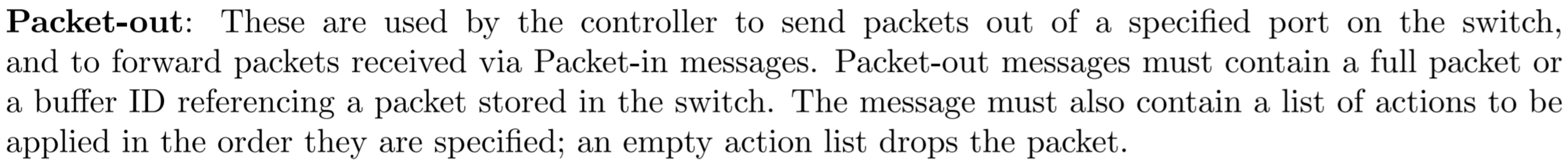
Lab 3 Recall

- Packet-out defined in OpenFlow 1.3

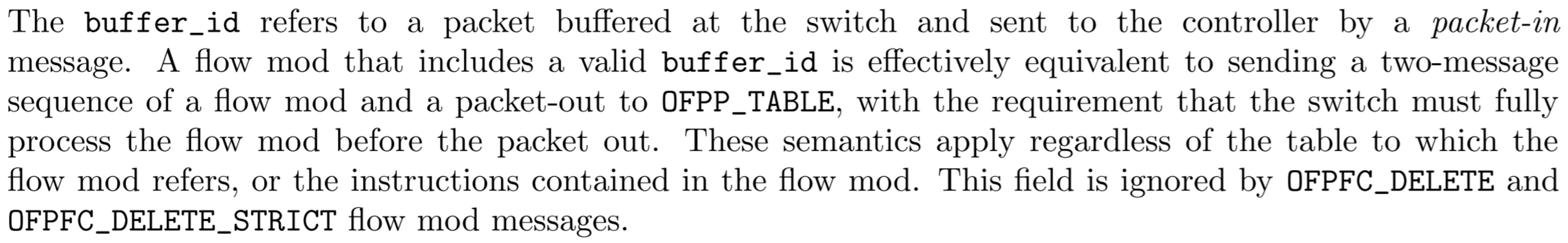


Packet-out: These are used by the controller to send packets out of a specified port on the switch, and to forward packets received via Packet-in messages. Packet-out messages must contain a full packet or a buffer ID referencing a packet stored in the switch. The message must also contain a list of actions to be applied in the order they are specified; an empty action list drops the packet.

- Make sure to Packet-out when you send Flow-mod
  - Since flow modification message only install flow rule on the switch

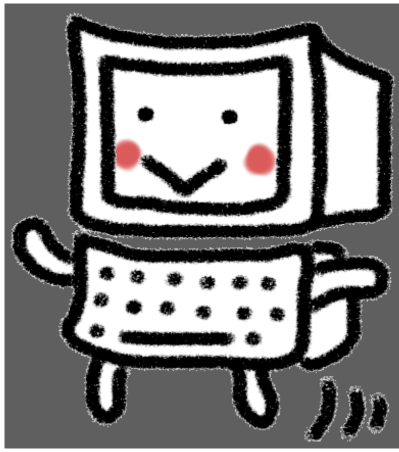
… without buffer ID

- Flow modification messages with buffer ID



The `buffer_id` refers to a packet buffered at the switch and sent to the controller by a *packet-in* message. A flow mod that includes a valid `buffer_id` is effectively equivalent to sending a two-message sequence of a flow mod and a packet-out to `OFPP_TABLE`, with the requirement that the switch must fully process the flow mod before the packet out. These semantics apply regardless of the table to which the flow mod refers, or the instructions contained in the flow mod. This field is ignored by `OFPFC_DELETE` and `OFPFC_DELETE_STRICT` flow mod messages.

Refer from OpenFlow specification v1.3.0



Project 4

Path Service

Date:	2019/04/11 (Thu.)
Deadline:	2019/04/28 (Sun.)

Ryu – What Should an SDN Application do?

- Request Packet-in
  - switch_features_handler()
- Handle Packet-in
  - _packet_in_handler()
- Packet-out
  - OFPPacketOut()
- Flow-mod
  - add_flow()   // OFPFlowMod()

Refer from Ryu SimpleSwitch13

ONOS – What Should an SDN Application do?

- Request Packet-in
  - requestIntercepts()
- Handle Packet-in
  - PacketProcessor{process()}
- Packet-out
  - packetOut() // PacketContext.setOutput().send()
- Flow-mod
  - installRule() // FlowObjectiveService.forward()

Refer from ONOS v1.15 ReactiveForwarding

Outline

- Project 4 Requirements
- Reference

Outline

- Project 4 Requirements
- Reference

Path Service

Project 4 Requirements

Requirement

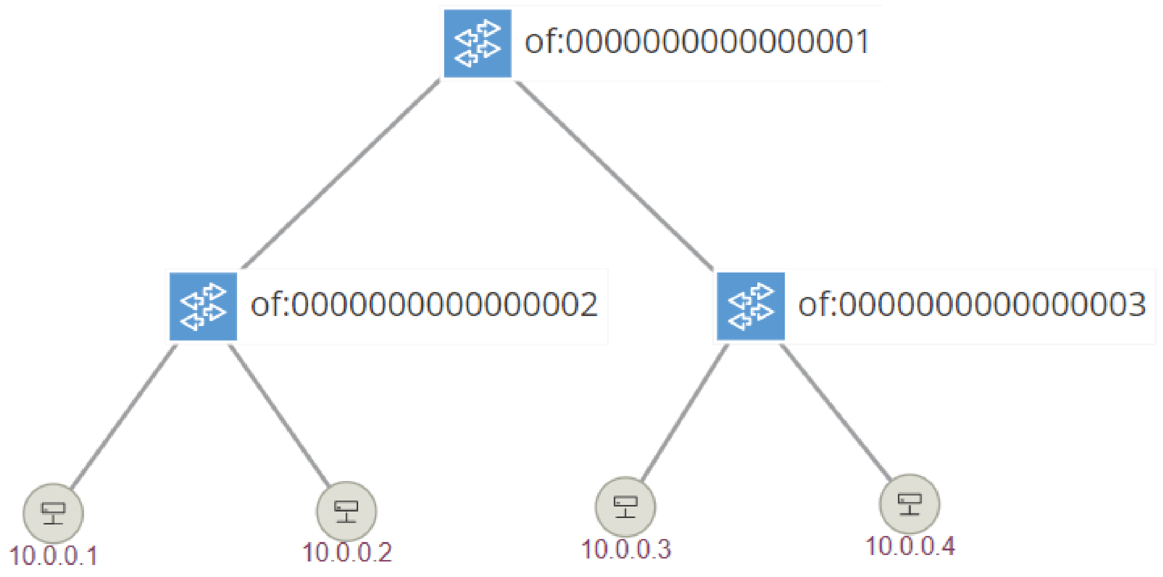
- Write a path service application which finds a path
    - Construct a graph of network topology when Packet-in
    - Find a path between the source and destination
    - You could use algorithms like BFS, Dijkstra or spanning tree
    - Shortest path algorithms are not required

- Proactive install flow rules on each devices in the path
    - Each packet from hosts would only trigger one Packet-in
    - Flow rules would be proactively installed on devices in the following path
    - Compare with reactive forwarding such as SimpleSwitch13 in Ryu or ReactiveForwarding in ONOS

Demonstration (I)

- Using Mininet tree topology with depth in 2

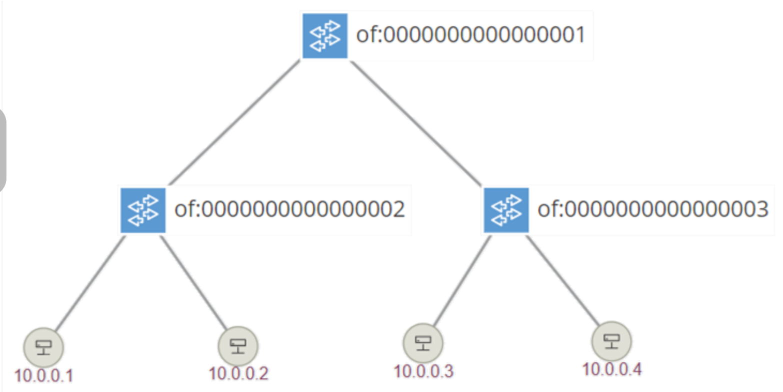
```
$ sudo mn --controller=remote --topo=tree,depth=2
```



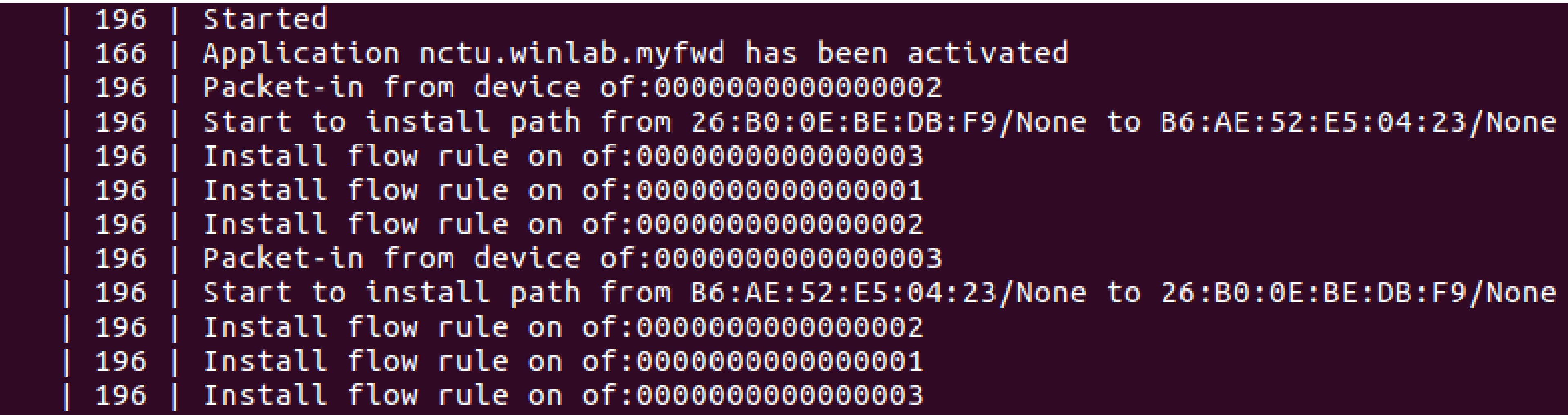
Demonstration (II)

- Ping from h1 to h3

```
mininet> h1 ping h3
```



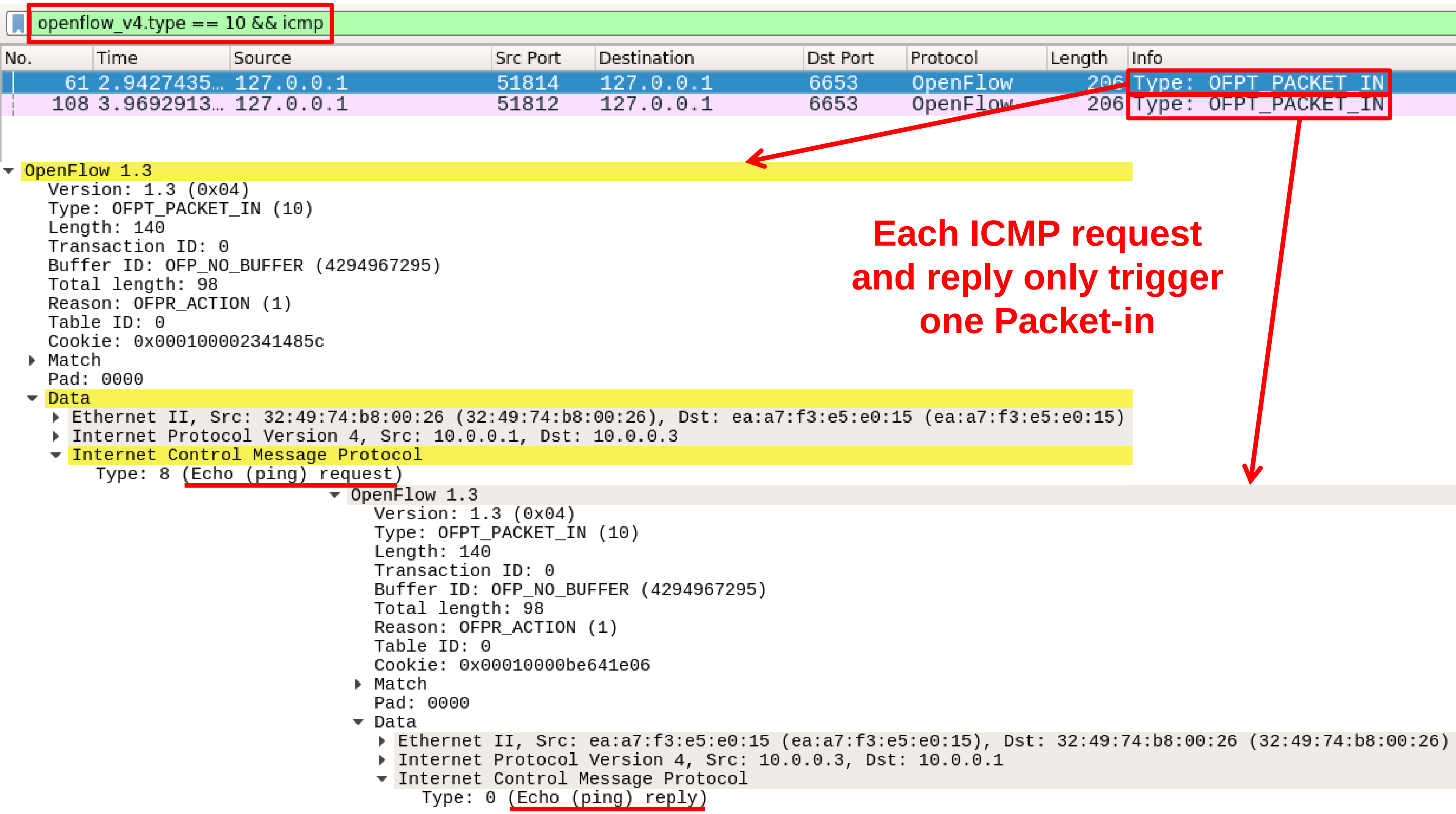
- ONOS log
  - Only devices connected source hosts would send Packet-in
  - Flow rules would be installed on devices in the following path



```
| 196 | Started
| 166 | Application nctu.winlab.myfwd has been activated
| 196 | Packet-in from device of:0000000000000002
| 196 | Start to install path from 26:B0:0E:BE:DB:F9/None to B6:AE:52:E5:04:23/None
| 196 | Install flow rule on of:0000000000000003
| 196 | Install flow rule on of:0000000000000001
| 196 | Install flow rule on of:0000000000000002
| 196 | Packet-in from device of:0000000000000003
| 196 | Start to install path from B6:AE:52:E5:04:23/None to 26:B0:0E:BE:DB:F9/None
| 196 | Install flow rule on of:0000000000000002
| 196 | Install flow rule on of:0000000000000001
| 196 | Install flow rule on of:0000000000000003
```

Demonstration (III)

- Capture OpenFlow packets from Loopback interface



```
openflow_v4.type == 10 && icmp
```

No.	Time	Source	Src Port	Destination	Dst Port	Protocol	Length	Info
61	2.9427435…	127.0.0.1	51814	127.0.0.1	6653	OpenFlow	206	Type: OFPT_PACKET_IN
108	3.9692913…	127.0.0.1	51812	127.0.0.1	6653	OpenFlow	206	Type: OFPT_PACKET_IN

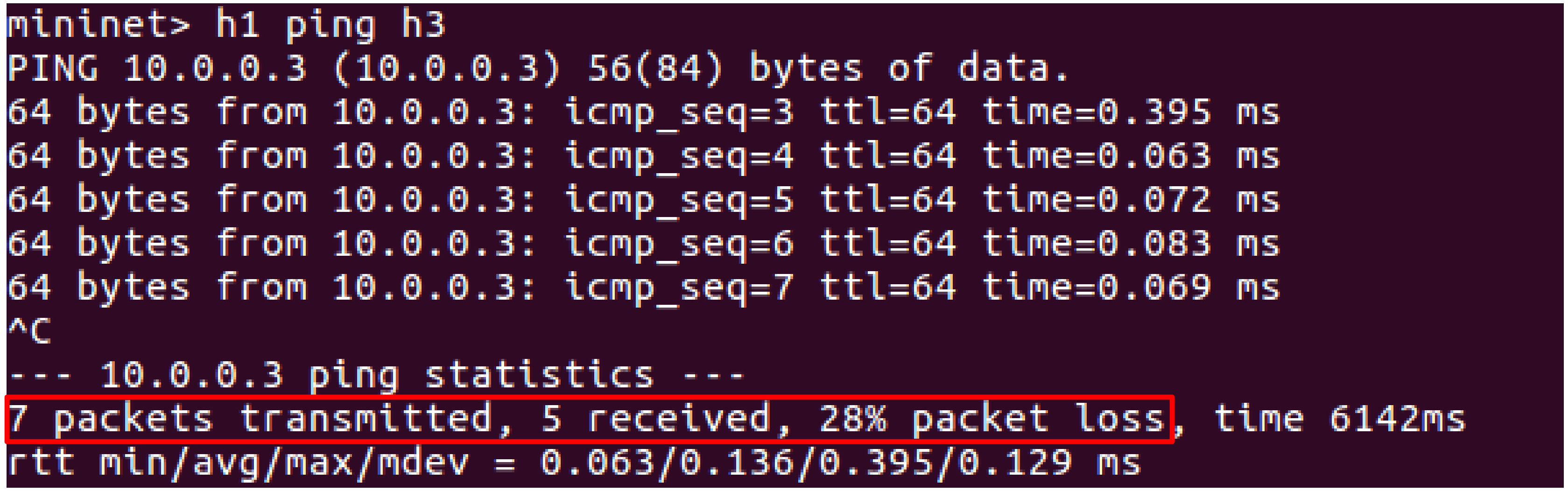
```
OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_PACKET_IN (10)
  Length: 140
  Transaction ID: 0
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Total length: 98
  Reason: OFPR_ACTION (1)
  Table ID: 0
  Cookie: 0x000100002341485c
  Match
  Pad: 0000
  Data
    Ethernet II, Src: 32:49:74:b8:00:26 (32:49:74:b8:00:26), Dst: ea:a7:f3:e5:e0:15 (ea:a7:f3:e5:e0:15)
    Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.3
    Internet Control Message Protocol
      Type: 8 (Echo (ping) request)
```

```
OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_PACKET_IN (10)
  Length: 140
  Transaction ID: 0
  Buffer ID: OFP_NO_BUFFER (4294967295)
  Total length: 98
  Reason: OFPR_ACTION (1)
  Table ID: 0
  Cookie: 0x00010000be641e06
  Match
  Pad: 0000
  Data
    Ethernet II, Src: ea:a7:f3:e5:e0:15 (ea:a7:f3:e5:e0:15), Dst: 32:49:74:b8:00:26 (32:49:74:b8:00:26)
    Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.1
    Internet Control Message Protocol
      Type: 0 (Echo (ping) reply)
```

Each ICMP request and reply only trigger one Packet-in

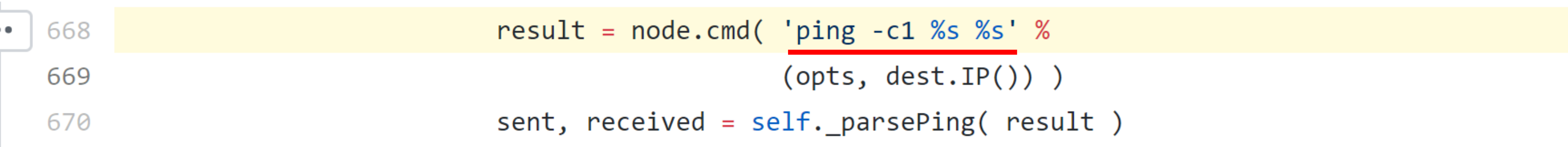
Demonstration (IV)

- Forwarding packets in data plane is much faster than installing flow rules in controller plane



```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.395 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.063 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.072 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.083 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.069 ms
^C
--- 10.0.0.3 ping statistics ---
7 packets transmitted, 5 received, 28% packet loss, time 6142ms
rtt min/avg/max/mdev = 0.063/0.136/0.395/0.129 ms
```

- Pingall in Mininet only take one ICMP packet into consideration
  - Thus, pingall would not work



```
                result = node.cmd( 'ping -c1 %s %s' %
                                   (opts, dest.IP()) )
                sent, received = self._parsePing( result )
```

  - ➢ Use ping to test your program

Refer from GitHub mininet/mininet - mininet/net.py

Hints

- Since forwarding packets in data plane is much faster than installing flow rules in controller plane

- Install flow rules on network devices from destination to source in the path

- Do not Packet-out when installing flow rules
  - Or the packet which triggers Packet-in would reach the next device before the flow rule successfully installed
  - The first packet is inevitably lost due to timeout
    - That is why always two packet loss in every ping

Outline

- Project 4 Requirements
- Reference

Path Service

Reference

Reference

- Ryu
  - Ryu API Reference
    - https://ryu.readthedocs.io/en/latest/api_ref.html
  - Ryu GitHub – ryu/ryu/topology/api.py
    - https://github.com/osrg/ryu/blob/master/ryu/topology/api.py
- ONOS
  - ONOS Java API 1.15.0
    - http://api.onosproject.org/1.15.0/apidocs/
  - ONOS TopologyService.getGraph() would return the topology graph
  - **DO NOT** use ONOS built-in APIs to find paths (e.g. TopologyService.getPath())

Submit to e3

- Files
  - All files of your application
- Submit
  - Upload ".zip" file to e3
    - Named: **project4_studentID.zip**
  - Wrong file name or format would not be scored

Hints for Demo Scenario

- Finding paths
  - Your application should be able to deal with loops in the topology
- Proactive installing flow rules
  - Please check if only two Packet-in are captured from Wireshark instead of logging
    - Since controller like ONOS would abstract some operations below
  - Probably do not set timeout too short in case that switches trigger another Packet-in after flow rules timeout

Q & A

Thank you