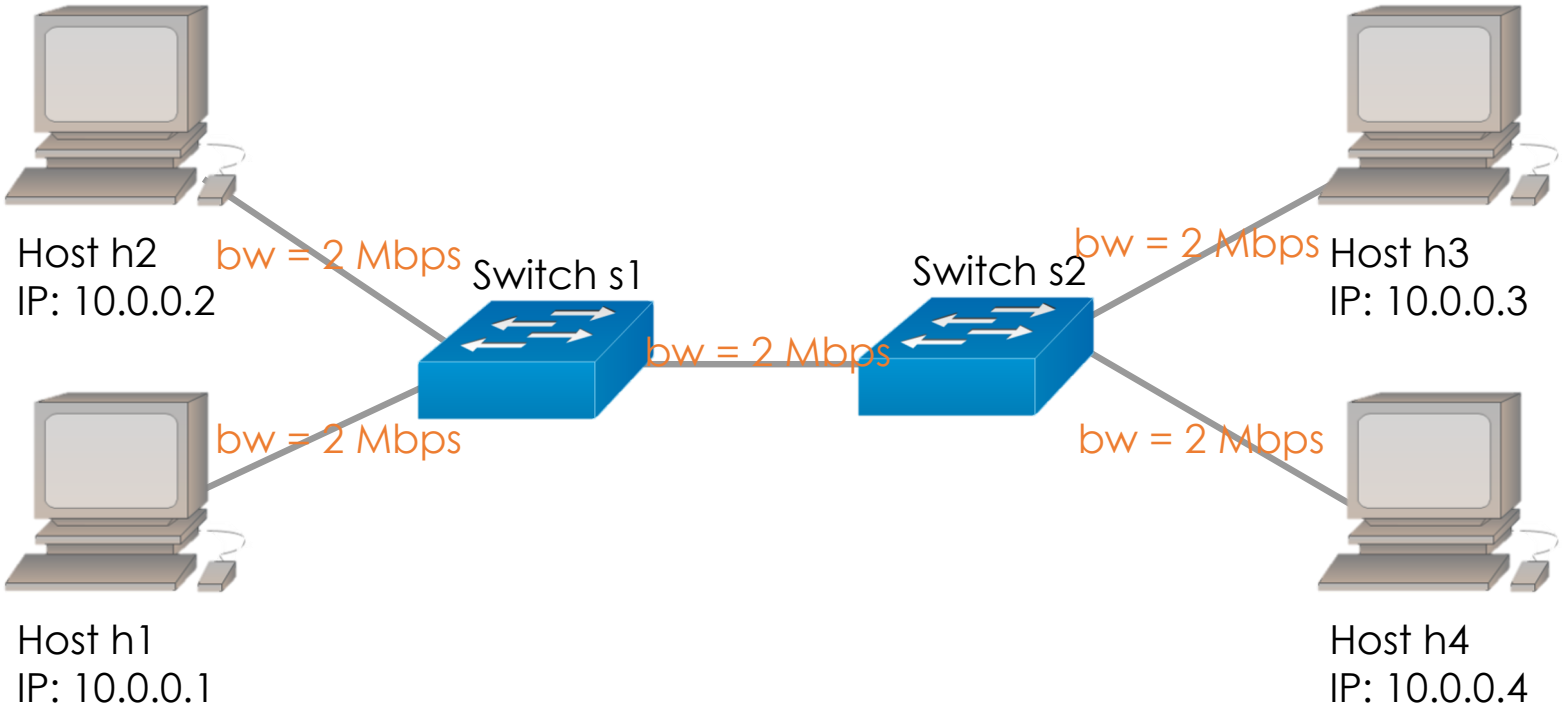
- You can try some command in page 9 and page 10, and use "exit" to terminate it

```
*** Starting CLI:
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> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['1.92 Mbits/sec', '2.18 Mbits/sec']
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

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

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 the example code 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 complete

Task 3. Generate Flow via iPerf (Cont.)

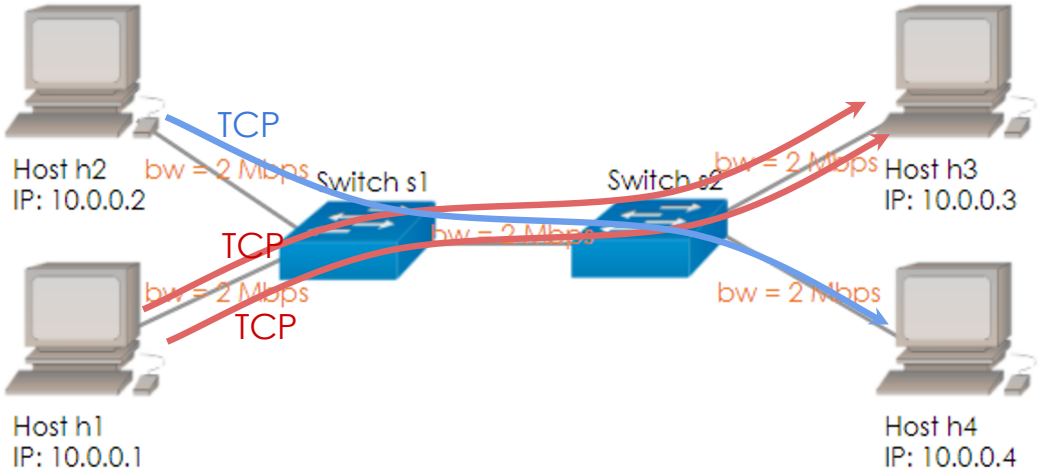
- Write Python programs to generate **two** flows from **h1 to h4** and **one** flow from **h2 to h3** in the topology you create in Task 2
  - topo_TCP.py: create topology and generate TCP flows
    - save the packet data into "../out/TCP_h3.pcap" and "../out/TCP_h4.pcap"
    - save the iPerf data into "../out/TCP_s_h3_<n>.txt", "../out/TCP_s_h4.txt", "../out/TCP_c_h1_<n>.txt" and "../out/TCP_c_h2.txt"
  - topo_UDP.py: create topology and generate UDP flows
    - save the packet data into "../out/UDP_h3.pcap" and "../out/UDP_h4.pcap"
    - save the iPerf data into "../out/UDP_s_h3_<n>.txt", "../out/UDP_s_h4.txt", "../out/UDP_c_h1_<n>.txt" and "../out/UDP_c_h2.txt"

Task 3. Generate Flow via iPerf (Cont.)

- topo_TCP.py

output: TCP_c_h2.txt

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



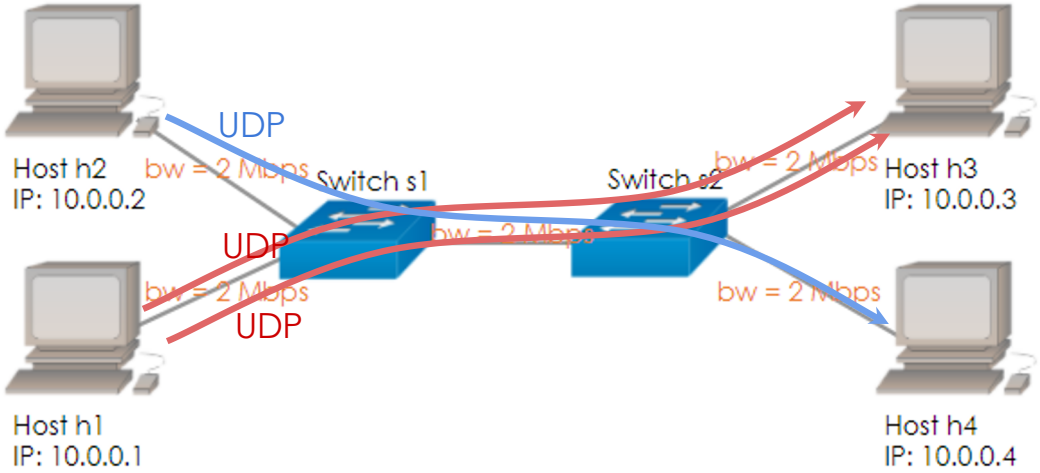
output:
1. TCP_h3.pcap
2. TCP_s_h3_1.txt
3. TCP_s_h3_2.txt

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

- topo_UDP.py

output: UDP_c_h2.txt

output:
1. UDP_c_h1_1.txt
2. UDP_c_h1_2.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

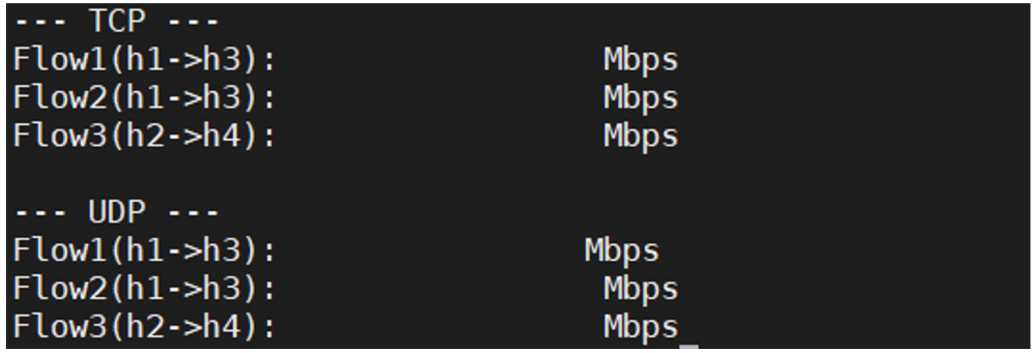
Task 4. Compute Throughput

- Run parser.py

  ```
  $ sudo python parser.py <pcap file path>
  ```

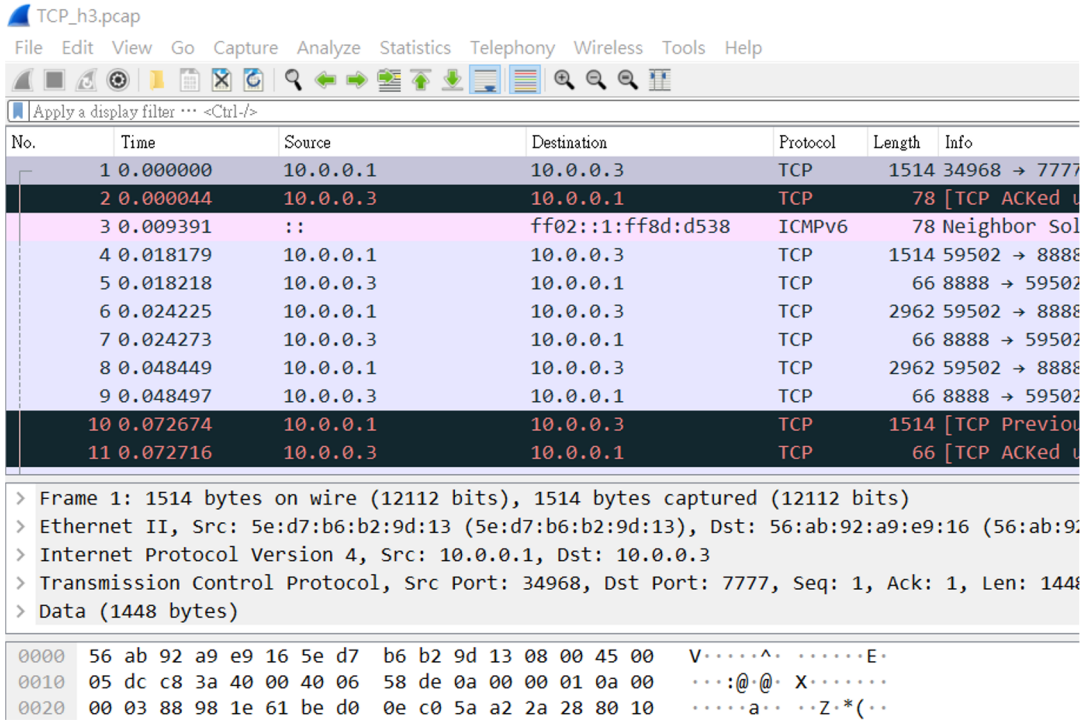
  ```
  ex. sudo python parser.py ../out/h2_output.pcap
  ```

  - It will parse the pcap file and print some information of packets
- Refer to the parser.py and write a Python program named "computRate.py" to compute throughput of each flow in Task 3
  - Save the screenshot of the result and insert to your report



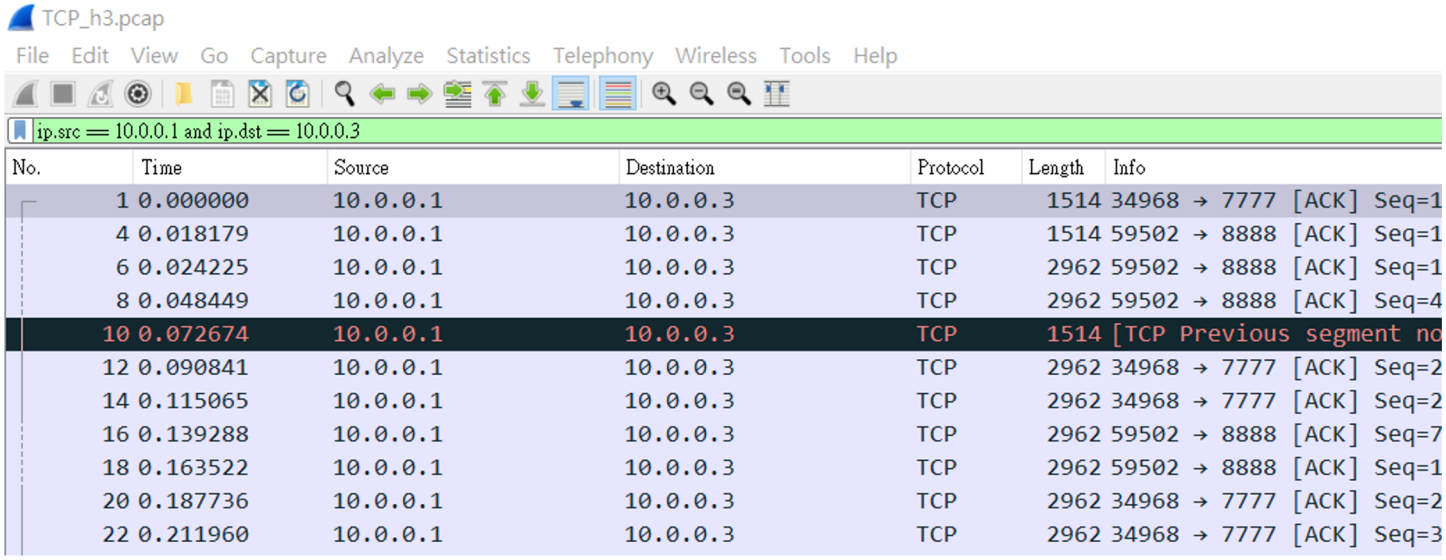
Task 5. Check Your Answer

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



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



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.1	10.0.0.3	TCP	1514	34968 → 7777 [ACK] Seq=1
4	0.018179	10.0.0.1	10.0.0.3	TCP	1514	59502 → 8888 [ACK] Seq=1
6	0.024225	10.0.0.1	10.0.0.3	TCP	2962	59502 → 8888 [ACK] Seq=1
8	0.048449	10.0.0.1	10.0.0.3	TCP	2962	59502 → 8888 [ACK] Seq=4
10	0.072674	10.0.0.1	10.0.0.3	TCP	1514	[TCP Previous segment no
12	0.090841	10.0.0.1	10.0.0.3	TCP	2962	34968 → 7777 [ACK] Seq=2
14	0.115065	10.0.0.1	10.0.0.3	TCP	2962	34968 → 7777 [ACK] Seq=2
16	0.139288	10.0.0.1	10.0.0.3	TCP	2962	59502 → 8888 [ACK] Seq=7
18	0.163522	10.0.0.1	10.0.0.3	TCP	2962	59502 → 8888 [ACK] Seq=1
20	0.187736	10.0.0.1	10.0.0.3	TCP	2962	34968 → 7777 [ACK] Seq=2
22	0.211960	10.0.0.1	10.0.0.3	TCP	2962	34968 → 7777 [ACK] Seq=3

Task 5. Check Your Answer (Cont.)

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

Statistics

Measurement	Captured	Displayed	Marked
Packets	699	191 (27.3%)	—
Time span, s	9.546	5.830	—
Average pps	73.2	32.8	—
Average packet size, B	1426	2848	—
Bytes	997002	544022 (54.6%)	0
Average bytes/s	104 k	93 k	—
Average bits/s	835 k	746 k	—

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

Task 5. 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
  - Bonus
    - What you have learned from this lab?
    - What difficulty you have met in this lab?

Notice: You can write your report in English or Chinese

Submission

- push all your files and report to Github (nctucn/Lab-<GITHUB_ID> repository)
- Make sure the filename of each file is correct
- File Structure:

```
.
|-- 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
```

Notice: No need to submit to new E3

Grading Policy

- Deadline – **2020.10.24 0:00** (by Friday night)
- 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

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