Project 2

OpenFlow Protocol Observation & Flow Rule installation

Deadline 2022/10/5(WED) 23:59

Email sdnta@win.cs.nctu.edu.tw

New update for Lab1 PDF

- source ~/.bashrc

```
Finished the installation of Open vSwitch.
************************************************
* The environment setup finished successfully! *
************************************************
demo@SDN-NFV:~/Desktop$
```

- Run the command

```
demo@SDN-NFV:~$ source ./bashrc
```

- 30GB

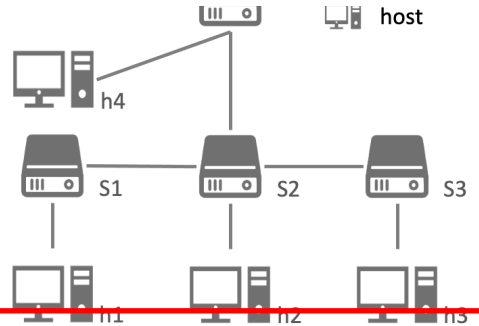
Experiment Environment

- Ubuntu Desktop 22.04
- Minimum Hardware Requirements:
  - 2 Cores (CPUs)
  - 8GB RAM
  - 30GB HDD

- HW1
  - 192.168.0.0/27
  - netmask 255.255.255.224

Host	IP Address
h1	192.168.0.1
h2	192.168.0.2
	…

- **Statically assign IP addresses with Python** and hand in the Python script you've edited
- Start mn with your Python script and take screenshots with command "dump" and "ifconfig" for all host.



- Run your Python script and use command "pingall". Then take a screenshot of topology on GUI.

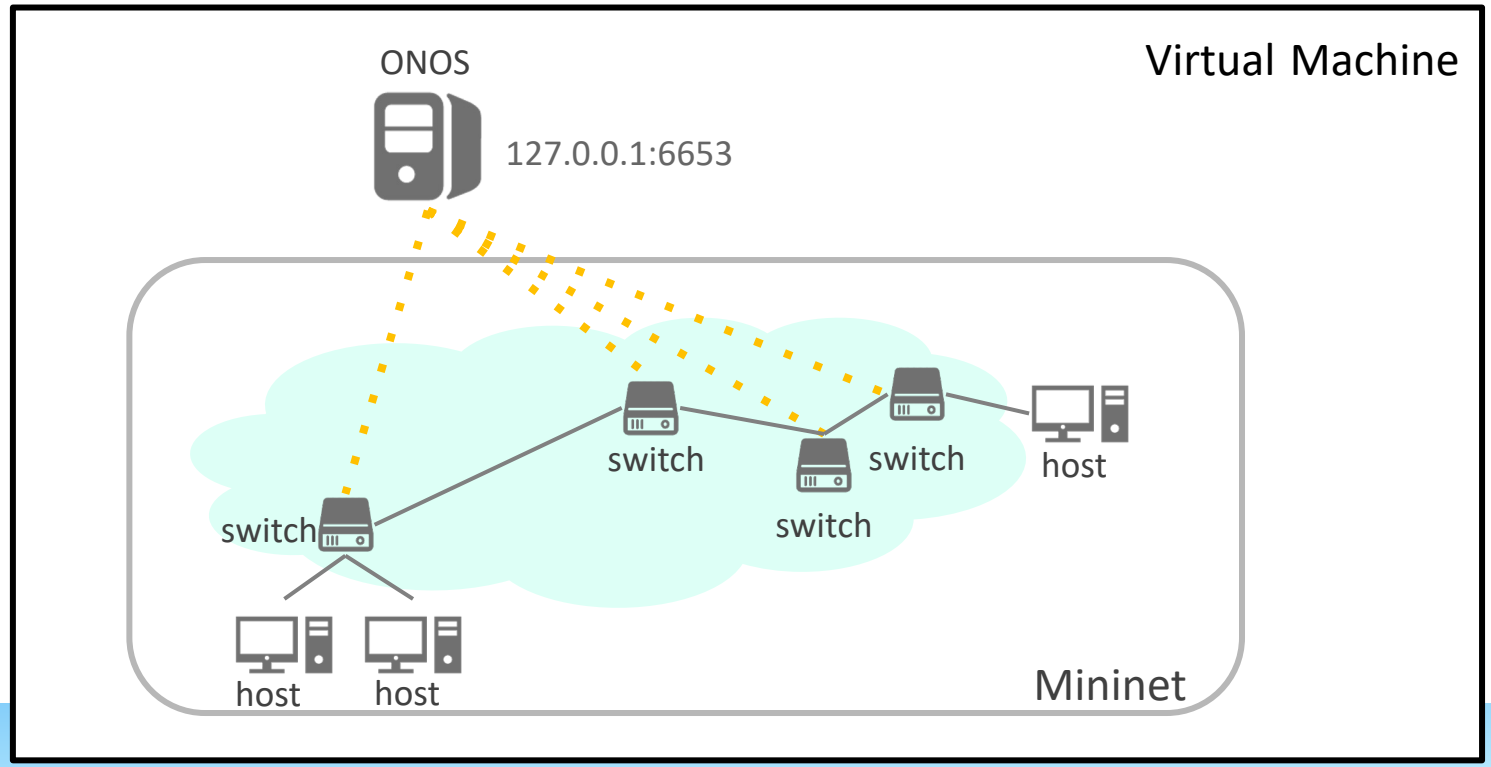
- MacBook M1 chip

Outline

- OpenFlow Messages
  - Monitor traffic between ONOS & Switches
  - OpenFlow Message Observation
- Install/ Delete Flow Rules
  - Rest, JSON file, and Curl introduction
  - ONOS and Topology Setup
  - Method 1: via Command “curl”
  - Method 2: via ONOS Web GUI
- Project 2 Requirements
  - Part 1: Answer Questions
  - Part 2: Install Flow Rules
  - Part 3: Create Broadcast Storm
  - Part 4: Trace ReactiveForwarding

OpenFlow Protocol

- OpenFlow is a Protocol for Software Defined Networks
- ONOS SDN controller uses OpenFlow messages to communicate with OVS switches.
  - Hello, Packet-in/out messages, Flow Install/Remove messages, etc.



Wireshark Installation

- Wireshark is an open-source and widely-used network packet analyzer
  - Can capture packets on any specified interface
- Installation steps:
  1. Download package information

```
$ sudo apt update          # update all packages information
```

  2. Install Wireshark

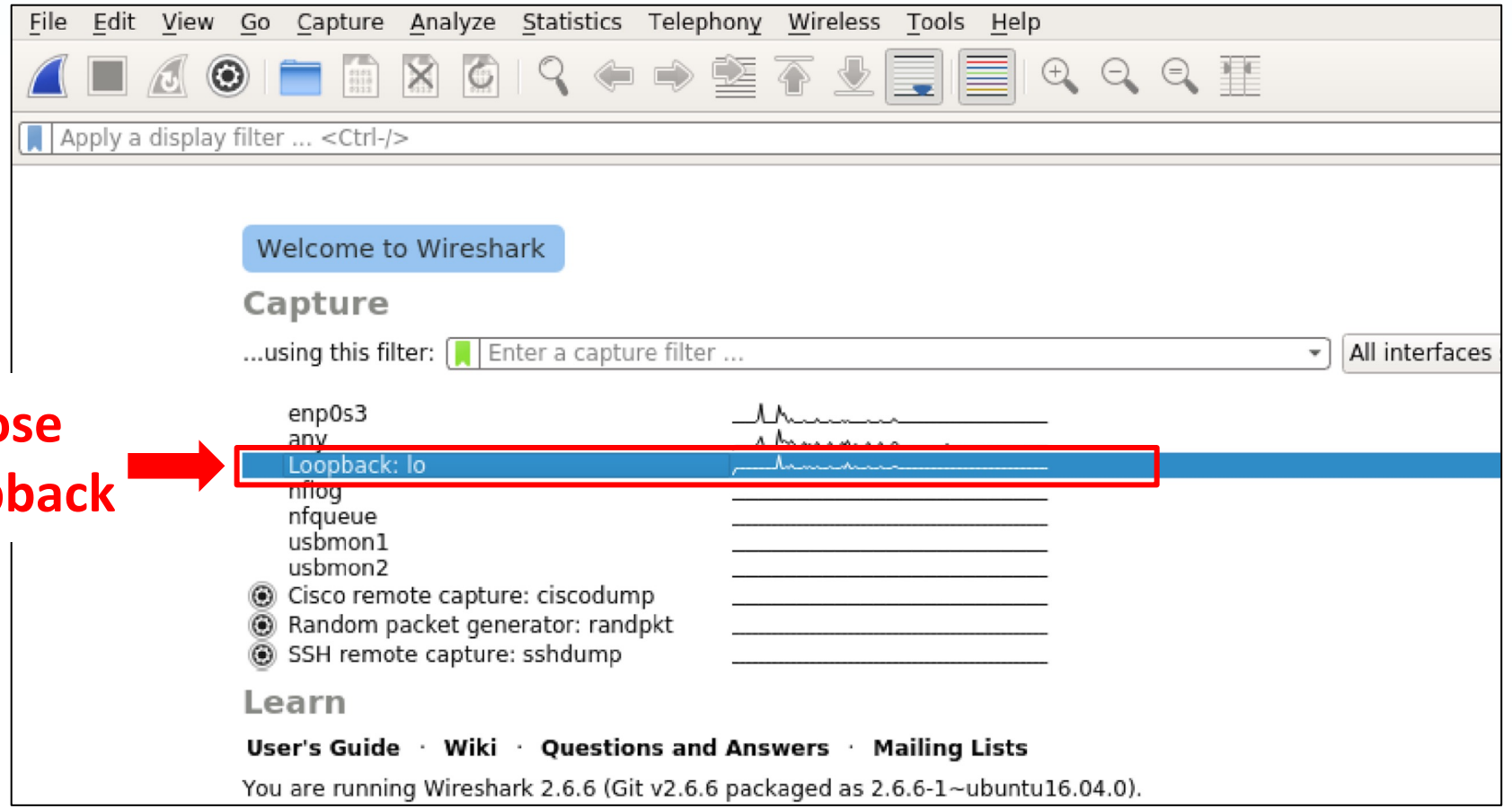
```
$ sudo apt install wireshark
```

- Start Wireshark

```
$ sudo wireshark
```

Capture Packets in Wireshark

- Both ONOS and Mininet run locally in VM
  - We can capture packets on the Loopback (lo) interface

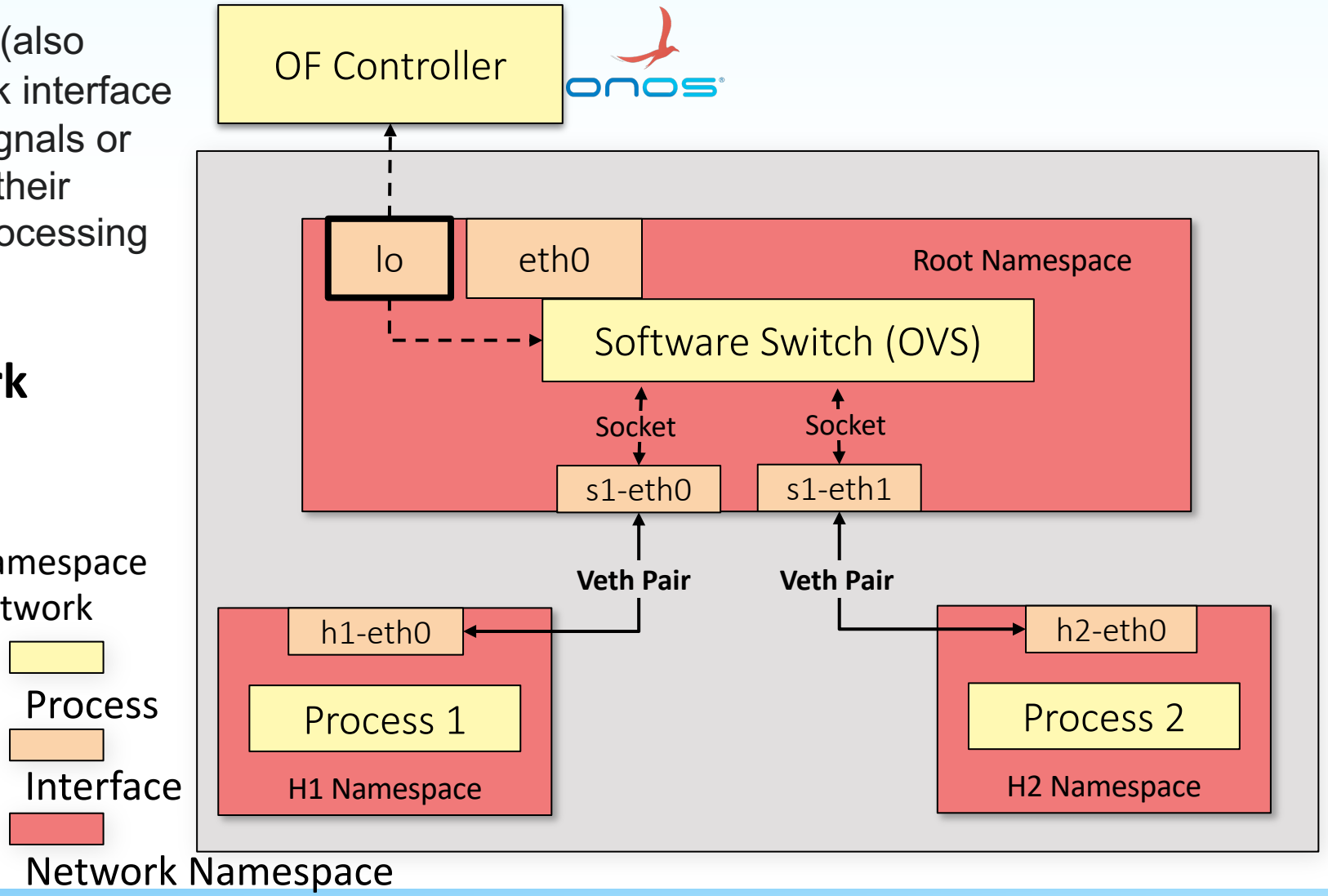


Choose Loopback

Mininet and Network Namespace

- **Loopback Interface** (also written **loop-back**) Loopback interface is the routing of electronic signals or digital data streams back to their source without intentional processing or modification.

- Mininet utilizes **network namespace** to emulate networks
  - OVS runs in root network namespace
  - Each host runs in its own network namespace



Sending OpenFlow Messages

1. Start ONOS
2. Activate ReactiveForwarding

```
onos> apps -a -s                    # (optional) check activated application
onos> app activate fwd              # activate ReactiveForwarding
```

3. Start Mininet with default topology

```
$ sudo mn --controller=remote,127.0.0.1:6653 --switch=ovs,protocols=OpenFlow14
```

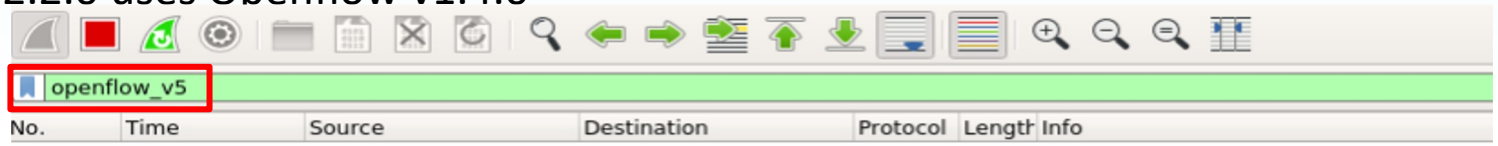
4. H1 ping H2 in Mininet

```
mininet> h1 ping h2 -c 5            # send five ICMP echo_request packets
```

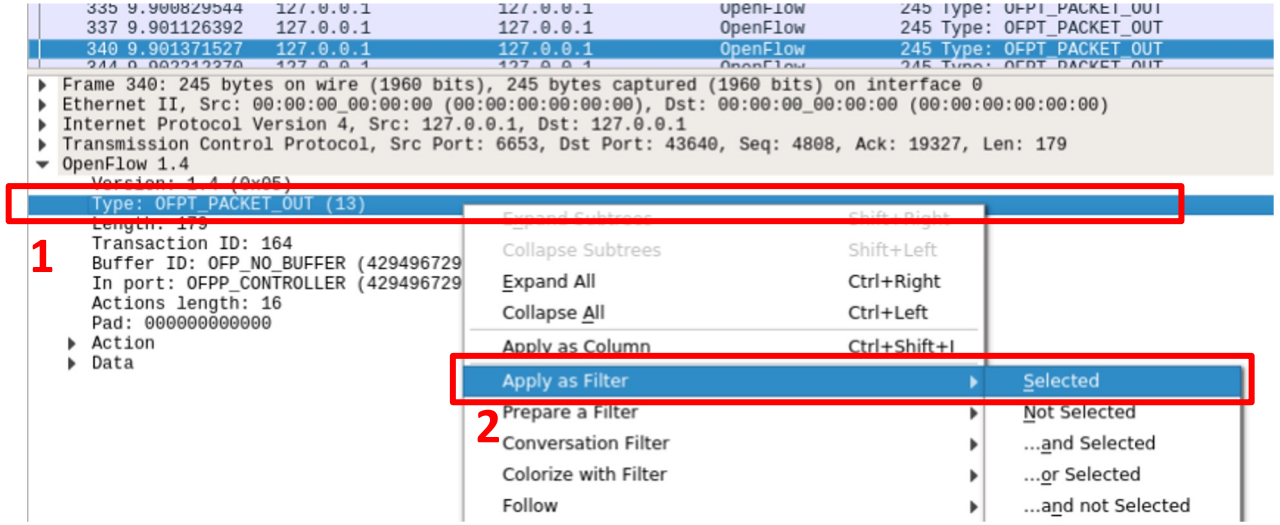
5. Exit Mininet and stop capturing packets in Wireshark when ping terminates
6. Observe captured OpenFlow packets

Filter in Wireshark

- Use keyword “openflow_v5” to filter **OpenFlow v1.4.0** packets
  - ONOS v2.2.0 uses Openflow v1.4.0



- Alternatively, apply filter in the following steps:
  1. Right click on the packet header field which you want to apply as filter
  2. Choose “Apply as Filter” and click “Selected”
  3. Wireshark will immediately filter out all the relevant packets

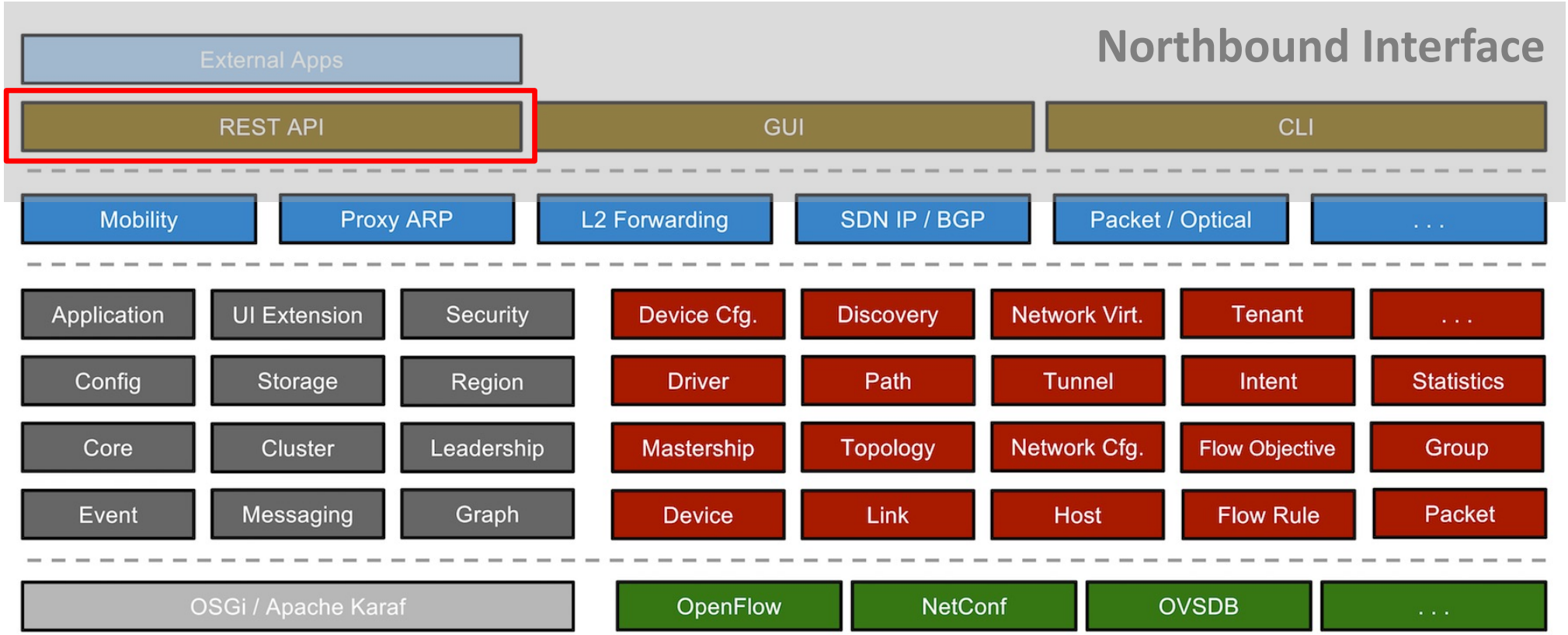


Outline

- OpenFlow Messages
- Install/Delete Flow Rules
  - Rest, JSON file, and Curl introduction
  - ONOS and Topology Setup
  - Method 1: via Command “curl”
  - Method 2: via ONOS Web GUI
- Project 2 Requirements

ONOS Northbound Interface

- Northbound Interface of ONOS is the interface that interact with programmers
- REST is a **software architectural style** for creating Web services
- We will use the REST API to install/delete flow rules



Create a JSON file of flow rules

- What is a JSON ?
  - JSON file is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of **attribute–value pairs** and arrays (or other serializable values)
- Example: JSON file for a flow rule
  - flows1.json

Hint: The priority of this flow rule MUST be higher than that of initially installed flow rule(40000), but not greater than 65535.

- Flow Rule Criteria & Instructions

flows1.json

```
{
    "priority": 50000,
    "timeout": 0,
    "isPermanent": true,
    "selector": {
        "criteria": [
            {
                "type": "IN_PORT",
                "port": 1
            }
        ]
    },
    "treatment": {
        "instructions": [
            {
                "type": "OUTPUT",
                "port": 2
            }
        ]
    }
}
```

JSON File: Match Fields

flows1.json

```
{
    "priority": 50000,
    "timeout": 0,
    "isPermanent": true,
    "selector": {
        "criteria": [
            {
                "type": "IN_PORT",
                "port": 1
            }
        ]
    },
    "treatment": {
        "instructions": [
            {
                "type": "OUTPUT",
                "port": 2
            }
        ]
    }
}
```

- ***Match fields may have dependency; please refer to OpenFlow spec v1.5.1.***

```
"selector": {
    "criteria": [
        {
            "type": "IN_PORT",
            "port": 1
        },
        { … },
        …
    ]
},
```

JSON File: Actions

flows1.json

```
{
    "priority": 50000,
    "timeout": 0,
    "isPermanent": true,
    "selector": {
        "criteria": [
            {
                "type": "IN_PORT",
                "port": 1
            }
        ]
    },
    "treatment": {
        "instructions": [
            {
                "type": "OUTPUT",
                "port": 2
            }
        ]
    }
}
```

```
"treatment": {
    "instructions": [
        {
            "type": "OUTPUT",
            "port": 2
        },
        { … },
        …
    ]
}
```

Curl– Command Tool For Transferring Data

- Command format

```
curl [options] [URL...]
```

- Transferring data with URL

```
$ curl -u <user:password> -X <method-type> -H <header> -d <data> [URL...]
                # option "-X" specifies a HTTP request method
                # option “-H” includes extra header in the HTTP request
                # option “-d” sends specified data in a POST request
                # URL (Uniform Resource Locator)
```

  - "<data>" can be a file name with prefix “@”

```
$ curl -u <user:password> -X <method-type> -H <header> -d @<filename> [URL...]
```

- “request methods” in HTTP
- Manpage for command "curl”

Outline

- OpenFlow Messages
- Install/ Delete Flow Rules
  - Rest, JSON file, and Curl introduction
  - ONOS and Topology Setup
  - Method 1: via Command “curl”
  - Method 2: via ONOS Web GUI
- Project 2 Requirements

ONOS & Topology Setup

1. Restart ONOS
   1. <ctrl+c> in the ONOS log panel to shutdown the ONOS instance
   2. Start ONOS

```
demo@SDN-NFV:~/onos$ ok clean
   # ok is an alias of command “bazel run onos-local -- ”
```

2. Deactivate ReactiveForwarding APP

```
onos> app deactivate fwd    # deactivate ReactiveForwarding
```

3. Start Mininet with default (minimal) topology

```
$ sudo mn --controller=remote,127.0.0.1:6653 --switch=ovs,protocols=OpenFlow14
```

4. Make sure that two hosts CAN NOT ping each other

```
mininet> h1 ping h2
```

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
```

Outline

- OpenFlow Messages
- Install/ Delete Flow Rules
  - Rest, JSON file, and Curl introduction
  - ONOS and Topology Setup
  - Method 1: via Command “curl”
  - Method 2: via ONOS Web GUI
- Project 2 Requirements

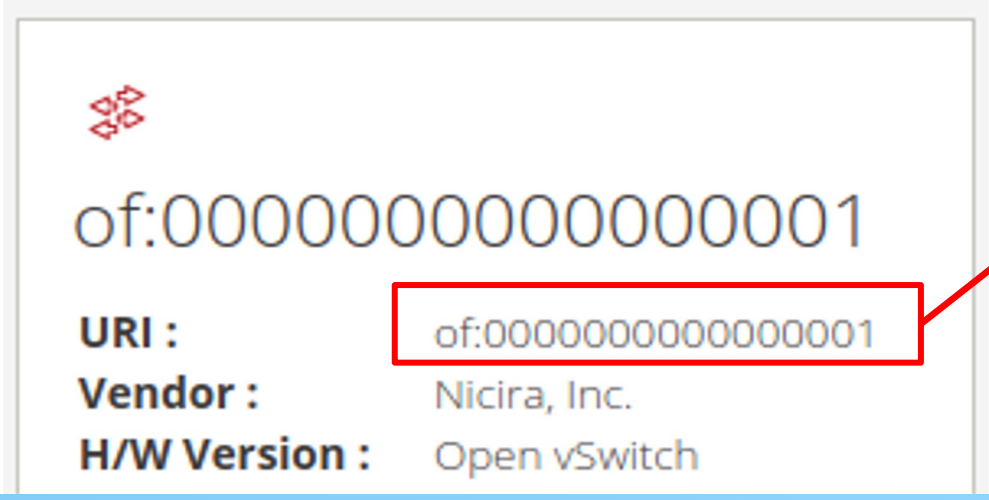
Upload JSON File to ONOS

- Install flow rules on ONOS with JSON file (**flows1.json**)

```
$ curl -u onos:rocks \
> -X POST \
> -H 'Content-Type: application/json' \
> -d @flows1.json \
> 'http://localhost:8181/onos/v1/flows/of:0000000000000001'
```

```
#Recall command from p.15
$ curl -u <user:password>
-X <method-type>
-H <header>
-d @<filename>
[URL...]
```

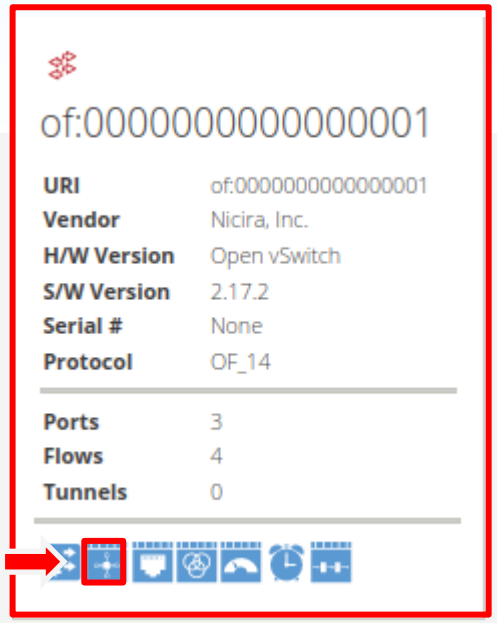
Device ID



- DeviceID MUST be the URI shown in the ONOS web GUI
- DeviceID is set by either ONOS or user specified topology file (i.e. *.py)

Check whether the flow rule is installed (1/2)

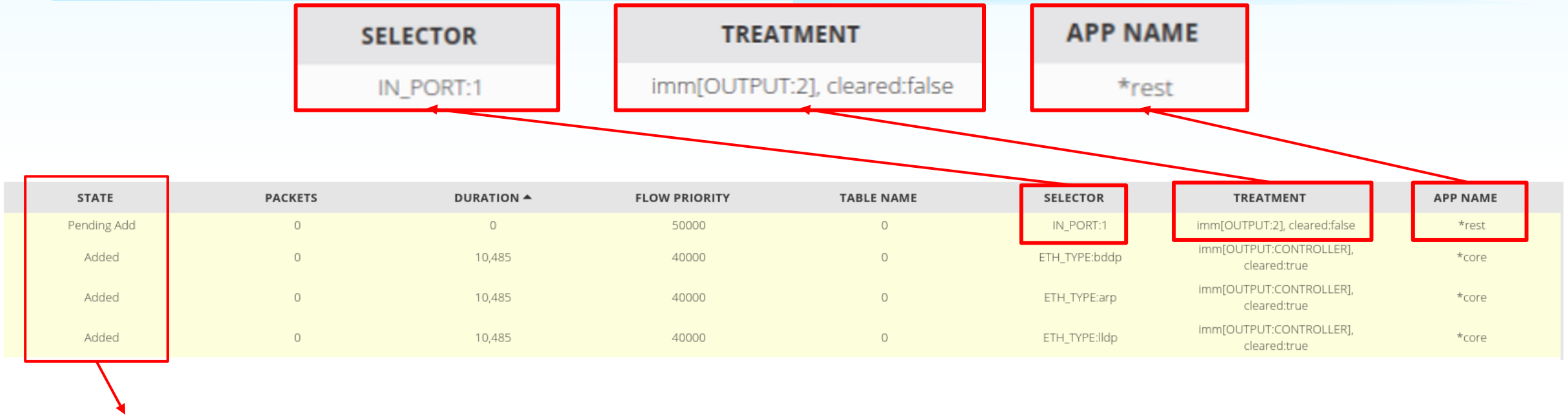
1. Go to ONOS web GUI (http://localhost:8181/onos/ui)
2. Left click on  . Then, the panel of switch info will pop out
3. Left click on  .



Flow rules in switch

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	36	50000	0	IN_PORT:1	imm[OUTPUT:2], cleared:false	*rest
Added	0	960	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	960	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	12	960	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core

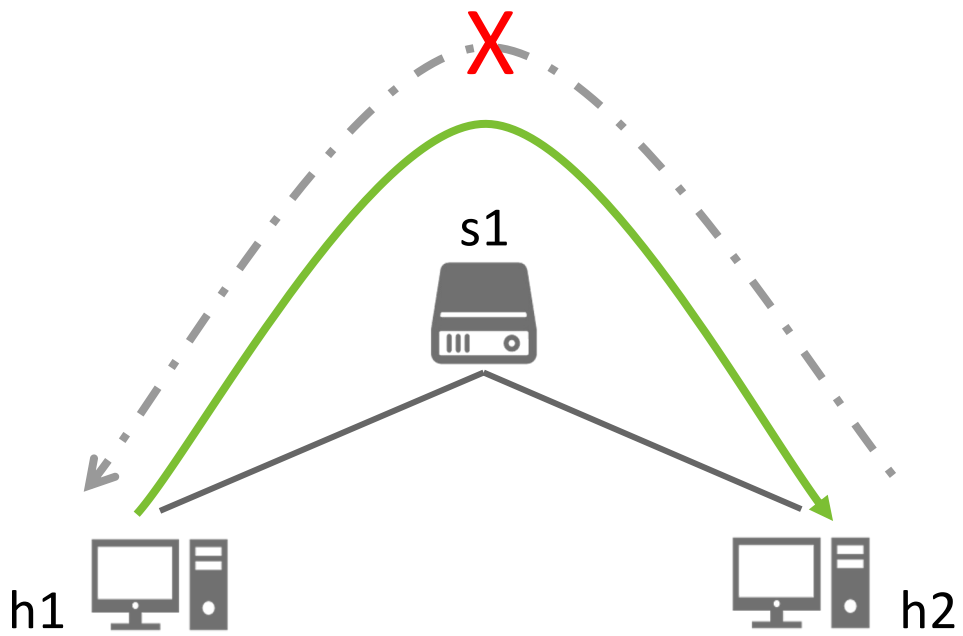
Check whether the flow rule is installed (2/2)



- Flow Rule States:
  - **PENDING_ADD—** ONOS has received a request from the application to install the flow rule, but that flow has NOT yet been observed on the device.
  - **ADDED—** Once the flow rule subsystem observes the flow on the device it will transition to this state.

Why Hosts Still Can't Ping Each Other?

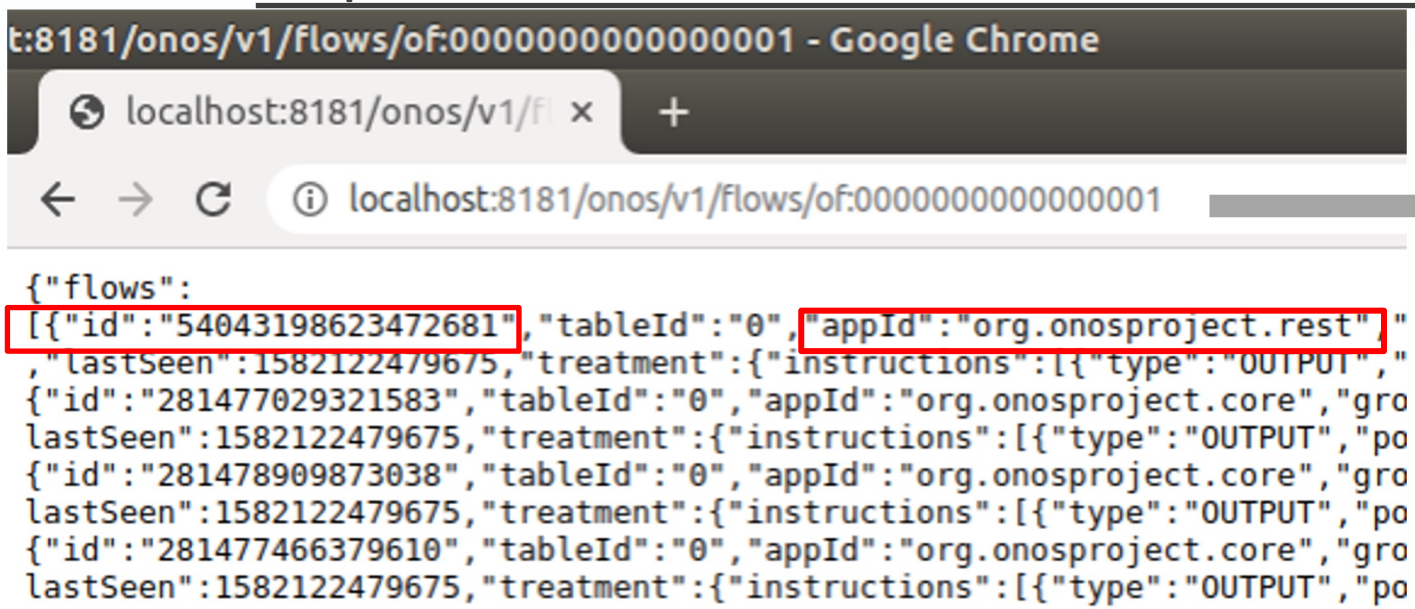
- Because we have **only** installed a flow rule for one direction
  - s1 can forward packets from h1 to h2
  - But, s1 CANNOT forward packets from h2 to h1
    - By default, s1 drops a packet if the packet does not match any flow rule (i.e. **table-miss**)



Delete Flow Rules (1/2)

- Use URL to find the **ID** of particular flow rules **in switch**

  Ex. http://localhost:8181/onos/v1/flows/of:0000000000000001



- ID of the flow we just added is 54043198623472681

- Alternatively, we could use “curl” to get flow information

```
$ curl -u onos:rocks -X GET -H ‘Accept: application/json’ \
> ‘http://localhost:8181/onos/v1/flows/of:0000000000000001’
```

Delete Flow Rules (2/2)

- Then, delete the flow rule with flowID 5404319862347268l

```
$ curl -u onos:rocks \
> -X DELETE \
> -H 'Accept: application/json' \
> 'http://localhost:8181/onos/v1/flows/of:0000000000000001\
> /5404319862347268l'
```

Device ID

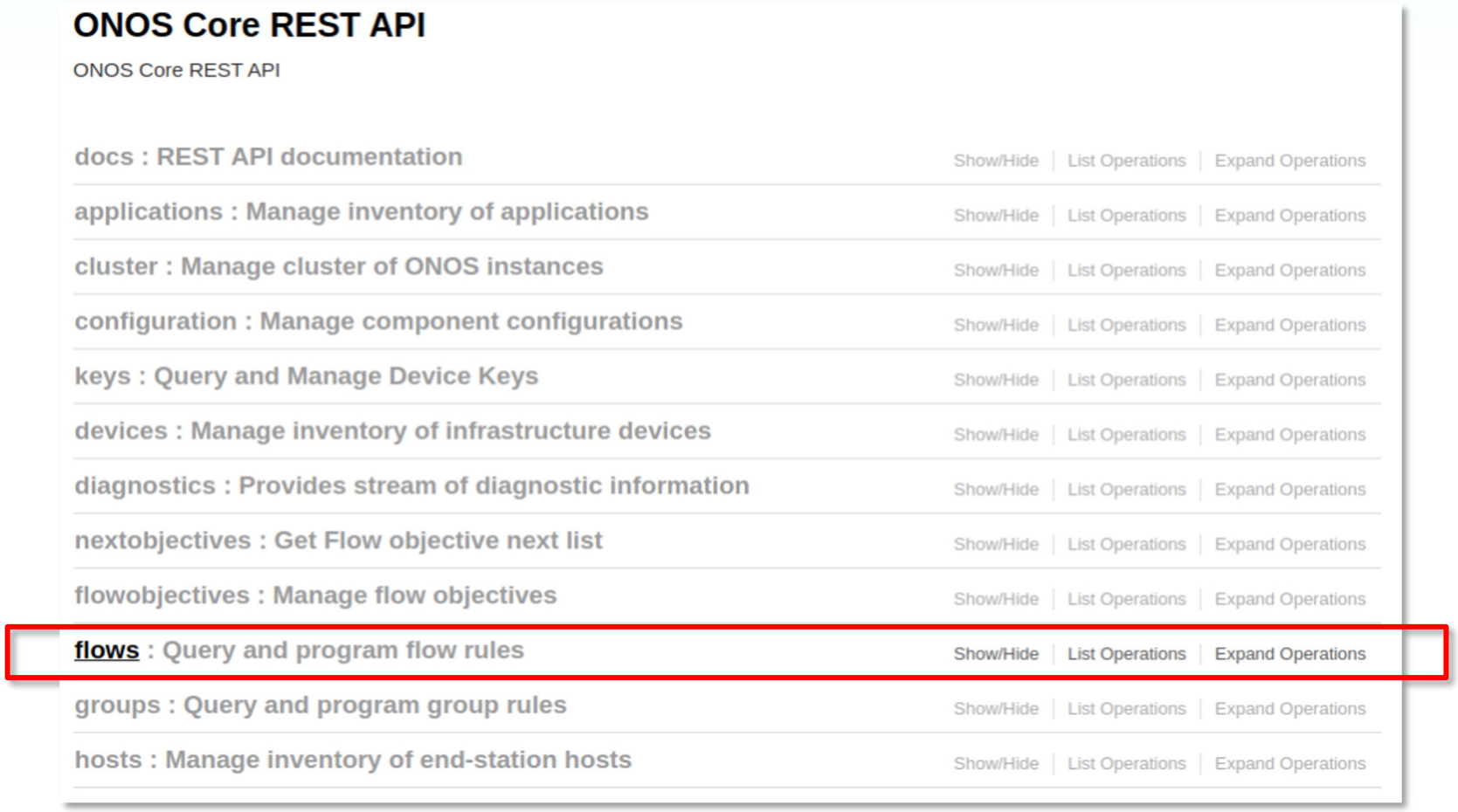
Flow ID

Outline

- OpenFlow Messages
- Install/ Delete Flow Rules
  - Rest, JSON file, and Curl introduction
  - ONOS and Topology Setup
  - Method 1: via Command “curl”
  - Method 2: via ONOS Web GUI
- Project 2 Requirements

REST API on ONOS Web GUI

- Browse http://localhost:8181/onos/v1/docs

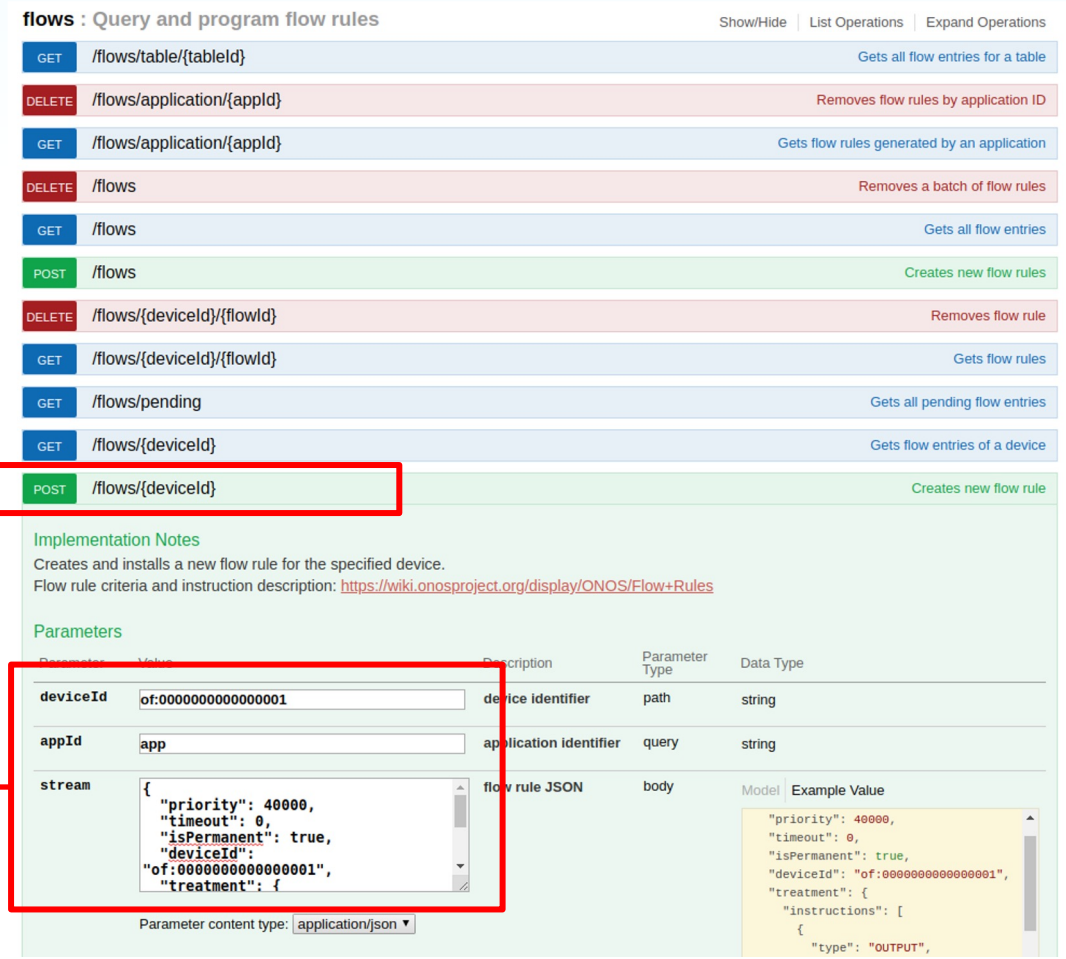


Using Web GUI to Install Flow Rule (1/2)

- Fill out required fields ("appId" could be arbitrary string)

Select

[POST] /flows/{deviceId}



- Compare GUI with curl command

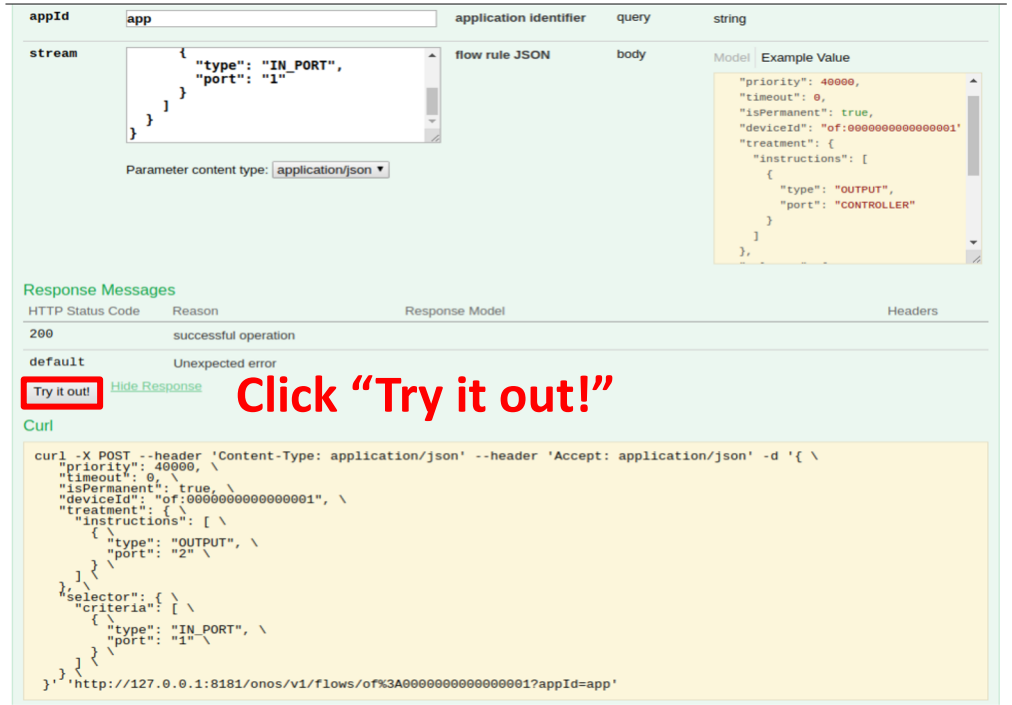
```
$ curl -u onos:rocks \
> -X POST \
> -H 'Content-Type: application/json' \
> -d @flows1.json \
>'http://localhost:8181/onos/v1/flows/of:0000000000000001'
```

Type In

deviceId
appId
JSON file

Using Web GUI to Install Flow Rule (2/2)

- Click “Try it out!”
  - Web will pass the JSON stream to ONOS
  - Status code 201 represents HTTP Request is granted



HTTP response replied by ONOS

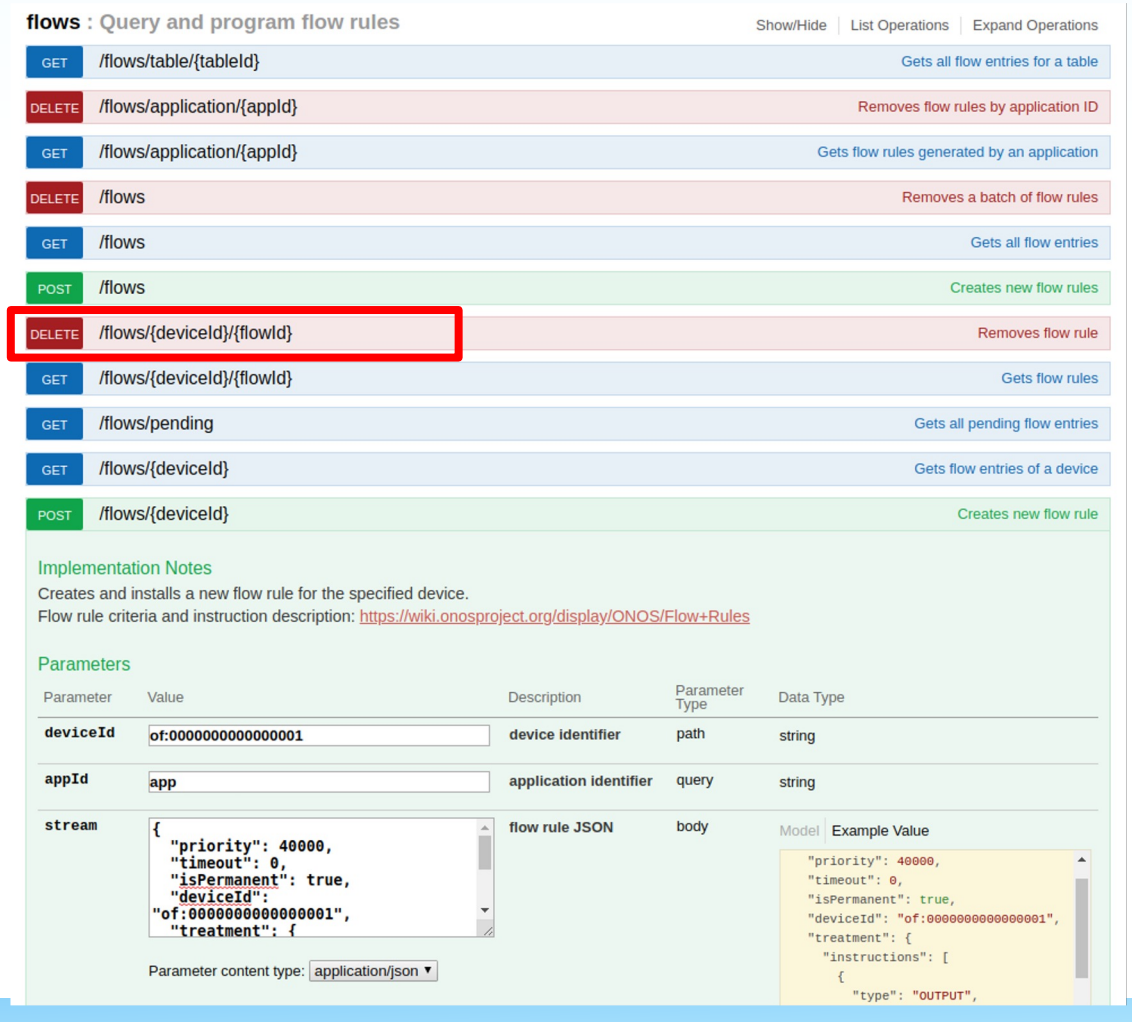
  - In case of “curl”, use “-i” option to include HTTP Response header in the output

Delete Flow Rule via ONOS Web GUI

- Same procedure as installing flow rules

Select

[DELETE] /flows/{deviceId}/{flowId}



Outline

- OpenFlow Messages
- Install/ Delete Flow Rules
- Project 2 Requirements
  - Part 1: Answer Questions (15%)
  - Part 2: Install Flow Rules (15%)
  - Part 3: Create Broadcast Storm (20%)
  - Part 4: Trace ReactiveForwarding (10%)
  - Project 2 Demo(40%)

Part 1: Answer Questions (1/2)

- Preparation:
  1. Start capturing packets on the loopback interface (lo) with Wireshark.
  2. Create a default topology topology.
  3. Activate “org.onosproject.fwd”.
  4. Execute command “h1 ping h2 -c 5” in Mininet CLI.
  5. Exit Mininet and stop capturing packets once ping terminates.

- Questions:
  1. How many **OpenFlow headers** with type “OFPT_FLOW_MOD” and command “OFPFC_ADD” are there among all the packets?
  2. What are the **match fields** and the corresponding **actions** in each “OFPT_FLOW_MOD” message?
  3. What are the **timeout values** for all flow rules on s1?

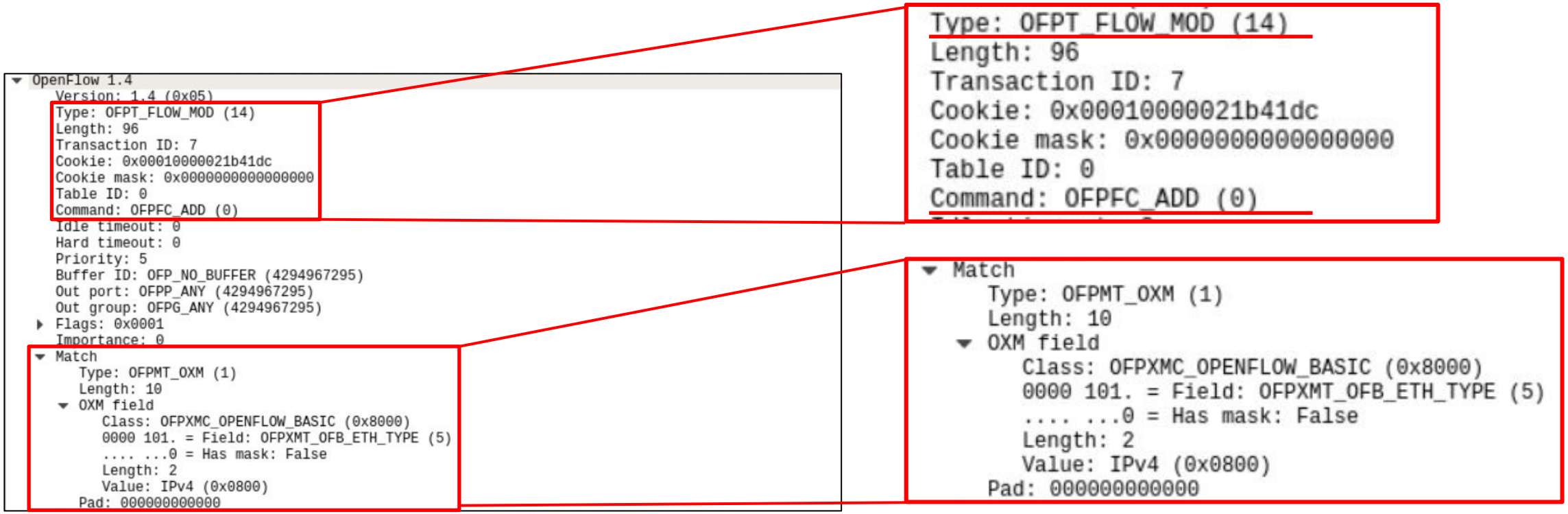
Report format

Ex:There are **x** distinct “OFPT_FLOW_MOD” headers during the experiment.

Math fields	actions	Timeout values
IN_PORT=1	Output port=4	0
……		

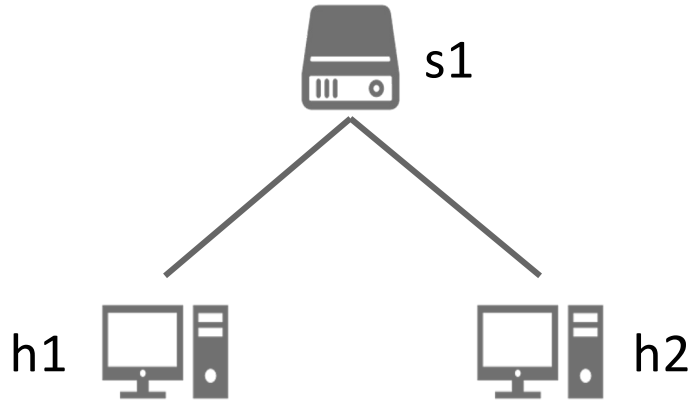
Part 1: Answer Questions (2/2)

- Hints
  - A single OpenFlow packet may contain multiple OpenFlow message headers
  - Only count the number of **distinct** OpenFlow headers
    - If match fields of two headers are the same, just count once
  - Value of timeout can be zero



Part 2: Install Flow Rules (1/3)

- Please deactivate all the apps, **except those** initially activated.
  “org.onosproject.hostprovider”,
  “org.onosproject.lldpprovider”,
  “org.onosproject.optical-model”,
  “org.onosproject.openflow-base”,
  “org.onosproject.openflow”,
  “org.onosproject.drivers”
  and “org.onosproject.gui2”.
- Use the following topology (i.e. h1-s1-h2):




```
$ sudo mn --controller=remote,127.0.0.1:6653 --switch=ovs,protocols=OpenFlow14
```

Part 2: Install Flow Rules (2/3)

- Install **one** flow rule to forward ARP packets with below condition
  - Match Fields
    - Ethernet type (ARP)
  - Actions
    - Forwarding ARP packets to all port in one instruction
- Verify the flow rules your installed and take screenshot

```
mininet> h1 arping h2                # send ARP request
```

```
mininet> h1 arping h2
ARPING 10.0.0.2 from 10.0.0.1 h1-eth0
Unicast reply from 10.0.0.2 [42:75:EB:67:61:F6]  4.324ms
Unicast reply from 10.0.0.2 [42:75:EB:67:61:F6]  4.957ms
Unicast reply from 10.0.0.2 [42:75:EB:67:61:F6]  4.928ms
Unicast reply from 10.0.0.2 [42:75:EB:67:61:F6]  4.834ms
```

Part 2: Install Flow Rules (3/3)

- Install **two** flow rules to forward IPv4 packets
  - Match Fields
    - IPv4 destination address and other required dependencies
  - Actions
    - Forwarding IPv4 packets to the right host
- Verify the flow rules your installed and take screenshot

```
mininet> h1 ping h2        # send ICMP request
```

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=9.00 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.54 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.188 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.075 ms
```

Hint:

1. *Match fields may have dependency; please refer to OpenFlow spec v1.5.1.*

Part 3: Create Topology with Broadcast Storm

- Steps:
  - Create a topology that may cause a “Broadcast Storm”.
  - Install flow rules on switches of the network.
  - Send packets from a host to another host.
  - Observe link status of the network and the CPUs utilization of VM
- Do NOT activate any other APPs, except for those initially activated by ONOS
- Describe what you have observed and explain why the broadcast storm occurred
- Take screenshot of CPU’s utilization
- Hand in both Topology file (*.py) and flow rule files (*.json)

Hint: ONOS would initially install several flow rules.

Part 4: Trace ReactiveForwarding

- Activate only “org.onosproject.fwd” and other initially activated APPs.
- Use Mininet default topology and let h1 ping h2.
- Observe what happens in data and control planes
  - From the time when h1 pings h2 until h2 receives the first ICMP request
  - Write down each operation made by data and control planes
  - Please refer to the ONOS ReactiveForwarding application
  - Source Code
- Describe the what you observe step by step and write in report
- Don’t need to take screenshot

Hint: Tracing Source code with wireshark to finish this Part

Submission (1/2)

- Files:
  - A report: project2_<studentID>.pdf

1. Part 1: Answers to those three questions in p.31 format
2. Part 2: Take screenshots of arping/ping result
3. Part 3: Take screenshots and answer the question
4. Part 4: Write down what you have observed step by step
5. What you've learned or solved

  - JSON files for installing flow rules in part 2 and part 3
    - Please follow naming convention
  - A Python script for creating topology in part 3

Naming Convention

- Use the following convention to name the files created in both part 2 and part 3.
  1. Python script for the topology: topo_<studentID>.py
  2. JSON files for flow rules: flows_s<i>-<j>_<studentID>.json
     - “i” is the switch number
     - “j” is the flow rule number, starting from 1, on a switch.

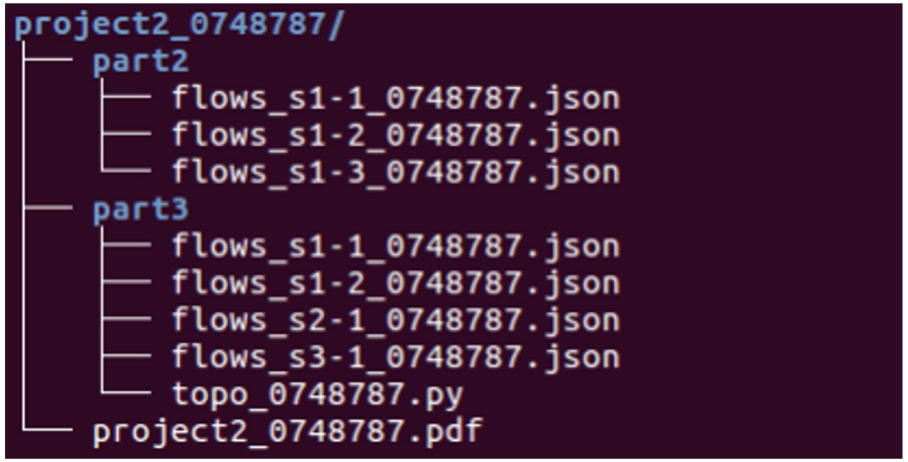
     e.g.

File Name	Meaning
flows_s1-1_0748787.json	#1 flow rule to install on s1
flows_s1-2_0748787.json	#2 flow rule to install on s1
flows_s2-1_0748787.json	#1 flow rule to install on s2

Submission (2/2)

- Directory structure:
  - Create root folder: project2_<studentID>
  - In root folder, create part2 and part3 folders and move files (i.e. *.json, *.py) into the corresponding folders

    e.g.



  - Zip root folder: project2_<studentID>.zip
- Wrong file name or format will result in 10 points deduction

Project 2 Demo

- Date: TA will open a demo time-reserved table one week before demo. The demo dates will be in the week after project 2 deadline.
- Demo question will show when demo start.
- There will be 3 questions, and questions will be similar to project 2.

About help!

- **For lab problem, ask at e3 forum**
  - Ask at the e3 forum
  - TAs will help to clarify Lab contents instead of giving answers!
  - Please describe your questions with sufficient context,
    - , e.g., Environment setup, Input/Output, Screenshots, …
- **For personal problem mail to sdnta@win.cs.nctu.edu.tw**
  - You have special problem and you can't meet the deadline
  - You got weird score with project
- **No Fixed TA hour**

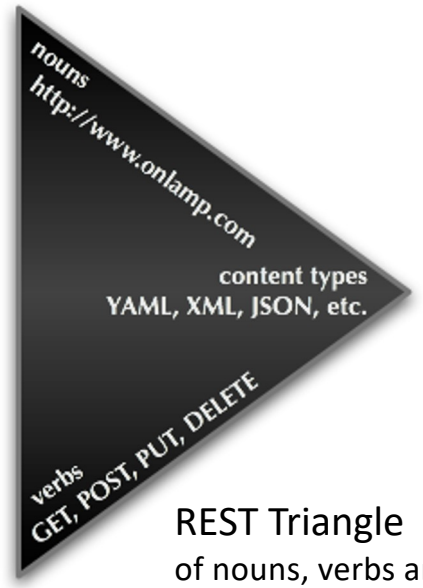
Q & A

References

- OpenFlow spec v1.5.1
  - https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
- Wireshark wiki
  - https://wiki.wireshark.org/Home
- ONOS REST API
  - https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API
- JSON Format for Installing Flow Rules
  - https://wiki.onosproject.org/display/ONOS/Flow+Rules

Appendix—REST (REpresentational State Transfer)

- REST is a **software architectural style** for creating Web services
- Architectural constraints:
  - Client-server architecture
  - Stateless
  - Cacheable
  - Uniform interface
  - Layered system



REST Triangle
of nouns, verbs and content types

Source: Soul & Shell Blog

- Allow us to access and manipulate web resources
  - Commonly we use HTTP method
    - Payload could be formatted in HTML, XML, JSON