Using OpenFlow 1.3

RYU

SDN Framework

RYU project team

This specialized book is for the Ryu development framework, which is used to achieve Software Defined Networking (SDN).

Why Ryu?

We hope you can find the answer in this book.

We recommend that you read Chapters 1 to 5, in that order. In Chapter 1, a simple switch hub is implemented, and in later chapters, traffic monitor and link aggregation functions are added. Through actual examples, we describe programming using Ryu.

Chapters 6 to 8 provide details about the OpenFlow protocol and the packet libraries that are necessary for programming using Ryu. In Chapters 9 to 11, we talk about how to use the firewall and test tool included in the Ryu package as sample applications. Chapters 12 to 14 introduce Ryu's architecture and introduction cases.

Finally, we would like to say thank you to those people, in particular users, who supported the Ryu project. We are waiting for your opinions via the mailing list.

Let's develop Ryu together!

CHAPTER

ONE

SWITCHING HUB

This section uses implementation of a simple switching hub as a material to describes the method of implementing applications using Ryu.

1.1 Switching Hub

Switching hubs have a variety of functions. Here, we take a look at a switching hub having the following simple functions.

- Learns the MAC address of the host connected to a port and retains it in the MAC address table.
- When receiving packets addressed to a host already learned, transfers them to the port connected to the host.
- When receiving packets addressed to an unknown host, performs flooding.

Let's use Ryu to implement such a switch.

1.2 Switching Hub by OpenFlow

OpenFlow switches can perform the following by receiving instructions from OpenFlow controllers such as Ryu.

- Rewrites the address of received packets or transfers the packets from the specified port.
- Transfers the received packets to the controller (Packet-In).
- Transfers the packets forwarded by the controller from the specified port (Packet-Out).

It is possible to achieve a switching hub having those functions combined.

First of all, you need to use the Packet-In function to learn MAC addresses. The controller can use the Packet-In function to receive packets from the switch. The switch analyzes the received packets to learn the MAC address of the host and information about the connected port.

After learning, the switch transfers the received packets. The switch investigates whether the destination MAC address of the packets belong to the learned host. Depending on the investigation results, the switch performs the following processing.

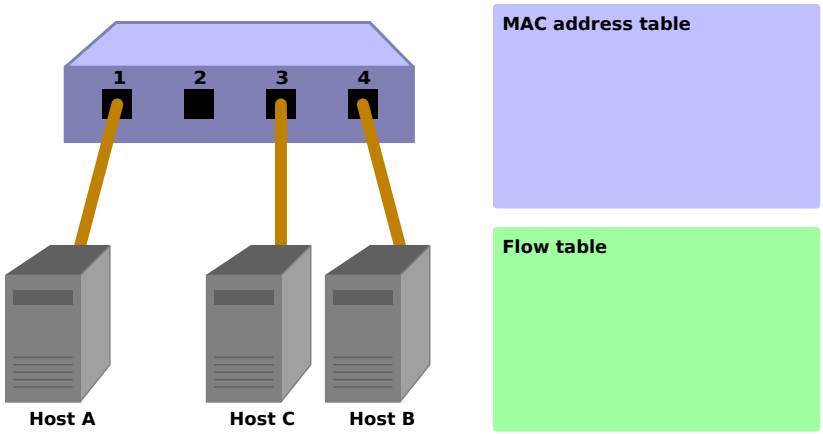
- If the host is already a learned host ... Uses the Packet-Out function to transfer the packets from the connected port.
- If the host is unknown host ... Use the Packet-Out function to perform flooding.

The following explains the above operation in a step-by-step way using figures.

1. Initial status

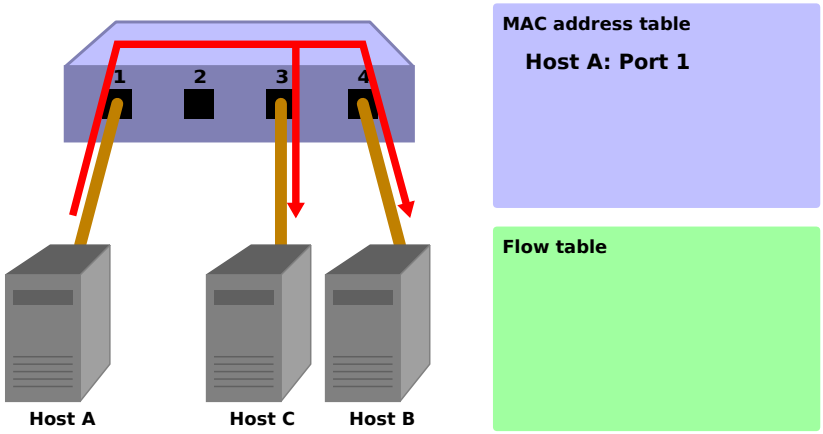
   This is the initial status where the flow table is empty.

   Assuming host A is connected to port 1, host B to part 4, and host C to port 3.



2. Host A -> Host B

   When packets are sent from host A to host B, a Packet-In message is sent and the MAC address of host A is learned by port 1. Because the port for host B has not been found, the packets are flooded and are received by host B and host C.



   Packet-In:

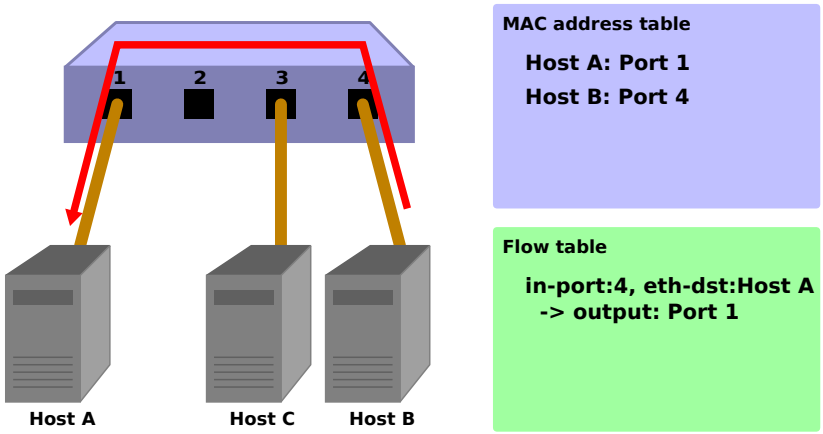
```
in-port: 1
eth-dst: Host B
eth-src: Host A
```

   Packet-Out:

```
action: OUTPUT:Flooding
```

3. Host B -> Host A

   When the packets are returned from host B to host A, an entry is added to the flow table and also the packets are transferred to port 1. For that reason, the packets are not received by host C.



Packet-In:

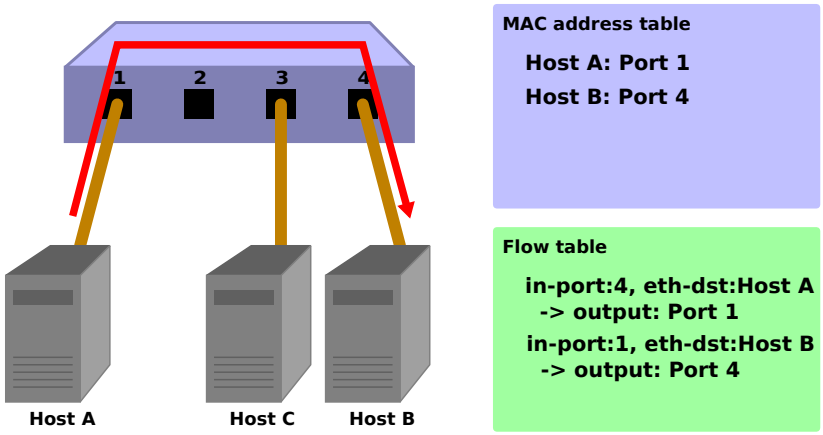
```
in-port: 4
eth-dst: Host A
eth-src: Host B
```

Packet-Out:

```
action: OUTPUT:Port 1
```

4. Host A -> Host B

   Again, when packets are sent from host A to host B, an entry is added to the flow table and also the packets are transferred to port 4.



Packet-In:

```
in-port: 1
eth-dst: Host B
eth-src: Host A
```

Packet-Out:

```
action: OUTPUT:Port 4
```

Next, let's take a look at the source code of a switching hub implemented using Ryu.

1.3 Implementation of Switching Hub Using Ryu

The source code of the switching hub is in Ryu's source tree.

ryu/app/simple_switch_13.py

Other than the above, there are simple_switch.py(OpenFlow 1.0) and simple_switch_12.py(OpenFlow 1.2), depending on the version of OpenFlow but we take a look at implementation supporting OpenFlow 1.3.

The source code is short thus we shown the entire source code below.

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet


class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]

        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
```

```
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            self.add_flow(datapath, 1, match, actions)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions, data=data)
        datapath.send_msg(out)
```

Let's examine the respective implementation content.

1.3.1 Class Definition and Initialization

In order to implement as a Ryu application, ryu.base.app_manager.RyuApp is inherited. Also, to use OpenFlow 1.3, the OpenFlow 1.3 version is specified for `OFP_VERSIONS`.

Also, MAC address table mac_to_port is defined.

In the OpenFlow protocol, some procedures such as handshake required for communication between the OpenFlow switch and the controller have been defined. However, because Ryu's framework takes care of those procedures thus it is not necessary to be aware of those in Ryu applications.

```
class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    # ...
```

1.3.2 Event Handler

With Ryu, when an OpenFlow message is received, an event corresponding to the message is generated. The Ryu application implements an event handler corresponding to the message desired to be received.

The event handler defines a function having the event object for the argument and use the `ryu.controller.handler.set_ev_cls` decorator to decorate.

set_ev_cls specifies the event class supporting the received message and the state of the OpenFlow switch for the argument.

The event class name is `ryu.controller.ofp_event.EventOFP` + <OpenFlow message name>. For example, in case of a Packet-In message, it becomes `EventOFPPacketIn`. For details, refer to Ryu's document titled API Reference . For the state, specify one of the following or list.

Definition	Explanation
ryu.controller.handler.HANDSHAKE_DISPATCHER	Exchange of HELLO message
ryu.controller.handler.CONFIG_DISPATCHER	Waiting to receive SwitchFeatures message
ryu.controller.handler.MAIN_DISPATCHER	Normal status
ryu.controller.handler.DEAD_DISPATCHER	Disconnection of connection

Adding Table-miss Flow Entry

After handshake with the OpenFlow switch is completed, the Table-miss flow entry is added to the flow table to get ready to receive the Packet-In message.

Specifically, upon receiving the Switch Features(Features Reply) message, the Table-miss flow entry is added.

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # ...
```

In `ev.msg`, the instance of the OpenFlow message class corresponding to the event is stored. In this case, it is `ryu.ofproto.ofproto_v1_3_parser.OFPSwitchFeatures`.

In `msg.datapath`, the instance of the `ryu.controller.controller.Datapath` class corresponding to the OpenFlow switch that issued this message is stored.

The Datapath class performs important processing such as actual communication with the OpenFlow switch and issuance of the event corresponding to the received message.

The main attributes used by the Ryu application are as follows:

Attribute name	Explanation
id	ID (data path ID) of the connected OpenFlow switch.

ofproto	Indicates the ofproto module that supports the OpenFlow version in use. At this point, it is one of the following modules.
	`ryu.ofproto.ofproto_v1_0`
	`ryu.ofproto.ofproto_v1_2`
	`ryu.ofproto.ofproto_v1_3`

ofproto_parser	Same as ofproto, indicates the ofproto_parser module. At this point, it is one of the following modules.
	`ryu.ofproto.ofproto_v1_0_parser`
	`ryu.ofproto.ofproto_v1_2_parser`
	`ryu.ofproto.ofproto_v1_3_parser`

The main methods of the Datapath class used in the Ryu application are as follows:

send_msg(msg)

> Sends the OpenFlow message. msg is a sub class of `ryu.ofproto.ofproto_parser.MsgBase` corresponding to the send OpenFlow message.

The switching hub does not particularly use the received Switch Features message itself. It is handled as an event to obtain timing to add the Table-miss flow entry.

```
def switch_features_handler(self, ev):
    # ...

    # install table-miss flow entry
    #
    # We specify NO BUFFER to max_len of the output action due to
    # OVS bug. At this moment, if we specify a lesser number, e.g.,
    # 128, OVS will send Packet-In with invalid buffer_id and
    # truncated packet data. In that case, we cannot output packets
    # correctly.
    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)
```

The Table-miss flow entry has the lowest (0) priority and this entry matches all packets. In the instruction of this entry, by specifying the output action to output to the controller port, in case the received packet does not match any of the normal flow entries, Packet-In is issued.

Note: As of January 2014, Open vSwitch does not fully support OpenFlow 1.3 and as with versions prior to OpenFlow1.3, Packet-In is issued by default. Also, the Table-miss flow entry is not supported at this time and is handled as a normal flow entry.

An empty match is generated to match all packets. Match is expressed in the `OFPMatch` class.

Next, an instance of the OUTPUT action class (`OFPActionOutput`) is generated to transfer to the controller port. The controller is specified as the output destination and `OFPCML_NO_BUFFER` is specified to max_len in order to send all packets to the controller.

Finally, 0 (lowest) is specified for priority and the `add_flow()` method is executed to send the Flow Mod message. The content of the add_flow() method is explained in a later section.

Packet-in Message

Create the handler of the Packet-In event handler in order to accept received packets with an unknown destination.

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # ...
```

Frequently used OFPPacketIn class attributes are as follows:

Attribute name	Explanation
match	`ryu.ofproto.ofproto_v1_3_parser.OFPMatch` class instance in which the meta information of received packets is set.
data	Binary data indicating received packets themselves.
total_len	Data length of the received packets.
buffer_id	When received packets are buffered on the OpenFlow switch, indicates its ID. If not buffered, `ryu.ofproto.ofproto_v1_3.OFP_NO_BUFFER` is set.

Updating the MAC Address Table

```
def _packet_in_handler(self, ev):
    # ...

    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]

    dst = eth.dst
    src = eth.src

    dpid = datapath.id
    self.mac_to_port.setdefault(dpid, {})

    self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

    # learn a mac address to avoid FLOOD next time.
    self.mac_to_port[dpid][src] = in_port

    # ...
```

Get the receive port (`in_port`) from the OFPPacketIn match. The destination MAC address and sender MAC address are obtained from the Ethernet header of the received packets using Ryu's packet library.

Based on the acquired sender MAC address and received port number, the MAC address table is updated.

In order to support connection with multiple OpenFlow switches, the MAC address table is so designed to be managed for each OpenFlow switch. The data path ID is used to identify OpenFlow switches.

Judging the Transfer Destination Port

The corresponding port number is used when the destination MAC address exists in the MAC address table. If not found, the instance of the OUTPUT action class specifying flooding (`OFPP_FLOOD`) for the output port is generated.

```
def _packet_in_handler(self, ev):
    # ...

    if dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dst]
    else:
        out_port = ofproto.OFPP_FLOOD

    actions = [parser.OFPActionOutput(out_port)]

    # install a flow to avoid packet_in next time
    if out_port != ofproto.OFPP_FLOOD:
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
        self.add_flow(datapath, 1, match, actions)

    # ...
```

If the destination MAC address is found, an entry is added to the flow table of the OpenFlow switch.

As with addition of the Table-miss flow entry, specify match and action, and execute add_flow() to add a flow entry.

Unlike the Table-miss flow entry, set conditions for match this time. Implementation of the switching hub this time, the receive port (in_port) and destination MAC address (eth_dst) have been specified. For example, packets addressed to host B received by port 1 is the target.

For the flow entry this time, the priority is specified to 1. The greater the value, the higher the priority, therefore, the flow entry added here will be evaluated before the Table-miss flow entry.

Based on the summary including the aforementioned actions, add the following entry to the flow table.

Transfer packets addressed to host B (the destination MAC address is B) received by port 1 to port 4.

Hint: With OpenFlow, a logical output port called NORMAL is prescribed in option and when NORMAL is specified for the output port, the L2/L3 function of the switch is used to process the packets. That means, by instructing to output all packets to the NORMAL port, it is possible to make the switch operate as a switching hub. However, we implement each processing using OpenFlow.

Adding Processing of Flow Entry

Processing of the Packet-In handler has not been done yet but here take a look at the method to add flow entries.

```
def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]

    # ...
```

For flow entries, set match that indicates the target packet conditions, and instruction that indicates the operation on the packet, entry priority level, and effective time.

In the switching hub implementation, Apply Actions is used for the instruction to set so that the specified action is immediately used.

Finally, add an entry to the flow table by issuing the Flow Mod message.

```
def add_flow(self, datapath, port, dst, actions):
    # ...

    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                            match=match, instructions=inst)
    datapath.send_msg(mod)
```

The class corresponding to the Flow Mod message is the `OFPFlowMod` class. The instance of the OFPFlowMod class is generated and the message is sent to the OpenFlow switch using the Datapath.send_msg() method.

There are many arguments of constructor of the OFPFlowMod class. Many of them generally can be the default as is. Inside the parenthesis is the default.

datapath

> This is the Datapath class instance supporting the OpenFlow switch subject to flow table operation. Normally, specify the one acquired from the event passed to the handler such as the Packet-In message.

cookie (0)

> An optional value specified by the controller and can be used as a filter condition when updating or deleting entries. This is not used for packet processing.

cookie_mask (0)

> When updating or deleting entries, if a value other than 0 is specified, it is used as the filter of the operation target entry using the cookie value of the entry.

table_id (0)

> Specifies the table ID of the operation target flow table.

command (ofproto_v1_3.OFPFC_ADD)

> Specify whose operation is to be performed.

Value	Explanation
OFPFC_ADD	Adds new flow entries.
OFPFC_MODIFY	Updates flow entries.
OFPFC_MODIFY_STRICT	Update strictly matched flow entries
OFPFC_DELETE	Deletes flow entries.
OFPFC_DELETE_STRICT	Deletes strictly matched flow entries.

idle_timeout (0)

> Specifies the validity period of this entry, in seconds. If the entry is not referenced and the time specified by idle_timeout elapses, that entry is deleted. When the entry is referenced, the elapsed time is reset.
>
> When the entry is deleted, a Flow Removed message is sent to the controller.

hard_timeout (0)

> Specifies the validity period of this entry, in seconds. Unlike dle_timeout, with hard_timeout, even though the entry is referenced, the elapsed time is not reset. That is, regardless of the reference of the entry, the entry is deleted when the specified time elapsed.
>
> As with idle_timeout, when the entry is deleted, a Flow Removed message is sent.

priority (0)

> Specifies the priority order of this entry. The greater the value, the higher the priority.

buffer_id (ofproto_v1_3.OFP_NO_BUFFER)

> Specifies the buffer ID of the packet buffered on the OpenFlow switch. The buffer ID is notified in the packet-In message and when the specified processing is the same as when two messages are sent,

i.e., the Packet-Out message for which OFPP_TABLE is specified for the output port and Flow Mod message. This is ignored when the command is OFPFC_DELETE or OFPFC_DELETE_STRICT.

When the buffer ID is not specified, set `OFP_NO_BUFFER`.

out_port (0)

If the command is OFPFC_DELETE or OFPFC_DELETE_STRICT, the target entry is filtered by the output port. If the command is OFPFC_ADD, OFPFC_MODIFY, or OFPFC_MODIFY_STRICT, it is ignored.

To disable filtering by the output port, specify `OFPP_ANY`.

out_group (0)

As with out_port, filters by the output group.

To disable, specify `OFPG_ANY`.

flags (0)

You can specify the following combinations of flags.

Value	Explanation
OFPFF_SEND_FLOW_REM	Issues the Flow Removed message to the controller when this entry is deleted.
OFPFF_CHECK_OVERLAP	When the command is OFPFC_ADD, checks duplicated entries. If duplicated entries are found, Flow Mod fails and an error is returned.
OFPFF_RESET_COUNTS	Resets the packet counter and byte counter of the relevant entry.
OFPFF_NO_PKT_COUNTS	Disables the packet counter of this entry.
OFPFF_NO_BYT_COUNTS	Disables the byte counter of this entry.

match (None)

Specifies match.

instructions ([])

Specifies a list of instructions.

Packet Transfer

Now we return to the Packet-In handler and explain about final processing.

Regardless whether the destination MAC address is found from the MAC address table, at the end the Packet-Out message is issued and received packets are transferred.

```
def _packet_in_handler(self, ev):
    # ...

    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data

    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)
```

The class corresponding to the Packet-Out message is `OFPPacketOut` class.

The arguments of the constructor of OFPPacketOut are as follows:

datapath

Specifies the instance of the Datapath class corresponding to the OpenFlow switch.

buffer_id

Specifies the buffer ID of the packets buffered on the OpenFlow. If not buffered, `OFP_NO_BUFFER` is specified.

in_port

Specifies the port that received packets. if it is not the received packet, `OFPP_CONTROLLER` is specified.

actions

Specifies the list of actions.

data

Specifies the binary data of packets. This is used when `OFP_NO_BUFFER` is specified for buffer_id. When the OpenFlow switch's buffer is used, this is omitted.

In the switching hub implementation, buffer_id of the Packet-In message has been specified for buffer_id. If the buffer-id of the Packet-In message has been disabled, the received packet of Packet-In is specified for data to send the packets.

This is the end of explanation of the source code of switching hub. Next, let's execute this switching hub to confirm actual operation.

1.4 Execution of Ryu Application

To run the switching hub, we use Open vSwitch for the OpenFlow switch and mininet for the execution environment.

Because OpenFlow Tutorial VM images for Ryu have been prepared, using those VM images makes it easy to prepare the execution environment.

VM image

http://sourceforge.net/projects/ryu/files/vmimages/OpenFlowTutorial/

OpenFlow_Tutorial_Ryu3.2.ova (approx. 1.4GB)

Related document (Wiki page)

https://github.com/osrg/ryu/wiki/OpenFlow_Tutorial

Because the Open vSwitch and Ryu version used for the VM images in the document are old, you must be careful.

If you do not want to use these VM images, you of course can build an environment yourself. For your reference, when building an environment yourself, the version of each software used for the VM images is as follows.

Mininet VM version 2.0.0 http://mininet.org/download/

Open vSwitch version 1.11.0 http://openvswitch.org/download/

Ryu version 3.2 https://github.com/osrg/ryu/

```
$ sudo pip install ryu
```

Here, we use the VM images of the OpenFlow Tutorial for Ryu.

1.4.1 Execution of Mininet

Because xterm is started from mininet, an environment where X can be used is necessary.

Because VM of OpenFlow Tutorial is used, log in using ssh with X11 Forwarding enabled.

```
$ ssh -X ryu@<VM address>
```

The user name is `ryu`, and the password is `ryu`.

When you log in, use the `mn` command to start the Mininet environment.

The environment to be built has a simple structure with three hosts and one switch.

mn command parameters are as follows:

Parameter	Value	Explanation
topo	single,3	Topology of one switch and three hosts
mac	None	Automatically sets the MAC address of the host
switch	ovsk	Uses Open vSwitch
controller	remote	Uses external OpenFlow controller
x	None	Starts xterm

An execution example is as follows:

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Running terms on localhost:10.0
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

When executing the command, five xterm start on the desktop PC. Each xterm corresponds to hosts 1 to 3, the switch and the controller.

Execute the command from the xterm for the switch to set the OpenFlow version to be used. The xterm for which the window title is “switch:s1 (root)” is the one for the switch.

First of all, let’s take a look at the status of Open vSwitch.

switch: s1:

```
root@ryu-vm:~# ovs-vsctl show
fdec0957-12b6-4417-9d02-847654e9cc1f
Bridge "s1"
    Controller "ptcp:6634"
    Controller "tcp:127.0.0.1:6633"
    fail_mode: secure
    Port "s1-eth3"
        Interface "s1-eth3"
    Port "s1-eth2"
        Interface "s1-eth2"
    Port "s1-eth1"
        Interface "s1-eth1"
    Port "s1"
        Interface "s1"
            type: internal
ovs_version: "1.11.0"
root@ryu-vm:~# ovs-dpctl show
system@ovs-system:
        lookups: hit:14 missed:14 lost:0
        flows: 0
        port 0: ovs-system (internal)
        port 1: s1 (internal)
        port 2: s1-eth1
        port 3: s1-eth2
        port 4: s1-eth3
root@ryu-vm:~#
```

Switch (bridge) *s1* has been created and three ports corresponding to hosts have been added.

Next, set 1.3 for the OpenFlow version.

switch: s1:

```
root@ryu-vm:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
root@ryu-vm:~#
```

Let's check the empty flow table.

switch: s1:

```
root@ryu-vm:~# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
root@ryu-vm:~#
```

The ovs-ofctl command needs to specify the OpenFlow version to be used as an option. The default is *OpenFlow10*.

1.4.2 Executing the Switching Hub

Preparation is now done and we will run the Ryu application.

From the xterm for which the window title is "controller: c0 (root)", execute the following commands.

controller: c0:

```
root@ryu-vm:~# ryu-manager --verbose ryu.app.simple_switch_13
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13
instantiating app ryu.controller.ofp_handler
BRICK SimpleSwitch13
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPacketIn
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.GreenSocket object at 0x2e2c050> address:('127.0.0.1',
53937)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x2e2a550>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0xff9ad15b OFPSwitchFeatures(auxiliary_id=0,
capabilities=71,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
```

It may take time to connect to OVS but after you wait for a while, as shown above...

```
connected socket:<....
hello ev ...
...
move onto main mode
```

„„is displayed.

Now OVS has been connected, handshake has been performed, the Table-miss flow entry has been added and the switching hub is in the status waiting for Packet-In.

Confirm that the Table-miss flow entry has been added.

switch: s1:

```
root@ryu-vm:~# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=105.975s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER
:65535
root@ryu-vm:~#
```

The priority level is 0, no match, and CONTROLLER is specified for action, and transfer data size of 65535(0xffff = OFPCML_NO_BUFFER) is specified.

1.4.3 Confirming Operation

Execute ping from host 1 to host 2.

1. ARP request

   At this point, host 1 does not know the MAC address of host 2, therefore, before ICMP echo request, an ARP request is supposed to be broadcast. The broadcast packet is received by host 2 and host 3.

2. ARP reply

   In response to the ARP, host 2 returns an ARP reply to host 1.

3. ICMP echo request

   Now host 1 knows the MAC address of host 2, host 1 sends an echo request to host 2.

4. ICMP echo reply

   Because host 2 already knows the MAC address of host 1, host 2 returns an echo reply to host 1.

Communications like those above are supposed to take place.

Before executing the ping command, execute the tcpdump command so that it is possible to check what packets were received by each host.

host: h1:

```
root@ryu-vm:~# tcpdump -en -i h1-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

host: h2:

```
root@ryu-vm:~# tcpdump -en -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

host: h3:

```
root@ryu-vm:~# tcpdump -en -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

Use the console where the mn command is executed first, execute the following command to issue ping from host 1 to host 2.

```
mininet> h1 ping -c1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=97.5 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 97.594/97.594/97.594/0.000 ms
mininet>
```

ICMP echo reply has returned normally.

First of all, check the flow table.

switch: s1:

```
root@ryu-vm:~# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=417.838s, table=0, n_packets=3, n_bytes=182, priority=0 actions=
CONTROLLER:65535
 cookie=0x0, duration=48.444s, table=0, n_packets=2, n_bytes=140, priority=1,in_port=2,dl_dst
=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=48.402s, table=0, n_packets=1, n_bytes=42, priority=1,in_port=1,dl_dst
=00:00:00:00:00:02 actions=output:2
root@ryu-vm:~#
```

In addition to the Table-miss flow entry, tow flow entries of priority level 1 have been registered.

1. Receive port (in_port):2, Destination MAC address (dl_dst):host 1 -> Action (actions):Transfer to port 1
2. Receive port (in_port):1, Destination MAC address (dl_dst): host 2 -> Action (actions): Transfer to port 2

Entry (1) was referenced twice (n_packets) and entry (2) was referenced once. Because (1) is a communication from host 2 to host 1, ARP reply and ICMP echo reply must have matched. (2) is a communication from host 1 to host 2 and because ARP request is broadcast, this is supposed to be by ICMP echo request.

Now, let's look at the log output of simple_switch_13.

controller: c0:

```
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 00:00:00:00:00:02 1
```

The first Packet-In is the ARP request issued by host 1 and is a broadcast, the flow entry is not registered and only Packet-Out is issued.

The second one is the ARP reply returned from host 2 and because its destination MAC address is host 1, the aforementioned flow entry (1) is registered.

The third one is the ICMP echo request sent from host 1 to host 2 and flow entry (2) is registered.

The ICMP echo reply returned from host 2 to host 1 matches the already registered flow entry (1) thus is transferred to host 1 without issuing Packet-In.

Finally, let's take a look at the output of tcpdump executed on each host.

host: h1:

```
root@ryu-vm:~# tcpdump -en -i h1-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
20:38:04.625473 00:00:00:00:00:01 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42:
Request who-has 10.0.0.2 tell 10.0.0.1, length 28
20:38:04.678698 00:00:00:00:00:02 > 00:00:00:00:00:01, ethertype ARP (0x0806), length 42:
Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
20:38:04.678731 00:00:00:00:00:01 > 00:00:00:00:00:02, ethertype IPv4 (0x0800), length 98:
10.0.0.1 > 10.0.0.2: ICMP echo request, id 3940, seq 1, length 64
20:38:04.722973 00:00:00:00:00:02 > 00:00:00:00:00:01, ethertype IPv4 (0x0800), length 98:
10.0.0.2 > 10.0.0.1: ICMP echo reply, id 3940, seq 1, length 64
```

Host 1 first broadcast the ARP request and then received the ARP reply returned from host 2. Next, host 1 issued the ICMP echo request and received the ICMP echo reply returned from host 2.

host: h2:

```
root@ryu-vm:~# tcpdump -en -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

```
20:38:04.637987 00:00:00:00:00:01 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42:
Request who-has 10.0.0.2 tell 10.0.0.1, length 28
20:38:04.638059 00:00:00:00:00:02 > 00:00:00:00:00:01, ethertype ARP (0x0806), length 42:
Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
20:38:04.722601 00:00:00:00:00:01 > 00:00:00:00:00:02, ethertype IPv4 (0x0800), length 98:
10.0.0.1 > 10.0.0.2: ICMP echo request, id 3940, seq 1, length 64
20:38:04.722747 00:00:00:00:00:02 > 00:00:00:00:00:01, ethertype IPv4 (0x0800), length 98:
10.0.0.2 > 10.0.0.1: ICMP echo reply, id 3940, seq 1, length 64
```

Host 2 received the ARP request issued by host 1 and returned the ARP reply to host 1. Then, host 2 received the ICMP echo request from host 1 and returned the echo reply to host 1.

host: h3:

```
root@ryu-vm:~# tcpdump -en -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
20:38:04.637954 00:00:00:00:00:01 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42:
Request who-has 10.0.0.2 tell 10.0.0.1, length 28
```

Host 3 only received the ARP request broadcast by host 1 at first.

1.5 Conclusion

This section used implementation of a simple switching hub as material to describe the basic procedures of implementation of a Ryu application and a simple method of controlling the OpenFlow switch using OpenFlow.

CHAPTER

TWO

TRAFFIC MONITOR

This section describes how to add a function to monitor OpenFlow switch statistical information to the switching hub explained in " *Switching Hub* ".

2.1 Routine Examination of Network

Networks have already become the infrastructure of many services and businesses, so maintaining of normal and stable operation is expected. Having said that, problems always occur.

When an error occurred on network, the cause must be identified and operation restored quickly. Needless to say, in order to detect errors and identify causes, it is necessary to understand the network status on a regular basis. For example, assuming the traffic volume of a port of some network device indicates a very high value, whether it is an abnormal state or is usually that way and when it became that way cannot be determined if the port's traffic volume has not been measured continuously.

For this reason, constant monitoring of the health of a network is essential for continuous and safe operation of the services or businesses that use that network. As a matter of course, simply monitoring traffic information does not provide a perfect guarantee but this section describes how to use OpenFlow to acquire statistical information for a switch.

2.2 Implementation of Traffic Monitor

The following is source code in which a traffic monitoring function has been added to the switching hub explained in " *Switching Hub* ".

```
from operator import attrgetter

from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub


class SimpleMonitor(simple_switch_13.SimpleSwitch13):

    def __init__(self, *args, **kwargs):
        super(SimpleMonitor, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)

    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if not datapath.id in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
```

```
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]

    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)

    def _request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

        req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
        datapath.send_msg(req)

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         '
                         'in-port  eth-dst           '
                         'out-port packets  bytes')
        self.logger.info('---------------- '
                         '-------- ----------------- '
                         '-------- -------- --------')
        for stat in sorted([flow for flow in body if flow.priority == 1],
                           key=lambda flow: (flow.match['in_port'],
                                             flow.match['eth_dst'])):
            self.logger.info('%016x %8x %17s %8x %8d %8d',
                             ev.msg.datapath.id,
                             stat.match['in_port'], stat.match['eth_dst'],
                             stat.instructions[0].actions[0].port,
                             stat.packet_count, stat.byte_count)

    @set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
    def _port_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         port     '
                         'rx-pkts  rx-bytes rx-error '
                         'tx-pkts  tx-bytes tx-error')
        self.logger.info('---------------- -------- '
                         '-------- -------- -------- '
                         '-------- -------- --------')
        for stat in sorted(body, key=attrgetter('port_no')):
            self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                             ev.msg.datapath.id, stat.port_no,
                             stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                             stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```

The traffic monitor function has been implemented in the SimpleMonitor class which inherited SimpleSwitch13, therefore, there is no packet transfer-related processing here.

2.2.1 Fixed-Cycle Processing

In parallel with switching hub processing, create a thread to periodically issue a request to the OpenFlow switch to acquire statistical information.

```
from operator import attrgetter

from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
```

```
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub


class SimpleMonitor(simple_switch_13.SimpleSwitch13):

    def __init__(self, *args, **kwargs):
        super(SimpleMonitor, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)
# ...
```

There are some eventlet wrappers and basic class implementation in `ryu.lib.hub`. Here, we use `hub.spawn()`, which creates threads. The thread actually created is an eventlet green thread.

```
# ...
@set_ev_cls(ofp_event.EventOFPStateChange,
            [MAIN_DISPATCHER, DEAD_DISPATCHER])
def _state_change_handler(self, ev):
    datapath = ev.datapath
    if ev.state == MAIN_DISPATCHER:
        if not datapath.id in self.datapaths:
            self.logger.debug('register datapath: %016x', datapath.id)
            self.datapaths[datapath.id] = datapath
    elif ev.state == DEAD_DISPATCHER:
        if datapath.id in self.datapaths:
            self.logger.debug('unregister datapath: %016x', datapath.id)
            del self.datapaths[datapath.id]

def _monitor(self):
    while True:
        for dp in self.datapaths.values():
            self._request_stats(dp)
        hub.sleep(10)
# ...
```

In thread function `_monitor()`, issuance of a statistical information acquisition request for the registered switch is repeated infinitely every 10 seconds.

In order to make sure the connected switch is monitored, `EventOFPStateChange` event is used for detecting connection and disconnection. This event is issued by the Ryu framework and is issued when the Datapath state is changed.

Here, when the Datapath state becomes `MAIN_DISPATCHER`, that switch is registered as the monitor target and when it becomes `DEAD_DISPATCHER`, the registration is deleted.

```
# ...
def _request_stats(self, datapath):
    self.logger.debug('send stats request: %016x', datapath.id)
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    req = parser.OFPFlowStatsRequest(datapath)
    datapath.send_msg(req)

    req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
    datapath.send_msg(req)
# ...
```

With periodically called `_request_stats()`, `OFPFlowStatsRequest` and `OFPPortStatsRequest` are issued to the switch.

`OFPFlowStatsRequest` requests that the switch provide statistical information related to flow entry. The requested target flow entry can be narrowed down by conditions such as table ID, output port, cookie value and match but here all entries are made subject to the request.

`OFPPortStatsRequest` request that the switch provide port-related statistical information. It is possible to specify the desired port number to acquire information from. Here, `OFPP_ANY` is specified to request information from all ports.

2.2.2 FlowStats

In order to receive a response from the switch, create an event handler that receives the FlowStatsReply message.

```
# ...
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def _flow_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.info('datapath         '
                     'in-port  eth-dst           '
                     'out-port packets  bytes')
    self.logger.info('---------------- '
                     '-------- ----------------- '
                     '-------- -------- --------')
    for stat in sorted([flow for flow in body if flow.priority == 1],
                       key=lambda flow: (flow.match['in_port'],
                                         flow.match['eth_dst'])):
        self.logger.info('%016x %8x %17s %8x %8d %8d',
                         ev.msg.datapath.id,
                         stat.match['in_port'], stat.match['eth_dst'],
                         stat.instructions[0].actions[0].port,
                         stat.packet_count, stat.byte_count)
# ...
```

`OPFFlowStatsReply` class's attribute `body` is the list of `OFPFlowStats` and stores the statistical information of each flow entry, which was subject to FlowStatsRequest.

All flow entries are selected except the Table-miss flow of priority 0. The number of packets and bytes matched to the respective flow entry are output by being sorted by the received port and destination MAC address.

Here, only part of values are output to the log but in order to continuously collect and analyze information, linkage with external programs may be required. In such a case, the content of `OFPFlowStatsReply` can be converted to the JSON format.

For example, it can be written as follows:

```
import json

# ...

self.logger.info('%s', json.dumps(ev.msg.to_jsondict(), ensure_ascii=True,
                                  indent=3, sort_keys=True))
```

In this case, the output is as follows:

```
{
   "OFPFlowStatsReply": {
      "body": [
         {
            "OFPFlowStats": {
               "byte_count": 0,
               "cookie": 0,
               "duration_nsec": 680000000,
               "duration_sec": 4,
               "flags": 0,
               "hard_timeout": 0,
               "idle_timeout": 0,
               "instructions": [
                  {
                     "OFPInstructionActions": {
                        "actions": [
                           {
                              "OFPActionOutput": {
                                 "len": 16,
                                 "max_len": 65535,
                                 "port": 4294967293,
                                 "type": 0
                              }
                           }
                        ],
```

```
                    "len": 24,
                    "type": 4
                }
            }
        ],
        "length": 80,
        "match": {
            "OFPMatch": {
                "length": 4,
                "oxm_fields": [],
                "type": 1
            }
        },
        "packet_count": 0,
        "priority": 0,
        "table_id": 0
    }
},
{
    "OFPFlowStats": {
        "byte_count": 42,
        "cookie": 0,
        "duration_nsec": 72000000,
        "duration_sec": 57,
        "flags": 0,
        "hard_timeout": 0,
        "idle_timeout": 0,
        "instructions": [
            {
                "OFPInstructionActions": {
                    "actions": [
                        {
                            "OFPActionOutput": {
                                "len": 16,
                                "max_len": 65509,
                                "port": 1,
                                "type": 0
                            }
                        }
                    ],
                    "len": 24,
                    "type": 4
                }
            }
        ],
        "length": 96,
        "match": {
            "OFPMatch": {
                "length": 22,
                "oxm_fields": [
                    {
                        "OXMTlv": {
                            "field": "in_port",
                            "mask": null,
                            "value": 2
                        }
                    },
                    {
                        "OXMTlv": {
                            "field": "eth_dst",
                            "mask": null,
                            "value": "00:00:00:00:00:01"
                        }
                    }
                ],
                "type": 1
            }
        },
        "packet_count": 1,
        "priority": 1,
        "table_id": 0
    }
}
```

```
        ],
        "flags": 0,
        "type": 1
    }
}
```

2.2.3 PortStats

In order to receive a response from the switch, create an event handler that receives the PortStatsReply message.

```
# ...
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
    body = ev.msg.body

    self.logger.info('datapath         port     '
                     'rx-pkts  rx-bytes rx-error '
                     'tx-pkts  tx-bytes tx-error')
    self.logger.info('---------------- -------- '
                     '-------- -------- -------- '
                     '-------- -------- --------')
    for stat in sorted(body, key=attrgetter('port_no')):
        self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                         ev.msg.datapath.id, stat.port_no,
                         stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                         stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```

`OPFPortStatsReply` class's attribute `body` is the list of `OFPPortStats`.

`OFPPortStats` stores statistical information such as port numbers, send/receive packet count, respectively, byte count, drop count, error count, frame error count, overrun count, CRC error count, and collision count.

Here, being sorted by port number, receive packet count, receive byte count, receive error count, send packet count, send byte count, and send error count are output.

2.3 Executing Traffic Monitor

Now, let's actually execute this traffic monitor.

First of all, as with " *Switching Hub* ", execute Mininet. Do not forget to set OpenFlow13 for the OpenFlow version.

Next, finally, let's execute the traffic monitor.

controller: c0:

```
ryu@ryu-vm:~# ryu-manager --verbose ./simple_monitor.py
loading app ./simple_monitor.py
loading app ryu.controller.ofp_handler
instantiating app ./simple_monitor.py
instantiating app ryu.controller.ofp_handler
BRICK SimpleMonitor
  CONSUMES EventOFPStateChange
  CONSUMES EventOFPFlowStatsReply
  CONSUMES EventOFPPortStatsReply
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPStateChange TO {'SimpleMonitor': set(['main', 'dead'])}
  PROVIDES EventOFPFlowStatsReply TO {'SimpleMonitor': set(['main'])}
  PROVIDES EventOFPPortStatsReply TO {'SimpleMonitor': set(['main'])}
  PROVIDES EventOFPPacketIn TO {'SimpleMonitor': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'SimpleMonitor': set(['config'])}
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
```

```
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.GreenSocket object at 0x343fb10> address:('127.0.0.1',
55598)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x343fed0>
move onto config mode
EVENT ofp_event->SimpleMonitor EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0x7dd2dc58 OFPSwitchFeatures(auxiliary_id=0,
capabilities=71,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
EVENT ofp_event->SimpleMonitor EventOFPStateChange
register datapath: 0000000000000001
send stats request: 0000000000000001
EVENT ofp_event->SimpleMonitor EventOFPFlowStatsReply
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
EVENT ofp_event->SimpleMonitor EventOFPPortStatsReply
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1        0        0        0        0        0        0
0000000000000001        2        0        0        0        0        0        0
0000000000000001        3        0        0        0        0        0        0
0000000000000001 fffffffe        0        0        0        0        0        0
```

In " *Switching Hub* ", the SimpleSwitch13 module name (ryu.app.simple_switch_13) was specified for the ryu-manager command. However, the SimpleMonitor file name (./simple_monitor.py) is specified here.

At this point, there is no flow entry (Table-miss flow entry is not displayed) and the count of each port is all 0.

Let's execute ping from host 1 to host 2.

host: h1:

```
root@ryu-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=94.4 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 94.489/94.489/94.489/0.000 ms
root@ryu-vm:~#
```

Packet transfer and flow entry are registered and statistical information is changed.

controller: c0:

```
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000001        1 00:00:00:00:00:02        2        1       42
0000000000000001        2 00:00:00:00:00:01        1        2      140
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1        3      182        0        3      182        0
0000000000000001        2        3      182        0        3      182        0
0000000000000001        3        0        0        0        1       42        0
0000000000000001 fffffffe        0        0        0        1       42        0
```

According to the flow entry statistical information, traffic matched to the receive port 1's flow is recorded as 1 packet, 42 bytes. With receive port 2, it is 2 packets, 140 bytes.

According to the port statistical information, the receive packet count (rx-pkts) of port 1 is 3, the receive byte count (rx-bytes) is 182 bytes. With port 2, it is 3 packets and 182 bytes, respectively.

Figures do not match between the statistical information of flow entry and that of port. The reason for that is because the flow entry statistical information is the information of packets that match the entry and were transferred. That means, packets for which Packet-In is issued by Table-miss and are transferred by Packet-Out are not included in these statistics.

In this case, three packets that is the ARP request first broadcast by host 1, the ARP reply returned by host 2 to host 1, and the echo request issued by host 1 to host 2, are transferred by Packet-Out. For the above reason, the amount of port statistics is larger than that of flow entry.

2.4 Conclusion

The section described the following items using a statistical information acquisition function as material.

- Thread generation method by Ryu application
- Capturing of Datapath status changes
- How to acquire FlowStats and PortStats

CHAPTER

THREE

REST LINKAGE

This section describes how to add a REST link function to the switching hub explained in " *Switching Hub*".

3.1 Integrating REST API

Ryu has a Web server function corresponding to WSGI. By using this function, it is possible to create a REST API, which is useful to link with other systems or browsers.

Note: WSGI means a unified framework for connecting Web applications and Web servers in Python.

3.2 Implementing a Switching Hub with REST API

Let's add the following two REST APIs to the switching hub explained in " *Switching Hub* ".

1. MAC address table acquisition API

   Returns the content of the MAC address table held by the switching hub. Returns a pair of MAC address and port number in JSON format.

2. MAC address table registration API

   Registers a pair of MAC address and port number in the MAC address table and adds a flow entry to the switch.

Let's take a look at the source code.

```
import json
import logging

from ryu.app import simple_switch_13
from webob import Response
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.app.wsgi import ControllerBase, WSGIApplication, route
from ryu.lib import dpid as dpid_lib

simple_switch_instance_name = 'simple_switch_api_app'
url = '/v1/simpleswitch/mactable/{dpid}'

class SimpleSwitchRest13(simple_switch_13.SimpleSwitch13):

    _CONTEXTS = { 'wsgi': WSGIApplication }

    def __init__(self, *args, **kwargs):
        super(SimpleSwitchRest13, self).__init__(*args, **kwargs)
        self.switches = {}
        wsgi = kwargs['wsgi']
        wsgi.register(SimpleSwitchController, {simple_switch_instance_name : self})

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
```

```
    def switch_features_handler(self, ev):
        super(SimpleSwitchRest13, self).switch_features_handler(ev)
        datapath = ev.msg.datapath
        self.switches[datapath.id] = datapath
        self.mac_to_port.setdefault(datapath.id, {})

    def set_mac_to_port(self, dpid, entry):
        mac_table = self.mac_to_port.setdefault(dpid, {})
        datapath = self.switches.get(dpid)

        entry_port = entry['port']
        entry_mac = entry['mac']

        if datapath is not None:
            parser = datapath.ofproto_parser
            if entry_port not in mac_table.values():

                for mac, port in mac_table.items():

                    # from known device to new device
                    actions = [parser.OFPActionOutput(entry_port)]
                    match = parser.OFPMatch(in_port=port, eth_dst=entry_mac)
                    self.add_flow(datapath, 1, match, actions)

                    # from new device to known device
                    actions = [parser.OFPActionOutput(port)]
                    match = parser.OFPMatch(in_port=entry_port, eth_dst=mac)
                    self.add_flow(datapath, 1, match, actions)

                mac_table.update({entry_mac : entry_port})
        return mac_table

class SimpleSwitchController(ControllerBase):

    def __init__(self, req, link, data, **config):
        super(SimpleSwitchController, self).__init__(req, link, data, **config)
        self.simpl_switch_spp = data[simple_switch_instance_name]

    @route('simpleswitch', url, methods=['GET'], requirements={'dpid': dpid_lib.DPID_PATTERN})
    def list_mac_table(self, req, **kwargs):

        simple_switch = self.simpl_switch_spp
        dpid = dpid_lib.str_to_dpid(kwargs['dpid'])

        if dpid not in simple_switch.mac_to_port:
            return Response(status=404)

        mac_table = simple_switch.mac_to_port.get(dpid, {})
        body = json.dumps(mac_table)
        return Response(content_type='application/json', body=body)

    @route('simpleswitch', url, methods=['PUT'], requirements={'dpid': dpid_lib.DPID_PATTERN})
    def put_mac_table(self, req, **kwargs):

        simple_switch = self.simpl_switch_spp
        dpid = dpid_lib.str_to_dpid(kwargs['dpid'])
        new_entry = eval(req.body)

        if dpid not in simple_switch.mac_to_port:
            return Response(status=404)

        try:
            mac_table = simple_switch.set_mac_to_port(dpid, new_entry)
            body = json.dumps(mac_table)
            return Response(content_type='application/json', body=body)
        except Exception as e:
            return Response(status=500)
```

With simple_switch_rest_13.py, two classes are defined.

The first class is controller class `SimpleSwitchController`, which defines the URL to receive the HTTP request and its corresponding method.

The second class is `SimpleSwitchRest13`, which is extension of " *Switching Hub* ", to be able to update the MAC address table.

With `SimpleSwitchRest13`, because flow entry is added to the switch, the FeaturesReply method is overridden and holds the datapath object.

3.3 Implementing SimpleSwitchRest13 Class

```
class SimpleSwitchRest13(simple_switch_13.SimpleSwitch13):

    _CONTEXTS = { 'wsgi': WSGIApplication }
...
```

Class variable `_CONTEXT` is used to specify Ryu's WSGI-compatible Web server class. By doing so, WSGI's Web server instance can be acquired by a key called the `wsgi` key.

```
def __init__(self, *args, **kwargs):
    super(SimpleSwitchRest13, self).__init__(*args, **kwargs)
    self.switches = {}
    wsgi = kwargs['wsgi']
    wsgi.register(SimpleSwitchController, {simple_switch_instance_name : self})
...
```

Constructor acquires the instance of `WSGIApplication` in order to register the controller class, which is explained in a later section. For registration, the `register` method is used. When executing the `register` method, the dictionary object is passed in the key name `simple_switch_api_app` so that the constructor of the controller can access the instance of the `SimpleSwitchRest13` class.

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    super(SimpleSwitchRest13, self).switch_features_handler(ev)
    datapath = ev.msg.datapath
    self.switches[datapath.id] = datapath
    self.mac_to_port.setdefault(datapath.id, {})
...
```

Parent class `switch_features_handler` is overridden. This method, upon rising of the SwitchFeatures event, acquires the `datapath` object stored in event object `ev` and stores it in instance variable `switches`. Also, at this time, an empty dictionary is set as the initial value in the MAC address table.

```
def set_mac_to_port(self, dpid, entry):
    mac_table = self.mac_to_port.setdefault(dpid, {})
    datapath = self.switches.get(dpid)

    entry_port = entry['port']
    entry_mac = entry['mac']

    if datapath is not None:
        parser = datapath.ofproto_parser
        if entry_port not in mac_table.values():

            for mac, port in mac_table.items():

                # from known device to new device
                actions = [parser.OFPActionOutput(entry_port)]
                match = parser.OFPMatch(in_port=port, eth_dst=entry_mac)
                self.add_flow(datapath, 1, match, actions)

                # from new device to known device
                actions = [parser.OFPActionOutput(port)]
                match = parser.OFPMatch(in_port=entry_port, eth_dst=mac)
                self.add_flow(datapath, 1, match, actions)

            mac_table.update({entry_mac : entry_port})
    return mac_table
...
```

This method registers the MAC address and port to the specified switch. The method is executed when REST API is called by the PUT method.

In argument `entry`. a pair of the desired MAC address and connection port is stored.

While referencing the MAC address table `self.mac_to_port` information, the flow entry to be registered in the switch is searched.

For example, a pair of the following MAC address and connection port is registered in the MAC address table,

- 00:00:00:00:00:01, 1

and a pair of the MAC address and port passed by the argument `entry` is

- 00:00:00:00:00:02, 2

, so the flow entry necessary to register in the switch is as follows:

- Matching condition: in_port = 1, dst_mac = 00:00:00:00:00:02 Action: output=2
- Matching condition: in_port = 2, dst_mac = 00:00:00:00:00:01 Action: output=1

To register flow entry, the parent class `add_flow` method is used. At the end, the information passed by argument `entry` is stored in the MAC address table.

3.4 Implementing SimpleSwitchController Class

Next, let's talk about the controller class that accepts HTTP requests to REST API. The class name is `SimpleSwitchController`.

```
class SimpleSwitchController(ControllerBase):
    def __init__(self, req, link, data, **config):
        super(SimpleSwitchController, self).__init__(req, link, data, **config)
        self.simpl_switch_spp = data[simple_switch_instance_name]
...
```

The instance of the `SimpleSwitchRest13` class is acquired by the contractor.

```
@route('simpleswitch', url, methods=['GET'], requirements={'dpid': dpid_lib.DPID_PATTERN})
def list_mac_table(self, req, **kwargs):

    simple_switch = self.simpl_switch_spp
    dpid = dpid_lib.str_to_dpid(kwargs['dpid'])

    if dpid not in simple_switch.mac_to_port:
        return Response(status=404)

    mac_table = simple_switch.mac_to_port.get(dpid, {})
    body = json.dumps(mac_table)
    return Response(content_type='application/json', body=body)
...
```

This part is to implement REST API's URL and its corresponding processing. To associate this method and URL, the `route` decorator defined in Ryu is used.

The content specified by the decorator is as follows:

- First argument

  Any name

- Second argument

  Specify URL. Make URL to be `http://<server IP>:8080/simpleswitch/mactable/<data path ID>`.

- Third argument

  Specify the HTTP method. Specify the GET method.

- Fourth argument

  Specify the format of the specified location. The condition is that the {dpid} part of the URL(/simpleswitch/mactable/{dpid}) matches the expression of a 16-digit hex value defined by `DPID_PATTERN` of ryu/lib/dpid.py.

The REST API is called by the URL specified by the second argument. If the HTTP method at that time is GET, the `list_mac_table` method is called. This method acquires the MAC address table corresponding to the data path ID specified in the {dpid} part, converts it to the JSON format and returns it to the caller.

If the data path ID of an unknown switch, which is not connected to Ryu, is specified, response code 404 is returned.

```
@route('simpleswitch', url, methods=['PUT'], requirements={'dpid': dpid_lib.DPID_PATTERN})
def put_mac_table(self, req, **kwargs):

    simple_switch = self.simpl_switch_spp
    dpid = dpid_lib.str_to_dpid(kwargs['dpid'])
    new_entry = eval(req.body)

    if dpid not in simple_switch.mac_to_port:
        return Response(status=404)

    try:
        mac_table = simple_switch.set_mac_to_port(dpid, new_entry)
        body = json.dumps(mac_table)
        return Response(content_type='application/json', body=body)
    except Exception as e:
        return Response(status=500)
...
```

Let's talk about REST API that registers MAC address table.

URL is the same as API when the MAC address table is acquired but when the HTTP method is PUT, the `put_mac_table` method is called. With this method, the `set_mac_to_port` method of the switching hub instance is called inside. When an exception is raised inside the `put_mac_table` method, response code 500 is returned. Also, as with the `list_mac_table` method, if the data path ID of an unknown switch, which is not connected to Ryu, is specified, response code 404 is returned.

3.5 Executing REST API Added Switching Hub

Let's execute the switching hub to which REST API has been added.

First, as with " *Switching Hub* ", execute Mininet. Here again, don't forget to set OpenFlow13 for the OpenFlow version. Then, start the switching hub added with REST API.

```
ryu@ryu-vm:~/ryu/ryu/app$ cd ~/ryu/ryu/app
ryu@ryu-vm:~/ryu/ryu/app$ sudo ovs-vsctl set Bridge s1 protocols=OpenFlow13
ryu@ryu-vm:~/ryu/ryu/app$ ryu-manager --verbose ./simple_switch_rest_13.py
loading app ./simple_switch_rest_13.py
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app ryu.controller.ofp_handler
instantiating app ./simple_switch_rest_13.py
BRICK SimpleSwitchRest13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'SimpleSwitchRest13': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitchRest13': set(['config'])}
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPHello
(31135) wsgi starting up on http://0.0.0.0:8080/
connected socket:<eventlet.greenio.GreenSocket object at 0x318c6d0> address:('127.0.0.1',
48914)
```

```
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x318cc10>
move onto config mode
EVENT ofp_event->SimpleSwitchRest13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0x78dd7a72 OFPSwitchFeatures(auxiliary_id=0,
capabilities=71,datapath_id=1,n_buffers=256,n_tables=254)
move onto main mode
```

In the message at the time of start, there is a line stating "(31135) wsgi starting up on http://0.0.0.0:8080/" and this indicates that the Web server started at port number 8080.

Next, issue a ping from host 1 to host 2 on the mininet shell.

```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=84.1 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 84.171/84.171/84.171/0.000 ms
```

At this time, Packet-In to Ryu occurred three times.

```
EVENT ofp_event->SimpleSwitchRest13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->SimpleSwitchRest13 EventOFPPacketIn
packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
EVENT ofp_event->SimpleSwitchRest13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 00:00:00:00:00:02 1
```

Let's execute REST API that acquires the MAC table of the switching hub. This time, use the curl command to call REST API.

```
ryu@ryu-vm:~$ curl -X GET http://127.0.0.1:8080/simpleswitch/mactable/0000000000000001
{"00:00:00:00:00:02": 2, "00:00:00:00:00:01": 1}
```

You can find that two hosts host 1 and host 2 have been learned on the MAC address table.

This time, store the two hosts, host 1 and host 2, in the MAC address table beforehand and execute ping. Temporarily stop the switching hub and Mininet once. Then, start Mininet again, set the OpenFlow version to OpenFlow13 and then start the switching hub.

```
...
(26759) wsgi starting up on http://0.0.0.0:8080/
connected socket:<eventlet.greenio.GreenSocket object at 0x2afe6d0> address:('127.0.0.1',
48818)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x2afec10>
move onto config mode
EVENT ofp_event->SimpleSwitchRest13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0x96681337 OFPSwitchFeatures(auxiliary_id=0,
capabilities=71,datapath_id=1,n_buffers=256,n_tables=254)
switch_features_handler inside sub class
move onto main mode
```

Next, call REST API for updating of the MAC address table for each host. The data format when calling REST API shall be {"mac" : "MAC address", "port" : Connection port number}.

```
ryu@ryu-vm:~$ curl -X PUT -d '{"mac" : "00:00:00:00:00:01", "port" : 1}' http
://127.0.0.1:8080/simpleswitch/mactable/0000000000000001
{"00:00:00:00:00:01": 1}
ryu@ryu-vm:~$ curl -X PUT -d '{"mac" : "00:00:00:00:00:02", "port" : 2}' http
://127.0.0.1:8080/simpleswitch/mactable/0000000000000001
{"00:00:00:00:00:02": 2, "00:00:00:00:00:01": 1}
```

When those commands are executed, the flow entry corresponding to host 1 and host 2 are registered.

Now, let's execute a ping from host 1 to host 2.

```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=4.62 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.623/4.623/4.623/0.000 ms
```

```
...
move onto main mode
(28293) accepted ('127.0.0.1', 44453)
127.0.0.1 - - [19/Nov/2013 19:59:45] "PUT /simpleswitch/mactable/0000000000000001 HTTP/1.1"
200 124 0.002734
EVENT ofp_event->SimpleSwitchRest13 EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
```

At this time, the flow entry exists for the switches, Packet-In only occurs when an ARP request is sent from host 1 to host 2 and is not raised for subsequent packet exchange.

3.6 Conclusion

This section used a function to reference or update the MAC address table as material to explain how to add REST API. As another practical application, it may be a good idea to create REST API that can add the desired flow entry to a switch and make it possible to operate from a browser.

CHAPTER

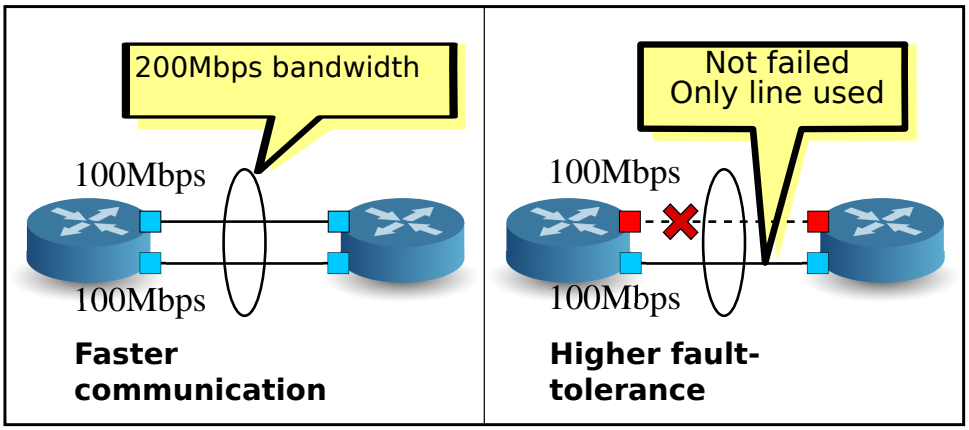
FOUR

LINK AGGREGATION

This section describes how to implement the link aggregation function using Ryu.

4.1 Link Aggregation

Link aggregation is a technology defined in IEEE802.1AX-2008, which combines multiple physical lines to be used as a logical link. With the link aggregation function, it is possible to increase communication speed between specific network devices. At the same time, by securing redundancy, it is possible to improve fault tolerance.



In order to use the link aggregation function, it is necessary to configure beforehand the respective network devices as to which interfaces are aggregated as one group.

There are two methods used to start the link aggregation function, the static method, in which each each network device is instructed directly, and the dynamic method, in which the function is started dynamically using the protocol called Link Aggregation Control Protocol (LACP).

When the dynamic method is adopted, counterpart interfaces of the network devices periodically exchange LACP data units to continuously check with each other that communication is available. When exchange of LACP data units is interrupted, the occurrence of a failure is assumed and the relevant network device becomes unavailable. As a result, sending or receiving of packets is only performed by the remaining interfaces. This method has the advantage that even when a relay device such as a media converter is installed between network devices, link down of the other side of the relay device can be detected. This chapter discusses the dynamic link aggregation function using LACP.

4.2 Executing the Ryu Application

Let's put off explaining the source and first execute Ryu's link aggregation application.

simple_switch_lacp.py provided in Ryu's source tree is an application dedicated to OpenFlow 1.0, therefore, we will create simple_switch_lacp_13.py, which supports OpenFlow 1.3. This program is an application to which the link aggregation function has been added to the switching hub of " *Switching Hub*."

Source name: `simple_switch_lacp_13.py`

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib import lacplib
from ryu.lib.dpid import str_to_dpid
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet


class SimpleSwitchLacp13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'lacplib': lacplib.LacpLib}

    def __init__(self, *args, **kwargs):
        super(SimpleSwitchLacp13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self._lacp = kwargs['lacplib']
        self._lacp.add(
            dpid=str_to_dpid('0000000000000001'), ports=[1, 2])

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]

        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

    def del_flow(self, datapath, match):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        mod = parser.OFPFlowMod(datapath=datapath,
                                command=ofproto.OFPFC_DELETE,
                                out_port=ofproto.OFPP_ANY,
                                out_group=ofproto.OFPG_ANY,
                                match=match)
        datapath.send_msg(mod)

    @set_ev_cls(lacplib.EventPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
```

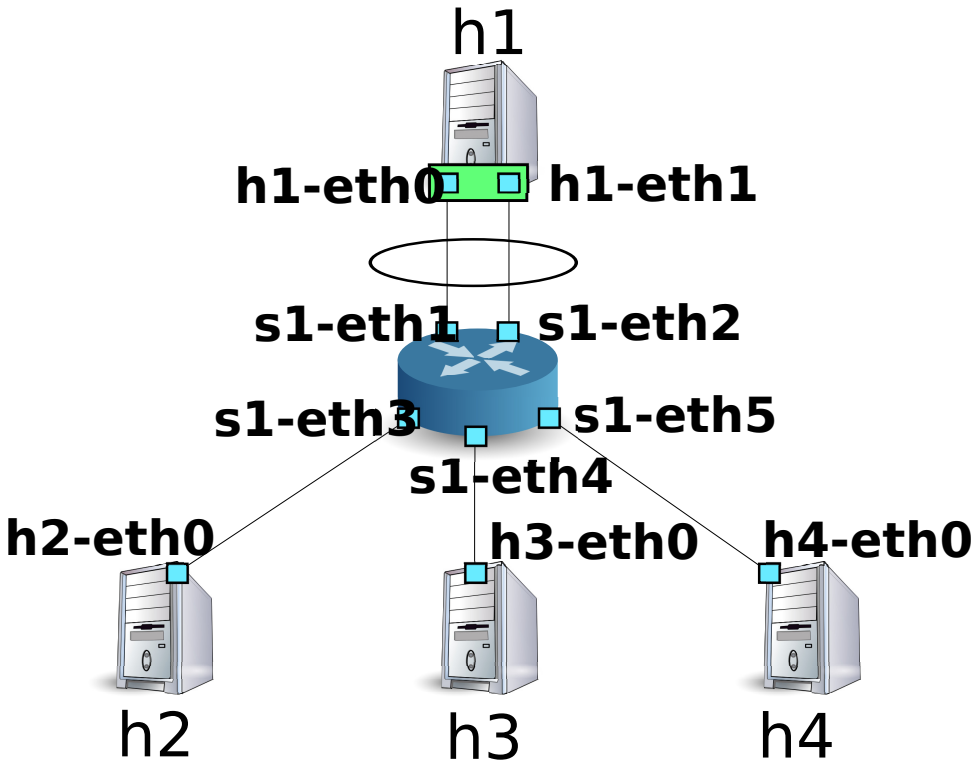
```
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            self.add_flow(datapath, 1, match, actions)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions, data=data)
        datapath.send_msg(out)

    @set_ev_cls(lacplib.EventSlaveStateChanged, MAIN_DISPATCHER)
    def _slave_state_changed_handler(self, ev):
        datapath = ev.datapath
        dpid = datapath.id
        port_no = ev.port
        enabled = ev.enabled
        self.logger.info("slave state changed port: %d enabled: %s",
                         port_no, enabled)
        if dpid in self.mac_to_port:
            for mac in self.mac_to_port[dpid]:
                match = datapath.ofproto_parser.OFPMatch(eth_dst=mac)
                self.del_flow(datapath, match)
            del self.mac_to_port[dpid]
        self.mac_to_port.setdefault(dpid, {})
```

4.2.1 Configuring an Experimental Environment

Let's configure a link aggregation between the OpenFlow switch and Linux host.

For details on the environment setting and login method, etc. to use the VM images, refer to " *Switching Hub*."

First of all, using Mininet, create the topology shown below.



Create a script to call Mininet's API and configure the necessary topology.

Source name: `link_aggregation.py`

```
#!/usr/bin/env python

from mininet.cli import CLI
from mininet.link import Link
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.term import makeTerm

if '__main__' == __name__:
    net = Mininet(controller=RemoteController)

    c0 = net.addController('c0')

    s1 = net.addSwitch('s1')

    h1 = net.addHost('h1')
    h2 = net.addHost('h2', mac='00:00:00:00:00:22')
    h3 = net.addHost('h3', mac='00:00:00:00:00:23')
    h4 = net.addHost('h4', mac='00:00:00:00:00:24')

    Link(s1, h1)
    Link(s1, h1)
    Link(s1, h2)
    Link(s1, h3)
    Link(s1, h4)

    net.build()
    c0.start()
    s1.start([c0])

    net.terms.append(makeTerm(c0))
    net.terms.append(makeTerm(s1))
    net.terms.append(makeTerm(h1))
    net.terms.append(makeTerm(h2))
    net.terms.append(makeTerm(h3))
    net.terms.append(makeTerm(h4))

    CLI(net)
```

```
    net.stop()
```

By executing this script, a topology is created in which two links exist between host h1 and switch s1. It is possible to use the net command to check the created topology.

```
ryu@ryu-vm:~$ sudo ./link_aggregation.py
Unable to contact the remote controller at 127.0.0.1:6633
mininet> net
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h1-eth1 s1-eth3:h2-eth0 s1-eth4:h3-eth0 s1-eth5:h4-eth0
h1 h1-eth0:s1-eth1 h1-eth1:s1-eth2
h2 h2-eth0:s1-eth3
h3 h3-eth0:s1-eth4
h4 h4-eth0:s1-eth5
mininet>
```

4.2.2 Setting Link Aggregation in Host h1

Make necessary settings on Linux of host h1 beforehand. About command input in this section, input them on xterm of host h1.

First of all, load the driver module to perform link aggregation. In Linux, the link aggregation function is taken care of by the bonding driver. Create the /etc/modprobe.d/bonding.conf configuration file beforehand.

File name: `/etc/modprobe.d/bonding.conf`

```
alias bond0 bonding
options bonding mode=4
```

Node: h1:

```
root@ryu-vm:~# modprobe bonding
```

mode=4 indicates that dynamic link aggregation is performed using LACP. Setting is omitted here because it is the default but it has been set so that the exchange interval of the LACP data units is SLOW (30-second intervals) and the sort logic is based on the destination MAC address.

Next, create a new logical interface named bond0. Also, set an appropriate value for the MAC address of bond0.

Node: h1:

```
root@ryu-vm:~# ip link add bond0 type bond
root@ryu-vm:~# ip link set bond0 address 02:01:02:03:04:08
```

Add the physical interfaces of h1-eth0 and h1-eth1 to the created local interface group. At that time, you need to make the physical interface to have been down. Also, rewrite the MAC address of the physical interface, which was randomly decided, to an easy-to-understand value beforehand.

Node: h1:

```
root@ryu-vm:~# ip link set h1-eth0 down
root@ryu-vm:~# ip link set h1-eth0 address 00:00:00:00:00:11
root@ryu-vm:~# ip link set h1-eth0 master bond0
root@ryu-vm:~# ip link set h1-eth1 down
root@ryu-vm:~# ip link set h1-eth1 address 00:00:00:00:00:12
root@ryu-vm:~# ip link set h1-eth1 master bond0
```

Assign an IP address to the logical interface. Here, let's assign 10.0.0.1. Because an IP address has been assigned to h1-eth0, delete this address.

Node: h1:

```
root@ryu-vm:~# ip addr add 10.0.0.1/8 dev bond0
root@ryu-vm:~# ip addr del 10.0.0.1/8 dev h1-eth0
```

Finally, make the logical interface up.

Node: h1:

```
root@ryu-vm:~# ip link set bond0 up
```

Now, let's check the state of each interface.

Node: h1:

```
root@ryu-vm:~# ifconfig
bond0     Link encap:Ethernet  HWaddr 02:01:02:03:04:08
          inet addr:10.0.0.1  Bcast:0.0.0.0  Mask:255.0.0.0
          UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:1240 (1.2 KB)

h1-eth0   Link encap:Ethernet  HWaddr 02:01:02:03:04:08
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:620 (620.0 B)

h1-eth1   Link encap:Ethernet  HWaddr 02:01:02:03:04:08
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:620 (620.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

You can see that logical interface bond0 is the MASTER and physical interface h1-eth0 and h1-eth1 are the SLAVE. Also, you can see that all of the MAC addresses of bond0, h1-eth0, and h1-eth1 are the same.

Check the state of the bonding driver as well.

Node: h1:

```
root@ryu-vm:~# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: slow
Min links: 0
Aggregator selection policy (ad_select): stable
Active Aggregator Info:
        Aggregator ID: 1
        Number of ports: 1
        Actor Key: 33
        Partner Key: 1
        Partner Mac Address: 00:00:00:00:00:00

Slave Interface: h1-eth0
MII Status: up
Speed: 10000 Mbps
Duplex: full
```

```
Link Failure Count: 0
Permanent HW addr: 00:00:00:00:00:11
Aggregator ID: 1
Slave queue ID: 0

Slave Interface: h1-eth1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:00:00:00:00:12
Aggregator ID: 2
Slave queue ID: 0
```

You can check the exchange intervals (LACP rate: slow) of the LACP data units and sort logic setting (Transmit Hash Policy: layer2 (0)). You can also check the MAC address of the physical interfaces h1-eth0 and h1-eth1.

Now pre-setting for host h1 has been completed.

4.2.3 Setting OpenFlow Version

Set the OpenFlow version of switch s1 to 1.3. Input this command on xterm of switch s1.

Node: s1:

```
root@ryu-vm:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

4.2.4 Executing the Switching Hub

This completes the preparation portion so let's move on to executing the Ryu application created at the beginning of the document.

Execute the following commands on xterm having the window title "Node: c0 (root)".

Node: c0:

```
ryu@ryu-vm:~$ ryu-manager ./simple_switch_lacp_13.py
loading app ./simple_switch_lacp_13.py
loading app ryu.controller.ofp_handler
creating context lacplib
instantiating app ./simple_switch_lacp_13.py
instantiating app ryu.controller.ofp_handler
...
```

Host h1 sends one LACP data unit every 30 seconds. A while after start, the switch receives the LACP data unit from host h1 and outputs it to the operation log.

Node: c0:

```
...
[LACP][INFO] SW=0000000000000001 PORT=1 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=1 the slave i/f has just been up.
[LACP][INFO] SW=0000000000000001 PORT=1 the timeout time has changed.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP sent.
slave state changed port: 1 enabled: True
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 the slave i/f has just been up.
[LACP][INFO] SW=0000000000000001 PORT=2 the timeout time has changed.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
slave state changed port: 2 enabled: True
...
```

The log indicates the following items:

- LACP received.

  An LACP data unit was received.

- the slave i/f has just been up.

  The port, which was in a disabled state, was enabled.

- the timeout time has changed.

  The communication monitoring time of the LACP data unit was changed (in this case, the default state of 0 seconds was changed to LONG_TIMEOUT_TIME 90 seconds).

- LACP sent.

  The response LACP data unit was sent.

- slave state changed ...

  The application received an `EventSlaveStateChanged` event from the LACP library (details of the event are explained later).

The switch sends response LACP data unit each time it receives LACP data unit from host h1.

Node: c0:

```
...
[LACP][INFO] SW=0000000000000001 PORT=1 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
...
```

Let's check flow entry.

Node: s1:

```
root@ryu-vm:~# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=14.565s, table=0, n_packets=1, n_bytes=124, idle_timeout=90,
send_flow_rem priority=65535,in_port=2,dl_src=00:00:00:00:00:12,dl_type=0x8809 actions=
CONTROLLER:65509
 cookie=0x0, duration=14.562s, table=0, n_packets=1, n_bytes=124, idle_timeout=90,
send_flow_rem priority=65535,in_port=1,dl_src=00:00:00:00:00:11,dl_type=0x8809 actions=
CONTROLLER:65509
 cookie=0x0, duration=24.821s, table=0, n_packets=2, n_bytes=248, priority=0 actions=
CONTROLLER:65535
```

In the switch,

- The Packet-In message is sent when the LACP data unit (ethertype is 0x8809) is sent from h1's h1-eth1 (the input port is s1-eth2 and the MAC address is 00:00:00:00:00:12).
- The Packet-In message is sent when the LACP data unit (ethertype is 0x8809) is sent from h1's h1-eth0 (the input port is s1-eth1 and the MAC address is 00:00:00:00:00:11)
- The same Table-miss flow entry as that of " *Switching Hub*".

The above three flow entries have been registered.

4.2.5 Checking the Link Aggregation Function

Improving Communication Speed

First of all, check improvement in the communication speed as a result of link aggregation. Let's take a look at the ways of using different links depending on communication.

First, execute ping from host h2 to host h1.

Node: h2:

```
root@ryu-vm:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_req=1 ttl=64 time=93.0 ms
64 bytes from 10.0.0.1: icmp_req=2 ttl=64 time=0.266 ms
64 bytes from 10.0.0.1: icmp_req=3 ttl=64 time=0.075 ms
64 bytes from 10.0.0.1: icmp_req=4 ttl=64 time=0.065 ms
...
```

While continuing to send pings, check the flow entry of switch s1.

Node: s1:

```
root@ryu-vm:~# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=22.05s, table=0, n_packets=1, n_bytes=124, idle_timeout=90,
send_flow_rem priority=65535,in_port=2,dl_src=00:00:00:00:00:12,dl_type=0x8809 actions=
CONTROLLER:65509
 cookie=0x0, duration=22.046s, table=0, n_packets=1, n_bytes=124, idle_timeout=90,
send_flow_rem priority=65535,in_port=1,dl_src=00:00:00:00:00:11,dl_type=0x8809 actions=
CONTROLLER:65509
 cookie=0x0, duration=33.046s, table=0, n_packets=6, n_bytes=472, priority=0 actions=
CONTROLLER:65535
 cookie=0x0, duration=3.259s, table=0, n_packets=3, n_bytes=294, priority=1,in_port=3,dl_dst
=02:01:02:03:04:08 actions=output:1
 cookie=0x0, duration=3.262s, table=0, n_packets=4, n_bytes=392, priority=1,in_port=1,dl_dst
=00:00:00:00:00:22 actions=output:3
```

After the previous check point, two flow entries have been added. They are the 4th and 5th entries with a small duration value.

The respective flow entry is as follows:

- When a packet address to bond0 of h1 is received from the 3rd port (s1-eth3, that is, the counterpart interface of h2), it is output from the first port (s1-eth1).
- When a packet addressed to h2 is received from the 1st port (s1-eth1), it is output from the 3rd port (s1-eth3).

You can tell that s1-eth1 is used for communication between h2 and h1.

Next, execute ping from host h3 to host h1.

Node: h3:

```
root@ryu-vm:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_req=1 ttl=64 time=91.2 ms
64 bytes from 10.0.0.1: icmp_req=2 ttl=64 time=0.256 ms
64 bytes from 10.0.0.1: icmp_req=3 ttl=64 time=0.057 ms
64 bytes from 10.0.0.1: icmp_req=4 ttl=64 time=0.073 ms
...
```

While continuing to send pings, check the flow entry of switch s1.

Node: s1:

```
root@ryu-vm:~# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=99.765s, table=0, n_packets=4, n_bytes=496, idle_timeout=90,
send_flow_rem priority=65535,in_port=2,dl_src=00:00:00:00:00:12,dl_type=0x8809 actions=
CONTROLLER:65509
 cookie=0x0, duration=99.761s, table=0, n_packets=4, n_bytes=496, idle_timeout=90,
send_flow_rem priority=65535,in_port=1,dl_src=00:00:00:00:00:11,dl_type=0x8809 actions=
CONTROLLER:65509
 cookie=0x0, duration=110.761s, table=0, n_packets=10, n_bytes=696, priority=0 actions=
CONTROLLER:65535
 cookie=0x0, duration=80.974s, table=0, n_packets=82, n_bytes=7924, priority=1,in_port=3,
dl_dst=02:01:02:03:04:08 actions=output:1
 cookie=0x0, duration=2.677s, table=0, n_packets=2, n_bytes=196, priority=1,in_port=2,dl_dst
=00:00:00:00:00:23 actions=output:4
 cookie=0x0, duration=2.675s, table=0, n_packets=1, n_bytes=98, priority=1,in_port=4,dl_dst
=02:01:02:03:04:08 actions=output:2
```

```
 cookie=0x0, duration=80.977s, table=0, n_packets=83, n_bytes=8022, priority=1,in_port=1,
dl_dst=00:00:00:00:00:22 actions=output:3
```

After the previous check point, two flow entries have been added. They are the 5th and 6th entries with a small duration value.

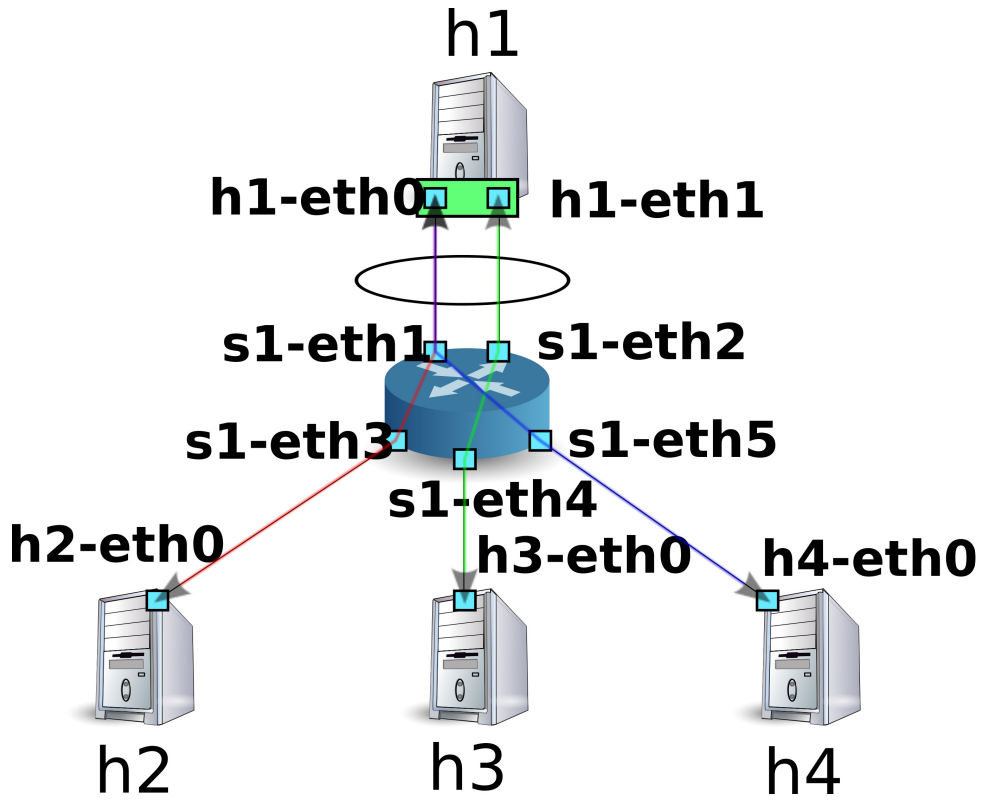
The respective flow entry is as follows:

- When a packet addressed to h3 is received from the 2nd port (s1-eth2), it is output from the 4th port (s1-eth4).
- When a packet address to bond0 of h1 is received from the 4th port (s1-eth4, that is, the counterpart interface of h3), it is output from the 2nd port (s1-eth2).

You can tell that s1-eth2 is used for communication between h3 and h1.

As a matter of course, ping can be executed from host H4 to host h1 as well. As before, new flow entries are registered and s1-eth1 is used for communication between h4 and h1.

Destination host	Port used
h2	1
h3	2
h4	1



As shown above, we were able to confirm use of different links depending on communication.

Improving Fault Tolerance

Check improvement in fault tolerance as a result of link aggregation. The current state is that when h2 and h4 communicate with h1, s1-eth2 is used and when h3 communicates with h1, s1-eth1 is used.

Here, we separate h1-eth0, which is the counterpart interface of s1-eth1, from the link aggregation group.

Node: h1:

```
root@ryu-vm:~# ip link set h1-eth0 nomaster
```

Because h1-eth0 has stopped, pings can no longer be sent from host h3 to host h1. When 90 seconds of no communication monitoring time elapses, the following message is output to the controller's operation log.

Node: c0:

```
...
[LACP][INFO] SW=0000000000000001 PORT=1 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP exchange timeout has occurred.
slave state changed port: 1 enabled: False
...
```

"LACP exchange timeout has occurred." indicates that the no communication monitoring time has elapsed. Here, by deleting all learned MAC addresses and flow entries for transfer, the switch is returned to the state that was in effect just after it started.

If new communication arises, the new MAC address is learned and flow entries are registered again using only living links.

New flow entries are registered related to communication between host h3 and host h1.

Node: s1:

```
root@ryu-vm:~# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=364.265s, table=0, n_packets=13, n_bytes=1612, idle_timeout=90,
send_flow_rem priority=65535,in_port=2,dl_src=00:00:00:00:00:12,dl_type=0x8809 actions=
CONTROLLER:65509
 cookie=0x0, duration=374.521s, table=0, n_packets=25, n_bytes=1830, priority=0 actions=
CONTROLLER:65535
 cookie=0x0, duration=5.738s, table=0, n_packets=5, n_bytes=490, priority=1,in_port=3,dl_dst
=02:01:02:03:04:08 actions=output:2
 cookie=0x0, duration=6.279s, table=0, n_packets=5, n_bytes=490, priority=1,in_port=2,dl_dst
=00:00:00:00:00:23 actions=output:5
 cookie=0x0, duration=6.281s, table=0, n_packets=5, n_bytes=490, priority=1,in_port=5,dl_dst
=02:01:02:03:04:08 actions=output:2
 cookie=0x0, duration=5.506s, table=0, n_packets=5, n_bytes=434, priority=1,in_port=4,dl_dst
=02:01:02:03:04:08 actions=output:2
 cookie=0x0, duration=5.736s, table=0, n_packets=5, n_bytes=490, priority=1,in_port=2,dl_dst
=00:00:00:00:00:21 actions=output:3
 cookie=0x0, duration=6.504s, table=0, n_packets=6, n_bytes=532, priority=1,in_port=2,dl_dst
=00:00:00:00:00:22 actions=output:4
```

ping that had been stopped at host h3 resumes.

Node: h3:

```
...
64 bytes from 10.0.0.1: icmp_req=144 ttl=64 time=0.193 ms
64 bytes from 10.0.0.1: icmp_req=145 ttl=64 time=0.081 ms
64 bytes from 10.0.0.1: icmp_req=146 ttl=64 time=0.095 ms
64 bytes from 10.0.0.1: icmp_req=237 ttl=64 time=44.1 ms
64 bytes from 10.0.0.1: icmp_req=238 ttl=64 time=2.52 ms
64 bytes from 10.0.0.1: icmp_req=239 ttl=64 time=0.371 ms
64 bytes from 10.0.0.1: icmp_req=240 ttl=64 time=0.103 ms
64 bytes from 10.0.0.1: icmp_req=241 ttl=64 time=0.067 ms
...
```

As explained above, even though a failure occurs in some links, we were able to check that it can be automatically recovered using other links.

4.3 Implementing the Link Aggregation Function with Ryu

Now we are going to see how the link aggregation function is implemented using OpenFlow.

With a link aggregation using LACP, the behavior is like this: "While LACP data units are exchanged normally, the relevant physical interface is enabled" and "If exchange of LACP data units is suspended, the physical interface becomes disabled". Disabling of a physical interface means that no flow entries exist that use that interface. Therefore, by implementing the following processing:

- Create and send a response when an LACP data unit is received.
- If an LACP data unit cannot be received for a certain period of time, the flow entry that uses the physical interface and after that flow entries that use the interface are not registered.
- If an LACP data unit is received by the disabled physical interface, said interface is enabled again.
- Packets other than the LACP data unit are learned and transferred, as with " *Switching Hub*".

...basic operation of link aggregation becomes possible. Because the part related to LACP and the part not related to LACP are clearly separated, you can implement by cutting out the part related to LACP as an LACP library and extending the switching hub of "*Switching Hub*" for the part not related to LACP.

Because creation and sending of responses after an LACP data unit is received cannot be achieved only by flow entries, we use the Packet-In message for processing at the OpenFlow controller side.

Note: Physical interfaces that exchange LACP data units are classified as ACTIVE and PASSIVE, depending on their role. ACTIVE sends LACP data units at specified intervals to actively check communication. PASSIVE passively checks communication by returning a response after receiving the LACP data unit sent from ACTIVE.

Ryu's link aggregation application implements only the PASSIVE mode.

If no LACP data unit is received for a predetermined period of time, the physical interface is disabled. Because of this processing, by setting idle_timeout for the flow entry that performs Packet-In of the LACP data unit, when timeout occurs, by sending the FlowRemoved message, it is possible for the OpenFlow controller to handle it when the interface is disabled.

Processing when the exchange of LACP data units is resumed with the disabled interface is achieved by the handler of the Packet-In message to determine and change the enable/disable state of the interface upon receiving a LACP data unit.

When the physical interface is disabled, as OpenFlow controller processing, it looks OK to simply "delete the flow entry that uses the interface" but it is not sufficient to do so.

For example, assume there is a logical interface using a group of three physical interfaces and the sort logic is "Surplus of MAC address by the number of enabled interfaces".

Interface 1	Interface 2	Interface 3
Surplus of MAC address:0	Surplus of MAC address:1	Surplus of MAC address:2

Then, assume that flow entry that uses each physical interface has been registered for three entries, each.

Interface 1	Interface 2	Interface 3
Address:00:00:00:00:00:00	Address:00:00:00:00:00:01	Address:00:00:00:00:00:02
Address:00:00:00:00:00:03	Address:00:00:00:00:00:04	Address:00:00:00:00:00:05
Address:00:00:00:00:00:06	Address:00:00:00:00:00:07	Address:00:00:00:00:00:08

Here, if interface 1 is disabled, according to the sort logic "Surplus of MAC address by the number of enabled interfaces", it must be sorted as follows:

Interface 1	Interface 2	Interface 3
Disabled	Surplus of MAC address:0	Surplus of MAC address:1

Interface 1	Interface 2	Interface 3
	Address:00:00:00:00:00:00	Address:00:00:00:00:00:01
	Address:00:00:00:00:00:02	Address:00:00:00:00:00:03
	Address:00:00:00:00:00:04	Address:00:00:00:00:00:05
	Address:00:00:00:00:00:06	Address:00:00:00:00:00:07
	Address:00:00:00:00:00:08	

In addition to the flow entry that used interface 1, you can see it is also necessary to rewrite the flow entry of interface 2 and interface 3 as well. This is the same for both when the physical interface is disabled and when it is enabled.

Therefore, when the enable/disable state of a physical interface is changed, processing is to delete all flow entries that use the physical interfaces included in the logical interface to which the said physical interface belongs.

Note: The sort logic is not defined in the specification and it is up to the implementation of each device. In Ryu's link aggregation application, unique sort processing is not used and the path sorted by the counterpart device is used.

Here, implement the following functions.

LACP library

- When an LACP data unit is received, a response is created and sent.
- When reception of LACP data units is interrupted, the corresponding physical interface is assumed to be disabled and the switching hub is notified accordingly.
- When reception of LACP data unit is resumed, the corresponding physical interface is assumed to be enabled and the switching hub is notified accordingly.

Switching hub

- Receives notification from the LACP library and deletes the flow entry that needs initialization.
- Learns and transfers packets other than LACP data units as usual

The source code of the LACP library and switching hub are in Ryu's source tree.

ryu/lib/lacplib.py

ryu/app/simple_switch_lacp.py

Note: Because simple_switch_lacp.py is an application dedicated to OpenFlow 1.0, this section describes details of the application based on simple_switch_lacp_13.py, which supports OpenFlow 1.3 indicated in "Executing the Ryu Application".

4.3.1 Implementing the LACP Library

In the following section, we take a look at how the aforementioned functions are implemented in the LACP library. The quoted sources are excerpts. For the entire picture, refer to the actual source.

Creating a Logical Interface

In order to use the link aggregation function, it is necessary to configure beforehand the respective network devices as to which interfaces are aggregated as one group. The LACP library uses the following method to configure this setting.

```
def add(self, dpid, ports):
    # ...
    assert isinstance(ports, list)
    assert 2 <= len(ports)
    ifs = {}
    for port in ports:
        ifs[port] = {'enabled': False, 'timeout': 0}
    bond = {}
    bond[dpid] = ifs
    self._bonds.append(bond)
```

The content of the arguments are as follows:

dpid

Specifies the data path ID of the OpenFlow switch.

ports

Specifies the list of port numbers to be grouped.

By calling this method, the LACP library assumes ports specified by the OpenFlow switch of the specified data path ID as one group. If you wish to create multiple groups, repeat calling the add() method. For the MAC address assigned to a logical interface, the same address of the LOCAL port having the OpenFlow switch is used automatically.

Tip: Some OpenFlow switches provide a link aggregation function as thje switches' own function (Open vSwitch, etc.). Here, we don't use such functions unique to the switch and instead implement the link aggregation function through control by the OpenFlow controller.

Packet-In Processing

" *Switching Hub* " performs flooding on the received packet when the destination MAC address has not been learned. LACP data units should only be exchanged between adjacent network devices and if transferred to another device the link aggregation function does not operate correctly. Therefore, operation is that if a packet received by Packet-In is an LACP data unit, it is snatched and if the packet is not a LACP data unit, it is left up to the operation of the switching hub. In this operation LACP data units are not shown to the switching hub.

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, evt):
    """PacketIn event handler. when the received packet was LACP,
    proceed it. otherwise, send a event."""
    req_pkt = packet.Packet(evt.msg.data)
    if slow.lacp in req_pkt:
        (req_lacp, ) = req_pkt.get_protocols(slow.lacp)
        (req_eth, ) = req_pkt.get_protocols(ethernet.ethernet)
        self._do_lacp(req_lacp, req_eth.src, evt.msg)
    else:
        self.send_event_to_observers(EventPacketIn(evt.msg))
```

The event handler itself is the same as "*Switching Hub*". Processing is branched depending on whether or not the LACP data unit is included in the received massage.

When the LACP data unit is included, the LACP library's LACP data unit receive processing is performed. If the LACP data unit is not included, a method named send_event_to_observers() is called. This method is used to send an event defined in the ryu.base.app_manager.RyuApp class.

In *Switching Hub*, we mentioned the OpenFlow message receive event defined in Ryu, but users can define their own event. The event called `EventPacketIn`, which is sent in the above source, is a user-defined event created in the LACP library.

```
class EventPacketIn(event.EventBase):
    """a PacketIn event class using except LACP."""
    def __init__(self, msg):
        """initialization."""
        super(EventPacketIn, self).__init__()
        self.msg = msg
```

User-defined events are created by inheriting the ryu.controller.event.EventBase class. There is no limit on data enclosed in the event class. In the `EventPacketIn` class, the ryu.ofproto.OFPPacketIn instance received by the Packet-In message is used as is.

The method of receiving user-defined events is explained in a later section.

Processing Accompanying Port Enable/Disable State Change

The LACP data unit reception processing of the LACP library consists of the following processing.

1. If the port that received an LACP data unit is in disabled state, it is changed to enabled state and the state change is notified by the event.
2. When the waiting time of the no communication timeout was changed, a flow entry to send Packet-In is re-registered when the LACP data unit is received.
3. Creates and sends a response for the received LACP data unit.

The processing of 2 above is explained in Registering Flow Entry Sending Packet-In of an LACP Data Unit in a later section and the processing of 3 above is explained in Send/Receive Processing for LACP DATA Unit in a later section, respectively. In this section, we explain the processing of 1 above.

```
def _do_lacp(self, req_lacp, src, msg):
    # ...

    # when LACP arrived at disabled port, update the state of
    # the slave i/f to enabled, and send a event.
    if not self._get_slave_enabled(dpid, port):
        self.logger.info(
            "SW=%s PORT=%d the slave i/f has just been up.",
            dpid_to_str(dpid), port)
        self._set_slave_enabled(dpid, port, True)
        self.send_event_to_observers(
            EventSlaveStateChanged(datapath, port, True))
```

The _get_slave_enabled() method acquires information as to whether or not the port specified by the specified switch is enabled. The _set_slave_enabled() method sets the enable/disable state of the port specified by the specified switch.

In the above source, when an LACP data unit is received by a port in the disabled state, the user-defined event called `EventSlaveStateChanged` is sent, which indicates that the port state has been changed.

```
class EventSlaveStateChanged(event.EventBase):
    """a event class that notifies the changes of the statuses of the
    slave i/fs."""
    def __init__(self, datapath, port, enabled):
        """initialization."""
        super(EventSlaveStateChanged, self).__init__()
        self.datapath = datapath
        self.port = port
        self.enabled = enabled
```

Other than when a port is enabled, the `EventSlaveStateChanged` event is also sent when a port is disabled. Processing when disabled is implemented in "Receive Processing of FlowRemoved Message".

The `EventSlaveStateChanged` class includes the following information:

- OpenFlow switch where port enable/disable state has been changed
- Port number where port enable/disable state has been changed
- State after the change

Registering Flow Entry Sending Packet-In of an LACP Data Unit

For exchange intervals of LACP data units, two types have been defined, FAST (every 1 second) and SLOW (every 30 seconds). In the link aggregation specifications, if no communication status continues for three times the exchange interval, the interface is removed from the link aggregation group and is no longer used for packet transfer.

The LACP library monitors no communication by setting three times the exchange interval (SHORT_TIMEOUT_TIME is 3 seconds, and LONG_TIMEOUT_TIME is 90 seconds) as idle_timeout for the flow entry sending Packet-In when an LACP data unit is received.

If the exchange interval was changed, it is necessary to re-set the idle_timeout time, which the LACP library implements as follows:

```
def _do_lacp(self, req_lacp, src, msg):
    # ...

    # set the idle_timeout time using the actor state of the
    # received packet.
    if req_lacp.LACP_STATE_SHORT_TIMEOUT == \
       req_lacp.actor_state_timeout:
        idle_timeout = req_lacp.SHORT_TIMEOUT_TIME
    else:
        idle_timeout = req_lacp.LONG_TIMEOUT_TIME

    # when the timeout time has changed, update the timeout time of
    # the slave i/f and re-enter a flow entry for the packet from
    # the slave i/f with idle_timeout.
    if idle_timeout != self._get_slave_timeout(dpid, port):
        self.logger.info(
            "SW=%s PORT=%d the timeout time has changed.",
            dpid_to_str(dpid), port)
        self._set_slave_timeout(dpid, port, idle_timeout)
        func = self._add_flow.get(ofproto.OFP_VERSION)
        assert func
        func(src, port, idle_timeout, datapath)

    # ...
```

The _get_slave_timeout() method acquires the current idle_timeout value of the port specified by the specified switch. The _set_slave_timeout() method registers the idle_timeout value of the port specified by the specified switch. In initial status or when the port is removed from the link aggregation group, because the idle_timeout value is set to 0, if a new LACP data unit is received, the flow entry is registered regardless of which exchange interval is used.

Depending on the OpenFlow version used, the argument of the constructor of the `OFPFlowMod` class is different, an the flow entry registration method according to the version is acquired. The following is the flow entry registration method used by OpenFlow 1.2 and later.

```
def _add_flow_v1_2(self, src, port, timeout, datapath):
    """enter a flow entry for the packet from the slave i/f
    with idle_timeout. for OpenFlow ver1.2 and ver1.3."""
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    match = parser.OFPMatch(
        in_port=port, eth_src=src, eth_type=ether.ETH_TYPE_SLOW)
    actions = [parser.OFPActionOutput(
        ofproto.OFPP_CONTROLLER, ofproto.OFPCML_MAX)]
    inst = [parser.OFPInstructionActions(
        ofproto.OFPIT_APPLY_ACTIONS, actions)]
    mod = parser.OFPFlowMod(
        datapath=datapath, command=ofproto.OFPFC_ADD,
        idle_timeout=timeout, priority=65535,
        flags=ofproto.OFPFF_SEND_FLOW_REM, match=match,
        instructions=inst)
    datapath.send_msg(mod)
```

In the above source, the flow entry that "sends Packet-In when the LACP data unit is received form the counterpart interface" is set with the highest priority with no communication monitoring time.

Send/Receive Processing for LACP DATA Unit

When an LACP data unit is received, after performing "Processing Accompanying Port Enable/Disable State Change" or "Registering Flow Entry Sending Packet-In of an LACP Data Unit", processing creates and sends the response LACP data unit.

```
def _do_lacp(self, req_lacp, src, msg):
    # ...
```

```
    # create a response packet.
    res_pkt = self._create_response(datapath, port, req_lacp)

    # packet-out the response packet.
    out_port = ofproto.OFPP_IN_PORT
    actions = [parser.OFPActionOutput(out_port)]
    out = datapath.ofproto_parser.OFPPacketOut(
        datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER,
        data=res_pkt.data, in_port=port, actions=actions)
    datapath.send_msg(out)
```

The _create_response() method called in the above source is response packet creation processing. Using the _create_lacp() method called there, a response LACP data unit is created. The created response packet is Packet-Out from the port that received the LACP data unit.

In the LACP data unit, the send side (Actor) information and receive side (Partner) information are set. Because the counterpart interface information is described in the send side information of the received LACP data unit, that is set as the receive side information when a response is returned from the OpenFlow switch.

```
def _create_lacp(self, datapath, port, req):
    """create a LACP packet."""
    actor_system = datapath.ports[datapath.ofproto.OFPP_LOCAL].hw_addr
    res = slow.lacp(
        # ...
        partner_system_priority=req.actor_system_priority,
        partner_system=req.actor_system,
        partner_key=req.actor_key,
        partner_port_priority=req.actor_port_priority,
        partner_port=req.actor_port,
        partner_state_activity=req.actor_state_activity,
        partner_state_timeout=req.actor_state_timeout,
        partner_state_aggregation=req.actor_state_aggregation,
        partner_state_synchronization=req.actor_state_synchronization,
        partner_state_collecting=req.actor_state_collecting,
        partner_state_distributing=req.actor_state_distributing,
        partner_state_defaulted=req.actor_state_defaulted,
        partner_state_expired=req.actor_state_expired,
        collector_max_delay=0)
    self.logger.info("SW=%s PORT=%d LACP sent.",
                     dpid_to_str(datapath.id), port)
    self.logger.debug(str(res))
    return res
```

Receive Processing of FlowRemoved Message

When LACP data units are not exchanged during the specified period, the OpenFlow switch sends a FlowRemoved message to the OpenFlow controller.

```
@set_ev_cls(ofp_event.EventOFPFlowRemoved, MAIN_DISPATCHER)
def flow_removed_handler(self, evt):
    """FlowRemoved event handler. when the removed flow entry was
    for LACP, set the status of the slave i/f to disabled, and
    send a event."""
    msg = evt.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    dpid = datapath.id
    match = msg.match
    if ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION:
        port = match.in_port
        dl_type = match.dl_type
    else:
        port = match['in_port']
        dl_type = match['eth_type']
    if ether.ETH_TYPE_SLOW != dl_type:
        return
    self.logger.info(
        "SW=%s PORT=%d LACP exchange timeout has occurred.",
```

```
            dpid_to_str(dpid), port)
        self._set_slave_enabled(dpid, port, False)
        self._set_slave_timeout(dpid, port, 0)
        self.send_event_to_observers(
            EventSlaveStateChanged(datapath, port, False))
```

When a FlowRemoved message is received, the OpenFlow controller uses the _set_slave_enabled() method to set port disabled state, uses the _set_slave_timeout() method to set the idle_timeout value to 0, and uses the send_event_to_observers() method to send an `EventSlaveStateChanged` event.

4.3.2 Implementing the Application

We explain the difference between the link aggregation application (simple_switch_lacp_13.py) that supports OpenFlow 1.3 described in Executing the Ryu Application and the switching hub of " *Switching Hub*", in order.

Setting "_CONTEXTS"

A Ryu application that inherits ryu.base.app_manager.RyuApp starts other applications using separate threads by setting other Ryu applications in the "_CONTEXTS" dictionary. Here, the LacpLib class of the LACP library is set in "_CONTEXTS" in the name of " lacplib".

```
from ryu.lib import lacplib

# ...

class SimpleSwitchLacp13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'lacplib': lacplib.LacpLib}

    # ...
```

Applications set in "_CONTEXTS" can acquire instances from the kwargs of the __init__() method.

```
# ...
def __init__(self, *args, **kwargs):
    super(SimpleSwitchLacp13, self).__init__(*args, **kwargs)
    self.mac_to_port = {}
    self._lacp = kwargs['lacplib']
# ...
```

Initial Setting of the Library

Initialize the LACP library set in "_CONTEXTS". For the initial setting, execute the add() method provided by the LACP library. Here, set the following values.

Parameter	Value	Explanation
dpid	str_to_dpid('0000000000000001')	Data path ID
ports	[1, 2]	List of port to be grouped

With this setting, part 1 and port 2 of the OpenFlow switch of data path ID "0000000000000001" operate as one link aggregation group.

```
# ...
    self._lacp = kwargs['lacplib']
    self._lacp.add(
        dpid=str_to_dpid('0000000000000001'), ports=[1, 2])
# ...
```

Receiving User-defined Events

As explained in Implementing the LACP Library, the LACP library sends a Packet-In message that does not contain the LACP data unit as a user-defined event called `EventPacketIn`. The event handler of the user-defined event uses the `ryu.controller.handler.set_ev_cls` decorator to decorate, as with the event handler provided by Ryu.

```
@set_ev_cls(lacplib.EventPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    # ...
```

Also, when the enable/disable condition of a port is changed, the LACP library sends an `EventSlaveStateChanged` event, therefore, create an event handler for this as well.

```
@set_ev_cls(lacplib.EventSlaveStateChanged, lacplib.LAG_EV_DISPATCHER)
def _slave_state_changed_handler(self, ev):
    datapath = ev.datapath
    dpid = datapath.id
    port_no = ev.port
    enabled = ev.enabled
    self.logger.info("slave state changed port: %d enabled: %s",
                     port_no, enabled)
    if dpid in self.mac_to_port:
        for mac in self.mac_to_port[dpid]:
            match = datapath.ofproto_parser.OFPMatch(eth_dst=mac)
            self.del_flow(datapath, match)
        del self.mac_to_port[dpid]
    self.mac_to_port.setdefault(dpid, {})
```

As explained at the beginning of this document, when the enable/disable state of a port is changed, the actual physical interface used by the packet that passes through the logical interface may be changed. For that reason, all registered flow entries are deleted.

```
def del_flow(self, datapath, match):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    mod = parser.OFPFlowMod(datapath=datapath,
                            command=ofproto.OFPFC_DELETE,
                            match=match)
    datapath.send_msg(mod)
```

Flow entries are deleted by the instance of the `OFPFlowMod` class.

As explained above, a switching hub application having an link aggregation function is achieved by a library that provides the link aggregation function and applications that use the library.

4.4 Conclusion

This section uses the link aggregation library as material to explain the following items:

- How to use the library using "_CONTEXTS"
- Method of defining user-defined events and method of raising event triggers

CHAPTER

FIVE

SPANNING TREE

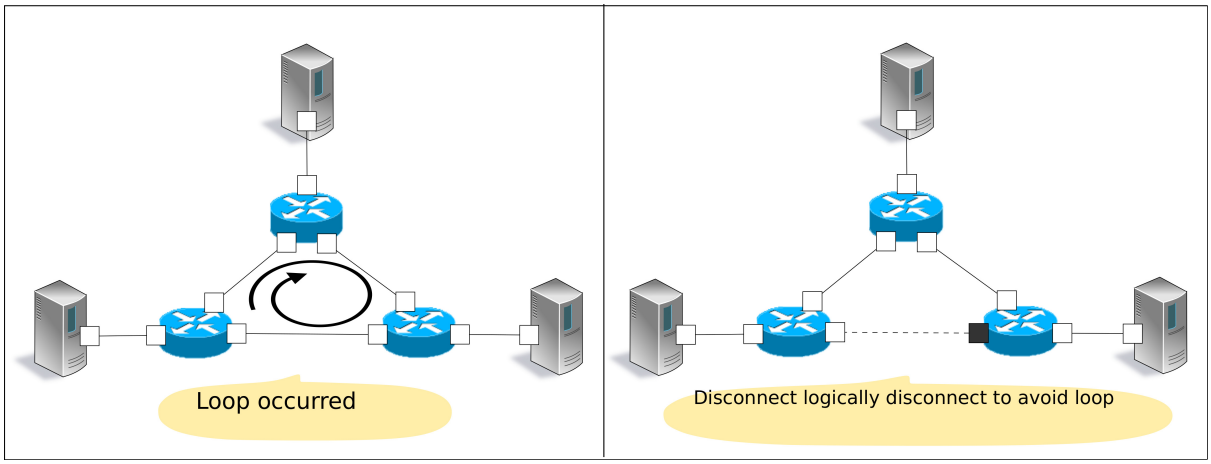
This section describes how to implement spanning tree using Ryu.

5.1 Spanning tree

Spanning tree is a function that suppresses occurrence of broadcast streams in a network having a loop structure. Also, applying the original function that is preventing the loop, it is used as a means to secure network redundancy to automatically switch the path in case of a network failure.

There are various types of spanning tree, including STP, RSTP, PVST+, and MSTP. In this section, we will take a look at implementation of the most basic STP.

Spanning Tree Protocol (STP: IEEE 802.1D) handles a network as a logical tree and by setting the ports of each switch (sometimes called a bridge in this section) to transfer frame or not it suppresses occurrence of broadcast streams in a network having a loop structure.



With STP, Bridge Protocol Data Unit (BPDU) packets are exchanged between bridges to compare the bridge and port information and decide whether or not frame transfer of each port is available.

Specifically, this is achieved by the following procedure:

1 Selecting the root bridge

The bridge having the smallest bridge ID is selected as the root bridge through BPDU packet exchange between bridges. After that, only the root bridge sends the original BPDU packet and other bridges transfer BPDU packets received from the root bridge.

Note: The bridge ID is calculated through a combination of the bridge priority set for each bridge and the MAC address of the specific port.

Bridge ID

Upper 2byte	Lower 6byte
Bridge priority	MAC address

2 Deciding the role of ports

Based on the cost of each port to reach the root bridge, decide the role of the ports.

- Root port

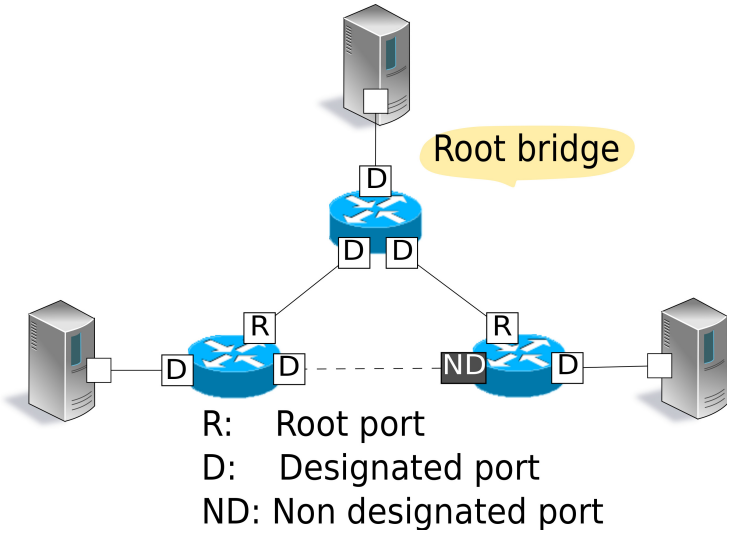
  The port having the smallest cost among bridges to reach the root bridge. This port receives BPDU packets from the root bridge.

- Designated ports

  Ports at the side having the small cost to reach the root bridge of each link. These ports sends BPDU packets received from the root bridge. Root bridge ports are all designated ports.

- Non designated ports

  Ports other than the root port and designated port. These ports suppress frame transfer.



Note: The cost to reach the root bridge is compared as follows based on the setting value of the BPDU packet received by each port.

Priority 1: Compares by the root path cost value.

When each bridge transfers a BPDU packet, the path cost value set for the output port is added to the root path cost value of the BPDU packet. Because of this, the root path cost value is the total value of the path cost value of each link passed through to reach the root bridge.

Priority 2: When the root path cost is the same, compares using the bridge ID of the counterpart bridges.

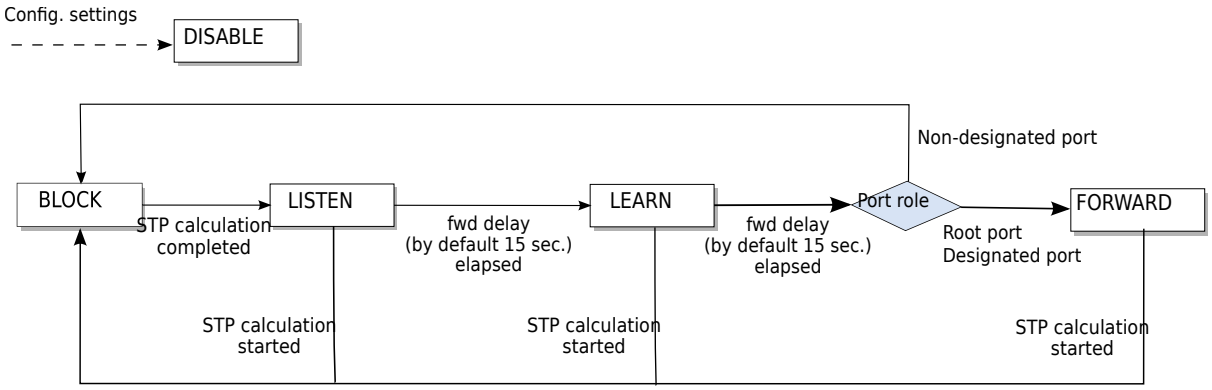
Priority 3: When the bridge ID of the counterpart bridges are the same (in cases in which each port is connected to the same bridge), compare using the port ID of the counterpart ports.

Port ID

Upper 2 bytes	Lower 2 bytes
Port priority	Port number

3 Port state change

After the port role is decided (STP calculation is completed), each port becomes LISTEN state. After that, the state changes as shown below and according to the role of each port, it eventually becomes FORWARD state or BLOCK state. Ports set as disabled ports in the configuration become DISABLE state and after that the change of state does not take place.



When that processing is executed at each bridge, ports that transfer frames and ports that suppress frame transfer are decided to dissolve loops inside the network.

Also, when failure is detected due to link down or no reception of BPDU packet for the max age (default: 20 seconds), or a change in the network topology is detected as a result of the addition of a port, each bridge executes 1, 2, and 3 above to reconfigure the tree (STP re-calculation).

5.2 Executing the Ryu Application

Let's execute the Ryu's spanning tree application for which the spanning function is achieved using OpenFlow.

simple_switch_stp.py provided in Ryu's source tree is an application dedicated to OpenFlow 1.0, therefore, we will create simple_switch_stp_13.py, which supports OpenFlow 1.3. This program is an application to which the spanning tree function has been added to the switching hub of " *Switching Hub* .

Source name: `simple_switch_stp_13.py`

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib import dpid as dpid_lib
from ryu.lib import stplib
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet


class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'stplib': stplib.Stp}

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.stp = kwargs['stplib']

        # Sample of stplib config.
        #  please refer to stplib.Stp.set_config() for details.
        config = {dpid_lib.str_to_dpid('0000000000000001'):
                     {'bridge': {'priority': 0x8000}},
                  dpid_lib.str_to_dpid('0000000000000002'):
                     {'bridge': {'priority': 0x9000}},
                  dpid_lib.str_to_dpid('0000000000000003'):
                     {'bridge': {'priority': 0xa000}}}
        self.stp.set_config(config)

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
```

```
        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]

        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

    def delete_flow(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        for dst in self.mac_to_port[datapath.id].keys():
            match = parser.OFPMatch(eth_dst=dst)
            mod = parser.OFPFlowMod(
                datapath, command=ofproto.OFPFC_DELETE,
                out_port=ofproto.OFPP_ANY, out_group=ofproto.OFPG_ANY,
                priority=1, match=match)
            datapath.send_msg(mod)

    @set_ev_cls(stplib.EventPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            self.add_flow(datapath, 1, match, actions)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data
```

```
        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions, data=data)
        datapath.send_msg(out)

    @set_ev_cls(stplib.EventTopologyChange, MAIN_DISPATCHER)
    def _topology_change_handler(self, ev):
        dp = ev.dp
        dpid_str = dpid_lib.dpid_to_str(dp.id)
        msg = 'Receive topology change event. Flush MAC table.'
        self.logger.debug("[dpid=%s] %s", dpid_str, msg)

        if dp.id in self.mac_to_port:
            self.delete_flow(dp)
            del self.mac_to_port[dp.id]

    @set_ev_cls(stplib.EventPortStateChange, MAIN_DISPATCHER)
    def _port_state_change_handler(self, ev):
        dpid_str = dpid_lib.dpid_to_str(ev.dp.id)
        of_state = {stplib.PORT_STATE_DISABLE: 'DISABLE',
                    stplib.PORT_STATE_BLOCK: 'BLOCK',
                    stplib.PORT_STATE_LISTEN: 'LISTEN',
                    stplib.PORT_STATE_LEARN: 'LEARN',
                    stplib.PORT_STATE_FORWARD: 'FORWARD'}
        self.logger.debug("[dpid=%s][port=%d] state=%s",
                          dpid_str, ev.port_no, of_state[ev.port_state])
```

5.2.1 Configuring the Experimental Environment

Let's configure an experimental environment to confirm operation of the spanning tree application.

For details on environment configuration and the login method, etc. to use VM images, refer to " *Switching Hub* ".

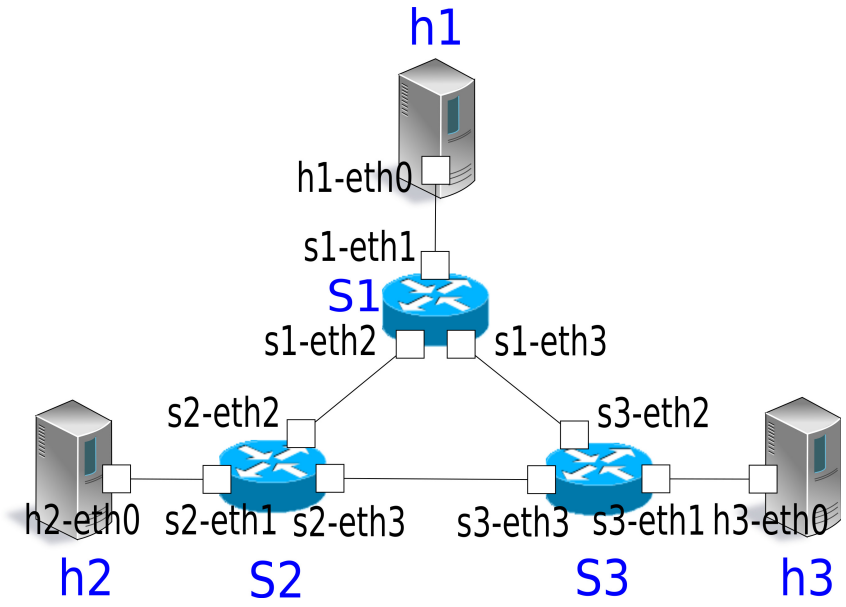
To operate using a special topology having a loop structure, as with " *Link Aggregation* ", using the topology configuration script, configure a mininet environment.

Source name: `spanning_tree.py`

```
#!/usr/bin/env python

from mininet.cli import CLI
from mininet.link import Link
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.term import makeTerm

if '__main__' == __name__:
    net = Mininet(controller=RemoteController)

    c0 = net.addController('c0')

    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')

    h1 = net.addHost('h1')
    h2 = net.addHost('h2')
    h3 = net.addHost('h3')

    Link(s1, h1)
    Link(s2, h2)
    Link(s3, h3)

    Link(s1, s2)
    Link(s2, s3)
    Link(s3, s1)


    net.build()
    c0.start()
```

```
    s1.start([c0])
    s2.start([c0])
    s3.start([c0])

    net.terms.append(makeTerm(c0))
    net.terms.append(makeTerm(s1))
    net.terms.append(makeTerm(s2))
    net.terms.append(makeTerm(s3))
    net.terms.append(makeTerm(h1))
    net.terms.append(makeTerm(h2))
    net.terms.append(makeTerm(h3))

    CLI(net)

    net.stop()
```

By executing the program in the VM environment, a topology is created in which a loop exists between switches s1, s2, and s3.



The execution result of the net command is as follows:

```
ryu@ryu-vm:~$ sudo ./spanning_tree.py
Unable to contact the remote controller at 127.0.0.1:6633
mininet> net
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2 s1-eth3:s3-eth3
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s1-eth3
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
```

5.2.2 Setting the OpenFlow Version

Set the OpenFlow version to 1.3. Input this command on xterm of switches s1, s2, and x3.

Node: s1:

```
root@ryu-vm:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

Node: s2:

```
root@ryu-vm:~# ovs-vsctl set Bridge s2 protocols=OpenFlow13
```

Node: s3:

```
root@ryu-vm:~# ovs-vsctl set Bridge s3 protocols=OpenFlow13
```

5.2.3 Executing the Switching Hub

This completes preparation so let's move on to executing the Ryu application. Execute the following commands from xterm for which the window title is "Node: c0 (root)".

Node: c0:

```
root@ryu-vm:~$ ryu-manager ./simple_switch_stp_13.py
loading app simple_switch_stp_13.py
loading app ryu.controller.ofp_handler
loading app ryu.controller.ofp_handler
instantiating app None of Stp
creating context stplib
instantiating app simple_switch_stp_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

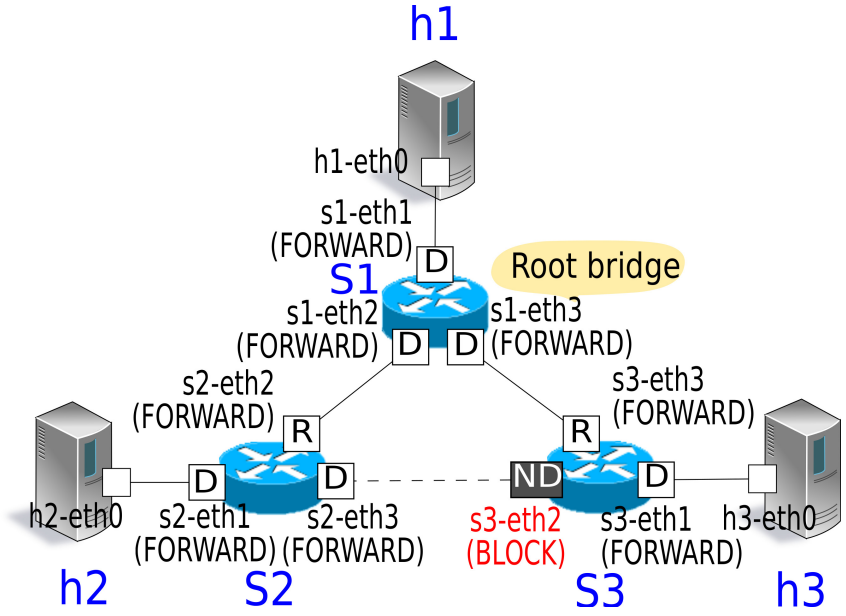
Calculating STP Upon OpenFlow Swtich Starts

When connection between each OpenFlow switch and the controller is completed, exchange of BPDU packets starts and root bridge selection, port role setting, and port state change takes place.

```
[STP][INFO] dpid=0000000000000001: Join as stp bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: Join as stp bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: Join as stp bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
```

```
[STP][INFO] dpid=0000000000000003: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / FORWARD
```

As a result, each port eventually becomes FORWARD state or BLOCK state.



Next, in order to confirm that packets are not looped, execute ping from host 1 to host 2.

Before executing the ping command, execute the tcpdump command.

Node: s1:

```
root@ryu-vm:~# tcpdump -i s1-eth2 arp
```

Node: s2:

```
root@ryu-vm:~# tcpdump -i s2-eth2 arp
```

Node: s3:

```
root@ryu-vm:~# tcpdump -i s3-eth2 arp
```

On the console where the topology configuration script is executed, execute the following commands to issue a ping from host 1 to host 2.

```
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_req=1 ttl=64 time=84.4 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.657 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_req=5 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_req=6 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_req=7 ttl=64 time=0.041 ms
64 bytes from 10.0.0.2: icmp_req=8 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_req=9 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_req=10 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_req=11 ttl=64 time=0.068 ms
^C
--- 10.0.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 9998ms
rtt min/avg/max/mdev = 0.041/7.784/84.407/24.230 ms
```

As a result of tcpdump output, you can confirm that ARP is not looped.

Node: s1:

```
root@ryu-vm:~# tcpdump -i s1-eth2 arp
tcpdump: WARNING: s1-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s1-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
11:30:24.692797 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
11:30:24.749153 ARP, Reply 10.0.0.2 is-at 82:c9:d7:e9:b7:52 (oui Unknown), length 28
11:30:29.797665 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
11:30:29.798250 ARP, Reply 10.0.0.1 is-at c2:a4:54:83:43:fa (oui Unknown), length 28
```

Node: s2:

```
root@ryu-vm:~# tcpdump -i s2-eth2 arp
tcpdump: WARNING: s2-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s2-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
11:30:24.692824 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
11:30:24.749116 ARP, Reply 10.0.0.2 is-at 82:c9:d7:e9:b7:52 (oui Unknown), length 28
11:30:29.797659 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
11:30:29.798254 ARP, Reply 10.0.0.1 is-at c2:a4:54:83:43:fa (oui Unknown), length 28
```

Node: s3:

```
root@ryu-vm:~# tcpdump -i s3-eth2 arp
tcpdump: WARNING: s3-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
11:30:24.698477 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
```

Re-Calculation When a Failure is Detected

Next, let's check re-calculation operation of STP in case of link down. In the state in which STP calculation has been completed after each OpenFlow switch starts, execute the following commands to make the port down.

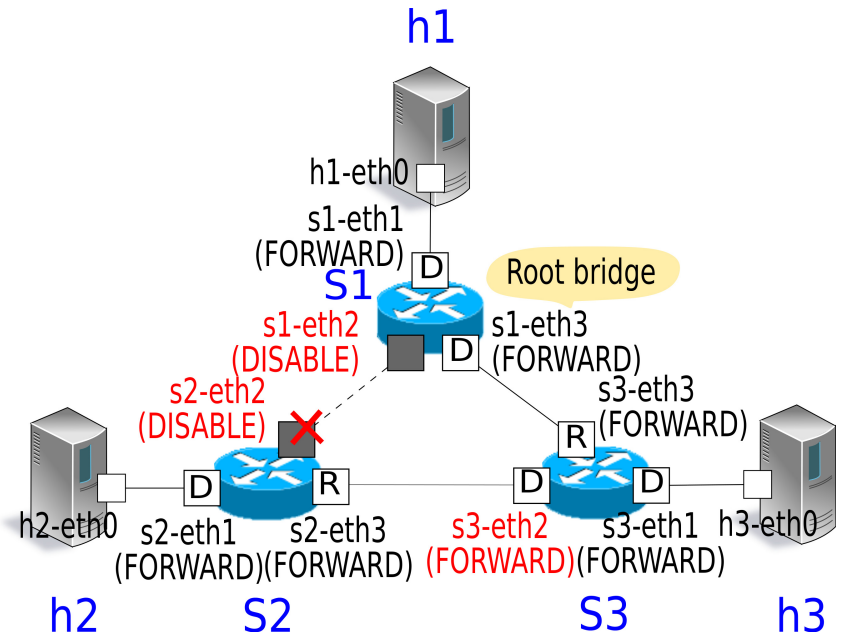
Node: s2:

```
root@ryu-vm:~# ifconfig s2-eth2 down
```

Link down is detected and recalculation of STP is executed.

```
[STP][INFO] dpid=0000000000000002: [port=2] Link down.
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / DISABLE
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] Link down.
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / DISABLE
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] Wait BPDU timer is exceeded.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=3] ROOT_PORT           / FORWARD
```

You can confirm that the s3-eth2 port, which was in BLOCK state, became FORWARD state and frame transfer became available again.



Recalculation of STP At Failure Recovery

Next, check operation of recalculation of STP when link down is recovered. To start the port execute the following commands while the link is down.

Node: s2:

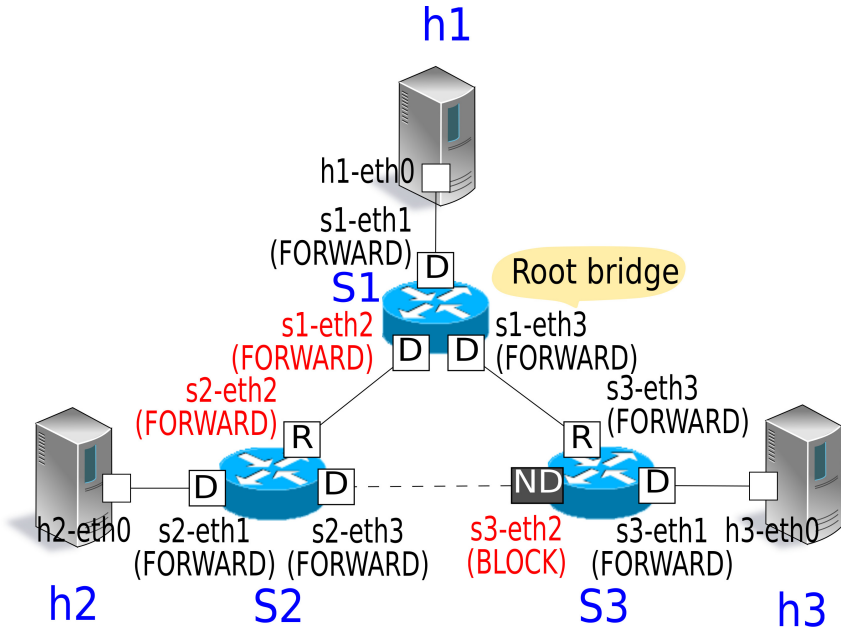
```
root@ryu-vm:~# ifconfig s2-eth2 up
```

Link recovery is detected and STP re-calculation is executed.

```
[STP][INFO] dpid=0000000000000002: [port=2] Link down.
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / DISABLE
[STP][INFO] dpid=0000000000000002: [port=2] Link up.
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] Link up.
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
```

```
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
```

You can confirm that the tree structure becomes the same as that in effect when the application starts and frame transfer becomes available again.



5.3 Spanning Tree by OpenFlow

In Ryu's spanning tree application, let's look at how spanning tree is implemented using OpenFlow.

OpenFlow 1.3 provides config to configure the following port operation. By issuing a Port Modification message to the OpenFlow switch, it is possible to control operations such as availability of port frame transfer.

Value	Explanation
OFPPC_PORT_DOWN	Status in which maintenance personnel has set it to disable.
OFPPC_NO_RECV	Discards all packets received by the port.
OFPPC_NO_FWD	Packets are not transferred from the port.
OFPPC_NO_PACKET_IN	In case of table-miss, Packet-In messages are not sent.

Also, in order to control BPDU packet reception and reception of packets other than BPDU for each port, flow entry that sends Packet-In of BPDU packets and flow entry that drops packets other than BPDU are registered in the OpenFlow switch using Flow Mod messages, respectively.

The controller controls sending/receiving of BPDU packets depending on the port status, learning of MAC addresses (receiving packets other than BPDU), and frame transfer (sending packets other then BPDU) by setting the port configuration and flow entry on the OpenFlow switch as shown below.

Status	Port configuration	Flow entry
DISABLE	NO_RECV/NO_FWD	No setting
BLOCK	NO_FWD	BPDU Packet-In, drop packets other than BPDU
LISTEN	No setting	BPDU Packet-In, drop packets other than BPDU
LEARN	No setting	BPDU Packet-In, drop packets other than BPDU
FORWARD	No setting	BPDU Packet-In

Note: For simplification, the spanning tree library implemented by Ryu does not perform MAC address learning (receiving packets other than BPDU) in Learn status.

In addition to those settings, by building the BPDU packet for transmission based on the port information collected when connecting to an Open Flow switch and the root bridge information set in the BPDU packet received by each OpenFlow switch and issuing a Packet-Out message, the controller achieves BPDU packet exchange between OpenFlow switches.

5.4 Using Ryu to Implement Spanning Tree

Next, let's take a look at the source code of spanning tree implemented using Ryu. The spanning tree source code is in the Ryu's source tree.

ryu/lib/stplib.py

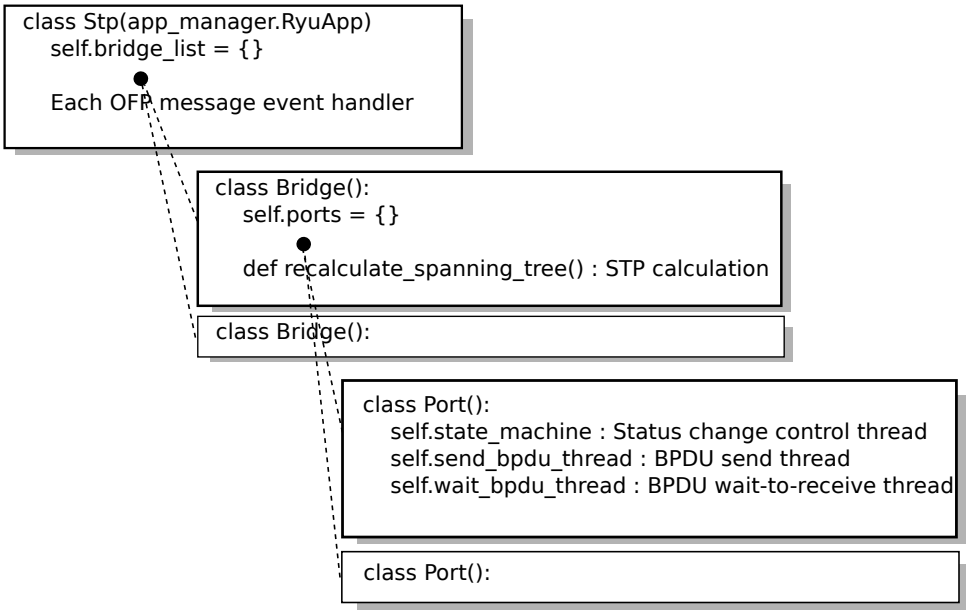
ryu/app/simple_switch_stp.py

stplib.py is a library that provides spanning tree functions such as BPDU packet exchange and management of rules, and the status of each port. The simple_switch_stp.py is an application program in which the spanning tree function is added to the switching hub application using the spanning tree library.

Attention: simple_switch_stp.py is an application dedicated to OpenFlow 1.0; this section describes details of the application based on simple_switch_stp_13.py, which supports OpenFlow 1.3, indicated in "Executing the Ryu Application ".

5.4.1 Implementing the Library

Library Overview



When the STP library (Stp class instance) detects connection of an OpenFlow switch to the controller, a Bridge class instance and Port class instance are generated. After each class instance is generated and started,

- Notification of the OpenFlow message reception from the Stp class instance
- STP calculation of the Bridge class instance (loot bridge selection and selection of the role of each port)
- Status change of the port of the Port class instance and send/receive of BPDU packets

work together to achieve the spanning tree function.

Configuration Settings Item

The STP library provides the bridge port config setting IF using the `Stp.set_config()` method. The following items can be set:

- bridge

Item	Explanation	Default value
`priority`	Bridge priority	0x8000
`sys_ext_id`	Sets VLAN-ID (*the current STP library does not support VLAN)	0
`max_age`	Timer value to wait to receive BPDU packets	20[sec]
`hello_time`	Send intervals of BPDU packets	2 [sec]
`fwd_delay`	Period that each port stays in LISTEN or LEARN status	15[sec]

- port

Item	Explanation	Default value
`priority`	Port priority	0x80
`path_cost`	Link cost value	Auto setting based on the link speed
`enable`	Port enable/disable setting	True

Sending BPDU Packet

BPDU packets are sent by the BPDU packet send thread (`Port.send_bpdu_thread`) of the Port class. When the port role is the designated port (`DESIGNATED_PORT`), a BPDU packet is generated (`Port._generate_config_bpdu()`) at the hello time (`Port.port_times.hello_time`: by default, 2 seconds) notified by the root bridge and the BPDU packet is sent (`Port.ofctl.send_packet_out()`).

```
class Port(object):

    def __init__(self, dp, logger, config, send_ev_func, timeout_func,
                 topology_change_func, bridge_id, bridge_times, ofport):
        super(Port, self).__init__()

        # ...

        # BPDU handling threads
        self.send_bpdu_thread = PortThread(self._transmit_bpdu)

    # ...

    def _transmit_bpdu(self):
        while True:
            # Send config BPDU packet if port role is DESIGNATED_PORT.
            if self.role == DESIGNATED_PORT:

                # ...

                bpdu_data = self._generate_config_bpdu(flags)
                self.ofctl.send_packet_out(self.ofport.port_no, bpdu_data)

                # ...

            hub.sleep(self.port_times.hello_time)
```

BPDU packets to be sent are configured based on the port information (`Port.ofport`) collected when the controller is connected to OpenFlow switches or the root bridge information (`Port.port_priority`, `Port.port_times`) set in the received BPDU packets.

```
class Port(object):

    def _generate_config_bpdu(self, flags):
        src_mac = self.ofport.hw_addr
        dst_mac = bpdu.BRIDGE_GROUP_ADDRESS
        length = (bpdu.bpdu._PACK_LEN + bpdu.ConfigurationBPDUs.PACK_LEN
                  + llc.llc._PACK_LEN + llc.ControlFormatU._PACK_LEN)

        e = ethernet.ethernet(dst_mac, src_mac, length)
        l = llc.llc(llc.SAP_BPDU, llc.SAP_BPDU, llc.ControlFormatU())
        b = bpdu.ConfigurationBPDUs(
            flags=flags,
            root_priority=self.port_priority.root_id.priority,
            root_mac_address=self.port_priority.root_id.mac_addr,
            root_path_cost=self.port_priority.root_path_cost+self.path_cost,
            bridge_priority=self.bridge_id.priority,
            bridge_mac_address=self.bridge_id.mac_addr,
            port_priority=self.port_id.priority,
            port_number=self.ofport.port_no,
            message_age=self.port_times.message_age+1,
            max_age=self.port_times.max_age,
            hello_time=self.port_times.hello_time,
            forward_delay=self.port_times.forward_delay)

        pkt = packet.Packet()
        pkt.add_protocol(e)
        pkt.add_protocol(l)
        pkt.add_protocol(b)
        pkt.serialize()

        return pkt.data
```

Receiving BPDU Packets

Reception of a BPDU packet is detected by the Packet-In event handler of the Stp class and is notified to the Port class instance via the Bridge class instance. For implementation of the event handler, refer to "*Switching Hub*".

The port that receives a BPDU packet compares (`Stp.compare_bpdu_info()`) the bridge ID of previously received BPDU packets and the BPDU packet received this time to determine the need for STP re-calculation. If a superior BPDU (`SUPERIOR`) than the previously received BPDU is received, it means there is a change in the network topology such as "a new root bridge is added", which is a a trigger for STP re-calculation.

```
class Port(object):

    def rcv_config_bpdu(self, bpdu_pkt):
        # Check received BPDU is superior to currently held BPDU.
        root_id = BridgeId(bpdu_pkt.root_priority,
                           bpdu_pkt.root_system_id_extension,
                           bpdu_pkt.root_mac_address)
        root_path_cost = bpdu_pkt.root_path_cost
        designated_bridge_id = BridgeId(bpdu_pkt.bridge_priority,
                                        bpdu_pkt.bridge_system_id_extension,
                                        bpdu_pkt.bridge_mac_address)
        designated_port_id = PortId(bpdu_pkt.port_priority,
                                    bpdu_pkt.port_number)

        msg_priority = Priority(root_id, root_path_cost,
                                designated_bridge_id,
                                designated_port_id)
        msg_times = Times(bpdu_pkt.message_age,
                          bpdu_pkt.max_age,
                          bpdu_pkt.hello_time,
                          bpdu_pkt.forward_delay)

        rcv_info = Stp.compare_bpdu_info(self.designated_priority,
                                         self.designated_times,
```

```
                                        msg_priority, msg_times)

        # ...

        return rcv_info, rcv_tc
```

Detecting Failures

When direct failure such as link down or indirect failure such as no reception of BPDU packet from the root bridge for the predetermined period of time is detected, it is a trigger for STP re-calculation.

Link down is detected by the PortStatus event handler of the Stp class and is notified to the Bridge class instance.

Timeout of BPDU packet receive waiting is detected by the BPDU packet receive waiting thread (`Port.wait_bpdu_thread`) of the Port class. When BPDU packets from the root bridge cannot be received for the maximum age (default: 20 seconds), an indirect failure is determined and is notified to the Bridge class instance.

For update of the BPDU receive waiting timer and detection of timeout, `hub.Event` and `hub.Timeout` of the hub module (ryu.lib.hub) are used. `hub.Event` enters wait status by `hub.Event.wait()` and the thread is suspended until `hub.Event.set()` is executed. `hub.Timeout` issues an `hub.Timeout` exception if processing of the `try` clause is not completed within the specified timeout time. When `hub.Event` enters wait status and `hub.Event.set()` is not executed within the timeout time specified in `hub.Timeout`, timeout of BPDU packet receive waiting is determined and STP re-calculation processing of the Bridge class is called.

```
class Port(object):

    def __init__(self, dp, logger, config, send_ev_func, timeout_func,
                 topology_change_func, bridge_id, bridge_times, ofport):
        super(Port, self).__init__()
        # ...
        self.wait_bpdu_timeout = timeout_func
        # ...
        self.wait_bpdu_thread = PortThread(self._wait_bpdu_timer)

    # ...

    def _wait_bpdu_timer(self):
        time_exceed = False

        while True:
            self.wait_timer_event = hub.Event()
            message_age = (self.designated_times.message_age
                           if self.designated_times else 0)
            timer = self.port_times.max_age - message_age
            timeout = hub.Timeout(timer)
            try:
                self.wait_timer_event.wait()
            except hub.Timeout as t:
                if t is not timeout:
                    err_msg = 'Internal error. Not my timeout.'
                    raise RyuException(msg=err_msg)
                self.logger.info('[port=%d] Wait BPDU timer is exceeded.',
                                 self.ofport.port_no, extra=self.dpid_str)
                time_exceed = True
            finally:
                timeout.cancel()
                self.wait_timer_event = None

            if time_exceed:
                break

        if time_exceed:  # Bridge.recalculate_spanning_tree
            hub.spawn(self.wait_bpdu_timeout)
```

When `SUPERIOR` or `REPEATED` is determined as a result of comparison processing (`Stp.compare_bpdu_info()`) of the received BPDU packet, it means that the BPDU packet

from the root bridge can be received. Therefore, the BPDU receive waiting timer is updated (`Port._update_wait_bpdu_timer()`). By the `set()` processing of `Port.wait_timer_event`, which is a `hub.Event`, the `Port.wait_timer_event` is released from wait status and the BPDU packet receive waiting thread (`Port.wait_bpdu_thread`) cancels the timer without entering timeout processing of the `except hub.Timeout` clause and sets the timer again to start waiting for the next BPDU packet to be received.

```
class Port(object):

    def rcv_config_bpdu(self, bpdu_pkt):
        # ...

        rcv_info = Stp.compare_bpdu_info(self.designated_priority,
                                         self.designated_times,
                                         msg_priority, msg_times)
        # ...

        if ((rcv_info is SUPERIOR or rcv_info is REPEATED)
                and (self.role is ROOT_PORT
                     or self.role is NON_DESIGNATED_PORT)):
            self._update_wait_bpdu_timer()

        # ...

    def _update_wait_bpdu_timer(self):
        if self.wait_timer_event is not None:
            self.wait_timer_event.set()
            self.wait_timer_event = None
```

STP Calculation

STP calculation (selection of the root bridge and selection of the role of each port) is executed by the Bridge class.

In cases where STP calculation is executed, a change in the network topology has occurred and it is possible for packets to be looped. Therefore, by setting all ports to BLOCK state (`port.down`) and also notifying the topology change event (`EventTopologyChange`) to high order APL, initialization of already learned MAC address information is promoted.

After that, the root bridge and the role of ports are selected by `Bridge._spanning_tree_algorithm()`, and each port is started in LISTEN status (`port.up`) to start port status change.

```
class Bridge(object):

    def recalculate_spanning_tree(self, init=True):
        """ Re-calculation of spanning tree. """
        # All port down.
        for port in self.ports.values():
            if port.state is not PORT_STATE_DISABLE:
                port.down(PORT_STATE_BLOCK, msg_init=init)

        # Send topology change event.
        if init:
            self.send_event(EventTopologyChange(self.dp))

        # Update tree roles.
        port_roles = {}
        self.root_priority = Priority(self.bridge_id, 0, None, None)
        self.root_times = self.bridge_times

        if init:
            self.logger.info('Root bridge.', extra=self.dpid_str)
            for port_no in self.ports.keys():
                port_roles[port_no] = DESIGNATED_PORT
        else:
            (port_roles,
             self.root_priority,
             self.root_times) = self._spanning_tree_algorithm()

        # All port up.
```

```
        for port_no, role in port_roles.items():
            if self.ports[port_no].state is not PORT_STATE_DISABLE:
                self.ports[port_no].up(role, self.root_priority,
                                       self.root_times)
```

To select the root bridge, own bridge information such as bridge ID, etc. is compared with other bridge's information set in the BPDU packet received by each port (`Bridge._select_root_port`).

As a result, when the root port is found (the other bridge's information received by the port is superior to that of the own bridge), the other bridge is determined to be the root bridge and the designated ports (`Bridge._select_designated_port`) and non-designated ports are selected (ports other than the root port/designated ports are selected as non-designated ports).

On the other hand, if the root port is not found (own bridge information is the most superior), the own bridge is determined to be the root bridge and all other ports are designated ports.

```
class Bridge(object):

    def _spanning_tree_algorithm(self):
        """ Update tree roles.
             - Root bridge:
                all port is DESIGNATED_PORT.
             - Non root bridge:
                select one ROOT_PORT and some DESIGNATED_PORT,
                and the other port is set to NON_DESIGNATED_PORT."""
        port_roles = {}

        root_port = self._select_root_port()

        if root_port is None:
            # My bridge is a root bridge.
            self.logger.info('Root bridge.', extra=self.dpid_str)
            root_priority = self.root_priority
            root_times = self.root_times

            for port_no in self.ports.keys():
                if self.ports[port_no].state is not PORT_STATE_DISABLE:
                    port_roles[port_no] = DESIGNATED_PORT
        else:
            # Other bridge is a root bridge.
            self.logger.info('Non root bridge.', extra=self.dpid_str)
            root_priority = root_port.designated_priority
            root_times = root_port.designated_times

            port_roles[root_port.ofport.port_no] = ROOT_PORT

            d_ports = self._select_designated_port(root_port)
            for port_no in d_ports:
                port_roles[port_no] = DESIGNATED_PORT

            for port in self.ports.values():
                if port.state is not PORT_STATE_DISABLE:
                    port_roles.setdefault(port.ofport.port_no,
                                          NON_DESIGNATED_PORT)

        return port_roles, root_priority, root_times
```

Port Status Change

Status change processing of ports is executed by the status change control thread (`Port.state_machine`) of the Port class. It uses `Port._get_timer()` to get the timer to change to the next status and after the timer elapses, uses `Port._get_next_state()` to get the next status to change the status. Also, the status is changed when `Port._change_status()` is executed in case that when STP re-calculation occurs and the status is changed to BLOCK status, regardless of the previous port status. This processing is achieved using `hub.Event` and `hub.Timeout` of the hub module, as with "Detecting Failures".

```
class Port(object):
```

```
    def _state_machine(self):
        """ Port state machine.
            Change next status when timer is exceeded
            or _change_status() method is called."""

        # ...

        while True:
            self.logger.info('[port=%d] %s / %s', self.ofport.port_no,
                             role_str[self.role], state_str[self.state],
                             extra=self.dpid_str)

            self.state_event = hub.Event()
            timer = self._get_timer()
            if timer:
                timeout = hub.Timeout(timer)
                try:
                    self.state_event.wait()
                except hub.Timeout as t:
                    if t is not timeout:
                        err_msg = 'Internal error. Not my timeout.'
                        raise RyuException(msg=err_msg)
                    new_state = self._get_next_state()
                    self._change_status(new_state, thread_switch=False)
                finally:
                    timeout.cancel()
            else:
                self.state_event.wait()

            self.state_event = None

    def _get_timer(self):
        timer = {PORT_STATE_DISABLE: None,
                 PORT_STATE_BLOCK: None,
                 PORT_STATE_LISTEN: self.port_times.forward_delay,
                 PORT_STATE_LEARN: self.port_times.forward_delay,
                 PORT_STATE_FORWARD: None}
        return timer[self.state]

    def _get_next_state(self):
        next_state = {PORT_STATE_DISABLE: None,
                      PORT_STATE_BLOCK: None,
                      PORT_STATE_LISTEN: PORT_STATE_LEARN,
                      PORT_STATE_LEARN: (PORT_STATE_FORWARD
                                         if (self.role is ROOT_PORT or
                                             self.role is DESIGNATED_PORT)
                                         else PORT_STATE_BLOCK),
                      PORT_STATE_FORWARD: None}
        return next_state[self.state]
```

5.4.2 Implementing the Application

This section explains the difference between the spanning tree application (simple_switch_stp_13.py), which supports OpenFlow 1.3 described in "Executing the Ryu Application" and the switching hub of " *Switching Hub*", in order.

Setting "_CONTEXTS"

As with " *Link Aggregation* ", register CONTEXT to use the STP library.

```
from ryu.lib import stplib

# ...

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'stplib': stplib.Stp}
```

```
    # ...
```

Setting Configuration

Use the `set_config()` method of the STP library to set configuration. Here, the following values are set as a sample.

OpenFlow switch	Item	Setting
dpid=0000000000000001	bridge.priority	0x8000
dpid=0000000000000002	bridge.priority	0x9000
dpid=0000000000000003	bridge.priority	0xa000

Using these settings, the bridge ID of the dpid=0000000000000001 OpenFlow switch is always the smallest value and is selected as the root bridge.

```
class SimpleSwitch13(app_manager.RyuApp):

    # ...

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.stp = kwargs['stplib']

        # Sample of stplib config.
        #  please refer to stplib.Stp.set_config() for details.
        config = {dpid_lib.str_to_dpid('0000000000000001'):
                     {'bridge': {'priority': 0x8000}},
                  dpid_lib.str_to_dpid('0000000000000002'):
                     {'bridge': {'priority': 0x9000}},
                  dpid_lib.str_to_dpid('0000000000000003'):
                     {'bridge': {'priority': 0xa000}}}
        self.stp.set_config(config)
```

STP Event Processing

As with " *Link Aggregation* ", prepare the event handler to receive events notified by the STP library.

By using the `stplib.EventPacketIn` event defined in the STP library, it is possible to receive packets other than BPDU packets; therefore, the same packet handling is performed as " *Switching Hub* ".

```
class SimpleSwitch13(app_manager.RyuApp):

    @set_ev_cls(stplib.EventPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):

        # ...
```

The change notification event (`stplib.EventTopologyChange`) of the network topology is received and the learned MAC address and registered flow entry are initialized.

```
class SimpleSwitch13(app_manager.RyuApp):

    def delete_flow(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        for dst in self.mac_to_port[datapath.id].keys():
            match = parser.OFPMatch(eth_dst=dst)
            mod = parser.OFPFlowMod(
                datapath, command=ofproto.OFPFC_DELETE,
                out_port=ofproto.OFPP_ANY, out_group=ofproto.OFPG_ANY,
                priority=1, match=match)
            datapath.send_msg(mod)

    # ...
```

```
    @set_ev_cls(stplib.EventTopologyChange, MAIN_DISPATCHER)
    def _topology_change_handler(self, ev):
        dp = ev.dp
        dpid_str = dpid_lib.dpid_to_str(dp.id)
        msg = 'Receive topology change event. Flush MAC table.'
        self.logger.debug("[dpid=%s] %s", dpid_str, msg)

        if dp.id in self.mac_to_port:
            self.delete_flow(dp)
            del self.mac_to_port[dp.id]
```

The change notification event (`stplib.EventPortStateChange`) of the port status is received and the debug log of the port status is output.

```
class SimpleSwitch13(app_manager.RyuApp):

    @set_ev_cls(stplib.EventPortStateChange, MAIN_DISPATCHER)
    def _port_state_change_handler(self, ev):
        dpid_str = dpid_lib.dpid_to_str(ev.dp.id)
        of_state = {stplib.PORT_STATE_DISABLE: 'DISABLE',
                    stplib.PORT_STATE_BLOCK: 'BLOCK',
                    stplib.PORT_STATE_LISTEN: 'LISTEN',
                    stplib.PORT_STATE_LEARN: 'LEARN',
                    stplib.PORT_STATE_FORWARD: 'FORWARD'}
        self.logger.debug("[dpid=%s][port=%d] state=%s",
                          dpid_str, ev.port_no, of_state[ev.port_state])
```

As explained above, by using the library that provides the spanning tree function and the application that uses the library, a switching hub application having a spanning tree function is achieved.

5.5 Conclusion

This section uses the spanning tree library as material to explain the following items:

- Method of implementing event waiting processing using hub.Event
- Method of implementing timer control processing using hub.Timeout

CHAPTER

SIX

OPENFLOW PROTOCOL

This section describes match, instructions and actions defined in the OpenFlow protocol.

6.1 Match

There are a variety of conditions that can be specified to match, and it grows each time OpenFlow is updated. OpenFlow 1.0 had 12 types but in OpenFlow 1.3 as many as 40 types of conditions are defined.

For details of individual matches, please refer to the OpenFlow specification. This section gives a brief description of the Match field in OpenFlow 1.3.

Match field name	Explanation
in_port	Port number of receiving port
in_phy_port	Physical port number of receiving port
metadata	Metadata used to pass information between tables
eth_dst	Destination MAC address of Ethernet
eth_src	Source MAC address of Ethernet
eth_type	Frame type of Ethernet
vlan_vid	VLAN ID
vlan_pcp	VLAN PCP
ip_dscp	IP DSCP
ip_ecn	IP ECN
ip_proto	Protocol type of IP
ipv4_src	Source IP address of IPv4
ipv4_dst	Destination IP address of IPv4
tcp_src	Source port number of TCP
tcp_dst	Destination port number of TCP
udp_src	Source port number of UDP
udp_dst	Destination port number of UDP
sctp_src	Source port number of SCTP
sctp_dst	Destination port number of SCTP
icmpv4_type	Type of ICMP
icmpv4_code	Code of ICMP
arp_op	Opcode of ARP
arp_spa	Source IP address of ARP
arp_tpa	Target IP address of ARP
arp_sha	Source MAC address of ARP
arp_tha	Target MAC address of ARP
ipv6_src	Source IP address of IPv6
ipv6_dst	Destination IP address of IPv6
ipv6_flabel	Flow label of IPv6
icmpv6_type	Type of ICMPv6
icmpv6_code	Code of ICMPv6
ipv6_nd_target	Target address of IPv6 neighbor discovery

Continued on next page

Table 6.1 – continued from previous page

Match field name	Explanation
ipv6_nd_sll	Source link-layer address of IPv6 neighbor discovery
ipv6_nd_tll	Target link-layer address of IPv6 neighbor discovery
mpls_label	MPLS label
mpls_tc	MPLS traffic class (TC)
mpls_bos	MPLS BoS bit
pbb_isid	I-SID of 802.1ah PBB
tunnel_id	Metadata about logical port
ipv6_exthdr	Pseudo-field of extension header of IPv6

Depending on fields such as the MAC address and IP address, you can further specify the mask.

6.2 Instruction

The instruction is intended to define what happens when a packet corresponding to the match is received. The following types are defined.

Instruction	Explanation
Goto Table (Required)	In OpenFlow 1.1 and later, multiple flow tables are supported. Using GotoTable, you can take over the process of matching packets to a flow table you specify. For example, you can set flow entry such as "Add a VLAN-ID200 to packets received on port 1 and send it to table 2". The table ID you specify must be a value greater than the current table ID.
Write Metadata (Optional)	Set the metadata that can be referenced in the following table.
Write Actions (Required)	Add an action that is specified in the current set of actions. If same type of action has been set already, it is replaced with the new action.
Apply Actions (Optional)	Immediately apply the specified action without changing the action set.
Clear Actions (Optional)	Delete all actions in the current action set.
Meter (Optional)	Apply the packet to the meter you specify.

The following classes corresponding to each instruction are implemented in Ryu.

- `OFPInstructionGotoTable`
- `OFPInstructionWriteMetadata`
- `OFPInstructionActions`
- `OFPInstructionMeter`

Write/Apply/Clear Actions is grouped into OPFInstructionActions and is selected at the time of instantiation.

Note: Support for Write Actions is said to be essential, but at the current time it is not supported in Open vSwitch. Apply Actions is supported, so you need to use it instead.

6.3 Action

The OFPActionOutput class is used to specify packet forwarding to be used in Packet-Out and Flow Mod messages. Specify the maximum data size (max_len) to be transmitted to the controller and the destination in the constructor arguments. For the destination, other than the physical port number of the switch, some defined value can be used.

Value	Explanation
OFPP_IN_PORT	Forwarded to the receive port
OFPP_TABLE	Applied to the first flow table.
OFPP_NORMAL	Forwarded by the L2/L3 switch function
OFPP_FLOOD	Flooded to all physical ports of the VLAN except blocked ports and receiving ports
OFPP_ALL	Forwarded to all physical ports except receiving ports
OFPP_CONTROLLER	Sent to the controller as a Packet-In message.
OFPP_LOCAL	Indicates a local port of the switch
OFPP_ANY	Meant to be used as a wild card when you select a port using Flow Mod (delete) or Stats Requests messages, and it's not used in packet forwarding.

When you specify 0 for max_len, binary data of packet is not attached to the Packet-In message. If `OFPCML_NO_BUFFER` is specified, the entire packet is attached to the Packet-In message without buffering the packet on the OpenFlow switch.

CHAPTER

SEVEN

PACKET LIBRARY

The Packet-Out and Packet-In message of OpenFlow have a field that enters a byte string that represents the contents of the raw packet. Ryu offers a library for easier handling of such raw packets from applications. This section describes this library.

7.1 Basic Usage

7.1.1 Protocol Header Class

The Ryu packet library offers classes corresponding to various protocol headers.

Protocols including the following are supported. For details of classes corresponding to each protocol, please refer to API Reference.

- arp
- bgp
- bpdu
- dhcp
- ethernet
- icmp
- icmpv6
- igmp
- ipv4
- ipv6
- llc
- lldp
- mpls
- ospf
- pbb
- sctp
- slow
- tcp
- udp
- vlan
- vrrp

__init__ argument name of each protocol header class is basically the same as the name that is used for RFC, etc. It is the same for the naming convention of instance attributes of the protocol header class. However, for __init__ argument name that corresponds to a field with a name that conflicts one built in to Python such as type, _ is attached at the end, such as type.

Some __init__ arguments have a default value and can be omitted. In the following example, version=4 etc. it omitted.

```
from ryu.lib.ofproto import inet
from ryu.lib.packet import ipv4

pkt_ipv4 = ipv4.ipv4(dst='192.0.2.1',
                     src='192.0.2.2',
                     proto=inet.IPPROTO_UDP)
```

```
print pkt_ipv4.dst
print pkt_ipv4.src
print pkt_ipv4.proto
```

7.1.2 Network Address

In the API of the library Ryu packet, basically the network address of the string representation is used. The following is an example.

Address type	Example of python string
MAC address	'00:03:47:8c:a1:b3'
IPv4 address	'192.0.2.1'
IPv6 address	'2001:db8::2'

7.1.3 Analysis of Packet (Parse)

Generate a corresponding python object from a sequence of bytes of the packet.

A specific example is as follows.

1. Generate ryu.lib.packet.packet.Packet class object. (Specify the byte string to be parsed into the data argument)
2. Using the get_protocol method etc. of object of 1 above, obtain the object corresponding to the respective protocol header.

```
pkt = packet.Packet(data=bin_packet)
pkt_ethernet = pkt.get_protocol(ethernet.ethernet)
if not pkt_ethernet:
    # non ethernet
    return
print pkt_ethernet.dst
print pkt_ethernet.src
print pkt_ethernet.ethertype
```

7.1.4 Generation of Packets (Serialization)

Generate a corresponding sequence of bytes of the packet from a python object.

A specific example is as follows.

1. Generate a ryu.lib.packet.packet.Packet class object.
2. Generate an object corresponding to each protocol header. (ethernet, ipv4, ...)
3. Using the add_protocol method of the object of 1. above, add a header of 2. above, in order.
4. Call the serialize method of the object object of 1. above, and generate the byte sequence.

Some fields such as payload length and checksum are calculated automatically at the time of serialization even if you do not explicitly specify a value. Please refer to the reference for each class for more information.

```
pkt = packet.Packet()
pkt.add_protocol(ethernet.ethernet(ethertype=...,
                                   dst=...,
                                   src=...))
pkt.add_protocol(ipv4.ipv4(dst=...,
                           src=...,
                           proto=...))
pkt.add_protocol(icmp.icmp(type_=...,
                           code=...,
                           csum=...,
                           data=...))
pkt.serialize()
bin_packet = pkt.data
```

An alternate API similar to Scapy is also available. Please use it according to your preference.

```
e = ethernet.ethernet(...)
i = ipv4.ipv4(...)
u = udp.udp(...)
pkt = e/i/u
```

7.2 Application Examples

The following is an example of an application that responds to a ping created using the above examples.

Receive ARP REQUEST and ICMP ECHO REQUEST with Packet-In and send a response with Packet-Out. IP addresses, etc. are hard-coded in the __init__ method.

```
# Copyright (C) 2013 Nippon Telegraph and Telephone Corporation.
# Copyright (C) 2013 YAMAMOTO Takashi <yamamoto at valinux co jp>
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# a simple ICMP Echo Responder

from ryu.base import app_manager

from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls

from ryu.ofproto import ofproto_v1_3

from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import arp
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp


class IcmpResponder(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(IcmpResponder, self).__init__(*args, **kwargs)
```

```
        self.hw_addr = '0a:e4:1c:d1:3e:44'
        self.ip_addr = '192.0.2.9'

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def _switch_features_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        actions = [parser.OFPActionOutput(port=ofproto.OFPP_CONTROLLER,
                                          max_len=ofproto.OFPCML_NO_BUFFER)]
        inst = [parser.OFPInstructionActions(type_=ofproto.OFPIT_APPLY_ACTIONS,
                                             actions=actions)]
        mod = parser.OFPFlowMod(datapath=datapath,
                                priority=0,
                                match=parser.OFPMatch(),
                                instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        port = msg.match['in_port']
        pkt = packet.Packet(data=msg.data)
        self.logger.info("packet-in %s" % (pkt,))
        pkt_ethernet = pkt.get_protocol(ethernet.ethernet)
        if not pkt_ethernet:
            return
        pkt_arp = pkt.get_protocol(arp.arp)
        if pkt_arp:
            self._handle_arp(datapath, port, pkt_ethernet, pkt_arp)
            return
        pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
        pkt_icmp = pkt.get_protocol(icmp.icmp)
        if pkt_icmp:
            self._handle_icmp(datapath, port, pkt_ethernet, pkt_ipv4, pkt_icmp)
            return

    def _handle_arp(self, datapath, port, pkt_ethernet, pkt_arp):
        if pkt_arp.opcode != arp.ARP_REQUEST:
            return
        pkt = packet.Packet()
        pkt.add_protocol(ethernet.ethernet(ethertype=pkt_ethernet.ethertype,
                                           dst=pkt_ethernet.src,
                                           src=self.hw_addr))
        pkt.add_protocol(arp.arp(opcode=arp.ARP_REPLY,
                                 src_mac=self.hw_addr,
                                 src_ip=self.ip_addr,
                                 dst_mac=pkt_arp.src_mac,
                                 dst_ip=pkt_arp.src_ip))
        self._send_packet(datapath, port, pkt)

    def _handle_icmp(self, datapath, port, pkt_ethernet, pkt_ipv4, pkt_icmp):
        if pkt_icmp.type != icmp.ICMP_ECHO_REQUEST:
            return
        pkt = packet.Packet()
        pkt.add_protocol(ethernet.ethernet(ethertype=pkt_ethernet.ethertype,
                                           dst=pkt_ethernet.src,
                                           src=self.hw_addr))
        pkt.add_protocol(ipv4.ipv4(dst=pkt_ipv4.src,
                                   src=self.ip_addr,
                                   proto=pkt_ipv4.proto))
        pkt.add_protocol(icmp.icmp(type_=icmp.ICMP_ECHO_REPLY,
                                   code=icmp.ICMP_ECHO_REPLY_CODE,
                                   csum=0,
                                   data=pkt_icmp.data))
        self._send_packet(datapath, port, pkt)

    def _send_packet(self, datapath, port, pkt):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        pkt.serialize()
```

```
        self.logger.info("packet-out %s" % (pkt,))
        data = pkt.data
        actions = [parser.OFPActionOutput(port=port)]
        out = parser.OFPPacketOut(datapath=datapath,
                                  buffer_id=ofproto.OFP_NO_BUFFER,
                                  in_port=ofproto.OFPP_CONTROLLER,
                                  actions=actions,
                                  data=data)
        datapath.send_msg(out)
```

Note: In OpenFlow 1.2 or later, you may retrieve the content of a parsed packet header from the match field of a Packet-In message. However, how much information is put in this field depends on the implementation of the switch. For example, Open vSwitch only puts in the minimum amount of information, and in many cases the content of the packet must be analyzed on the controller side. On the other hand, LINC puts in as much information as possible.

The following is an example of the log when you run ping-c 3.

```
EVENT ofp_event->IcmpResponder EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0xb63c802c OFPSwitchFeatures(auxiliary_id=0,
capabilities=71,datapath_id=11974852296259,n_buffers=256,n_tables=254)
move onto main mode
EVENT ofp_event->IcmpResponder EventOFPPacketIn
packet-in ethernet(dst='ff:ff:ff:ff:ff:ff',ethertype=2054,src='0a:e4:1c:d1:3e:43'), arp(dst_ip
='192.0.2.9',dst_mac='00:00:00:00:00:00',hlen=6,hwtype=1,opcode=1,plen=4,proto=2048,src_ip
='192.0.2.99',src_mac='0a:e4:1c:d1:3e:43'), '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00'
packet-out ethernet(dst='0a:e4:1c:d1:3e:43',ethertype=2054,src='0a:e4:1c:d1:3e:44'), arp(
dst_ip='192.0.2.99',dst_mac='0a:e4:1c:d1:3e:43',hlen=6,hwtype=1,opcode=2,plen=4,proto=2048,
src_ip='192.0.2.9',src_mac='0a:e4:1c:d1:3e:44')
EVENT ofp_event->IcmpResponder EventOFPPacketIn
packet-in ethernet(dst='0a:e4:1c:d1:3e:44',ethertype=2048,src='0a:e4:1c:d1:3e:43'), ipv4(csum
=47390,dst='192.0.2.9',flags=0,header_length=5,identification=32285,offset=0,option=None,proto
=1,src='192.0.2.99',tos=0,total_length=84,ttl=255,version=4), icmp(code=0,csum=38471,data=echo
(data='S,B\x00\x00\x00\x00\x00\x03L)(\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\
x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./\x00\x00\x00\x00\x00\x00\x00\x00',id=44565,seq=0),type
=8)
packet-out ethernet(dst='0a:e4:1c:d1:3e:43',ethertype=2048,src='0a:e4:1c:d1:3e:44'), ipv4(csum
=14140,dst='192.0.2.99',flags=0,header_length=5,identification=0,offset=0,option=None,proto=1,
src='192.0.2.9',tos=0,total_length=84,ttl=255,version=4), icmp(code=0,csum=40519,data=echo(
data='S,B\x00\x00\x00\x00\x00\x03L)(\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\
x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./\x00\x00\x00\x00\x00\x00\x00\x00',id=44565,seq=0),type
=0)
EVENT ofp_event->IcmpResponder EventOFPPacketIn
packet-in ethernet(dst='0a:e4:1c:d1:3e:44',ethertype=2048,src='0a:e4:1c:d1:3e:43'), ipv4(csum
=47383,dst='192.0.2.9',flags=0,header_length=5,identification=32292,offset=0,option=None,proto
=1,src='192.0.2.99',tos=0,total_length=84,ttl=255,version=4), icmp(code=0,csum=12667,data=echo
(data='T,B\x00\x00\x00\x00\x00Q\x17?(\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\
x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./\x00\x00\x00\x00\x00\x00\x00\x00',id=44565,seq=1),type
=8)
packet-out ethernet(dst='0a:e4:1c:d1:3e:43',ethertype=2048,src='0a:e4:1c:d1:3e:44'), ipv4(csum
=14140,dst='192.0.2.99',flags=0,header_length=5,identification=0,offset=0,option=None,proto=1,
src='192.0.2.9',tos=0,total_length=84,ttl=255,version=4), icmp(code=0,csum=14715,data=echo(
data='T,B\x00\x00\x00\x00\x00Q\x17?(\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\
x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./\x00\x00\x00\x00\x00\x00\x00\x00',id=44565,seq=1),type
=0)
EVENT ofp_event->IcmpResponder EventOFPPacketIn
packet-in ethernet(dst='0a:e4:1c:d1:3e:44',ethertype=2048,src='0a:e4:1c:d1:3e:43'), ipv4(csum
=47379,dst='192.0.2.9',flags=0,header_length=5,identification=32296,offset=0,option=None,proto
=1,src='192.0.2.99',tos=0,total_length=84,ttl=255,version=4), icmp(code=0,csum=26863,data=echo
(data='U,B\x00\x00\x00\x00\x00!\xa26(\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\
x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./\x00\x00\x00\x00\x00\x00\x00\x00',id=44565,seq=2),type
=8)
packet-out ethernet(dst='0a:e4:1c:d1:3e:43',ethertype=2048,src='0a:e4:1c:d1:3e:44'), ipv4(csum
=14140,dst='192.0.2.99',flags=0,header_length=5,identification=0,offset=0,option=None,proto=1,
src='192.0.2.9',tos=0,total_length=84,ttl=255,version=4), icmp(code=0,csum=28911,data=echo(
data='U,B\x00\x00\x00\x00\x00!\xa26(\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\
x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./\x00\x00\x00\x00\x00\x00\x00\x00',id=44565,seq=2),type
=0)
```

Handling of IP fragments will be an exercise for the reader. The OpenFlow protocol itself does not have a method to obtain the MTU, thus it needs to be hard-coded or requires some other idea. Also, since the Ryu packet library

will always parse or serialize the entire packet, you'll need API change to process packets that are fragmented.

CHAPTER

EIGHT

OF-CONFIG LIBRARY

This section describes Client library of OF-Config that comes with Ryu.

8.1 OF-Config Protocol

OF-Config is a protocol for the management of the OpenFlow switch. It has been defined as the NETCONF (RFC 6241) schema and can perform status acquisition and settings of logical switch, port, and queue.

It's formulated by ONF the same as OpenFlow and specifications can be obtained from the following website.

https://www.opennetworking.org/sdn-resources/onf-specifications/openflow-config

This library is in compliance with OF-Config 1.1.1.

Note: Currently Open vSwitch does not support OF-Config, but it does offer a service called OVSDB for the same purpose. OF-Config is a relatively new standard and did not yet exist when Open vSwitch implemented OVSDB.

The OVSDB protocol specification is published as 7047 RFC, but for all practical purposes it is become a protocol only for Open vSwitch. OF-Config only recently appeared but it is expected to be implemented a lot in future OpenFlow switches.

8.2 Library Configuration

8.2.1 ryu.lib.of_config.capable_switch.OFCapableSwitch Class

A class to handle NETCONF sessions.

```
from ryu.lib.of_config.capable_switch import OFCapableSwitch
```

8.2.2 ryu.lib.of_config.classes Module

A module to provide a set of classes to treat setting contents as python objects.

Note: The class name is basically the same as the names used by the grouping keyword in the yang specification of OF-Config 1.1.1. Example: OFPortType

```
import ryu.lib.of_config.classes as ofc
```

8.3 Usage Example

8.3.1 Connection to the Switch

Connect to the switch using SSH transport. For unknown_host_cb, specify a callback function that performs processing of an unknown SSH host key, but right now it is set to continue the connection unconditionally.

```
sess = OFCapableSwitch(
    host='localhost',
    port=1830,
    username='linc',
    password='linc',
    unknown_host_cb=lambda host, fingeprint: True)
```

8.3.2 GET

The following is an example to obtain the state using NETCONF GET. It displays /resources/port/resource-id and /resources/port/current-rate of all ports.

```
csw = sess.get()
for p in csw.resources.port:
    print p.resource_id, p.current_rate
```

8.3.3 GET-CONFIG

The following is an example to obtain settings using NETCONF GET-CONFIG.

Note: running is a currently running setting at data store of NETCONF. It depends on the implementation, but you can also use a data store such as startup (settings that are loaded when you start the device) and candidate (candidate set).

It displays the /resources/port/resource-id and /resources/port/configuration/admin-state of all ports.

```
csw = sess.get_config('running')
for p in csw.resources.port:
    print p.resource_id, p.configuration.admin_state
```

8.3.4 EDIT-CONFIG

The following is an example of how you can change settings using NETCONF EDIT-CONFIG. The basic procedure is to obtain settings using GET-CONFIG, edit them and send them back using EDIT-CONFIG.

Note: You can also partially edit settings in EDIT-CONFIG on the protocol, but this usage is safer.

Set /resources/port/configuration/admin-state of all ports to down.

```
csw = sess.get_config('running')
for p in csw.resources.port:
    p.configuration.admin_state = 'down'
sess.edit_config('running', csw)
```

CHAPTER

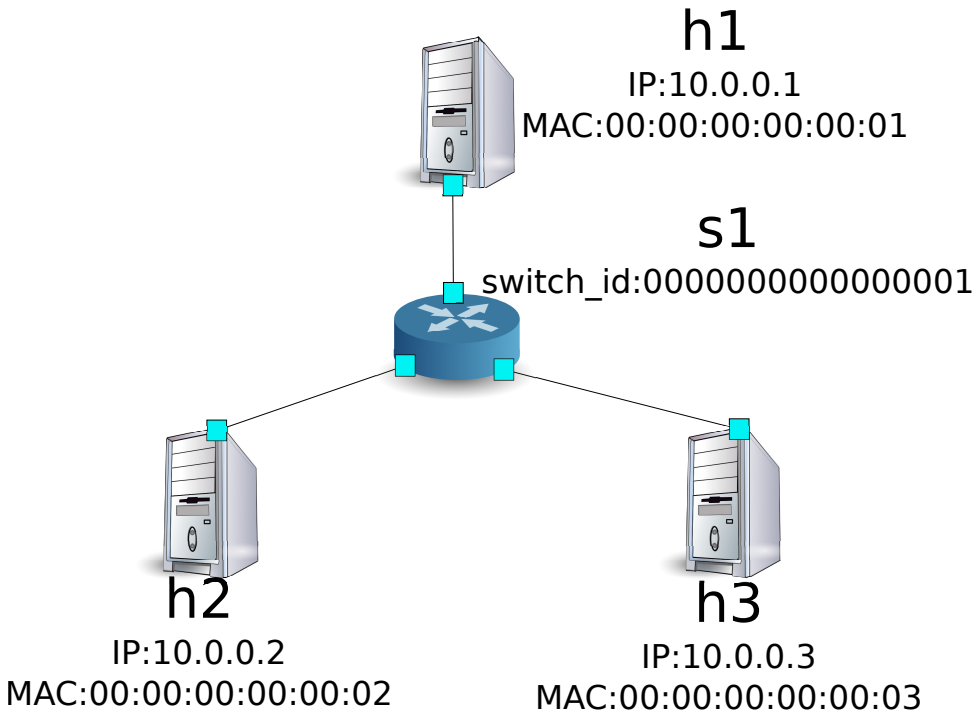
NINE

FIREWALL

This section describes how to use Firewall, which can be set using REST.

9.1 Example of operation of a single tenant

The following shows an example of creating a topology such as the following and adding or deleting a route or address for switch s1.



9.1.1 Environment Building

First, build an environment on Mininet. The commands to be entered are the same as "*Switching Hub*"

```
ryu@ryu-vm:~$ sudo mn --topo single,3 --mac --switch ovsk --controller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
```

```
*** Running terms on localhost:10.0
*** Starting controller
*** Starting 1 switches
s1

*** Starting CLI:
mininet>
```

Also, start another xterm for the controller.

```
mininet> xterm c0
mininet>
```

Next, set the version of OpenFlow to be used in each router to 1.3.

switch: s1 (root):

```
root@ryu-vm:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

Finally, start rest_firewall on xterm of the controller.

controller: c0 (root):

```
root@ryu-vm:~# ryu-manager ryu.app.rest_firewall
loading app ryu.app.rest_firewall
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.rest_firewall of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(2210) wsgi starting up on http://0.0.0.0:8080/
```

After a successful connection between the router and Ryu, the following message appears.

controller: c0 (root):

```
[FW][INFO] switch_id=0000000000000001: Join as firewall
```

9.1.2 Changing in the initial state

Immediately after starting the firewall, it was set to disable status to cut off all communication. Enable it with the following command.

Note: For details of REST API used in the following description, please see "REST API List" at the end of the section.

Node: c0 (root):

```
root@ryu-vm:~# curl -X PUT http://localhost:8080/firewall/module/enable/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": {
        "result": "success",
        "details": "firewall running."
      }
    }
  ]

root@ryu-vm:~# curl http://localhost:8080/firewall/module/status
  [
    {
      "status": "enable",
      "switch_id": "0000000000000001"
    }
  ]
```

Note: The result of the REST command is formatted for easy viewing.

Check ping communication from h1 to h2. Since access permission rules are not set, communication will be blocked.

host: h1:

```
root@ryu-vm:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
20 packets transmitted, 0 received, 100% packet loss, time 19003ms
```

Packets that are blocked are output to the log.

controller: c0 (root):

```
[FW][INFO] dpid=0000000000000001: Blocked packet = ethernet(dst='00:00:00:00:00:02',ethertype
=2048,src='00:00:00:00:00:01'), ipv4(csum=9895,dst='10.0.0.2',flags=2,header_length=5,
identification=0,offset=0,option=None,proto=1,src='10.0.0.1',tos=0,total_length=84,ttl=64,
version=4), icmp(code=0,csum=55644,data=echo(data='K\x8e\xaeR\x00\x00\x00\x00=\xc6\r\x00\x00\
x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()
*+,-./01234567',id=6952,seq=1),type=8)
...
```

9.1.3 Adding a Rule

Add a rule to permit pinging between h1 and h2. You need to add the rule for both ways.

Let's add the following rules. Rule ID is assigned automatically.

Source	Destination	Protocol	Permission	(Rule ID)
10.0.0.1/32	10.0.0.2/32	ICMP	Allow	1
10.0.0.2/32	10.0.0.1/32	ICMP	Allow	2

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"nw_src": "10.0.0.1/32", "nw_dst": "10.0.0.2/32", "nw_proto":
 "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Rule added. : rule_id=1"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst": "10.0.0.1/32", "nw_proto":
 "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Rule added. : rule_id=2"
        }
      ]
    }
  ]
```

Added rules are registered in the switch as flow entries.

switch: s1 (root):

```
root@ryu-vm:~# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=823.705s, table=0, n_packets=10, n_bytes=420, priority=65534,arp actions
=NORMAL
 cookie=0x0, duration=542.472s, table=0, n_packets=20, n_bytes=1960, priority=0 actions=
CONTROLLER:128
 cookie=0x1, duration=145.05s, table=0, n_packets=0, n_bytes=0, priority=1,icmp,nw_src
=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x2, duration=118.265s, table=0, n_packets=0, n_bytes=0, priority=1,icmp,nw_src
=10.0.0.2,nw_dst=10.0.0.1 actions=NORMAL
```

In addition, add a rule to allow all IPv4 packets, including ping, between h3 and h2.

Source	Destination	Protocol	Permission	(Rule ID)
10.0.0.2/32	10.0.0.3/32	any	Allow	3
10.0.0.3/32	10.0.0.2/32	any	Allow	4

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst": "10.0.0.3/32"}' http://
localhost:8080/firewall/rules/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Rule added. : rule_id=3"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"nw_src": "10.0.0.3/32", "nw_dst": "10.0.0.2/32"}' http://
localhost:8080/firewall/rules/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Rule added. : rule_id=4"
        }
      ]
    }
  ]
```

Added rules are registered in the switch as flow entries.

switch: s1 (root):

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x3, duration=12.724s, table=0, n_packets=0, n_bytes=0, priority=1,ip,nw_src=10.0.0.2,
nw_dst=10.0.0.3 actions=NORMAL
 cookie=0x4, duration=3.668s, table=0, n_packets=0, n_bytes=0, priority=1,ip,nw_src=10.0.0.3,
nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x0, duration=1040.802s, table=0, n_packets=10, n_bytes=420, priority=65534,arp
actions=NORMAL
 cookie=0x0, duration=759.569s, table=0, n_packets=20, n_bytes=1960, priority=0 actions=
CONTROLLER:128
 cookie=0x1, duration=362.147s, table=0, n_packets=0, n_bytes=0, priority=1,icmp,nw_src
=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x2, duration=335.362s, table=0, n_packets=0, n_bytes=0, priority=1,icmp,nw_src
=10.0.0.2,nw_dst=10.0.0.1 actions=NORMAL
```

Priority can be set to rules.

Try to add a rule to block pings (ICMP) between h3 and h2. Set a value greater than 1, which is default value of priority.

(Priority)	Source	Destination	Protocol	Permission	(Rule ID)
10	10.0.0.2/32	10.0.0.3/32	ICMP	Block	5
10	10.0.0.3/32	10.0.0.2/32	ICMP	Block	6

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"nw_src": "10.0.0.2/32", "nw_dst": "10.0.0.3/32", "nw_proto
": "ICMP", "actions": "DENY", "priority": "10"}' http://localhost:8080/firewall/rules
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Rule added. : rule_id=5"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"nw_src": "10.0.0.3/32", "nw_dst": "10.0.0.2/32", "nw_proto
": "ICMP", "actions": "DENY", "priority": "10"}' http://localhost:8080/firewall/rules
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Rule added. : rule_id=6"
        }
      ]
    }
  ]
```

Added rules are registered in the switch as flow entries.

switch: s1 (root):

```
root@ryu-vm:~# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x3, duration=242.155s, table=0, n_packets=0, n_bytes=0, priority=1,ip,nw_src
=10.0.0.2,nw_dst=10.0.0.3 actions=NORMAL
 cookie=0x4, duration=233.099s, table=0, n_packets=0, n_bytes=0, priority=1,ip,nw_src
=10.0.0.3,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x0, duration=1270.233s, table=0, n_packets=10, n_bytes=420, priority=65534,arp
actions=NORMAL
 cookie=0x0, duration=989s, table=0, n_packets=20, n_bytes=1960, priority=0 actions=CONTROLLER
:128
 cookie=0x5, duration=26.984s, table=0, n_packets=0, n_bytes=0, priority=10,icmp,nw_src
=10.0.0.2,nw_dst=10.0.0.3 actions=CONTROLLER:128
 cookie=0x1, duration=591.578s, table=0, n_packets=0, n_bytes=0, priority=1,icmp,nw_src
=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x6, duration=14.523s, table=0, n_packets=0, n_bytes=0, priority=10,icmp,nw_src
=10.0.0.3,nw_dst=10.0.0.2 actions=CONTROLLER:128
 cookie=0x2, duration=564.793s, table=0, n_packets=0, n_bytes=0, priority=1,icmp,nw_src
=10.0.0.2,nw_dst=10.0.0.1 actions=NORMAL
```

9.1.4 Confirming Rules

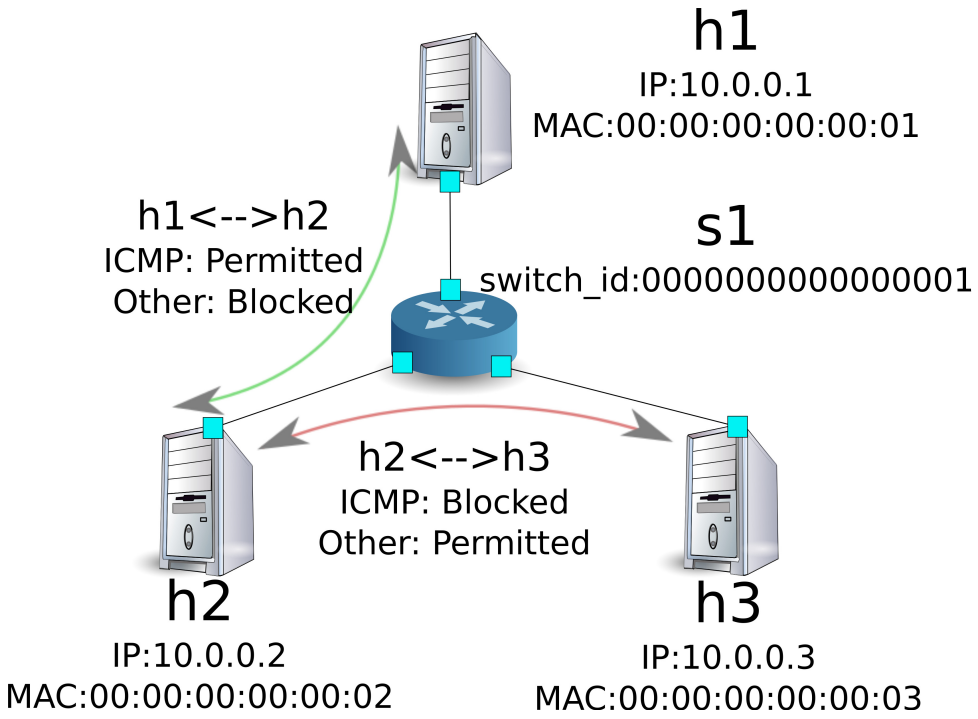
Confirm the rules that have been set.

Node: c0 (root):

```
root@ryu-vm:~# curl http://localhost:8080/firewall/rules/0000000000000001
  [
    {
      "access_control_list": [
        {
```

```
        "rules": [
          {
            "priority": 1,
            "dl_type": "IPv4",
            "nw_dst": "10.0.0.3",
            "nw_src": "10.0.0.2",
            "rule_id": 3,
            "actions": "ALLOW"
          },
          {
            "priority": 1,
            "dl_type": "IPv4",
            "nw_dst": "10.0.0.2",
            "nw_src": "10.0.0.3",
            "rule_id": 4,
            "actions": "ALLOW"
          },
          {
            "priority": 10,
            "dl_type": "IPv4",
            "nw_proto": "ICMP",
            "nw_dst": "10.0.0.3",
            "nw_src": "10.0.0.2",
            "rule_id": 5,
            "actions": "DENY"
          },
          {
            "priority": 1,
            "dl_type": "IPv4",
            "nw_proto": "ICMP",
            "nw_dst": "10.0.0.2",
            "nw_src": "10.0.0.1",
            "rule_id": 1,
            "actions": "ALLOW"
          },
          {
            "priority": 10,
            "dl_type": "IPv4",
            "nw_proto": "ICMP",
            "nw_dst": "10.0.0.2",
            "nw_src": "10.0.0.3",
            "rule_id": 6,
            "actions": "DENY"
          },
          {
            "priority": 1,
            "dl_type": "IPv4",
            "nw_proto": "ICMP",
            "nw_dst": "10.0.0.1",
            "nw_src": "10.0.0.2",
            "rule_id": 2,
            "actions": "ALLOW"
          }
        ]
      }
    ],
    "switch_id": "0000000000000001"
  }
]
```


The following shows the rules set.



Try to send a ping from h1 to h2. Since rules to allow communication are set, the ping will go through.

host: h1:

```
root@ryu-vm:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.419 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.033 ms
...
```

Packets from h1 to h2 other than ping are blocked by the firewall. For example, if you execute wget from h1 to h2, a log is output that packets were blocked.

host: h1:

```
root@ryu-vm:~# wget http://10.0.0.2
--2013-12-16 15:00:38--  http://10.0.0.2/
Connecting to 10.0.0.2:80... ^C
```

controller: c0 (root):

```
[FW][INFO] dpid=0000000000000001: Blocked packet = ethernet(dst='00:00:00:00:00:02',ethertype
=2048,src='00:00:00:00:00:01'), ipv4(csum=4812,dst='10.0.0.2',flags=2,header_length=5,
identification=5102,offset=0,option=None,proto=6,src='10.0.0.1',tos=0,total_length=60,ttl=64,
version=4), tcp(ack=0,bits=2,csum=45753,dst_port=80,offset=10,option='\x02\x04\x05\xb4\x04\x02
\x08\n\x00H:\x99\x00\x00\x00\x00\x01\x03\x03\t',seq=1021913463,src_port=42664,urgent=0,
window_size=14600)
...
```

Between h2 and 3h, packets other than ping can communicate. For example, if you execute ssh from h2 to h3, it will not output a log that packets were blocked. (Since sshd is not operating in h3, communication by ssh will fail.)

host: h2:

```
root@ryu-vm:~# ssh 10.0.0.3
ssh: connect to host 10.0.0.3 port 22: Connection refused
```

If you execute ping from h2 to h3, a log is output that packets were blocked by the firewall.

host: h2:

```
root@ryu-vm:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7055ms
```

controller: c0 (root):

```
[FW][INFO] dpid=0000000000000001: Blocked packet = ethernet(dst='00:00:00:00:00:03',ethertype
=2048,src='00:00:00:00:00:02'), ipv4(csum=9893,dst='10.0.0.3',flags=2,header_length=5,
identification=0,offset=0,option=None,proto=1,src='10.0.0.2',tos=0,total_length=84,ttl=64,
version=4), icmp(code=0,csum=35642,data=echo(data='\r\x12\xcaR\x00\x00\x00\x00\xab\x8b\t\x00\
x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()
*+,-./01234567',id=8705,seq=1),type=8)
...
```

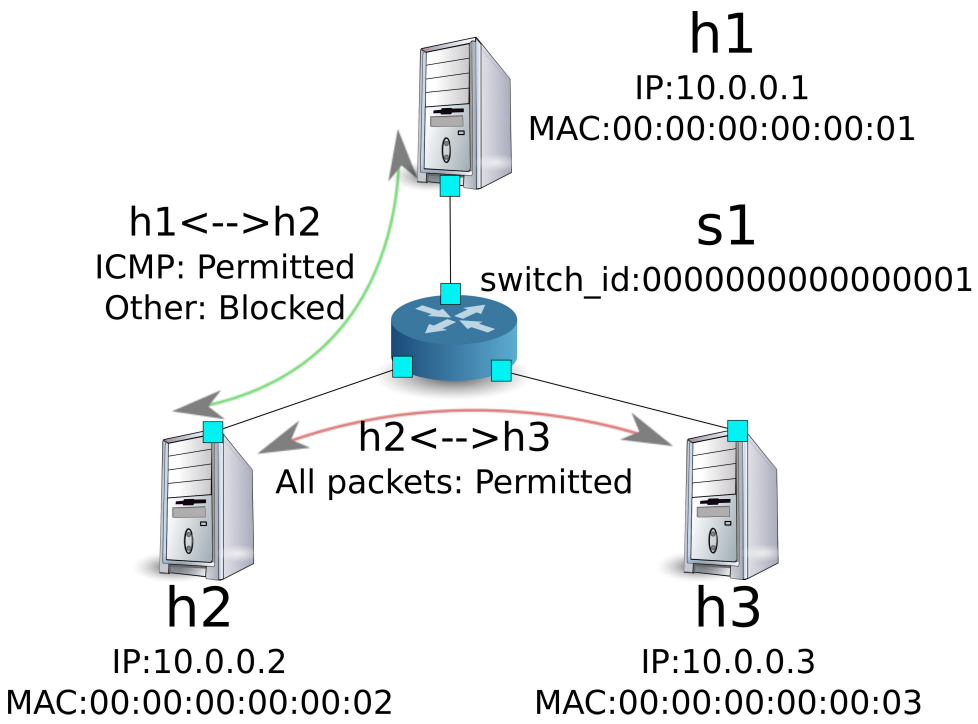
9.1.5 Deleting a Rule

Delete the "rule_id:5" and "rule_id:6" rules.

Node: c0 (root):

```
root@ryu-vm:~# curl -X DELETE -d '{"rule_id": "5"}' http://localhost:8080/firewall/rules
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Rule deleted. : ruleID=5"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X DELETE -d '{"rule_id": "6"}' http://localhost:8080/firewall/rules
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Rule deleted. : ruleID=6"
        }
      ]
    }
  ]
```

The following shows the current rules.



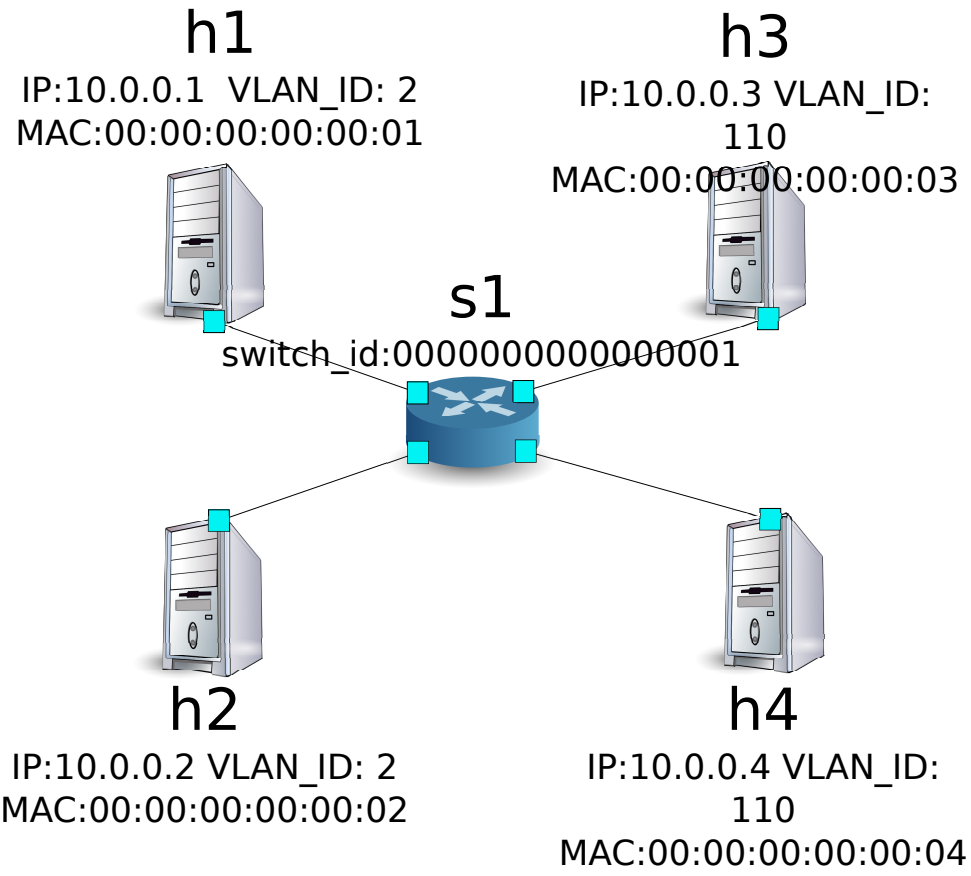
Let's confirm them. Since the rule to block pings (ICMP) between h2 and h3 is deleted, you can see that ping can now communicate.

host: h2:

```
root@ryu-vm:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.841 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.036 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=0.026 ms
64 bytes from 10.0.0.3: icmp_req=4 ttl=64 time=0.033 ms
...
```

9.2 Example of the Operation of a Multi-tenant

The following shows an example of creating a topology where tenants are divided by VLAN such as the following and routes or addresses for each switch s1 are added or deleted and communication availability between each host is verified.



9.2.1 Building an Environment

As with the example of Single-tenant, build an environment on Mininet and start another xterm for controller. Note that there is one more host to be used than before.

```
ryu@ryu-vm:~$ sudo mn --topo single,4 --mac --switch ovsk --controller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Running terms on localhost:10.0
*** Starting controller
*** Starting 1 switches
s1

*** Starting CLI:
mininet> xterm c0
mininet>
```

Next, set the VLAN ID to the interface of each host.

host: h1:

```
root@ryu-vm:~# ip addr del 10.0.0.1/8 dev h1-eth0
root@ryu-vm:~# ip link add link h1-eth0 name h1-eth0.2 type vlan id 2
root@ryu-vm:~# ip addr add 10.0.0.1/8 dev h1-eth0.2
root@ryu-vm:~# ip link set dev h1-eth0.2 up
```

host: h2:

```
root@ryu-vm:~# ip addr del 10.0.0.2/8 dev h2-eth0
root@ryu-vm:~# ip link add link h2-eth0 name h2-eth0.2 type vlan id 2
root@ryu-vm:~# ip addr add 10.0.0.2/8 dev h2-eth0.2
root@ryu-vm:~# ip link set dev h2-eth0.2 up
```

host: h3:

```
root@ryu-vm:~# ip addr del 10.0.0.3/8 dev h3-eth0
root@ryu-vm:~# ip link add link h3-eth0 name h3-eth0.110 type vlan id 110
root@ryu-vm:~# ip addr add 10.0.0.3/8 dev h3-eth0.110
root@ryu-vm:~# ip link set dev h3-eth0.110 up
```

host: h4:

```
root@ryu-vm:~# ip addr del 10.0.0.4/8 dev h4-eth0
root@ryu-vm:~# ip link add link h4-eth0 name h4-eth0.110 type vlan id 110
root@ryu-vm:~# ip addr add 10.0.0.4/8 dev h4-eth0.110
root@ryu-vm:~# ip link set dev h4-eth0.110 up
```

Then, set the version of OpenFlow to be used in each router to 1.3.

switch: s1 (root):

```
root@ryu-vm:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

Finally, start rest_firewall on an xterm of the controller.

controller: c0 (root):

```
root@ryu-vm:~# ryu-manager ryu.app.rest_firewall
loading app ryu.app.rest_firewall
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.rest_firewall of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(13419) wsgi starting up on http://0.0.0.0:8080/
```

After a successful connection between the router and Ryu, the following message appears.

controller: c0 (root):

```
[FW][INFO] switch_id=0000000000000001: Join as firewall
```

9.2.2 Changing the Initial State

Enable the firewall.

Node: c0 (root):

```
root@ryu-vm:~# curl -X PUT http://localhost:8080/firewall/module/enable/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": {
        "result": "success",
        "details": "firewall running."
      }
    }
  ]

root@ryu-vm:~# curl http://localhost:8080/firewall/module/status
  [
    {
      "status": "enable",
      "switch_id": "0000000000000001"
    }
  ]
```

9.2.3 Adding Rules

Add a rule to vlan_id = 2 that allows pings (ICMP packets) to be sent and received to 10.0.0.0/8. Since there is a need to set rules in both directions, add two rules.

(Priority)	VLAN ID	Source	Destination	Protocol	Permission	(Rule ID)
1	2	10.0.0.0/8	any	ICMP	Allow	1
1	2	any	10.0.0.0/8	ICMP	Allow	2

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"nw_src": "10.0.0.0/8", "nw_proto": "ICMP"}' http://localhost
:8080/firewall/rules/0000000000000001/2
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Rule added. : rule_id=1"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"nw_dst": "10.0.0.0/8", "nw_proto": "ICMP"}' http://localhost
:8080/firewall/rules/0000000000000001/2
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Rule added. : rule_id=2"
        }
      ]
    }
  ]
```

9.2.4 Confirming Rules

Confirm the rules that have been set.

Node: c0 (root):

```
root@ryu-vm:~# curl http://localhost:8080/firewall/rules/0000000000000001/all
  [
    {
      "access_control_list": [
        {
          "rules": [
            {
              "priority": 1,
              "dl_type": "IPv4",
              "nw_proto": "ICMP",
              "dl_vlan": 2,
              "nw_src": "10.0.0.0/8",
              "rule_id": 1,
              "actions": "ALLOW"
            },
            {
              "priority": 1,
              "dl_type": "IPv4",
              "nw_proto": "ICMP",
              "nw_dst": "10.0.0.0/8",
              "dl_vlan": 2,
              "rule_id": 2,
              "actions": "ALLOW"
```

```
            }
          ],
          "vlan_id": 2
        }
      ],
      "switch_id": "0000000000000001"
    }
  ]
```

Let's confirm them. When you execute ping from h1, which is vlan_id=2, to h2 which is also vlan_id=2, you can see that it can communicate per the rules that were added.

host: h1:

```
root@ryu-vm:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.893 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.098 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.122 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.047 ms
...
```

Since there is no rule between h3 and h4, which are both vlan_id=110, packets are blocked.

host: h3:

```
root@ryu-vm:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
^C
--- 10.0.0.4 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 4999ms
```

Since packets are blocked, a log is output.

controller: c0 (root):

```
[FW][INFO] dpid=0000000000000001: Blocked packet = ethernet(dst='00:00:00:00:00:04',ethertype
=33024,src='00:00:00:00:00:03'), vlan(cfi=0,ethertype=2048,pcp=0,vid=110), ipv4(csum=9891,dst
='10.0.0.4',flags=2,header_length=5,identification=0,offset=0,option=None,proto=1,src
='10.0.0.3',tos=0,total_length=84,ttl=64,version=4), icmp(code=0,csum=58104,data=echo(data='\
xb8\xa9\xaeR\x00\x00\x00\x00\xce\xe3\x02\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\
x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567',id=7760,seq=4),type=8)
...
```

In this section, you learned how to use the firewall with specific examples.

9.3 REST API List

List of REST API of rest_firewall, which is introduced in this section.

9.3.1 Acquiring Enable/Disable State of All Switches

Method	GET
URL	/firewall/module/status

9.3.2 Changing Enable/Disable State of All Switches

Method	PUT
URL	/firewall/module/{**op**}/{**switch**} –**op**: ["enable" | "disable"] –**switch**: ["all" | *Switch ID*]
Remarks	Initial state of each switch is "disable"

9.3.3 Acquiring All Rules

Method	GET
URL	/firewall/rules/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Remarks	Specification of VLAN ID is optional.

9.3.4 Adding Rules

Method	POST
URL	/firewall/rules/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Data	**priority**:[0 - 65535] **in_port**:[0 - 65535] **dl_src**:"<xx:xx:xx:xx:xx:xx>" **dl_dst**:"<xx:xx:xx:xx:xx:xx>" **dl_type**:["ARP" | "IPv4"] **nw_src**:"<xxx.xxx.xxx.xxx/xx>" **nw_dst**:"<xxx.xxx.xxx.xxx/xx">" **nw_proto**":["TCP" | "UDP" | "ICMP"] **tp_src**:[0 - 65535] **tp_dst**:[0 - 65535] **actions**: ["ALLOW" | "DENY"]
Remarks	When it is successfully registered, Rule ID is generated and is noted in the response. Specification of VLAN ID is optional.

9.3.5 Deleting Rules

Method	DELETE
URL	/firewall/rules/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Data	**rule_id**:["all" | 1 - ...]
Remarks	Specification of VLAN ID is optional.

9.3.6 Acquiring Log Output State of All Switches

Method	GET
URL	/firewall/log/status

9.3.7 Changing Log Output State of All Switches

Method	PUT
URL	/firewall/log/{**op**}/{**switch**} –**op**: ["enable" | "disable"] –**switch**: ["all" | *Switch ID*]
Remarks	Initial state of each switch is "enable"

CHAPTER

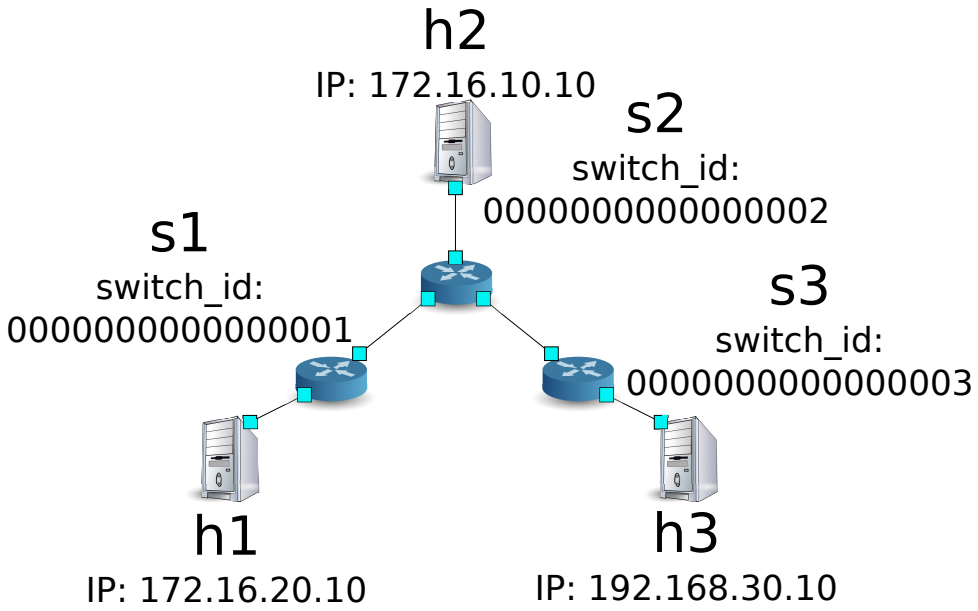
TEN

ROUTER

This section describes how to use a router that can be set using REST.

10.1 Example of the Operation of a Single Tenant

The following shows an example of creating topology such as the following and adding or deleting a route or address for each switch (router) and verifying communication availability between each host.



10.1.1 Building the environment

First, build an environment on Mininet. Parameters of the `mn` command are as follows.

Parameter	Value	Explanation
topo	linear,3	Topology where three switches are connected serially
mac	None	Set the MAC address of the host automatically
switch	ovsk	Use Open vSwitch
controller	remote	Use an external one for OpenFlow controller
x	None	Start xterm

An execution example is as follows.

```
ryu@ryu-vm:~$ sudo mn --topo linear,3 --mac --switch ovsk --controller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
```

```
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3
*** Running terms on localhost:10.0
*** Starting controller
*** Starting 3 switches
s1 s2 s3

*** Starting CLI:
mininet>
```

Also, start another xterm for the controller.

```
mininet> xterm c0
mininet>
```

Next, set the version of OpenFlow to be used in each router to 1.3.

switch: s1 (root):

```
root@ryu-vm:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

switch: s2 (root):

```
root@ryu-vm:~# ovs-vsctl set Bridge s2 protocols=OpenFlow13
```

switch: s3 (root):

```
root@ryu-vm:~# ovs-vsctl set Bridge s3 protocols=OpenFlow13
```

Then, delete the IP address that is assigned automatically on each host and set a new IP address.

host: h1:

```
root@ryu-vm:~# ip addr del 10.0.0.1/8 dev h1-eth0
root@ryu-vm:~# ip addr add 172.16.20.10/24 dev h1-eth0
```

host: h2:

```
root@ryu-vm:~# ip addr del 10.0.0.2/8 dev h2-eth0
root@ryu-vm:~# ip addr add 172.16.10.10/24 dev h2-eth0
```

host: h3:

```
root@ryu-vm:~# ip addr del 10.0.0.3/8 dev h3-eth0
root@ryu-vm:~# ip addr add 192.168.30.10/24 dev h3-eth0
```

Finally, start rest_router on xterm of controller.

controller: c0 (root):

```
root@ryu-vm:~# ryu-manager ryu.app.rest_router
loading app ryu.app.rest_router
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.rest_router of RestRouterAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(2212) wsgi starting up on http://0.0.0.0:8080/
```

After a successful connection between the router and Ryu, the following message appears.

controller: c0 (root):

```
[RT][INFO] switch_id=0000000000000003: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000003: Set ARP handling (packet in) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Set L2 switching (normal) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Set default route (drop) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000003: Join as router.
...
```

If the above log is displayed for the three routers, preparation is complete.

10.1.2 Setting the Address

Set the address for each router.

First, set the addresses “172.16.20.1/24” and “172.16.30.30/24” for router s1.

Note: For details of REST API used in the following description, see “REST API List” at the end of the section.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"address":"172.16.20.1/24"}' http://localhost:8080/router
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "172.16.30.30/24"}' http://localhost:8080/router
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]
```

Note: The result of the REST command is formatted for easy viewing.

Next, set the addresses “172.16.10.1/24”, “172.16.30.1/24” and “192.168.10.1/24” for router s2.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"address":"172.16.10.1/24"}' http://localhost:8080/router
/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]
```

```
root@ryu-vm:~# curl -X POST -d '{"address": "172.16.30.1/24"}' http://localhost:8080/router
/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "192.168.10.1/24"}' http://localhost:8080/router
/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=3]"
        }
      ]
    }
  ]
```

Then, set the addresses “192.168.30.1/24” and “192.168.10.20/24” for router s3.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"address": "192.168.30.1/24"}' http://localhost:8080/router
/0000000000000003
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "192.168.10.20/24"}' http://localhost:8080/router
/0000000000000003
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]
```

IP addresses to the router have been set. Register as the default gateway on each host.

host: h1:

```
root@ryu-vm:~# ip route add default via 172.16.20.1
```

host: h2:

```
root@ryu-vm:~# ip route add default via 172.16.10.1
```

host: h3:

```
root@ryu-vm:~# ip route add default via 192.168.30.1
```

10.1.3 Configuring the Default Route

Set the default route for each router.

First, set router s2 as the default route of router s1.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"gateway": "172.16.30.1"}' http://localhost:8080/router
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]
```

Set router s1 as the default route of router s2.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"gateway": "172.16.30.30"}' http://localhost:8080/router
/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]
```

Set router s2 as the default route of router s3.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"gateway": "192.168.10.1"}' http://localhost:8080/router
/0000000000000003
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]
```

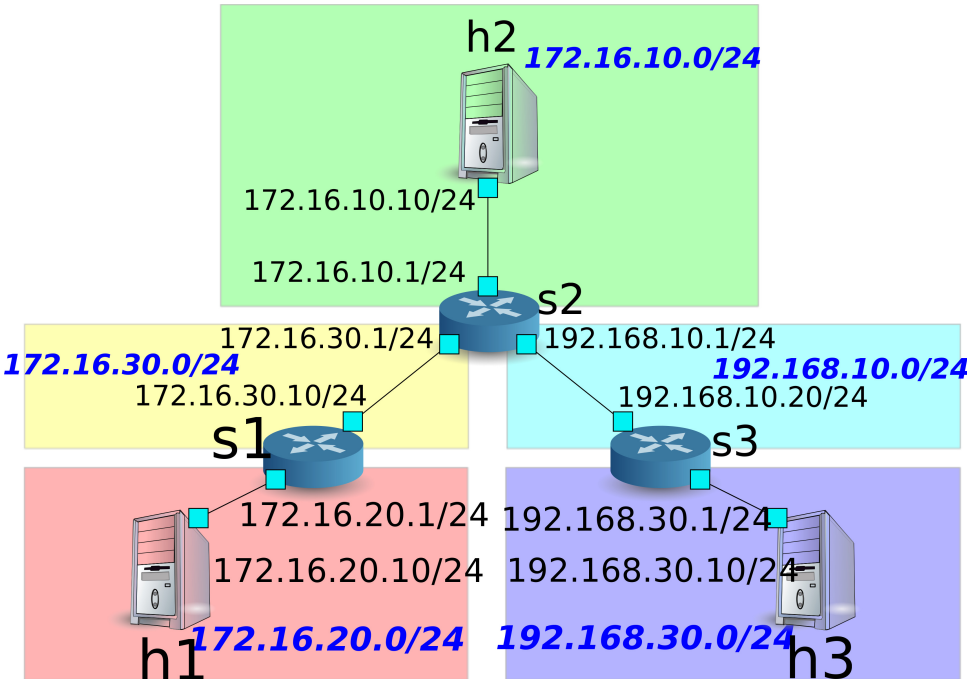
10.1.4 Setting Static Routes

For s2 router, set a static route to the host (192.168.30.0/24) under router s3.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"destination": "192.168.30.0/24", "gateway":
"192.168.10.20"}' http://localhost:8080/router/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Add route [route_id=2]"
        }
      ]
    }
  ]
```

The setting status of the route and address are as follows.



10.1.5 Verifying the Setting

Check the contents of the setting of each router.

Node: c0 (root):

```
root@ryu-vm:~# curl http://localhost:8080/router/0000000000000001
  [
    {
      "internal_network": [
        {
          "route": [
            {
              "route_id": 1,
              "destination": "0.0.0.0/0",
              "gateway": "172.16.30.1"
            }
          ],
          "address": [
            {
              "address_id": 1,
              "address": "172.16.20.1/24"
            },
            {
              "address_id": 2,
              "address": "172.16.30.30/24"
```

```
            }
          ]
        }
      ],
      "switch_id": "0000000000000001"
    }
  ]

root@ryu-vm:~# curl http://localhost:8080/router/0000000000000002
  [
    {
      "internal_network": [
        {
          "route": [
            {
              "route_id": 1,
              "destination": "0.0.0.0/0",
              "gateway": "172.16.30.30"
            },
            {
              "route_id": 2,
              "destination": "192.168.30.0/24",
              "gateway": "192.168.10.20"
            }
          ],
          "address": [
            {
              "address_id": 2,
              "address": "172.16.30.1/24"
            },
            {
              "address_id": 3,
              "address": "192.168.10.1/24"
            },
            {
              "address_id": 1,
              "address": "172.16.10.1/24"
            }
          ]
        }
      ],
      "switch_id": "0000000000000002"
    }
  ]

root@ryu-vm:~# curl http://localhost:8080/router/0000000000000003
  [
    {
      "internal_network": [
        {
          "route": [
            {
              "route_id": 1,
              "destination": "0.0.0.0/0",
              "gateway": "192.168.10.1"
            }
          ],
          "address": [
            {
              "address_id": 1,
              "address": "192.168.30.1/24"
            },
            {
              "address_id": 2,
              "address": "192.168.10.20/24"
            }
          ]
        }
      ],
      "switch_id": "0000000000000003"
    }
  ]
```

Check communication using ping in this state. First, run a ping from h3 to h2. You can verify that it can communicate successfully.

host: h2:

```
root@ryu-vm:~# ping 192.168.30.10
PING 192.168.30.10 (192.168.30.10) 56(84) bytes of data.
64 bytes from 192.168.30.10: icmp_req=1 ttl=62 time=48.8 ms
64 bytes from 192.168.30.10: icmp_req=2 ttl=62 time=0.402 ms
64 bytes from 192.168.30.10: icmp_req=3 ttl=62 time=0.089 ms
64 bytes from 192.168.30.10: icmp_req=4 ttl=62 time=0.065 ms
...
```

Next, run a ping from h2 to h1. You can also verify that it can communicate successfully.

host: h2:

```
root@ryu-vm:~# ping 172.16.20.10
PING 172.16.20.10 (172.16.20.10) 56(84) bytes of data.
64 bytes from 172.16.20.10: icmp_req=1 ttl=62 time=43.2 ms
64 bytes from 172.16.20.10: icmp_req=2 ttl=62 time=0.306 ms
64 bytes from 172.16.20.10: icmp_req=3 ttl=62 time=0.057 ms
64 bytes from 172.16.20.10: icmp_req=4 ttl=62 time=0.048 ms
...
```

10.1.6 Deleting the Static Route

Delete the static route to router s3 set in router s2.

Node: c0 (root):

```
root@ryu-vm:~# curl -X DELETE -d '{"route_id": "2"}' http://localhost:8080/router
/0000000000000002
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "details": "Delete route [route_id=2]"
        }
      ]
    }
  ]
```

Check the information that has been configured on router s2. You can see that the static route to router s3 has been deleted.

Node: c0 (root):

```
root@ryu-vm:~# curl http://localhost:8080/router/0000000000000002
  [
    {
      "internal_network": [
        {
          "route": [
            {
              "route_id": 1,
              "destination": "0.0.0.0/0",
              "gateway": "172.16.30.30"
            }
          ],
          "address": [
            {
              "address_id": 2,
              "address": "172.16.30.1/24"
            },
            {
              "address_id": 3,
              "address": "192.168.10.1/24"
```

```
            },
            {
              "address_id": 1,
              "address": "172.16.10.1/24"
            }
          ]
        }
      ],
      "switch_id": "0000000000000002"
    }
  ]
```

Check communication using ping. Since the root information from h3 to h2 was deleted, you will find that it cannot communicate.

host: h2:

```
root@ryu-vm:~# ping 192.168.30.10
PING 192.168.30.10 (192.168.30.10) 56(84) bytes of data.
^C
--- 192.168.30.10 ping statistics ---
12 packets transmitted, 0 received, 100% packet loss, time 11088ms
```

10.1.7 Deleting an Address

Delete the address "172.16.20.1/24", which is set in router s1.

Node: c0 (root):

```
root@ryu-vm:~# curl -X DELETE -d '{"address_id": "1"}' http://localhost:8080/router
/0000000000000001
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "details": "Delete address [address_id=1]"
        }
      ]
    }
  ]
```

Check the information that has been configured on router s1. You can see that of the IP addresses configured on router s1, "172.16.20.1/24" has been deleted.

Node: c0 (root):

```
root@ryu-vm:~# curl http://localhost:8080/router/0000000000000001
  [
    {
      "internal_network": [
        {
          "route": [
            {
              "route_id": 1,
              "destination": "0.0.0.0/0",
              "gateway": "172.16.30.1"
            }
          ],
          "address": [
            {
              "address_id": 2,
              "address": "172.16.30.30/24"
            }
          ]
        }
      ],
      "switch_id": "0000000000000001"
```

```
    }
  ]
```

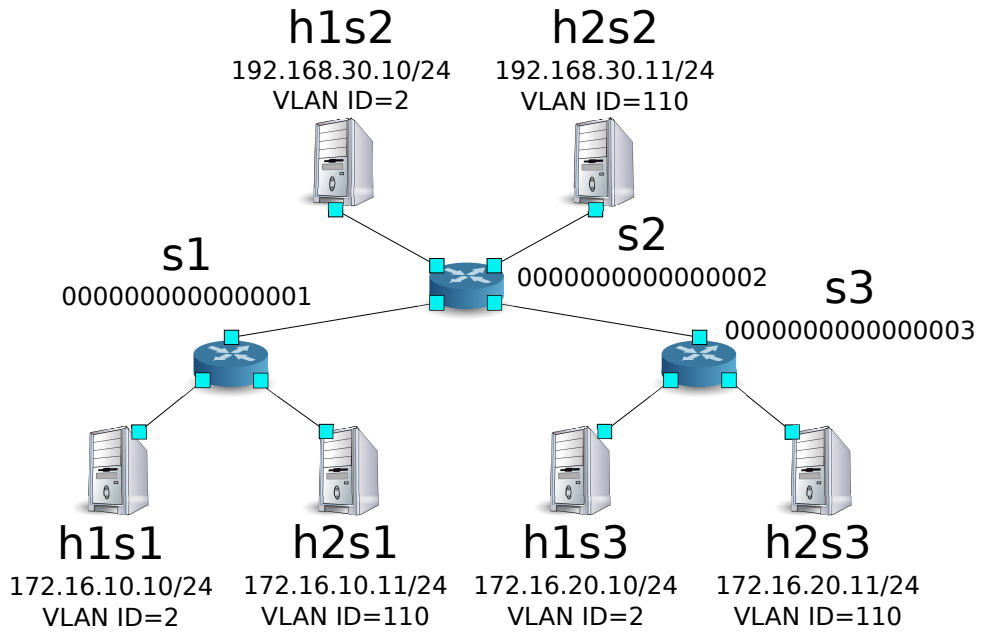
Check communication using ping. Since the information about the subnet to which h1 belongs has been removed from router s1, you can tell that communication from h2 to h1 is not possible.

host: h2:

```
root@ryu-vm:~# ping 172.16.20.10
PING 172.16.20.10 (172.16.20.10) 56(84) bytes of data.
^C
--- 172.16.20.10 ping statistics ---
19 packets transmitted, 0 received, 100% packet loss, time 18004ms
```

10.2 Example of the Operation of a Multi-tenant

The following shows an example of creating a topology where tenants are divided by VLAN such as the following and routes or addresses for each switch (router) are added or deleted and communication availability between each host is verified.



10.2.1 Environment building

First, build an environment on Mininet. Parameters of the `mn` command are as follows.

Parameter	Value	Example
topo	linear,3,2	Topology where three switches are connected serially (Two hosts are connected to each switch)
mac	None	Set the MAC address of the host automatically
switch	ovsk	Use Open vSwitch
controller	remote	Use an external one for OpenFlow controller
x	None	Start the xterm

A execution example is as follows.

```
ryu@ryu-vm:~$ sudo mn --topo linear,3,2 --mac --switch ovsk --controller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
```

```
*** Adding hosts:
h1s1 h1s2 h1s3 h2s1 h2s2 h2s3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1s1, s1) (h1s2, s2) (h1s3, s3) (h2s1, s1) (h2s2, s2) (h2s3, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1s1 h1s2 h1s3 h2s1 h2s2 h2s3
*** Running terms on localhost:10.0
*** Starting controller
*** Starting 3 switches
s1 s2 s3
*** Starting CLI:
mininet>
```

Also, start another xterm for the controller.

```
mininet> xterm c0
mininet>
```

Next, set the version of OpenFlow to be used in each router to 1.3.

switch: s1 (root):

```
root@ryu-vm:~# ovs-vsctl set Bridge s1 protocols=OpenFlow13
```

switch: s2 (root):

```
root@ryu-vm:~# ovs-vsctl set Bridge s2 protocols=OpenFlow13
```

switch: s3 (root):

```
root@ryu-vm:~# ovs-vsctl set Bridge s3 protocols=OpenFlow13
```

Then, set the VLAN ID to the interface of each host and set the new IP address.

host: h1s1:

```
root@ryu-vm:~# ip addr del 10.0.0.1/8 dev h1s1-eth0
root@ryu-vm:~# ip link add link h1s1-eth0 name h1s1-eth0.2 type vlan id 2
root@ryu-vm:~# ip addr add 172.16.10.10/24 dev h1s1-eth0.2
root@ryu-vm:~# ip link set dev h1s1-eth0.2 up
```

host: h2s1:

```
root@ryu-vm:~# ip addr del 10.0.0.4/8 dev h2s1-eth0
root@ryu-vm:~# ip link add link h2s1-eth0 name h2s1-eth0.110 type vlan id 110
root@ryu-vm:~# ip addr add 172.16.10.11/24 dev h2s1-eth0.110
root@ryu-vm:~# ip link set dev h2s1-eth0.110 up
```

host: h1s2:

```
root@ryu-vm:~# ip addr del 10.0.0.2/8 dev h1s2-eth0
root@ryu-vm:~# ip link add link h1s2-eth0 name h1s2-eth0.2 type vlan id 2
root@ryu-vm:~# ip addr add 192.168.30.10/24 dev h1s2-eth0.2
root@ryu-vm:~# ip link set dev h1s2-eth0.2 up
```

host: h2s2:

```
root@ryu-vm:~# ip addr del 10.0.0.5/8 dev h2s2-eth0
root@ryu-vm:~# ip link add link h2s2-eth0 name h2s2-eth0.110 type vlan id 110
root@ryu-vm:~# ip addr add 192.168.30.11/24 dev h2s2-eth0.110
root@ryu-vm:~# ip link set dev h2s2-eth0.110 up
```

host: h1s3:

```
root@ryu-vm:~# ip addr del 10.0.0.3/8 dev h1s3-eth0
root@ryu-vm:~# ip link add link h1s3-eth0 name h1s3-eth0.2 type vlan id 2
root@ryu-vm:~# ip addr add 172.16.20.10/24 dev h1s3-eth0.2
root@ryu-vm:~# ip link set dev h1s3-eth0.2 up
```

host: h2s3:

```
root@ryu-vm:~# ip addr del 10.0.0.6/8 dev h2s3-eth0
root@ryu-vm:~# ip link add link h2s3-eth0 name h2s3-eth0.110 type vlan id 110
root@ryu-vm:~# ip addr add 172.16.20.11/24 dev h2s3-eth0.110
root@ryu-vm:~# ip link set dev h2s3-eth0.110 up
```

Finally, start rest_router on xterm of controller.

controller: c0 (root):

```
root@ryu-vm:~# ryu-manager ryu.app.rest_router
loading app ryu.app.rest_router
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.app.rest_router of RestRouterAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(2447) wsgi starting up on http://0.0.0.0:8080/
```

After a successful connection between the router and Ryu, the following message appears.

controller: c0 (root):

```
[RT][INFO] switch_id=0000000000000003: Set SW config for TTL error packet in.
[RT][INFO] switch_id=0000000000000003: Set ARP handling (packet in) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Set L2 switching (normal) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Set default route (drop) flow [cookie=0x0]
[RT][INFO] switch_id=0000000000000003: Start cyclic routing table update.
[RT][INFO] switch_id=0000000000000003: Join as router.
...
```

If the above log is displayed for the three routers, preparation is complete.

10.2.2 Setting an Address

Set the address for each router.

First, set the addresses “172.16.20.1/24” and “10.10.10.1/24” to router s1. They must be set to each VLAN ID respectively.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"address": "172.16.10.1/24"}' http://localhost:8080/router
/0000000000000001/2
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "10.10.10.1/24"}' http://localhost:8080/router
/0000000000000001/2
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add address [address_id=2]"
        }
```

```
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "172.16.10.1/24"}' http://localhost:8080/router
/0000000000000001/110
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "10.10.10.1/24"}' http://localhost:8080/router
/0000000000000001/110
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]
```

Next, set the addresses "192.168.30.1/24" and "10.10.10.2/24" to router s2.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"address": "192.168.30.1/24"}' http://localhost:8080/router
/0000000000000002/2
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "10.10.10.2/24"}' http://localhost:8080/router
/0000000000000002/2
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "192.168.30.1/24"}' http://localhost:8080/router
/0000000000000002/110
  [
    {
      "switch_id": "0000000000000002",
```

```
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "10.10.10.2/24"}' http://localhost:8080/router
/0000000000000002/110
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]
```

Then, set the addresses "172.16.20.1/24" and "10.10.10.3/24" to router s3.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"address": "172.16.20.1/24"}' http://localhost:8080/router
/0000000000000003/2
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "10.10.10.3/24"}' http://localhost:8080/router
/0000000000000003/2
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"address": "172.16.20.1/24"}' http://localhost:8080/router
/0000000000000003/110
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add address [address_id=1]"
        }
      ]
    }
  ]
```

```
root@ryu-vm:~# curl -X POST -d '{"address": "10.10.10.3/24"}' http://localhost:8080/router
/0000000000000003/110
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add address [address_id=2]"
        }
      ]
    }
  ]
```

IP addresses to the routers have been set. Register as the default gateway on each host.

host: h1s1:

```
root@ryu-vm:~# ip route add default via 172.16.10.1
```

host: h2s1:

```
root@ryu-vm:~# ip route add default via 172.16.10.1
```

host: h1s2:

```
root@ryu-vm:~# ip route add default via 192.168.30.1
```

host: h2s2:

```
root@ryu-vm:~# ip route add default via 192.168.30.1
```

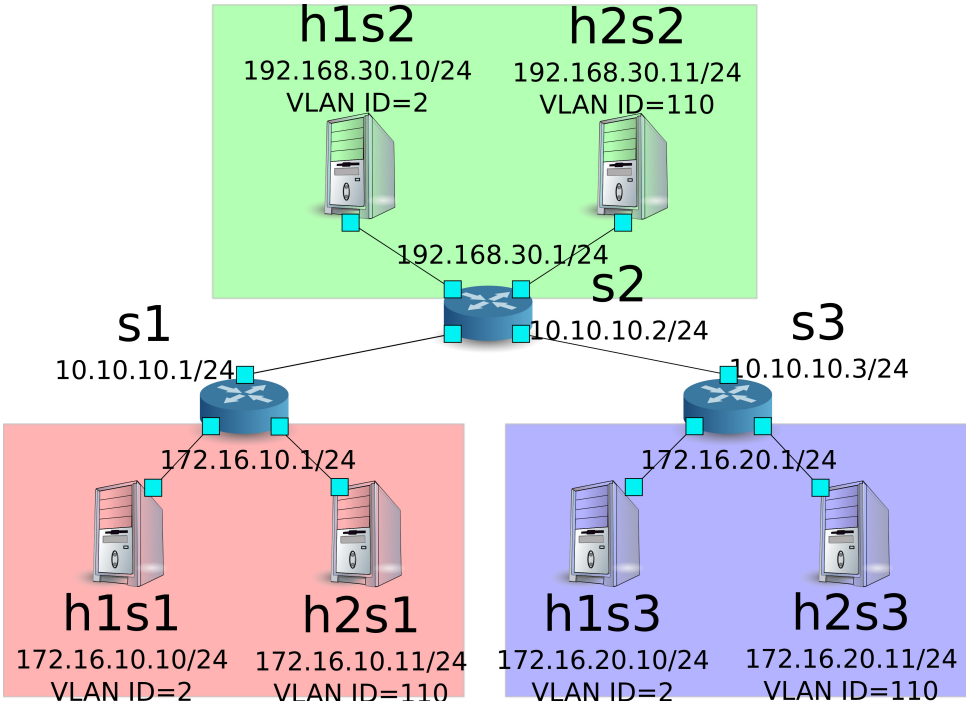
host: h1s3:

```
root@ryu-vm:~# ip route add default via 172.16.20.1
```

host: h2s3:

```
root@ryu-vm:~# ip route add default via 172.16.20.1
```

The addresses that have been set are as follows.



10.2.3 Setting Static Routes and the Default Route

Set static routes and the default route for each router.

First, set router s2 as the default route of router s1.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"gateway": "10.10.10.2"}' http://localhost:8080/router
/0000000000000001/2
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"gateway": "10.10.10.2"}' http://localhost:8080/router
/0000000000000001/110
  [
    {
      "switch_id": "0000000000000001",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]
```

Set router s1 as the default route of router s2.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"gateway": "10.10.10.1"}' http://localhost:8080/router
/0000000000000002/2
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"gateway": "10.10.10.1"}' http://localhost:8080/router
/0000000000000002/110
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]
```

Set router s2 as default route of router s3.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"gateway": "10.10.10.2"}' http://localhost:8080/router
/0000000000000003/2
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]

root@ryu-vm:~# curl -X POST -d '{"gateway": "10.10.10.2"}' http://localhost:8080/router
/0000000000000003/110
  [
    {
      "switch_id": "0000000000000003",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 110,
          "details": "Add route [route_id=1]"
        }
      ]
    }
  ]
```

Next, for s2 router, set a static route to the host (172.16.20.0/24) under router s3. Only set if vlan_id=2.

Node: c0 (root):

```
root@ryu-vm:~# curl -X POST -d '{"destination": "172.16.20.0/24", "gateway": "10.10.10.3"}'
http://localhost:8080/router/0000000000000002/2
  [
    {
      "switch_id": "0000000000000002",
      "command_result": [
        {
          "result": "success",
          "vlan_id": 2,
          "details": "Add route [route_id=2]"
        }
      ]
    }
  ]
```

10.2.4 Verifying the Settings

Check the contents of the settings for each router.

Node: c0 (root):

```
root@ryu-vm:~# curl http://localhost:8080/router/all/all
  [
    {
      "internal_network": [
        {},
        {
          "route": [
            {
              "route_id": 1,
              "destination": "0.0.0.0/0",
              "gateway": "10.10.10.2"
            }
```

```
        ],
        "vlan_id": 2,
        "address": [
          {
            "address_id": 2,
            "address": "10.10.10.1/24"
          },
          {
            "address_id": 1,
            "address": "172.16.10.1/24"
          }
        ]
      },
      {
        "route": [
          {
            "route_id": 1,
            "destination": "0.0.0.0/0",
            "gateway": "10.10.10.2"
          }
        ],
        "vlan_id": 110,
        "address": [
          {
            "address_id": 2,
            "address": "10.10.10.1/24"
          },
          {
            "address_id": 1,
            "address": "172.16.10.1/24"
          }
        ]
      }
    ],
    "switch_id": "0000000000000001"
  },
  {
    "internal_network": [
      {},
      {
        "route": [
          {
            "route_id": 2,
            "destination": "172.16.20.0/24",
            "gateway": "10.10.10.3"
          },
          {
            "route_id": 1,
            "destination": "0.0.0.0/0",
            "gateway": "10.10.10.1"
          }
        ],
        "vlan_id": 2,
        "address": [
          {
            "address_id": 2,
            "address": "10.10.10.2/24"
          },
          {
            "address_id": 1,
            "address": "192.168.30.1/24"
          }
        ]
      },
      {
        "route": [
          {
            "route_id": 1,
            "destination": "0.0.0.0/0",
            "gateway": "10.10.10.1"
          }
        ],
        "vlan_id": 110,
```

```
          "address": [
            {
              "address_id": 2,
              "address": "10.10.10.2/24"
            },
            {
              "address_id": 1,
              "address": "192.168.30.1/24"
            }
          ]
        }
      ],
      "switch_id": "0000000000000002"
    },
    {
      "internal_network": [
        {},
        {
          "route": [
            {
              "route_id": 1,
              "destination": "0.0.0.0/0",
              "gateway": "10.10.10.2"
            }
          ],
          "vlan_id": 2,
          "address": [
            {
              "address_id": 1,
              "address": "172.16.20.1/24"
            },
            {
              "address_id": 2,
              "address": "10.10.10.3/24"
            }
          ]
        },
        {
          "route": [
            {
              "route_id": 1,
              "destination": "0.0.0.0/0",
              "gateway": "10.10.10.2"
            }
          ],
          "vlan_id": 110,
          "address": [
            {
              "address_id": 1,
              "address": "172.16.20.1/24"
            },
            {
              "address_id": 2,
              "address": "10.10.10.3/24"
            }
          ]
        }
      ],
      "switch_id": "0000000000000003"
    }
  ]
```


A table of settings for each router is as follows.

Router	VLAN ID	IP address	Default route	Static route
s1	2	172.16.10.1/24, 10.10.10.1/24	10.10.10.2(s2)	
s1	110	172.16.10.1/24, 10.10.10.1/24	10.10.10.2(s2)	
s2	2	192.168.30.1/24, 10.10.10.2/24	10.10.10.1(s1)	Destination: 172.16.20.0/24, Gateway:10.10.10.3(s3)
s2	110	192.168.30.1/24, 10.10.10.2/24	10.10.10.1(s1)	
s3	2	172.16.20.1/24, 10.10.10.3/24	10.10.10.2(s2)	
s3	110	172.16.20.1/24, 10.10.10.3/24	10.10.10.2(s2)	

Send a ping from h1s1 to h1s3. Since they're the same host of vlan_id=2 and router 2 has a static route set to s3, it can communicate successfully.

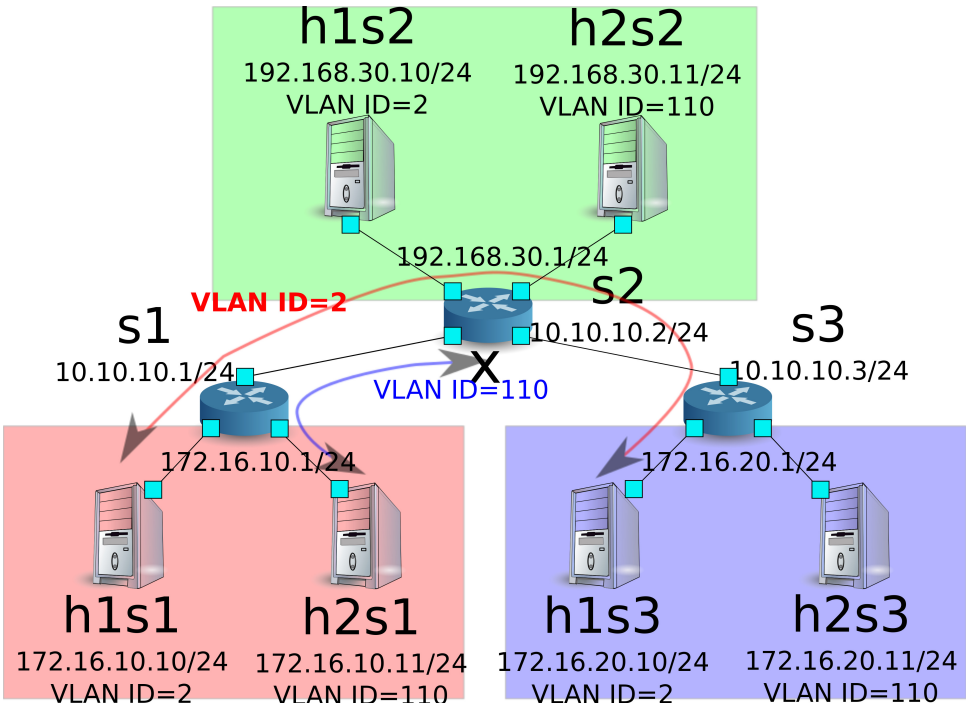
host: h1s1:

```
root@ryu-vm:~# ping 172.16.20.10
PING 172.16.20.10 (172.16.20.10) 56(84) bytes of data.
64 bytes from 172.16.20.10: icmp_req=1 ttl=61 time=45.9 ms
64 bytes from 172.16.20.10: icmp_req=2 ttl=61 time=0.257 ms
64 bytes from 172.16.20.10: icmp_req=3 ttl=61 time=0.059 ms
64 bytes from 172.16.20.10: icmp_req=4 ttl=61 time=0.182 ms
```

Send a ping from h2s1 to h2s3. They're the same host of vlan_id=2 but since router s2 doesn't have a static route set to s3, it cannot communicate successfully.

host: h2s1:

```
root@ryu-vm:~# ping 172.16.20.11
PING 172.16.20.11 (172.16.20.11) 56(84) bytes of data.
^C
--- 172.16.20.11 ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7009ms
```



In this section, you learned how to use routers with specific examples.

10.3 REST API List

A list of REST API of rest_router introduced in this section.

10.3.1 Acquiring the Setting

Method	GET
URL	/router/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Remarks	Specification of VLAN ID is optional.

10.3.2 Setting an Address

Method	POST
URL	/router/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Data	**address**:"<xxx.xxx.xxx.xxx/xx>"
Remarks	Perform address setting before performing route setting. Specification of VLAN ID is optional.

10.3.3 Setting Static Routes

Method	POST
URL	/router/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Data	**destination**:"<xxx.xxx.xxx.xxx/xx>" **gateway**:"<xxx.xxx.xxx.xxx>"
Remarks	Specification of VLAN ID is optional.

10.3.4 Setting Default Route

Method	POST
URL	/router/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Data	**gateway**:"<xxx.xxx.xxx.xxx>"
Remarks	Specification of VLAN ID is optional.

10.3.5 Deleting an Address

Method	DELETE
URL	/router/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Data	**address_id**:[1 - ...]
Remarks	Specification of VLAN ID is optional.

10.3.6 Deleting a Route

Method	DELETE
URL	/router/{**switch**}[/{**vlan**}] –**switch**: ["all" | *Switch ID*] –**vlan**: ["all" | *VLAN ID*]
Data	**route_id**:[1 - ...]
Remarks	Specification of VLAN ID is optional.

CHAPTER

ELEVEN

OPENFLOW SWITCH TEST TOOL

This section explains how to use the test tool to verify the degree of compliance of an OpenFlow switch with the OpenFlow specifications.

11.1 Overview of Test Tool

This tool is used to verify the status of support by OpenFlow switch for the OpenFlow specification by conducting a flow entry registration/packet application to the test subject OpenFlow switch according to a test pattern file and comparing the result of processing by the OpenFlow switch of packet rewriting and transfer (or discard) against the expected processing result described in the test pattern file.

This tool is compatible with FlowMod message test of OpenFlow version 1.3.

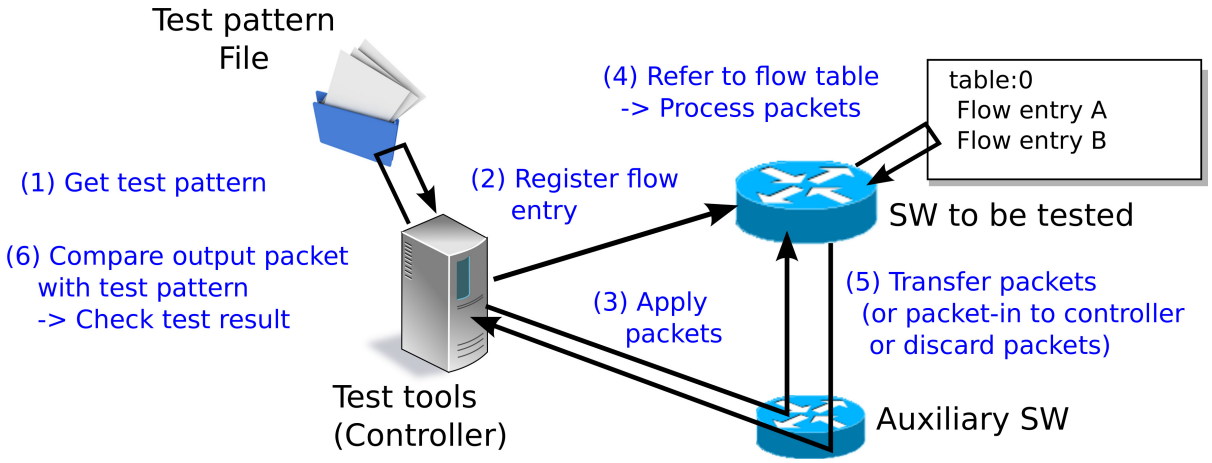
Test subject message	Corresponding parameters
OpenFlow1.3 FlowMod message	match (Excludes IN_PHY_PORT) actions (Excludes SET_QUEUE, GROUP)

"*Packet Library*" is used to confirm packet rewriting and results generation of packets to be applied.

11.1.1 Operation Overview

Test execution image

The following shows an image of operation when you run the test tool. "flow entry to be registered, "application packet" and "expected processing result to" are described in the test pattern file. How to set up your environment for the tool execution is described later (refer to Tool execution environment)



Output image of test results

The specified test items of the test pattern file are run in order and test results (OK / ERROR) are output. If a test result is ERROR, details of the error are also output.

```
--- Test start ---

match: 29_ICMPV6_TYPE
    ethernet/ipv6/icmpv6(type=128)-->'icmpv6_type=128,actions=output:2'               OK
    ethernet/ipv6/icmpv6(type=128)-->'icmpv6_type=128,actions=output:CONTROLLER'      OK
    ethernet/ipv6/icmpv6(type=135)-->'icmpv6_type=128,actions=output:2'               OK
    ethernet/vlan/ipv6/icmpv6(type=128)-->'icmpv6_type=128,actions=output:2'          ERROR
        Received incorrect packet-in: ethernet(ethertype=34525)
    ethernet/vlan/ipv6/icmpv6(type=128)-->'icmpv6_type=128,actions=output:CONTROLLER' ERROR
        Received incorrect packet-in: ethernet(ethertype=34525)
match: 30_ICMPV6_CODE
    ethernet/ipv6/icmpv6(code=0)-->'icmpv6_code=0,actions=output:2'                   OK
    ethernet/ipv6/icmpv6(code=0)-->'icmpv6_code=0,actions=output:CONTROLLER'          OK
    ethernet/ipv6/icmpv6(code=1)-->'icmpv6_code=0,actions=output:2'                   OK
    ethernet/vlan/ipv6/icmpv6(code=0)-->'icmpv6_code=0,actions=output:2'              ERROR
        Received incorrect packet-in: ethernet(ethertype=34525)
    ethernet/vlan/ipv6/icmpv6(code=0)-->'icmpv6_code=0,actions=output:CONTROLLER'     ERROR
        Received incorrect packet-in: ethernet(ethertype=34525)

---  Test end  ---
```

11.2 How to use

This section explains how to use the test tool.

11.2.1 Test Pattern File

You need to create a test pattern file in accordance with the test pattern that you want to test.

A test pattern file is a text file that has a ".json" extension. It is described using the following format.

```
[
    "xxxxxxxxxx",                     # Test item name
    {
        "description": "xxxxxxxxxx", # Description of the test content
        "prerequisite": [
            {
                "OFPFlowMod": {...}  # Flow entry to register
            },                       # (Describe OFPFlowMod of Ryu in json format)
            {...},                   #  In case of packet transfer (actions=output),
            {...}                    #  specify "2" as the output port number.
        ],
        "tests": [
            {
                "ingress": [         # Packet to be applied
                    "ethernet(...)", # (Describe in format of Ryu packet library constructor)
                    "ipv4(...)",
                    "tcp(...)"
                ],

                # Expected processing results
                # Depending on the type of processing results, describe either (a)(b)(c)
                #  (a) Confirmation test of packet transfer (actions=output:X)
                "egress": [          # Expected transfer packet
                    "ethernet(...)",
                    "ipv4(...)",
                    "tcp(...)"
                ]
                #  (b) Confirmation test of Packet-In (actions=CONTROLLER)
                "PACKET_IN": [       # Expected Packet-In data
                    "ethernet(...)",
                    "ipv4(...)",
```

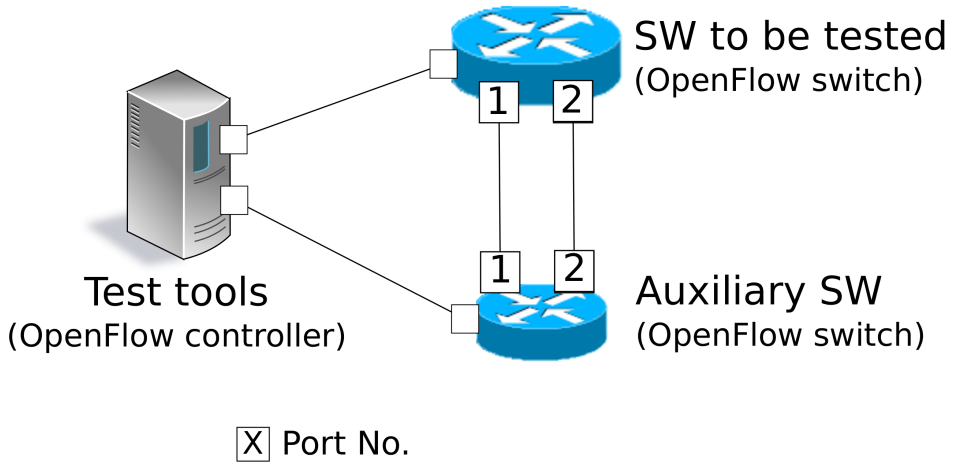
```
                    "tcp(...)"
                ]
                #   (c) Confirmation test of table-miss
                "table-miss": [      # flow table ID that is expected to be table-miss
                    0
                ]
            },
            {...},
            {...}
        ]
    },                                  # Test 1
    {...},                              # Test 2
    {...}                               # Test 3
]
```

Note: As a sample test pattern, the source tree of Ryu offers a test pattern file to check if each parameter that can be specified in the match/actions of OpenFlow1.3 FlowMod message works properly or not.

ryu/tests/switch/of13

11.2.2 Tool Execution Environment

The environment for test execution tools is described below.



As an auxiliary switch, an OpenFlow switch that can be used to perform following the operation successfully is required.

- Flow entry registration of actions=CONTROLLER
- Packet-In message transmission by flow entry of actions=CONTROLLER
- Packet transmission by Packet-Out message reception

Note: The source tree of Ryu offers an environment build script that allows realization of a tool execution environment on mininet that uses Open vSwitch as a test target switch.

ryu/tests/switch/run_mininet.py

A example script is described in "Test tool usage example".

11.2.3 How To Run The Test Tool

The test tool is available on the source tree on Ryu.

Source code	Explanation
ryu/tests/switch/tester.py	Test tool
ryu/tests/switch/of13	Sample of test pattern file
ryu/tests/switch/run_mininet.py	Test environment build script

The test tool is executed by the following command.

```
$ ryu-manager [--test-switch-target DPID] [--test-switch-tester DPID]
 [--test-switch-dir DIRECTORY] ryu/tests/switch/tester.py
```

Option	Explanation	Default value
–test-switch-target	Data path ID of test target switch	0000000000000001
–test-switch-tester	Data path ID of auxiliary switch	0000000000000002
–test-switch-dir	Directory path of test pattern file	ryu/tests/switch/of13

Note: Since the test tool is created as a Ryu application and inherits ryu.base.app_manager.RyuApp, it supports output of debugging information by the –verbose option, as with other Ryu applications.

After starting the test tool, when the auxiliary switch and test target switch are connected to the controller, the test starts based on the test pattern file that you specify.

11.3 Test Tool Usage Example

The following is the procedure to execute the test tool using a sample test pattern and original test pattern file.

11.3.1 Procedure for Executing Sample Test Pattern

The following shows the procedure of using sample test pattern (ryu/tests/switch/of13) of the source tree of the Ryu to check the through operation of match/actions of FlowMod messages.

In this procedure, the test environment is constructed using the test environment build script (ryu / tests / switch / run_mininet.py). Please refer to "*Switching Hub* " for environment settings and the login method for usage of the VM image.

1 Building the test environment

Log in to the VM environment and run the test environment build script.

```
ryu@ryu-vm:~$ sudo ryu/ryu/tests/switch/run_mininet.py
```

Execution result of the net command is as follows.

```
mininet> net
c0
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s2-eth2
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s1-eth2
```

2 Execution of the test tool

For execution of test tool, open xterm of controller.

```
mininet> xterm c0
```

Execute test tool from xterm of "Node: c0 (root)" At this time, as the directory for the test pattern file, specify the directory of the sample test pattern (ryu/tests/switch/of13). Since the data path ID of the test target switch and auxiliary switch in the mininet environment has the default value of each option of –test-switch-target / –test-switch-tester, it the option specification is omitted.

Node: c0:

```
root@ryu-vm:~$ ryu-manager --test-switch-dir ryu/ryu/tests/switch/of13 ryu/ryu/
tests/switch/tester.py
```

When the tool is executed it appears as follows and waits until the test target switch and auxiliary switch is connected to the controller.

```
root@ryu-vm:~$ ryu-manager --test-switch-dir ryu/ryu/tests/switch/of13/ ryu/ryu/
tests/switch/tester.py
loading app ryu/ryu/tests/switch/tester.py
loading app ryu.controller.ofp_handler
instantiating app ryu/ryu/tests/switch/tester.py of OfTester
target_dpid=0000000000000001
tester_dpid=0000000000000002
Test files directory = ryu/ryu/tests/switch/of13/
instantiating app ryu.controller.ofp_handler of OFPHandler
--- Test start ---
waiting for switches connection...
```

When the test target switch and auxiliary switch is connected to the controller, the test begins.

```
root@ryu-vm:~$ ryu-manager --test-switch-dir ryu/ryu/tests/switch/of13/ ryu/ryu/
tests/switch/tester.py
loading app ryu/ryu/tests/switch/tester.py
loading app ryu.controller.ofp_handler
instantiating app ryu/ryu/tests/switch/tester.py of OfTester
target_dpid=0000000000000001
tester_dpid=0000000000000002
Test files directory = ryu/ryu/tests/switch/of13/
instantiating app ryu.controller.ofp_handler of OFPHandler
--- Test start ---
waiting for switches connection...
dpid=0000000000000002 : Join tester SW.
dpid=0000000000000001 : Join target SW.
action: 00_OUTPUT
    ethernet/ipv4/tcp-->'actions=output:2'      OK
    ethernet/ipv6/tcp-->'actions=output:2'      OK
    ethernet/arp-->'actions=output:2'           OK
action: 11_COPY_TTL_OUT
    ethernet/mpls(ttl=64)/ipv4(ttl=32)/tcp-->'eth_type=0x8847,actions=copy_ttl_out,
output:2'        ERROR
        Failed to add flows: OFPErrorMsg[type=0x02, code=0x00]
    ethernet/mpls(ttl=64)/ipv6(hop_limit=32)/tcp-->'eth_type=0x8847,actions=
copy_ttl_out,output:2'  ERROR
        Failed to add flows: OFPErrorMsg[type=0x02, code=0x00]
...
```

When all testing of the sample test pattern file under ryu/tests/switch/of13 is complete, the test tool ends.

<Reference>

Sample test pattern file list

> Offers a test pattern that registers flow entries corresponding to each setting in the match/actions and applies multiple patterns of packets that match (or do not match) flow entries.

```
ryu/tests/switch/of13/action:
00_OUTPUT.json              20_POP_MPLS.json
11_COPY_TTL_OUT.json        23_SET_NW_TTL_IPv4.json
12_COPY_TTL_IN.json         23_SET_NW_TTL_IPv6.json
15_SET_MPLS_TTL.json        24_DEC_NW_TTL_IPv4.json
16_DEC_MPLS_TTL.json        24_DEC_NW_TTL_IPv6.json
17_PUSH_VLAN.json           25_SET_FIELD
17_PUSH_VLAN_multiple.json  26_PUSH_PBB.json
18_POP_VLAN.json            26_PUSH_PBB_multiple.json
19_PUSH_MPLS.json           27_POP_PBB.json
19_PUSH_MPLS_multiple.json

ryu/tests/switch/of13/action/25_SET_FIELD:
03_ETH_DST.json        14_TCP_DST_IPv4.json   24_ARP_SHA.json
04_ETH_SRC.json        14_TCP_DST_IPv6.json   25_ARP_THA.json
05_ETH_TYPE.json       15_UDP_SRC_IPv4.json   26_IPV6_SRC.json
```

```
06_VLAN_VID.json        15_UDP_SRC_IPv6.json    27_IPV6_DST.json
07_VLAN_PCP.json        16_UDP_DST_IPv4.json    28_IPV6_FLABEL.json
08_IP_DSCP_IPv4.json    16_UDP_DST_IPv6.json    29_ICMPV6_TYPE.json
08_IP_DSCP_IPv6.json    17_SCTP_SRC_IPv4.json   30_ICMPV6_CODE.json
09_IP_ECN_IPv4.json     17_SCTP_SRC_IPv6.json   31_IPV6_ND_TARGET.json
09_IP_ECN_IPv6.json     18_SCTP_DST_IPv4.json   32_IPV6_ND_SLL.json
10_IP_PROTO_IPv4.json   18_SCTP_DST_IPv6.json   33_IPV6_ND_TLL.json
10_IP_PROTO_IPv6.json   19_ICMPV4_TYPE.json     34_MPLS_LABEL.json
11_IPV4_SRC.json        20_ICMPV4_CODE.json     35_MPLS_TC.json
12_IPV4_DST.json        21_ARP_OP.json          36_MPLS_BOS.json
13_TCP_SRC_IPv4.json    22_ARP_SPA.json         37_PBB_ISID.json
13_TCP_SRC_IPv6.json    23_ARP_TPA.json         38_TUNNEL_ID.json

ryu/tests/switch/of13/match:
00_IN_PORT.json         13_TCP_SRC_IPv4.json    25_ARP_THA.json
02_METADATA.json        13_TCP_SRC_IPv6.json    25_ARP_THA_Mask.json
02_METADATA_Mask.json   14_TCP_DST_IPv4.json    26_IPV6_SRC.json
03_ETH_DST.json         14_TCP_DST_IPv6.json    26_IPV6_SRC_Mask.json
03_ETH_DST_Mask.json    15_UDP_SRC_IPv4.json    27_IPV6_DST.json
04_ETH_SRC.json         15_UDP_SRC_IPv6.json    27_IPV6_DST_Mask.json
04_ETH_SRC_Mask.json    16_UDP_DST_IPv4.json    28_IPV6_FLABEL.json
05_ETH_TYPE.json        16_UDP_DST_IPv6.json    29_ICMPV6_TYPE.json
06_VLAN_VID.json        17_SCTP_SRC_IPv4.json   30_ICMPV6_CODE.json
06_VLAN_VID_Mask.json   17_SCTP_SRC_IPv6.json   31_IPV6_ND_TARGET.json
07_VLAN_PCP.json        18_SCTP_DST_IPv4.json   32_IPV6_ND_SLL.json
08_IP_DSCP_IPv4.json    18_SCTP_DST_IPv6.json   33_IPV6_ND_TLL.json
08_IP_DSCP_IPv6.json    19_ICMPV4_TYPE.json     34_MPLS_LABEL.json
09_IP_ECN_IPv4.json     20_ICMPV4_CODE.json     35_MPLS_TC.json
09_IP_ECN_IPv6.json     21_ARP_OP.json          36_MPLS_BOS.json
10_IP_PROTO_IPv4.json   22_ARP_SPA.json         37_PBB_ISID.json
10_IP_PROTO_IPv6.json   22_ARP_SPA_Mask.json    37_PBB_ISID_Mask.json
11_IPV4_SRC.json        23_ARP_TPA.json         38_TUNNEL_ID.json
11_IPV4_SRC_Mask.json   23_ARP_TPA_Mask.json    38_TUNNEL_ID_Mask.json
12_IPV4_DST.json        24_ARP_SHA.json         39_IPV6_EXTHDR.json
12_IPV4_DST_Mask.json   24_ARP_SHA_Mask.json    39_IPV6_EXTHDR_Mask.json
```

11.3.2 Procedure for Executing Original Test Pattern

The following is the procedure to run the test tool by creating an original test pattern.

The following is an example of creating a test tool that checks if it has a function to process the match/actions required for OpenFlow switch to implement the router function.

1 Creating the test pattern file

It will test the following flow entry, which has a function for the router to forward packets according to the routing table, and check if it is working correctly.

match	actions
Destination IP address range “192.168.30.0/24”	Rewrite the source MAC address to “aa:aa:aa:aa:aa:aa”. Rewrite the destination MAC address to “bb:bb:bb:bb:bb:bb” TTL decrement Packet forwarding

Create a test pattern file to perform this test pattern.

File name: `sample_test_pattern.json`

```
[
   "sample: Router test",
   {
       "description": "static routing table",
       "prerequisite": [
           {
               "OFPFlowMod": {
                   "table_id": 0,
                   "match": {
                       "OFPMatch": {
                           "oxm_fields": [
                               {
```

```
                                    "OXMTlv": {
                                        "field": "eth_type",
                                        "value": 2048
                                    }
                                },
                                {
                                    "OXMTlv": {
                                        "field": "ipv4_dst",
                                        "mask": 4294967040,
                                        "value": "192.168.30.0"
                                    }
                                }
                            ]
                        }
                    },
                    "instructions":[
                        {
                            "OFPInstructionActions": {
                                "actions":[
                                    {
                                        "OFPActionSetField":{
                                            "field":{
                                                "OXMTlv":{
                                                    "field":"eth_src",
                                                    "value":"aa:aa:aa:aa:aa:aa"
                                                }
                                            }
                                        }
                                    },
                                    {
                                        "OFPActionSetField":{
                                            "field":{
                                                "OXMTlv":{
                                                    "field":"eth_dst",
                                                    "value":"bb:bb:bb:bb:bb:bb"
                                                }
                                            }
                                        }
                                    },
                                    {
                                        "OFPActionDecNwTtl":{}
                                    },
                                    {
                                        "OFPActionOutput": {
                                            "port":2
                                        }
                                    }
                                ],
                                "type": 4
                            }
                        }
                    ]
                }
            }
        ],
        "tests":[
            {
                "ingress":[
                    "ethernet(dst='22:22:22:22:22:22',src='11:11:11:11:11:11',ethertype=2048)",
                    "ipv4(tos=32, proto=6, src='192.168.10.10', dst='192.168.30.10', ttl=64)",
                    "tcp(dst_port=2222, option='\\x00\\x00\\x00\\x00', src_port=11111)",
                    "'\\x01\\x02\\x03\\x04\\x05\\x06\\x07\\x08\\t\\n\\x0b\\x0c\\r\\x0e\\x0f'"
                ],
                "egress":[
                    "ethernet(dst='bb:bb:bb:bb:bb:bb',src='aa:aa:aa:aa:aa:aa',ethertype=2048)",
                    "ipv4(tos=32, proto=6, src='192.168.10.10', dst='192.168.30.10', ttl=63)",
                    "tcp(dst_port=2222, option='\\x00\\x00\\x00\\x00', src_port=11111)",
                    "'\\x01\\x02\\x03\\x04\\x05\\x06\\x07\\x08\\t\\n\\x0b\\x0c\\r\\x0e\\x0f'"
                ]
            }
        ]
    }
]
```

2 Building a test environment

Build a test environment using a test environment build script. Please refer to the execution procedure in Procedure for Executing Sample Test Pattern.

3 Executing the test tool

Execute the test tool from Xterm from the controller by specifying the original test pattern you just created. For –test-switch-dir option, you can also directly specify a file as well as a directory. In order to confirm the contents of packets sent and received, the –verbose option is also specified.

Node: c0:

```
root@ryu-vm:~$ ryu-manager --verbose --test-switch-dir ./sample_test_pattern.json
ryu/ryu/tests/switch/tester.py
```

When the test target switch and auxiliary switch is connected to the controller, the test begins.

In log output of "dpid=0000000000000002 : receive_packet...", you can see that the expected output packet set in egress packed of the test pattern file was sent. Note that only logs the test tool outputs are excerpted.

```
root@ryu-vm:~$ ryu-manager --verbose --test-switch-dir ./sample_test_pattern.json
ryu/ryu/tests/switch/tester.py
loading app ryu/tests/switch/tester.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu/tests/switch/tester.py of OfTester
target_dpid=0000000000000001
tester_dpid=0000000000000002
Test files directory = ./sample_test_pattern.json

--- Test start ---
waiting for switches connection...

dpid=0000000000000002 : Join tester SW.
dpid=0000000000000001 : Join target SW.

sample: Router test

send_packet:[ethernet(dst='22:22:22:22:22:22',ethertype=2048,src
='11:11:11:11:11:11'), ipv4(csum=53560,dst='192.168.30.10',flags=0,header_length=5,
identification=0,offset=0,option=None,proto=6,src='192.168.10.10',tos=32,
total_length=59,ttl=64,version=4), tcp(ack=0,bits=0,csum=33311,dst_port=2222,offset
=6,option='\x00\x00\x00\x00',seq=0,src_port=11111,urgent=0,window_size=0), '\x01\
x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f']
egress:[ethernet(dst='bb:bb:bb:bb:bb:bb',ethertype=2048,src='aa:aa:aa:aa:aa:aa'),
ipv4(csum=53816,dst='192.168.30.10',flags=0,header_length=5,identification=0,offset
=0,option=None,proto=6,src='192.168.10.10',tos=32,total_length=59,ttl=63,version=4)
, tcp(ack=0,bits=0,csum=33311,dst_port=2222,offset=6,option='\x00\x00\x00\x00',seq
=0,src_port=11111,urgent=0,window_size=0), '\x01\x02\x03\x04\x05\x06\x07\x08\t\n\
x0b\x0c\r\x0e\x0f']
packet_in:[]
dpid=0000000000000002 : receive_packet[ethernet(dst='bb:bb:bb:bb:bb:bb',ethertype
=2048,src='aa:aa:aa:aa:aa:aa'), ipv4(csum=53816,dst='192.168.30.10',flags=0,
header_length=5,identification=0,offset=0,option=None,proto=6,src='192.168.10.10',
tos=32,total_length=59,ttl=63,version=4), tcp(ack=0,bits=0,csum=33311,dst_port
=2222,offset=6,option='\x00\x00\x00\x00',seq=0,src_port=11111,urgent=0,window_size
=0), '\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f']
    static routing table                              OK
---  Test end  ---
```

Actual flow entries registered in the OpenFlow switch are shown below. You can see that packets applied by the test tool match the flow entry and n_packets has been incremented.

Node: s1:

```
root@ryu-vm:~# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=56.217s, table=0, n_packets=1, n_bytes=73, priority=0,ip,
nw_dst=192.168.30.0/24 actions=set_field:aa:aa:aa:aa:aa:aa->eth_src,set_field:bb:bb
:bb:bb:bb:bb->eth_dst,dec_ttl,output:2
```

11.3.3 List of Error Messages

The following is a list of error messages that can be output with this tool.

Error message	Description
Failed to initialize flow tables: barrier request timeout.	Failed to delete the flow entry of the previous test (time-out of Barrier Request)
Failed to initialize flow tables: [err_msg]	Failed to delete the flow entry of the previous test (error message received for FlowMod)
Failed to add flows: barrier request timeout.	Failed to register the flow entry (time-out of Barrier Request)
Failed to add flows: [err_msg]	Failed to register flow entry (error message is received for FlowMod)
Added incorrect flows: [flows]	Flow entry registration confirmation error (unexpected flow entry is registered)
Failed to add flows: flow stats request timeout.	Flow entry registration confirmation failure (time-out of FlowStats Request)
Failed to add flows: [err_msg]	Flow entry registration confirmation failure (error message received for FlowStats Request)
Failed to request port stats from target: request timeout.	Failed to acquire PortStats of the tested SW (time-out of PortStats Request)
Failed to request port stats from target: [err_msg]	Failed to acquire PortStats of the tested SW (error message received for PortStats Request)
Failed to request port stats from tester: request timeout.	Failed to acquire PortStats of Auxiliary SW (time-out of PortStats Request)
Failed to request port stats from tester: [err_msg]	Failed to acquire PortStats of Auxiliary SW (error message received for PortStats Request)
Received incorrect [packet]	Reception error of output expected packets (received different packets)
Receiving timeout: [detail]	Reception error of expected output packets (time-out)
Faild to send packet: barrier request timeout.	Failed to apply packet (time-out of Barrier Request)
Faild to send packet: [err_msg]	Failed to apply packet (error message received for Packet-Out)
Table-miss error: increment in matched_count.	table-miss check error (matches the flow)
Table-miss error: no change in lookup_count.	table-miss check error (packet has not been processed by the flow table being checked)
Failed to request table stats: request timeout.	Failed to check table-miss (time-out of TableStats Request)
Failed to request table stats: [err_msg]	Failed to check table-miss (error message received for TableStats Request)

CHAPTER

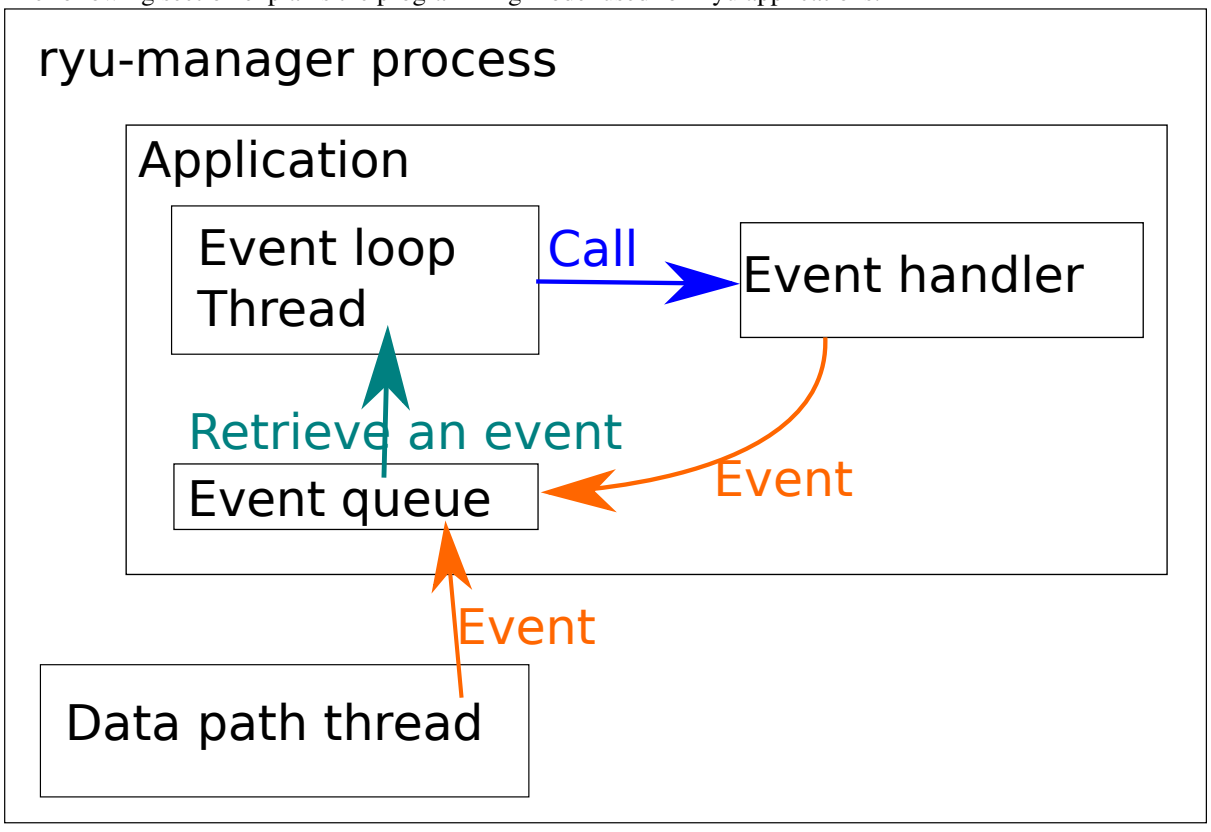
TWELVE

ARCHITECTURE

This section introduces the Ryu architecture. Refer to the API reference <http://ryu.readthedocs.org/en/latest/> for how to use each class.

12.1 Application Programming Model

The following section explains the programming model used for Ryu applications.



12.1.1 Applications

Applications are a class that inherits `ryu.base.app_manager.RyuApp`. User logic is described as an application.

12.1.2 Event

Events are class objects that inherit `ryu.controller.event.EventBase`. Communication between applications is performed by transmitting and receiving events.

12.1.3 Event Queue

Each application has a single queue for receiving events.

12.1.4 Threads

Ryu runs in multi-thread using eventlets. Because threads are non-preemptive, you need to be careful when performing time-consuming processes.

Event loops

One thread is automatically created for each application. This thread runs an event loop. If there is an event in the event queue, the event loop will load the event and call the corresponding event handler (described later).

Additional threads

You can create additional threads using the hub.spawn function to perform application-specific processing.

eventlets

It can also be used directly from the application function of eventlet, but it's not recommended. Please be sure to use the wrapper provided by the hub module if possible.

12.1.5 Event handlers

You can define an event handler by decorating application class method with an `ryu.controller.handler.set_ev_cls` decorator. When an event of the specified type occurs, the event handler is called from the application's event loop.

CHAPTER

THIRTEEN

CONTRIBUTION

One of the appeals of open source software is that you can participate in the development process yourself. This section introduces how to participate in the development of Ryu.

13.1 Development structure

Development of Ryu has been conducted around a mailing list. Let's begin by to joining the mailing list.

https://lists.sourceforge.net/lists/listinfo/ryu-devel

Information exchange on the mailing list is primarily done in English. When you have questions such as how to use, or if you encounter behavior that seems like a bug, do not hesitate to send mail. Because using open source software itself is an important contribution to the project.

13.2 Development Environment

This section describes the necessary environment and points to consider during Ryu development.

13.2.1 Python

Ryu supports Python 2.6 and later. Therefore, do not use syntax only available in Python 2.7.

Python 3.0 or later are not supported at the moment. However, it's best to keep in mind to write source code that avoids future changes as much as possible.

13.2.2 Coding Style

Ryu source code is in compliance with the PEP8 coding style. When sending a patch, which will be described later, please make sure in advance that the content is in compliance with PEP8.

http://www.python.org/dev/peps/pep-0008/

To check whether source code is compliant with PEP8, a checker is available along with the script introduced in the test section.

https://pypi.python.org/pypi/pep8

13.2.3 Test

Ryu has some automated testing, but the simplest and most frequently used one is a unit test that is completed only by Ryu. When sending a patch, which will be described later, please make sure in advance that the execution of unit tests do not fail due to changes made. As for newly added source code, it is desirable to describe unit tests as much as possible.

```
$ cd ryu/
$ ./run_tests.sh
```

13.3 Sending a Patch

When you want to change the source code repository due to adding features or bug fixes, create a patch of the changed contents and sent it to the mailing list. It is desirable to discuss major changes on the mailing list in advance.

Note: A repository of Ryu source code exists on GitHub, but please note that this is not a development process using a pull request.

For the format of a patch you're going to send, the style used in the development of the Linux kernel is expected. In this section we will show you an example of sending a patch of the same style to the mailing list, but please refer to related documents for more information.

http://lxr.linux.no/linux/Documentation/SubmittingPatches

The following is the procedure.

1 Check out the source code

First, check out the Ryu source code. You may also create a working repository for yourself by forking the source code on GitHub, but the example uses the exact original for the sake of simplicity.

```
$ git clone https://github.com/osrg/ryu.git $ cd ryu/
```

2 Make changes to the source code

Make the necessary changes to the Ryu source code. Let's commit the changes at the break of work.

```
$ git commit -a
```

3 Creating a patch

Create a patch of the difference between the changes. Please do not forget to include a Signed-off-by: line in the patch. This signature will be the declaration that for the patch you submitted there are problems with the open-source software license.

```
$ git format-patch origin -s
```

4 Sending the patch

After confirming that the content of the completed patch is correct, send it to the mailing list. You can send directly by a mailer, but you can also handle interactively by using git-send-email(1).

```
$ git send-email 0001-sample.patch
```

5 Wait for a response

Wait for a response to the patch. It may be taken as it is, but if issues are pointed out you'll need to correct the contents and send it again.

CHAPTER

FOURTEEN

INTRODUCTION EXAMPLE

This section shows examples of services and products that use Ryu.

14.1 Stratosphere SDN Platform (Stratosphere)

The Stratosphere SDN Platform (hereinafter abbreviated as SSP) is a software product developed by Stratosphere. Using SSP, you can construct a virtual network with an Edge Overlay-model using tunneling protocols such as VXLAN, STT, and MPLS.

Each tunneling protocol is converted to and from VLAN. Since the identifier of each tunneled protocol is larger than the 12 bits of VLAN, many more L2 segments can be managed than directly using VLAN. Also SSP can be used in conjunction with software such as OpenStack, CloudStack and IaaS.

SSP uses OpenFlow to implement functions and is adopting Ryu as the controller in version 1.1.4. One of the reasons for this is to support OpenFlow1.1 and later. Upon supporting MPLS to SSP, introduction of a framework that supports OpenFlow1.1 is being considered since it has support at the protocol level.

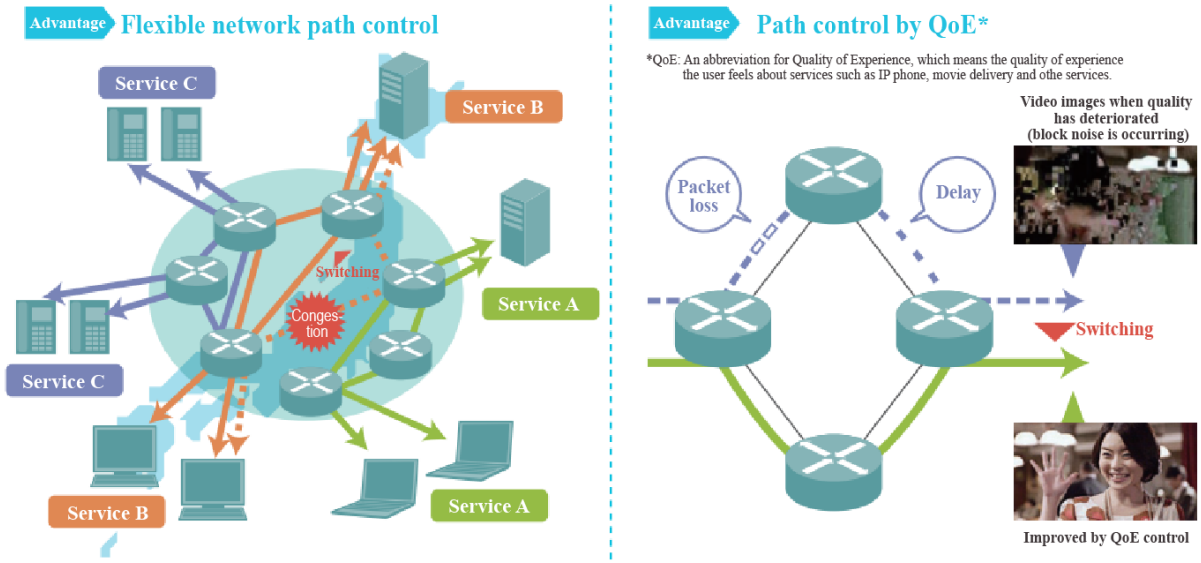
Note: Apart from support for the OpenFlow protocol itself, for items for which implementation is optional, it is also necessary to consider sufficient support of the OpenFlow switch side being used.

The fact that Python can be used as a development language is also a factor. Python is actively used in the development of Stratosphere, and many parts of SSP are written in Python as well. The outstanding descriptive power of Python and the fact that work can be performed using a familiar language results in improved development efficiency.

Software consists of multiple Ryu applications and interacts with other components of SSP through the REST API. The ability to divide software into multiple applications at the functional level is essential to maintaining good source code.

14.2 SmartSDN Controller (NTT COMWARE)

SmartSDN Controller is an SDN controller that provides centralized control functions of the network (network virtualization, optimization, etc.) to replace conventional autonomous distributed control.



SmartSDN Controller has the following two characteristics:

1. Flexible network routing by virtual networks

   By building multiple virtual networks on the same physical network, a flexible environment is provided to the network for requests from users, enabling reduced equipment cost through effective utilization of facilities. Also, by centrally managing the switches and routers in which information is individually referred and set, the entire network can be understood, allowing flexible route changes depending on the traffic situation and network failures.

   It focuses on Quality of Experience (QoE) of the user, and by determining QoE of network communication that is flowing (such as bandwidth, delay, loss, and fluctuation) and bypassing to a better path, stable maintenance of service quality is achieved.

2. Ensure network reliability with a high degree of maintenance and operation functionality

   It has a redundant configuration in order to continue service even in the event of controller failure. Also, by creating artificial communication packets that flow between sites and sending them on the path, early detection of failure on the path is provided, which cannot be detected by standard monitoring functions specified by the OpenFlow specification, allowing various tests (communication confirmation, route confirmation, etc.) to be performed.

   Furthermore, network design and network state confirmation is visualized using a GUI, allowing operation that does not depend on the skill level of maintenance personnel, which can reduce network operating costs.

In the development of SmartSDN Controller, it was necessary to select an OpenFlow framework that meets the following conditions.

- Framework that can comprehensively support the OpenFlow specification.
- Framework that allows updates relatively quickly because it is planning to follow updates to OpenFlow.

Within the above, Ryu had the following characteristics.

- Comprehensive support for functions in each version of OpenFlow.
- Quick compliance for updating of OpenFlow. The development community is also active and responds quickly to bugs.
- Substantial amounts of sample code and documentation.

Therefore, Ryu was deemed appropriate as a framework and has been adopted.