Software Defined Network
SDN

ch 5

Ryo Controller Programming

Ryu Application Framework

1. Introduction

Ryu applications are just Python scripts so you can save the file as (.py) on any location.

Components

ryu-manager :

- main executable

ryu.base.app_manager:

- The central management of Ryu applications
- Load Ryu applications
- Provide contexts to Ryu applications

ryu.ofproto:

- OpenFlow wire protocol encoder and decoder:

Ryu Application Framework

ryu.controller:

- openflow controller implementation
- generates openflow events

ryu.packet:

- all packet parsing libraries

Events:

RYU Application works based on Events. RYU Controller emits the events for the Openflow Messages received. This can be handled by the RYU Applicaltions.

Example Events:

ofp_event.EventOFPSwitchFeatures ofp_event.EventOFPPacketIn ofp_event.EventOFPFlowStatsReply

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html

Simple Switch application is a base application to refer. https://github.com/osrg/ryu/blob/master/ryu/app/simple_switch_13.py

Ryu Application Framework

2. Closer look on Simple_Switch_13 application

Lets have quick Recap on the OpenFlow Message transcations between Controller and Switch.

1. Hello Message
2. Feature Request/Response Message
3. Flow Modification message to install Table Miss Entry
4. Packet In Message
5. Packet Out Message
6. Flow Modification Message to Install the Flows

Ryu Application Framework

we can classify the program in few important parts.

1.Import the base classes / library

```
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 ether_types
```

2. Application Class (derived from app_manager)

```
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 = {}
```

Ryu Application Framework

3. Important openflow events handled

Features_Response_Received

PacketIn_Message_Received

we can handle this event, in our application as below,

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

```
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # If you hit this you might want to increase
        # the "miss_send_length" of your switch
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes",
                              ev.msg.msg_len, ev.msg.total_len)

        .....skipped......
```

In the switch_features_handler event, we add the TABLE MISS Entry . So we get the packet in messages.

In the packet_in_handler event, we process the PACKETs and parse the src_mac and dst_mac address and build the switching logic and install the flow.

Ryu Application Framework

Inserting(Adding) a flow

Flow table consists of Match, Actions and Counters.

A. Create a Match

Reference:

https://github.com/osrg/ryu/blob/master/ryu/ofproto/ofproto_v1_2_parser.py#L3403

This match is with no match fields. it means matching all the packets.

```
    match = parser.OFPMatch()
```

This matches with in_port, eth_dst and eth_src field.

```
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
```

Ryu Application Framework

B. Create a Actions

https://github.com/osrg/ryu/blob/master/ryu/ofproto/ofproto_v1_2_parser.py#L1252

Below action is send it to CONTROLLER(Reserved port).

```
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)
```

Below action , send it to port number 1.

```
out_port = 1
actions = [parser.OFPActionOutput(out_port)]
```

C. Create a Instruction list with Actions

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html?#ryu.ofproto.ofproto_v1_3_parser.OFPInstructionActions

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

Ryu Application Framework

D. Send OFP Flow Modifiation meassage for creating a new flow

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html?#ryu.ofproto.ofproto_v1_3_parser.OFPFlowMod

Most of the parameter are default. table_id, timeouts, cookies etc.

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

In Below example, explicitly specify tableid, timeouts.

```
mod = parser.OFPFlowMod(datapath=datapath, table_id=10,
                        buffer_id=buffer_id,idle_timeout=10,
                        hard_timeout=30, priority=priority,
                        match=match, instructions=inst)
```

Writing Basic Switching Applications

1. Introduction

we use simple_switch_13 application as base application, and will build hub, L3, L4 Match applications

2. HUB Application

Logic:

Create a Flow, all Matches with FLOOD Action

No need of TABLE MISS Entry.

So, we can simply modify the TABLE MISS Entry action as FLOOD to achieve the HUB Operations.

Code changes

In the Switch Features handler, modify the action as FLOOD.

```
actions = [parser.OFPActionOutput(port=ofproto.OFPP_FLOOD)]
```

We will never get Packet_in event , So we can remove those routines.

Save this file as hub.py

Writing Basic Switching Applications

Demo

1. Run Mininet topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4*

2. Run RYU hub application

```
ryu-manager hub.py
```

3. Check the openvswitch flows

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

4. do pingall from mininet.

Writing Basic Switching Applications

3. Switch Application With L3 Match

Logic

- simple_switch_13 application as a base application and switch learning process is same
- But Flow will be based on Layer3 Match (src ip and destination ip) instead of src_mac and dst_mac

Code changes

1. include the IP library

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

Writing Basic Switching Applications

2. Populate the Match based on IP.

- Check the packet is IP Packet, then decode the srcip and dstip from the packet header
- Populate the Match based on srcip and dstip.

```
    # check IP Protocol and create a match for IP
    if eth.ethertype == ether_types.ETH_TYPE_IP:
        ip = pkt.get_protocol(ipv4.ipv4)
        srcip = ip.src
        dstip = ip.dst
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
                                ipv4_src=srcip,
                                ipv4_dst=dstip
                                )
```

Writing Basic Switching Applications

Demo

1. Run Mininet topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4*

2. Run RYU hub application

```
ryu-manager l3_switch.py
```

3. do pingall from mininet.
4. Check the openvswitch flows

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

Writing Basic Switching Applications

4. Switch Application With L4 Match

Logic

- simple_switch_13 application as a base application and switch learning process is same
- But Flow will be based on Layer4 Match (src ip and destination ip, protocol and port number) instead of src_mac and dst_mac

- In the ethernet packet, check the ether frame type field is IP.
- if it is IP, load the IP Packet from the packet, extract the srcip , and dst ip from IP Packet.
- Check the IP Protocol(ICMP or TCP or UDP)
- If it is ICMP Prepare the openflow match with IP Src, IP dst and Protocol.
- If it is TCP, extract the tcp src port and tcp dst port fied Prepare the openflow match with IP Src, IP dst and Protocol, TCP Src Port and TCP dst port..
- If it is UDP Extract the udp src port and tcp dst port fied Prepare the openflow match with IP src, IP dst , protcol, udp src and udp dst port.

Writing Basic Switching Applications

1. include the header

```
from ryu.lib.packet import in_proto
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib.packet import tcp
from ryu.lib.packet import udp
```

2. Populate the Match

```
            # check IP Protocol and create a match for IP
            if eth.ethertype == ether_types.ETH_TYPE_IP:
                ip = pkt.get_protocol(ipv4.ipv4)
                srcip = ip.src
                dstip = ip.dst
                protocol = ip.proto

                # if ICMP Protocol
                if protocol == in_proto.IPPROTO_ICMP:
                    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ipv4_src=srcip,
ipv4_dst=dstip, ip_proto=protocol)

                #  if TCP Protocol
                elif protocol == in_proto.IPPROTO_TCP:
                    t = pkt.get_protocol(tcp.tcp)
                    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ipv4_src=srcip,
ipv4_dst=dstip, ip_proto=protocol, tcp_src=t.src_port, tcp_dst=t.dst_port,)

                #  If UDP Protocol
                elif protocol == in_proto.IPPROTO_UDP:
                    u = pkt.get_protocol(udp.udp)
                    match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ipv4_src=srcip,
ipv4_dst=dstip, ip_proto=protocol, udp_src=u.src_port, udp_dst=u.dst_port,)
```

Writing Basic Switching Applications

Demo

1. Run Mininet topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4*

2. Run RYU hub application

```
ryu-manager l4_switch.py
```

3. do pingall from mininet.
4. Check the openvswitch flows

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

Writing Basic Switching Applications

5. Flow Timeout

Idle Timeout:

If the flow is Idle(no packets hitting this flow) for the specified time(Idle Time), the flow will be expired.

Hard Timeout:

Flow entry must be expired in the specified number of seconds regardless of whether or not packets are hitting the entry

In simple_switch.py, the default timeout values(for idle and hard) are not set(0). it means, the flows are permanent. it will never expiry.

Writing Basic Switching Applications

Logic

Code changes

1. In the add_flow function, include idle, hard parameters with default value as 0.
2. OFPFlowMod API include the idle_timeout,hard_timeout parameters.

```
    mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    idle_timeout=idle, hard_timeout=hard, priority=priority,
match=match,
                                    instructions=inst)
```

3. Call the add_flow function with idle, hard value

Writing Basic Switching Applications

Demo

1. Run Mininet topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4*

2. Run RYU hub application

```
ryu-manager flow_timeout.py
```

3. do pingall from mininet.
4. Check the openvswitch flows

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

verify the timeout values in the flow,

5. Check the flows again after the timeout seconds.

Statistics Collection

1. Introduction

Openflow protocol provides extensive statistics, such as,

- flow statistics
- flow aggregate statistics
- port statistics
- group table statistics
- meter statistics

User can collect these statistics from the openflow switch and use it(its used for many applications)

User needs to send the statistics request meassge(Example: OpenFlow FlowStatistics Request Message). Switch reply with statistics respone message.

In this tutorial, we are going to write a RYU Code to display the statistics.

Statistics Collection

2. RYU Thread API

RYU comes with inbuilt thread implementation(hub library.)

Import the library

```
from ryu.lib import hub
```

We will write a piece of code, which will be executed in separate thread forever.

```
    def myfunction(self):
        self.logger.info(" started new thread")
        i = 0
        while True:
            hub.sleep(30)
            self.logger.info("printing %d", i)
            i = i + 1
```

Note: its a continuous run, so we use hub.sleep() to sleep for a time.

To initiate a thread

```
hub.spawn(self.myfunction)
```

Statistics Collection

Demo

> *ryu-manager thread.py*

Example:

```
$ ryu-manager thread1.py
loading app thread1.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app thread1.py of SimpleSwitch13
started new thread
printing 0
printing 1
printing 2
```

Statistics Collection

3. Flow Statistcs

Objective

Measure the flows statistics or utilization in regular interval(10second) and print it.

Logic

- Use Ryu Thread(HUB) mechanism,
- In the Thread, Send Openflow Flow Statistics Request message at regular interval
- Flow Statistics Response will be received as event, and display the bytes & packets counter values.

Coding

1. Use Simple Switch application as base application
2. we require datapaths object of all the switches for generating the stats request message. hence we store the datapaths in the dictionary.

Statistics Collection

In the init routine, we declare it.

```
self.datapaths = {}
```

In the switch features handler function, we store the the datapath object in the datapaths dictionary.

```
self.datapaths[datapath.id] = datapath
```

3. Include RYU Thread library

```
from ryu.lib import hub
```

3. Start the thread in the init routine

```
self.monitor_thread = hub.spawn(self.monitor)
```

Here thread function name is monitor.

Statistics Collection

4. Thread function

The thread function to be placed in side the RYU manager application class.

In the thread function, we process the datapaths(switches) dictionary, and generate the flowstats request message(without any match filter). It means we query the stats of all the flows in the switch.

```
def monitor(self):
   self.logger.info("start flow monitoring thread")
   while True:
       hub.sleep(10)
       for datapath in self.datapaths.values():
           ofp = datapath.ofproto
           ofp_parser = datapath.ofproto_parser
           req = ofp_parser.OFPFlowStatsRequest(datapath)
           datapath.send_msg(req)
```

Refer:

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPFlowStatsRequest

Statistics Collection

5. Flow Stats Response handling routine.

```
@set_ev_cls([ofp_event.EventOFPFlowStatsReply,
                 ], MAIN_DISPATCHER)
    def stats_reply_handler(self, ev):
        for stat in ev.msg.body:
            self.logger.info("Flow details:  %s ",stat)
            self.logger.info("byte_count:  %d ", stat.byte_count)
            self.logger.info("packet_count:  %d ", stat.packet_count)
```

Reference:

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPFlowStatsReply

Statistics Collection

Demo

1. start the mininet single topology

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --
topo=single,4
```

2. run our application.

```
ryu-manager flow_stats.py
```

3. ping h1 to h2 in mininet

```
mininet> h1 ping h2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=6.33 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.501 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=64 time=0.112 ms
```

Statistics Collection

4. In the ryu console, we can see the output

```
Flow details:
OFPFlowStats(byte_count=18480,cookie=0,duration_nsec=20000000,duration_sec=206,flags=0,hard_timeout=0,i
dle_timeout=0,instructions=[OFPInstructionActions(actions=
[OFPActionOutput(len=16,max_len=65509,port=1,type=0)],len=24,type=4)],length=104,match=OFPMatch(oxm_fie
lds={'eth_src': '00:00:00:00:00:02', 'eth_dst': '00:00:00:00:00:01', 'in_port':
2}),packet_count=192,priority=1,table_id=0)
byte_count:  18480
packet_count:  192
```

Statistics Collection

4. Aggregate Flow Stats

Objective

Measure Total number flows in regular interval(10second) and print it.

Logic

- Use Ryu Thread(HUB) mechanism,
- In the Thread, Send Openflow Aggregate Flow Statistics Request message at regular interval
- Aggregate Flow Statistics Response will be received as event, and display the bytes & packets counter values.

Statistics Collection

Coding

same as flow statistics example for thread stuff. But we send Aggregate flow stats request.

```
    def monitor(self):
        self.logger.info("start flow monitoring thread")
        while True:
            hub.sleep(10)
            for datapath in self.datapaths.values():
                ofp = datapath.ofproto
                ofp_parser = datapath.ofproto_parser
                cookie = cookie_mask = 0
                match = ofp_parser.OFPMatch()
                req = ofp_parser.OFPAggregateStatsRequest(datapath, 0,
                                                          ofp.OFPTT_ALL,
                                                          ofp.OFPP_ANY,
                                                          ofp.OFPG_ANY,
                                                          cookie, cookie_mask,
                                                          match)
                datapath.send_msg(req)
```

Reference:

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPAggregateStatsRequest

Statistics Collection

Aggregate Flow Stats Response handling routine.

```
    @set_ev_cls([ofp_event.EventOFPAggregateStatsReply,
                 ], MAIN_DISPATCHER)
    def stats_reply_handler(self, ev):
        result = ev.msg.body
        self.logger.info('AggregateFlowStats  %s', result)
        self.logger.info('FlowCount  %d', result.flow_count)
```

Reference:

https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPAggregateStatsReply

Statistics Collection

demo

1. start the mininet single topology

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --
switch=ovsk,protocols=OpenFlow13 --topo=single,4
```

2. run our application.

```
ryu-manager agg_flow_stats.py
```

3. ping h1 to h2 in mininet

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Statistics Collection

4. In the ryu console, we can see the output

Result:

```
AggregateFlowStats  OFPAggregateStats(packet_count=296,byte_count=24752,flow_count=13)
FlowCount  13
```

Statistics Collection

5. Port Statistics

This is similar to previous two exercises

we should use port stats request message.

```
req = ofp_parser.OFPPortStatsRequest(datapath)
```

For Port Stats response,

```
    @set_ev_cls([ofp_event.EventOFPPortStatsReply,
                 ], MAIN_DISPATCHER)
    def stats_reply_handler(self, ev):
        for stat in ev.msg.body:
            self.logger.info("Port statistics :  %s ", stat)
```

Demo same procedure as last two exercises

Statistics Collection

6. Further experiments

- In the stats request, you can specify the filter(Match) and get only subset of the flow stats.
- During this experiment, Capture the openflow packet traces in Wireshark and analyze it .
- Draw graph using python matplotlib library.
- Export the data in to CSV format
- Push the data to Grafana / Graphite and other Time Series Analysis (Graph) Softwate.

Providing Configuration Input to the RYU Application

1. Introduction

At various sceanarios, user need to provide input to the Application instead of hardcoded in the program. For example, user can provide INTERVAL interval in the statistics applications.

RYU supports configuration file input, which will be parsed by the RYU Application and we can use it as a variable.

Providing Configuration Input to the RYU Application

2. Configuration File

step1:

Enter the configuration in a file as below format,

> *cat params.conf*

```
[DEFAULT]

# interval parameter value in seconds
INTERVAL = 10
```

Here we specificed the INTERVAL as 10.

step2: In the Ryu application,

Include the library for configuration apis

```
from ryu import cfg
```

Providing Configuration Input to the RYU Application

Read the Configuration in the init routine of the application

```
self.CONF = cfg.CONF
self.CONF.register_opts([
            cfg.IntOpt('INTERVAL', default=10, help = ('Monitoring Interval'))]
            )
self.logger.info("Interval value %d ",self.CONF.INTERVAL)
```

step3:

When you run the ryu application, you should use --config-file filename options as below,

> *ryu-manager --config-file params.conf flow_stats_param.py*

3. Demo

Now we updated the flow statistics program (flow_stats.py) with configurable INTERVAL parameter.

Repeat the same demo flow statistics procedure.

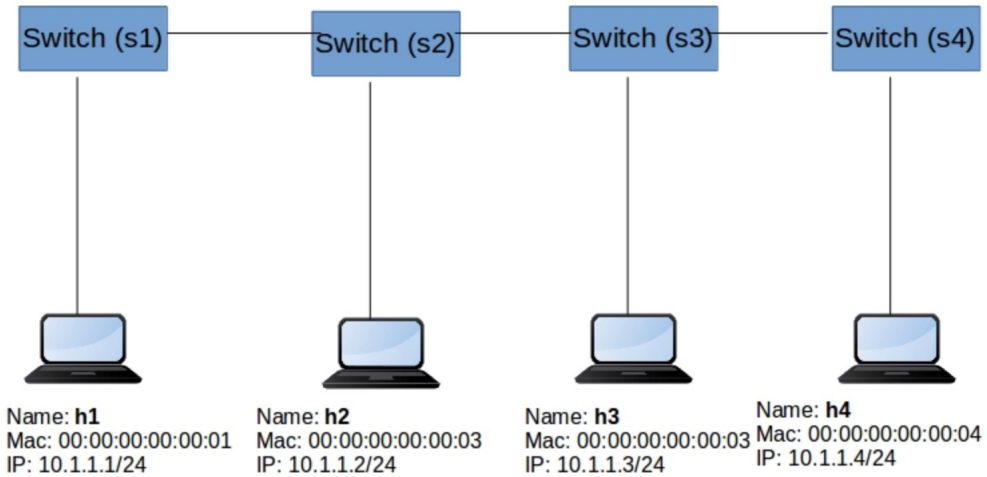
Group Table

1.Introduction

Please refer the group table theory/concepts from PART2 . In this guide, we are going to see, programming methods/api for for group table.

2. Sniffer

Objective



Group Table

Make h2 as Sniffer machine. it will sniff all the packets passing via S2.

Logic

Group table1(Group Table ID 50):

Create a Group table with TYPE=ALL(it means, copy a packet for each bucket. and each bucket will be processed). create two buckets. one bucket will send the packet to Port3, another bucket will send the packet to Port1

Group table2(Group ID 51):

Create a Group table with TYPE=ALL(it means, copy a packet for each bucket. and each bucket will be processed). create two buckets. one bucket will send the packet to Port2, another bucket will send the packet to Port1

Proactively We create group tables and flows in S2.

Group Table

Coding

Use simple_switch_13 as base code

1. Creat a group table create function.

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

        # Hardcoding the stuff, as we already know the topology diagram.
        # Group table1
        # Receiver port2, forward it to port1 and Port3

        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(3)]
        buckets = [parser.OFPBucket(actions=actions1),
                   parser.OFPBucket(actions=actions2)]
        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_ALL, 50, buckets)
        datapath.send_msg(req)

        # Group table2
        # Receive Port3, forward it to port1 and Port2
        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(2)]
        buckets = [parser.OFPBucket(actions=actions1),
                   parser.OFPBucket(actions=actions2)]
        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_ALL, 51, buckets)
        datapath.send_msg(req)
```

Group Table

Here we use OFPGroupMod API for creation groups. refer the API for more details https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPGroupMod

2. In Switch features handler function, call the Group table creation function and add flows for switch S2,

```
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        .........skipped......
        .........skipped......
        # switch s2
        if datapath.id == 2:
            # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)
            # entry 2
            actions = [parser.OFPActionGroup(group_id=51)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)
```

Group Table

Demo

1. run the topology in mininiet

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --
topo=linear,4
```

2. Run the ryu controller application(simple switch and ofctl)

```
ryu-manager sniffer.py
```

3. Do pingall from mininiet

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Group Table

4. Check the flows of switch s2

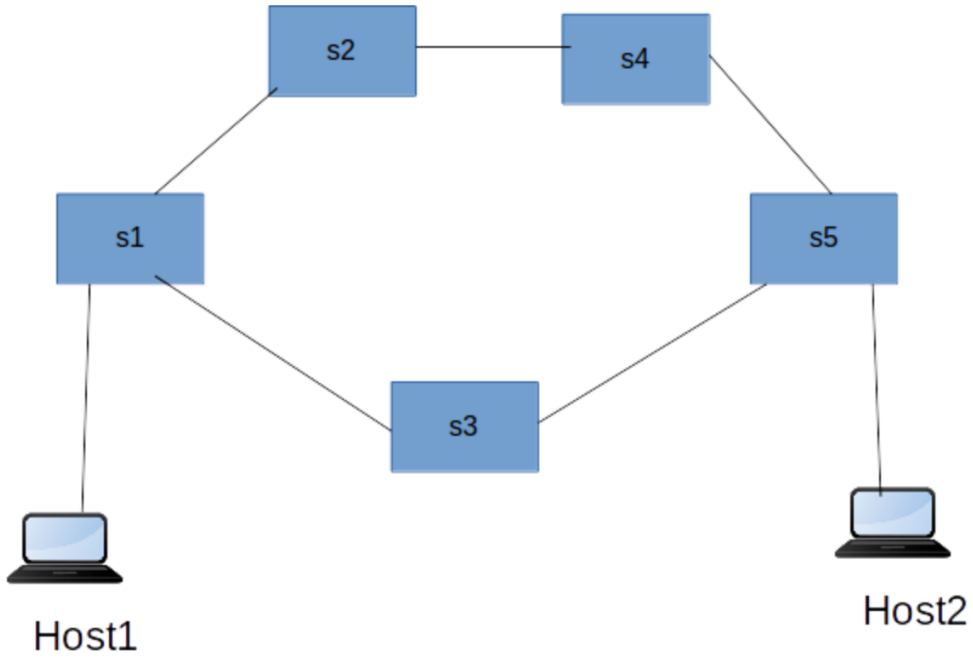
```
sudo ovs-vsctl show
sudo ovs-ofctl -O OpenFlow13 dump-flows s2
sudo ovs-ofctl -O OpenFlow13 dump-groups s2
sudo ovs-ofctl -O OpenFlow13 dump-group-stats s2
```

Group Table

3. Loadbalancer

Objective

Forward the packet to 1 bucket(out of N buckets) and process it. (Load Balancer)



Group Table

Logic

In Switch S1 ,

Create a Group table with TYPE=SELECT (it means, For each bucket, a bucket will be selected (based on weight - vendor implementation). and the selected bucket will be processed).

create two buckets. one bucket will send the packet to S2 Port, another bucket will send the packet to S3 Port

In Switch S5 ,

Create a Group table with TYPE=SELECT (it means, For each bucket, a bucket will be selected (based on weight - vendor implementation). and the selected bucket will be processed).

create two buckets. one bucket will send the packet to S4 Port, another bucket will send the packet to S3 Port

Proactively We create group tables and flows in S1 and S5.

Coding

Use simple_switch_13 as base code

Group Table

1. Creat a group table create function.

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

        # Hardcoding the stuff, as we already know the topology diagram.
        # Group table1
        # Receiver port3 (host connected), forward it to port1(switch) and Port2(switch)
        LB_WEIGHT1 = 30 #percentage
        LB_WEIGHT2 = 70 #percentage

        watch_port = ofproto_v1_3.OFPP_ANY
        watch_group = ofproto_v1_3.OFPQ_ALL

        actions1 = [parser.OFPActionOutput(1)]
        actions2 = [parser.OFPActionOutput(2)]
        buckets = [parser.OFPBucket(LB_WEIGHT1, watch_port, watch_group, actions=actions1),
                   parser.OFPBucket(LB_WEIGHT2, watch_port, watch_group, actions=actions2)]

        req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
                                 ofproto.OFPGT_SELECT, 50, buckets)
        datapath.send_msg(req)
```

Here we use OFPGroupMod API for creation groups. refer the API for more details https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#ryu.ofproto.ofproto_v1_3_parser.OFPGroupMod

Group Table

2. In Switch features handler function, call the Group table creation function and add flows for switch S1 and S5,

```
        # switch s1
        if datapath.id == 1:
            # add group tables
            self.send_group_mod(datapath)
            actions = [parser.OFPActionGroup(group_id=50)]
            match = parser.OFPMatch(in_port=3)
            self.add_flow(datapath, 10, match, actions)

            #add the return flow for h1 in s1.
            # h1 is connected to port 3.
            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=1)
            self.add_flow(datapath, 10, match, actions)

            actions = [parser.OFPActionOutput(3)]
            match = parser.OFPMatch(in_port=2)
            self.add_flow(datapath, 10, match, actions)
```

Group Table

```
# switch s5
 if datapath.id == 5:
 # add group tables
     self.send_group_mod(datapath)
     actions = [parser.OFPActionGroup(group_id=50)]
     match = parser.OFPMatch(in_port=3)
     self.add_flow(datapath, 10, match, actions)


     #add the return flow for h2 in s4.
     # h2 is connected to port 3.
     actions = [parser.OFPActionOutput(3)]
     match = parser.OFPMatch(in_port=1)
     self.add_flow(datapath, 10, match, actions)

     actions = [parser.OFPActionOutput(3)]
     match = parser.OFPMatch(in_port=2)
     self.add_flow(datapath, 10, match, actions)
```

Group Table

Demo

1. run the topology in mininiet

```
sudo python lb_topo.py
```

2. Add the static ARP Entry in the mininiet hosts

```
mininet> h1 arp -s 192.168.1.2 00:00:00:00:00:02
mininet> h2 arp -s 192.168.1.1 00:00:00:00:00:01
```

3. Run the ryu manager

```
ryu-manager loadbalancer.py
```

Group Table

4. Testing

Verify the flows

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
sudo ovs-ofctl -O OpenFlow13 dump-flows s2
sudo ovs-ofctl -O OpenFlow13 dump-flows s3
sudo ovs-ofctl -O OpenFlow13 dump-flows s4
sudo ovs-ofctl -O OpenFlow13 dump-flows s5
```

run iperf test between h1 and h5

```
mininet> h2 iperf -u  -s &
mininet> h2 iperf   -s &
mininet> h1 iperf -c h2 -t 60 -P 5 &
```

Check the flows and groups.

Group Table

4. Further experiments:

- Multipath load balancing with specified percentage
- Dynamically do the load balancing depends on the traffic condition.
- Failover

RYU Packet Parsing & Generation capability

1. Introduction

RYU comes with packent library (consists of plenty of protocols), we can use this library to parse the packets or generate the packets.

There are many usecase around this.

2. Packet Parsing Capbility

We can use the packet library to parse the PACKET IN Message and identify which application and use it.

1. Include the relevant packet library

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

2. In the Packet In handler,

Get the Packet Handler object

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

Now we can get the Protocol headers from the pkt

RYU Packet Parsing & Generation capability

To get the ethernet header

```
eth = pkt.get_protocols(ethernet.ethernet)
self.logger.info("Ethernet Header %s ",eth)
```

To Get the IP Header

```
ip = pkt.get_protocol(ipv4.ipv4)
self.logger.info("IPv4 Header %s ",eth)
```

To Get the TCP header

```
t = pkt.get_protocol(tcp.tcp)
self.logger.info("TCP Header %s ",eth)
```

To get the UDP header

```
u = pkt.get_protocol(udp.udp)
self.logger.info("UDP Header %s ",eth)
```

RYU Packet Parsing & Generation capability

Further we can get the Protocol header details, by processing it

```
u = pkt.get_protocol(udp.udp)
self.logger.info("UDP Header %s ",eth)
self.logger.info ("UDP Src port %d Destination Port %d ",u.src_port, u.dst_port)
```

RYU can handle these many protocols

https://github.com/osrg/ryu/tree/master/ryu/lib/packet

RYU Packet Parsing & Generation capability

3. Packet Generation Capbility

In SDN, SDN Controller can generate a packet and send it. There are many usecases/applications require this capability.

Router SDN Controller needs to have ARP Request & Response capability for neighbour discovery

Proxy Application (ARP Proxy, DNS Proxy) Proxy application need to handle the request of the specific proxy application protocol(such as DNS, ARP etc) and respond it.

Demo - Simple ARP Proxy application

Objective This application answers the ARP requests(instead of forwarding the ARP broadcasts to all hosts)

This Application act as ARP Proxy. It captures the ARP Packets, builds the ARP response and send it to the Src.

RYU Packet Parsing & Generation capability

Logic

1. MAC with IP Table will be hardcoded in the code.
2. In the Packet IN handler, check the packet is ARP Packet.
3. If it is ARP Packet,
    - In the ARP Request Packet, decode the destination IP from ARP header
    - Get the MAC address of the equivalent IP address
    - Build the ARP Response packet
    - Send it to the Received PORT.
4. If not ARP, normal switching routine.

RYU Packet Parsing & Generation capability

Coding

1. include the lib

```
from ryu.lib.packet import arp
```

2. define your table with MAC address and IP Address.(ARP Table)

```
arp_table = {"10.1.1.1": "00:00:00:00:00:01",
             "10.1.1.2": "00:00:00:00:00:02",
             "10.1.1.3": "00:00:00:00:00:03",
             "10.1.1.4": "00:00:00:00:00:04"
             }
```

RYU Packet Parsing & Generation capability

3. Write your arp proxy function

```
def arp_process(self, datapath, eth, a, in_port):
    r = arp_table.get(a.dst_ip)
    if r:
        self.logger.info("Matched MAC %s ", r)
        arp_resp = packet.Packet()
        arp_resp.add_protocol(ethernet.ethernet(ethertype=eth.ethertype,
                              dst=eth.src, src=r))
        arp_resp.add_protocol(arp.arp(opcode=arp.ARP_REPLY,
                              src_mac=r, src_ip=a.dst_ip,
                              dst_mac=a.src_mac,
                              dst_ip=a.src_ip))

        arp_resp.serialize()
        actions = []
        actions.append(datapath.ofproto_parser.OFPActionOutput(in_port))
        parser = datapath.ofproto_parser
        ofproto = datapath.ofproto
        out = parser.OFPPacketOut(datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER,
                              in_port=ofproto.OFPP_CONTROLLER, actions=actions, data=arp_resp)
        datapath.send_msg(out)
        self.logger.info("Proxied ARP Response packet")
```

RYU Packet Parsing & Generation capability

4. In the packet handler function, if it is arp packet, call the arp_proxy function.

```
        # Check whether is it arp packet
        if eth.ethertype == ether_types.ETH_TYPE_ARP:
            self.logger.info("Received ARP Packet %s %s %s ", dpid, src, dst)
            a = pkt.get_protocol(arp.arp)
            self.arp_process(datapath, eth, a, in_port)
            return
```

RYU Packet Parsing & Generation capability

Demo

1. Run Mininet topology

> *sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=single,4*

2. Run RYU hub application

```
ryu-manager arp_proxy.py
```

3. Check the openvswitch flows

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

RYU Packet Parsing & Generation capability

4. start the tcpdump capture in h2

```
xterm h2
tcpdump -i any -vvv
```

5. do ping h1 to h2 from mininet

```
mininet>h1 ping h2
```

6. Check the tcpdump traces in h2 you will not the ARP Request, Controller respond with ARP Reply.

4. Further Experiments

- Implement DNS Proxy Application
- Implement Simple Gateway / router Application

Topology Discovery

1. Introduction

RYU has Topology discovery mechnism. RYU uses LLDP(Link Layer Discovery Protocol) to discover the switches and links. Topology discovery feature will be enabled with **"--observe-links"** option.

Topology discovery feature is used in many applications such as identify the shortest path, avoid loops, multipath with load balancing etc.

Topology Discovery

2. How it works

Switch & Link Discovery:

- RYU Controller generates LLDP Packet(datapath id, port number) from each switch and each port number.
- LLDP Packet reaches the other end of the link (its another switch port).
- This packet will send it to the controller
- Controller decodes the packet and learn the Switch, and Link information from the LLDP Packet (sender switch details(datapath ID, Port number) and received switch details((datapath ID, Port number)))

Host discovery:

Host discovery is based on the incoming packet. Usually the host machines are keep sending some broadcast/multicast packets(some services will trigger this). RYU Controller uses these packets to discover the hosts.

Topology Discovery

3. Coding

Logic:

Topology discovery process may take few seconds to complete it . So we use RYU thread and print the topology information after 10seconds. We assume within 10 seconds, the topology discovery process will be completed.

1. Include the header

```
from ryu.topology.api import get_switch, get_link, get_host
```

2. Initiate a thread

```
        hub.spawn(self.myfunction)
```

Topology Discovery

3. Printing the topology discovery information

```
    def myfunction(self):
        self.logger.info("started new thread")
        hub.sleep(10)

        switch_list = get_switch(self.topology_api_app, None)
        self.switches = [switch.dp.id for switch in switch_list]
        links_list = get_link(self.topology_api_app, None)
        self.links = [(link.src.dpid, link.dst.dpid, {'port': link.src.port_no}) for link in
links_list]
        host_list = get_host(self.topology_api_app, None)
        self.hosts = [(host.mac, host.port.dpid, {'port': host.port.port_no}) for host in host_list]
        self.logger.info("*********Topology Information************")
        self.logger.info("Switches %s", self.switches)
        self.logger.info("Links %s", self.links)
        self.logger.info("Hosts %s", self.hosts)
```

we use three APIs **get_switch, get_link, get_host** . These api provides the discovered switch,link and host details in the list of objects.

This below line, we are processing the switch list, and get only switch dp id.

```
        self.switches = [switch.dp.id for switch in switch_list]
```

Simiar to other link and host.

Topology Discovery

4. Demo

1. Run Mininet Linear Topology

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --
topo=linear,4
```

2. Start the Wireshark Capture with loopback interface
3. Run ryu controller with "--observe-links" options

```
ryu-manager --observe-links topology_discovery.py
```

4. Topology information will be printed in the RYU Console.

```
*********Topology Information************
Switches [1, 2, 3, 4]
Links [(2, 3, {'port': 3}), (3, 2, {'port': 2}), (3, 4, {'port': 3}), (2, 1, {'port': 2}), (4, 3,
{'port': 2}), (1, 2, {'port': 2})]
Hosts [('00:00:00:00:00:03', 3, {'port': 1}), ('00:00:00:00:00:02', 2, {'port': 1}),
('00:00:00:00:00:01', 1, {'port': 1}), ('00:00:00:00:00:04', 4, {'port': 1})]
```

5. Analyze the Wireshark traces(LLDP Packets)

Topology Discovery

5. Further experiments

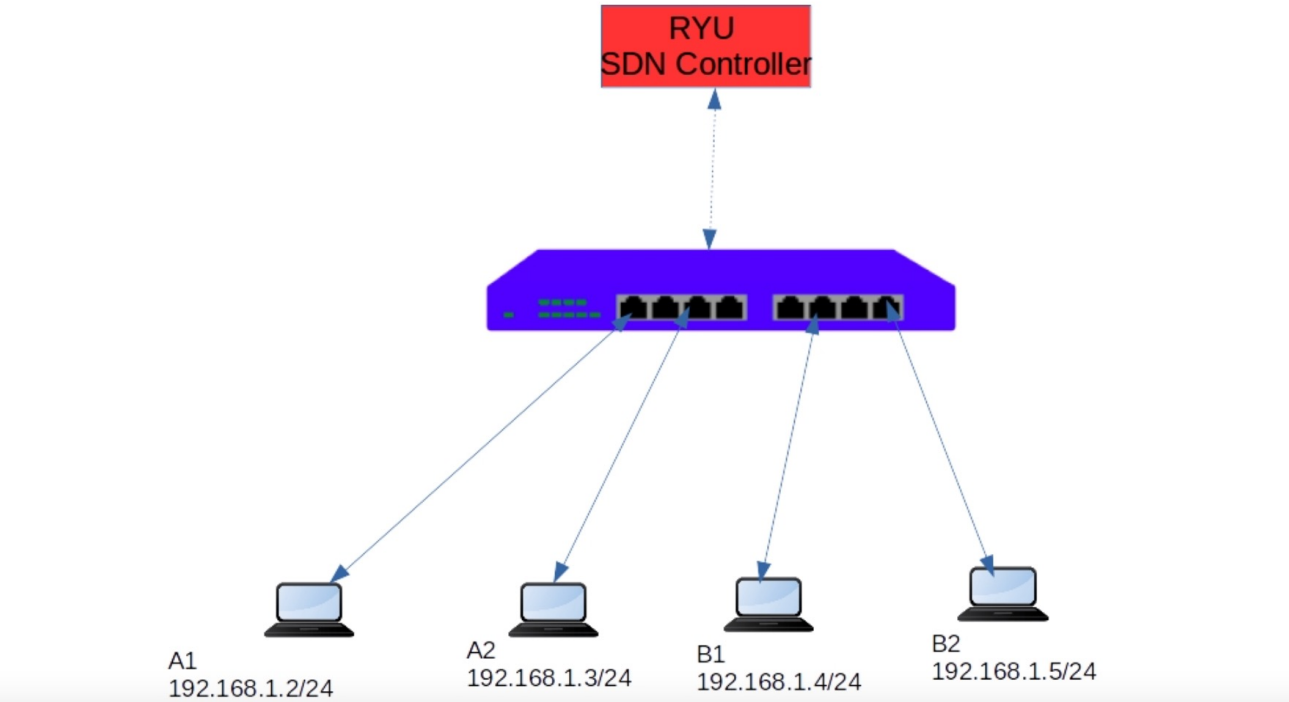
- Use matplotlib library to draw the topology diagram and save it as image.

Multicontroller

1. Introduction

Problem Statement

1. Controller failure in SDN Network is major failure affects the entire network topology
2. Various reasons for controller failure - security threats, hardware /software issues, manual errors etc.
3. Using Single Controller in SDN Network is high risk, high possibility for failure, not scalable and fault tolerant.



Multicontroller

Demo - Single Controller

1. Running Mininet topology with Linear topology.

```
    sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=single,4
```

2. Running a ryu controller with simple switch application(idle timeout/hard timeout to 30seconds)

```
ryu-manager flow_timeout.py
```

3. Pingall.
4. Stop the controller
5. Check the flows

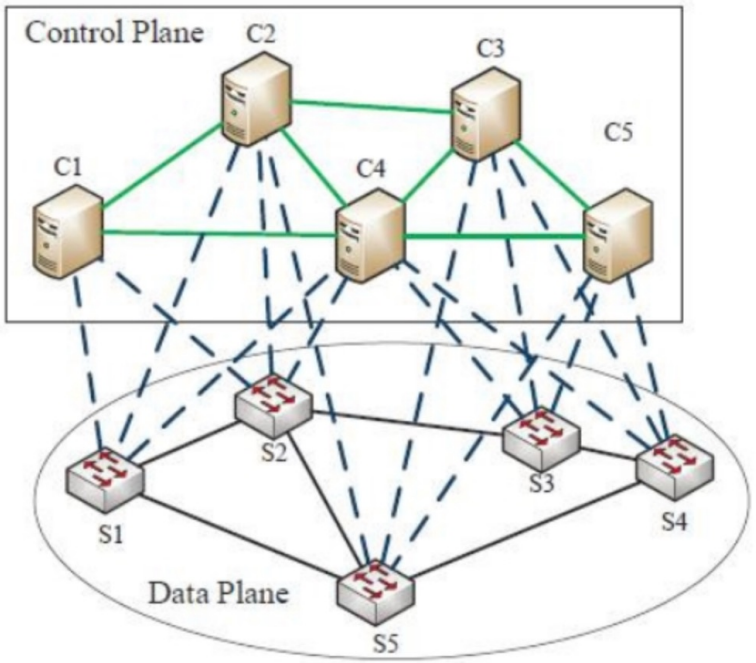
Observation: Once the controller is stopped, topology works till the flows are timeout(hard) after that, the entire topology datapath is not working. hosts cannot reach other hosts.

Its a single point failure.

Multicontroller

2. Cluster of SDN Controllers:

Openflow specification supports the Multiple controllers environment. User can configure the multiple controllers in the switch.



Multicontroller

3. Controller Roles

Equal Role

It means, all the controllers configured in the switch have full control to update/modify the flows.

Switch must send the PACKET IN Message to all the Controllers. Also switch process the PACKET OUT, FLOW UPDATES etc from all the controllers.

Master Role

It means, MASTER controller will be responsible for managing the switch openflow dataplane. Switch will send the control messages to only MASTER controller.

Slave Role

Slave plays backup role for MASTER. it also receives the HELLO and KEEPALIVE Message. But it cannot send and receive the Control message.

Multicontroller

4.Openflow ROLE Change Message:

ROLE_REQUEST and RESPONSE:

When the controller comes up, it will send the ROLE REQUEST with ROLE MASTER, other controller should send a role as SLAVE. Switch will communicate with MASTER.

5. Demo - Equal role

1. controller1

```
ryu-manager --ofp-tcp-listen-port 6653 flow_timeout.py
```

2. controller2

```
ryu-manager --ofp-tcp-listen-port 6654 flow_timeout.py
```

3. Mininet topology

```
sudo python topology.py
```

4. Start the wireshark capture for traces

5. pingall

Note : we can see the packet in comes from both controller

Multicontroller

6. Demo- Master/Slave Role

Here one controller going to act as Master Role, another controller going to act as SLAVE or BACKUP role.

1. Start the wireshark capture for traces in loopback interface
2. Mininet topology

```
sudo python topology.py
```

3. start the master controller

```
ryu-manager --ofp-tcp-listen-port 6653 l3switch_master.py
```

4. start the backup controller

```
ryu-manager --ofp-tcp-listen-port 6654 l3switch_slave.py
```

5. Check the traces in controller terminals. you can see only MASTER terminal you can see packet in traces.
6. pingall

only MASTER Controller manages the switches

7. Analyze the ROLE Messages in the Wireshark traces.

Multicontroller

7. Data Synchronization

In Multicontroller environement(specifically Master/slave), data(intelligence built by the controller. For example mac_to_port structure) needs to be synchronized across the controller.

There are many mechnisms available such as DB, InMemoryDB, Message Brokers etc.

8. Further experiments

- Use InmemoryDB or MessageBroker for DataSynchronization
- Implement Master Election Algorithm/ Failure Detection /Autofailover
- Distributed Master/Slaves