

Implementation and Evaluation of Routing
Algorithm in GENI Software-Defined
Networking

Wei-Hen Hsu

June 7, 2018

Contents

1	Introduction	3
2	General Background Information	5
2.1	Software-defined networking	5
2.2	OpenFlow	6
2.2.1	OpenFlow switch components	7
2.3	GENI	7
2.4	Open vSwitch	8
2.5	Ryu controller	9
3	System model	10
3.1	Workflow of GENI	10
3.2	Network in GENI	11
3.3	Ryu controller	11
4	Implementation of Routing Algorithm in GENI	17

4.1	The reservation of compute resources for customized topology	17
4.2	Offline algorithm	19
4.2.1	create the network graph for the input of routing algorithm	19
4.2.2	setting up the demands	20
4.2.3	setting up the controller	20
4.3	Online algorithm	20
4.3.1	ryu controller	20
5	Case study	22
6	Experiment results	23
7	Summary	24
7.1	conclusion	24
7.2	future work	24
8	References	25

Chapter 1

Introduction

Software-defined networking (SDN) has received a lot of attention in recent years, several routing algorithms have been proposed by researchers. The environment that evaluates a routing algorithm is quite important.

Commonly, building up a SDN environment with real OpenFlow switches for evaluating the performance of routing algorithms is very persuasive. Unfortunately, a real OpenFlow switch is expensive. It's costly to build a small-scale network with real OpenFlow switches.

Another tool for building the SDN environment is the network testbed. The network testbed is a platform to provide network researchers with a realistic environment for testing. In some network testbeds, they are supported by governments and it's free for researchers to use the testbed. Compared with

previous ways, it is more cost-effective for researchers to build the environment.

Therefore, in this thesis, we proposed a method to implement the routing algorithms in **Global Environment for Network Innovations (GENI)** SDN testbed. The routing algorithms fall into the following two main categories: (a) online routing algorithm; (b) offline routing algorithm. In each category, we describe the process of creating a SDN on GENI and implementing the routing algorithm on SDN.

Chapter 2

General Background Information

2.1 Software-defined networking

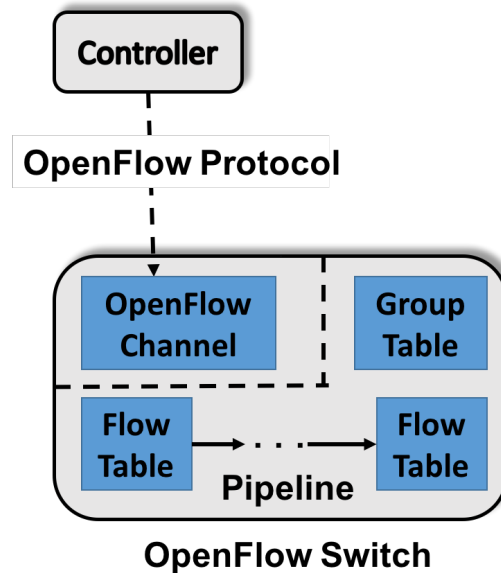
According to **ONF(Open Networking Foundation)** definition of software-defined networking, SDN is an architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. In traditional network, each network device usually has its own control plane. The transport of the packet is processed by network device individually. The SDN decouples the forwarding and control planes, removes the control plane from network device and implements it in software instead. The SDN enables the control plane to become directly programmable and

the underlying infrastructure to be abstracted for applications and network services. The OpenFlow protocol is a foundational element for building SDN solutions.

2.2 OpenFlow

OpenFlow is a communications protocol between SDN controller and OpenFlow switch that enables the SDN controller to direct interact with the forwarding plane of network devices such as switches and routers. Figure 2.1 shows the main components of an OpenFlow switch.

Figure 2.1: Main components of an OpenFlow switch.



Match Fields	Priority	Counter	Instructions	Timeouts	Cookie
--------------	----------	---------	--------------	----------	--------

Table 2.1: Main components of a flow entry in a flow table

2.2.1 OpenFlow switch components

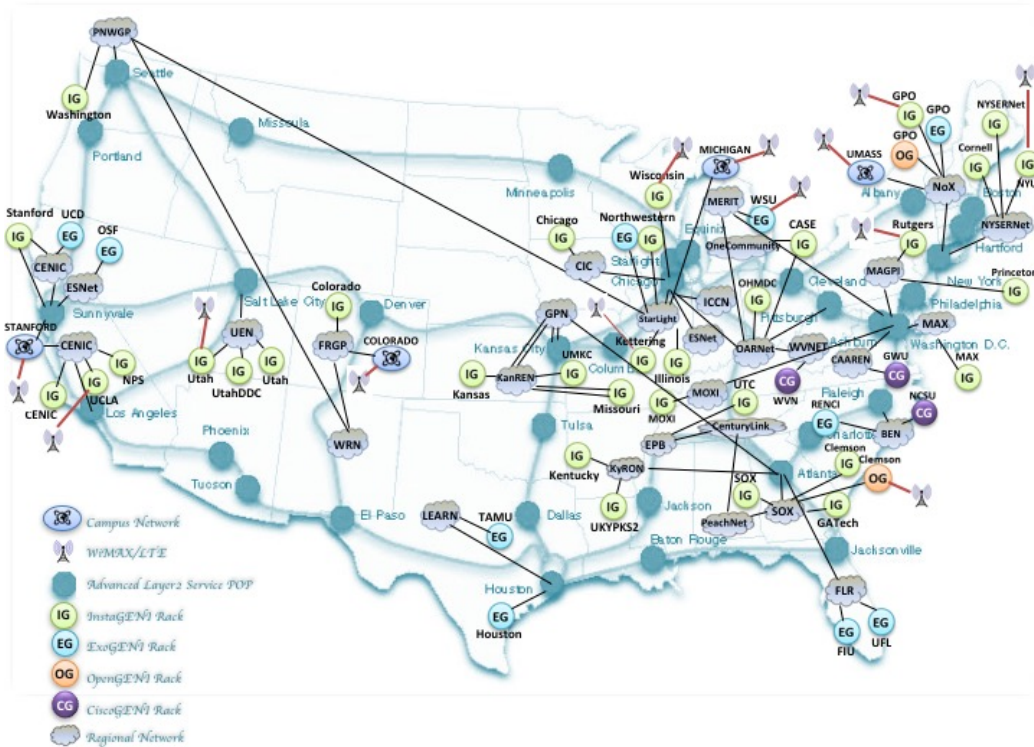
An OpenFlow switch consists one or more flow tables, which perform packet lookups and forwarding. The controller could use the OpenFlow protocol to add, update, and delete flow entries in flow tables. In each flow table, it contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets. (see Table 2.1)

2.3 GENI

GENI provides a virtual laboratory for networking and distributed systems research and education. Researcher could obtain compute resources from locations around the United States. Figure 2.2 show the map of GENI compute and network resources. GENI allows user to install custom software or even custom operating systems on compute resources. Furthermore, researcher could control how network switches in their experiment handle traffic flows. GENI is sponsored by the U.S. National Science Foundation (NSF). If the researchers' institution

belongs to an identity federation, the researchers are able to log in using their usual username and password. If their institution does not belong, they could apply for an account at the NCSA identity Provider.

Figure 2.2: The map of GENI compute and network resources.



2.4 Open vSwitch

Open vSwitch is a open source, multi layer virtual switch implemented in software. It can be intergrated with hardware where it can act as as control plane. One example is it can handle

OpenFlow control plane in a hardware silicon. There are three mainly components in Open vSwitch: (a) Control Plane; (b) Kernel Mode; (c) Command line interface.

2.5 Ryu controller

Ryu is a component-based SDN framework. Ryu is fully written in Python and provides software components with well defined API that make it easy for developers to create new management and control applications.

Chapter 3

System model

3.1 Workflow of GENI

Figure 3.1 shows the workflow of GENI. In the beginning, the GENI experimenters need to create a slice. A GENI slice is:

- (a) The unit of isolation for experiments. Only experimenters who are members of a slice can make changes to experiments in that slice;
- (b) A container for resources used in an experiment.

For creating a specific topology, there are three methods to achieve that. Resource Specification (RSpec) is a XML document in a prescribed format. Experimenters send to aggregates a request RSpec that describes the resources they want.

After finishing in reservation, experimenters could use GENI desktop tool to configure each node manually. Another method is to use the manifest RSpec, manifest RSpec describes the re-

sources the got. The manifest include the names and IP addresses and login accounts of compute resources. The experimenters could use SSH to login in the resources and configure the nodes.

When finished setting the nodes including controller, the experimenters use SSH to connect to each node and run the **iPerf** to start the traffic. **iPerf** is a software that could create the traffic in the networks.

Finally, the experiments could clone the mesaurment datas from each node and view the information they want.

3.2 Network in GENI

Figure 3.2 shows the network in GENI. Each nodes is a virtual machine installed the Open vSwitch. The controller is installed the RYU controller.

3.3 Ryu controller

In Ryu controller, the controller will make connection to each switch by using OpenFlow protocol to send the hello message. After receving the reply message, the controller will want to get the information of switches. The controller will send

Figure 3.1: The workflow of GENI.

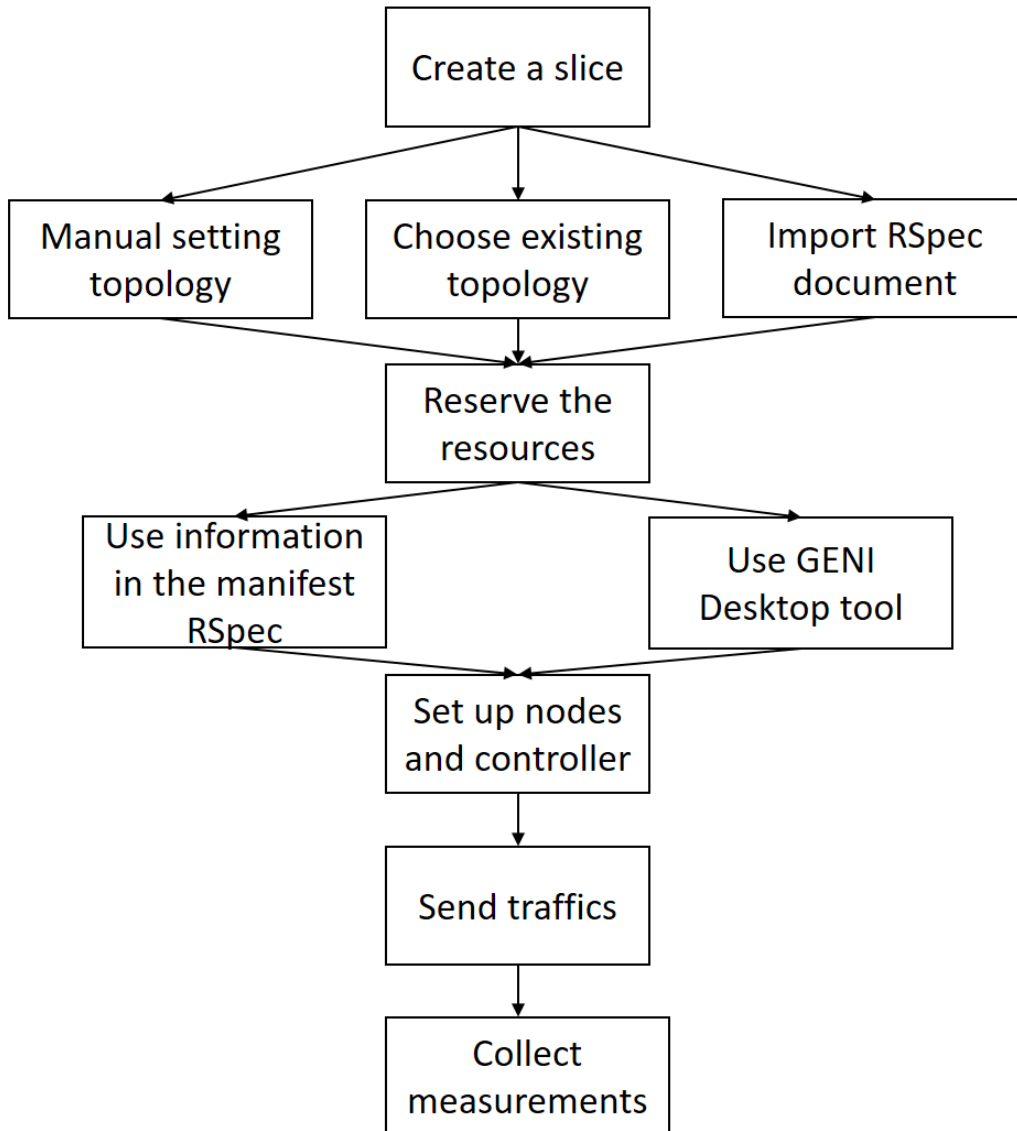
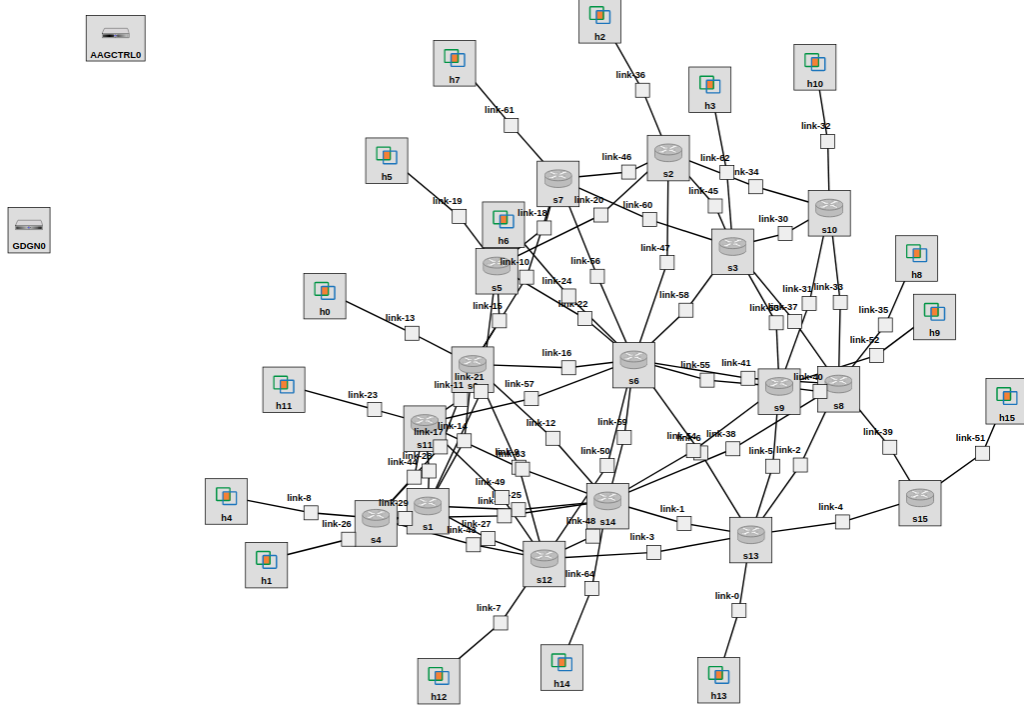


Figure 3.2: The network in GENI.



the features request to the switch and wait the reply message. Then, the controller will send to flow mod message to the switch which setting the switch how to handle the packet. After that, the controller will start to listen the packet. When a packet is sent to the controller by the switch. The controller will create an event. Because the event is casue by a pcket being sent, so the evnet will be sent to the function which handle with the packet in controller. Figures 3.3 shows the model of Ryu application. The data path thread will collect all the events from OpenFlow switch and dispatch the events to the event handler

to process it. The programmer should define the event handler how to process the event. Figure 3.4 shows the status of ryu from the beginning till the end. Figure 3.5 is a example code of how the controller handling the packet.

Figure 3.3: Ryu application programming model.

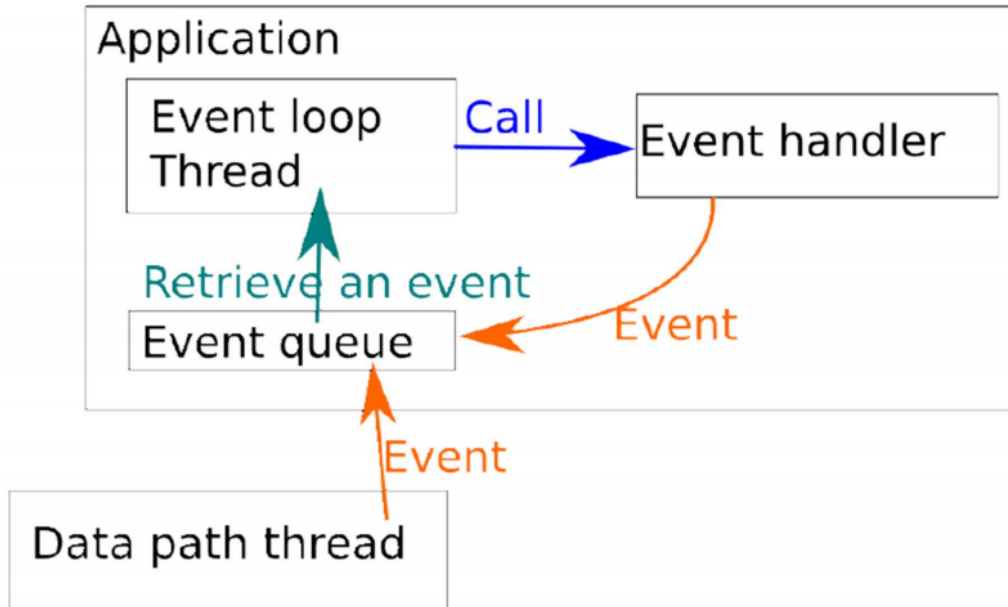


Figure 3.4: The four status in ryu controller

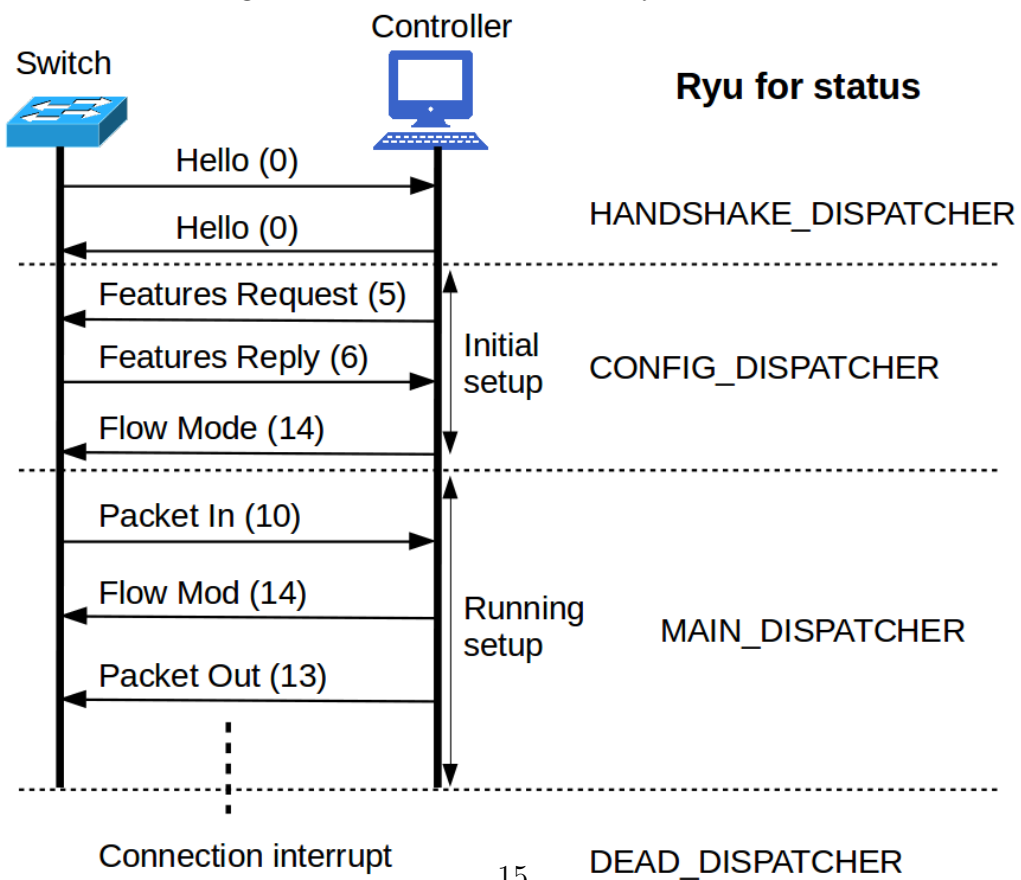


Figure 3.5: The example of the function which handle the packet

```
@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)
```

Chapter 4

Implementation of Routing Algorithm in GENI

4.1 The reservation of compute resources for customized topology

In GENI, There are three methods to reserve the resources: (a) setting the topology manually; (b) choose existing topology; (c) import Resource specification (Rspec) documents. Figure 4.1 to Figure 4.3 show the three methods to reserve the resources. In (a), you could just drag the components into the screen and create the topology you desire. In (b) choosing the existing topology that the GENI offering. In (c) To use the Rspec documents, you could write a program to output the Rspec document by following the Rspec structure.

Figure 4.1: (a) Setting the topology manually.

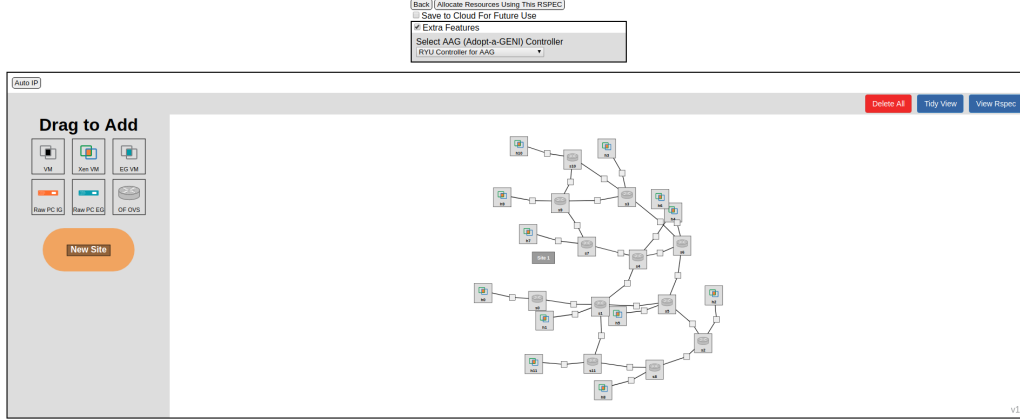


Figure 4.2: (b) Choose existing topology.



Figure 4.3: (c) Import Resource specification (Rspec) documents.

```

-⟨rspec⟩
-⟨node client_id="s2"⟩
+⟨sliver_type name="emulab-xen"⟩⟨/sliver_type⟩
  ⟨icon url="https://portal.geni.net/images/router.svg"⟩
  ⟨site id="Site 1"⟩
  ⟨interface client_id="interface-0"⟩
  ⟨interface client_id="interface-1"⟩
  ⟨interface client_id="interface-2"⟩
  ⟨/node⟩
+⟨node client_id="s4"⟩⟨/node⟩
+⟨node client_id="h7"⟩⟨/node⟩
+⟨node client_id="h11"⟩⟨/node⟩
+⟨node client_id="h9"⟩⟨/node⟩
+⟨node client_id="h10"⟩⟨/node⟩
+⟨node client_id="h5"⟩⟨/node⟩
+⟨node client_id="h0"⟩⟨/node⟩
+⟨node client_id="h8"⟩⟨/node⟩
+⟨node client_id="h1"⟩⟨/node⟩
+⟨node client_id="s5"⟩⟨/node⟩
+⟨node client_id="s9"⟩⟨/node⟩
+⟨node client_id="s10"⟩⟨/node⟩
+⟨node client_id="s0"⟩⟨/node⟩
+⟨node client_id="h4"⟩⟨/node⟩
+⟨node client_id="s1"⟩⟨/node⟩
+⟨node client_id="h3"⟩⟨/node⟩
+⟨node client_id="s6"⟩⟨/node⟩
+⟨node client_id="h2"⟩⟨/node⟩
+⟨node client_id="s3"⟩⟨/node⟩
+⟨node client_id="s11"⟩⟨/node⟩
+⟨node client_id="s7"⟩⟨/node⟩
+⟨node client_id="s8"⟩⟨/node⟩
+⟨node client_id="h6"⟩⟨/node⟩
-⟨link client_id="link-0"⟩
  ⟨interface_ref client_id="interface-0"⟩
  ⟨interface_ref client_id="interface-16"⟩
  ⟨property capacity="100000" dest_id="interface-16" source_id="interface-0"⟩
  ⟨/link⟩
  
```

Figure 4.4: The network structure in GENI

```
Aggregate CENIC InstaGENI's Raw Resources:
<?xml version="1.0"?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1" xmlns:tour="http://www.protogeni.net/resources/rspec/ext/emulab/1" client_id="s13" component_manager_id="urn:publicid:IDN+instageni.cenic.net">
  <sliver_type name="emulab-xen">
    <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops:UBUNTU14-0VS2.31"/>
  </sliver_type>
  <icon xmlns="http://www.protogeni.net/resources/rspec/ext/jacks/1" url="https://portal.geni.net/images/router.svg"/>
  <site id="Site 1"/>
  <interface client_id="interface-0" component_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" sliver_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" type="ipv4"/>
  </interface>
  <interface client_id="interface-1" component_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" sliver_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" type="ipv4"/>
  </interface>
  <interface client_id="interface-2" component_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:eth2" sliver_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:eth2" type="ipv4"/>
  </interface>
  <interface client_id="interface-3" component_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" sliver_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" type="ipv4"/>
  </interface>
  <interface client_id="interface-4" component_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" sliver_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" type="ipv4"/>
  </interface>
  <interface client_id="interface-6" component_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:eth3" sliver_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:eth3" type="ipv4"/>
  </interface>
  <interface client_id="interface-5" component_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:eth2" sliver_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:eth2" type="ipv4"/>
  </interface>
  <sliver_type name="emulab-xen">
    <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops:UBUNTU14-0VS2.31"/>
  </sliver_type>
  <services>
    <login authentication="ssh-keys" hostname="pc3.instageni.cenic.net" port="25409" username="mjtsai"/>
    <emulab:console server="vhost3.shared-nodes.emulab-ops.instageni.cenic.net"/>
  </services>
  <emulab:vnode name="pcvm3-35" hardware_type="pcvm"/>
  <host name="s13.newyork.ch-geni-net.instageni.cenic.net"/>
</node>
<node xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1" client_id="h13" component_manager_id="urn:publicid:IDN+instageni.cenic.net">
  <sliver_type name="emulab-xen">
    <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops:GDZEN160218"/>
  </sliver_type>
  <icon xmlns="http://www.protogeni.net/resources/rspec/ext/jacks/1" url="https://portal.geni.net/images/Xen-VM.svg"/>
  <site id="Site 1"/>
  <interface client_id="interface-7" component_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" sliver_id="urn:publicid:IDN+instageni.cenic.net+interface+pc3:lo0" type="ipv4"/>
  </interface>
  <sliver_type name="emulab-xen">
    <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops:GDZEN160218"/>
  </sliver_type>
  <services>
    <login authentication="ssh-keys" hostname="pc3.instageni.cenic.net" port="25409" username="mjtsai"/>
    <emulab:console server="vhost3.shared-nodes.emulab-ops.instageni.cenic.net"/>
  </services>
  <emulab:vnode name="pcvm3-35" hardware_type="pcvm"/>
  <host name="s13.newyork.ch-geni-net.instageni.cenic.net"/>
</node>
```

4.2 Offline algorithm

4.2.1 create the network graph for the input of routing algorithm

After reserve the resources, copy the network structure document from GENI online website. Figure 4.4 shows the the network structure document. The network structure document contains the information of nodes and links. Using this document to create the network graph for the input of the routing algorithm.

4.2.2 setting up the demands

After the routing algorithm compute the pathes of the demands. In order to let the Open vSwitch to identify each demands, we use ip of source, ip of destination, port of destination to identify each demands. For each nodes, we create a script to run the demands with given information.

4.2.3 setting up the controller

For controller, we create a dictionary for it. Every time when a node asks the controller how to process the demands, controller could use ip of source, ip of destination, port of destination as a key to find the path of the demand then send the OpenFlow message to the switch for adding the flow into the flow table.

4.3 Online algorithm

4.3.1 ryu controller

In ryu controller, we need to implement a topology discovery which the routing algorithm could use. Figure 4.5 is the package which the Ryu controller need to import. Figure 4.6 is the code to discover topology.

Figure 4.5: Ryu controller import package for topology discovery.

```
from ryu.topology import event, switches
from ryu.topology.api import get_switch, get_link
```

Figure 4.6: The code of topology discovery in Ryu controller.

```
@set_ev_cls(event.EventSwitchEnter)
def get_topology_data(self, ev):
    switch_list = get_switch(self.topology_api_app, None)
    switches=[switch.dp.id for switch in switch_list]
    links_list = get_link(self.topology_api_app, None)
    links=[(link.src.dpid,link.dst.dpid,{ 'port':link.src.port_no})
    for link in links_list]
```

Chapter 5

Case study

Chapter 6

Experiment results

Chapter 7

Summary

7.1 conclusion

7.2 future work

Chapter 8

References