國 立 清 華 大 學

碩　士　論　文

在GENI環境裡，將路由演算法實做進軟體定義網路裡並且評估其效能

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

系　　別： 資 訊 工 程 學 系　**組別：** ________ 中

學號姓名： 105062598　徐 偉 涵（Wei-Hen Hsu）

指導教授： 蔡 明 哲　博士　（Dr. Ming Jer Tsai）

華民國一百零四年七月

中文摘要

軟體定義網路越來越受到重視, 而許多的路由演算法相繼被提出, 評測演算法的表現能力成了重要的議題. 以實體的交換機搭建出的軟體定義網路環境雖然眞實, 但所需的成本極大. 實驗床是另一種搭建軟體定義網路環境的方法, 實驗床大部分是由某些組織或者企業所搭建出來的環境, 它提供給研究者搭建環境時所需要的計算及網路資源, 並且提供界面讓研究者可以在上面操作設定其所需的實驗環境, 因爲所使用的資源是跟許多人共享使用的, 所以成本相較於實體交換機少很多, 而所量測出的實驗數據又比依靠軟體模擬可信許多.

因此本論文中, 探討如何在一個GENI實驗床中建立出一個軟體定義網路, 並且我們將路由演算法區分爲線下與線上的演算法, 針對這兩種演算法, 我們逐一探討如何將其放入所搭建好的軟體定義網路環境中並且進行測試與收集研究者所需要的測試結果.

Abstract

Software-Defined networking has received more and more attentation recently, lots of routing algorithms have been proposed. For evaluating the performace of a routing algorithm becomes a important issue. Although Using real OpenFlow switch for constructing a software-defined networking is quite real, the cost for building is vrey high. The network testbed is another way for build software-defined networking. The network testbed is usually created by some organizations or enterprises. It offers the compute and network resources for the researchers. Also, it offers a platform for the researchers to operate and build the experimental environment they wnat. Because the compute and network resources are shared by other researchers, the cost is lower compared to building by real machines. Also, the result of the experiment is more persuasive than testing by some simulators.

Contents

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 importmant.

Commonly, building up a SDN environment with real OpenFlow switches for evaluating the porformance of routing algorithms is very persuasive. Unfortunately, a real OpenFlow switch is expensive. It's costly to build a small-scale netwrok 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 governmentsit 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 propsed 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 catergory, we describe the process of creating a SDN on GENI and implementing the rouing algorithm on SDN.

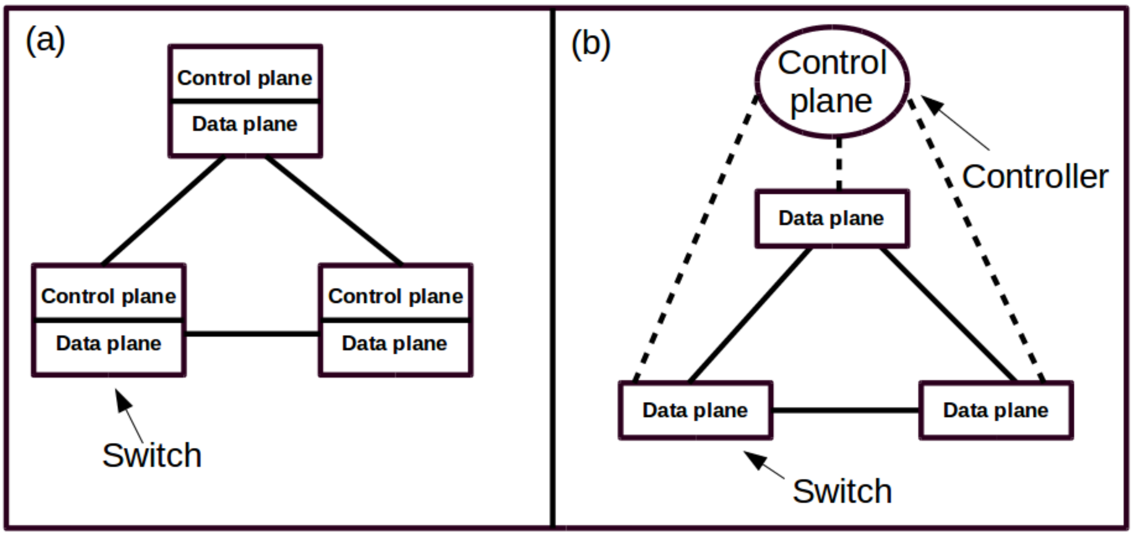
Chapter 2

General Background Information

Software-defined networking

According to **ONF(Open Networking Foundation)** [1] 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 sofware instead. In Figure 2.1, (a) is the traditional network and (b) is SDN. The SDN enables the control plane to beccome directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow® protocol is a foundational element

Figure 2.1: Traditional network and SDN.



for building SDN solutions.

OpenFlow

OpenFlow is a communication 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 2.2 shows the main components of an OpenFlow switch.

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 ta-

Match Fields	Priority	Counter	Instructions	Timeouts	Cookie

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

bles. 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)

GENI

GENI [2] provides a virtual laboratory for networking and distributed systems research and education.Researcher could obtain compute resources from locations around the United States. Figure 2.3 shows the map of GENI compute and netwrok resouces. 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 login using their usual username and password. If their institution does not belong, they could apply for an account at the NCSA identity Provider.

Open vSwitch

Open vSwitch [3] 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 OnpenFlow control plane in a hardware silicon. There are three mainly components in Open vSwitch: (a) Control Plane; (b) Kernel Mode; (c) Command line interface.

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.

Figure 2.2: Main components of an OpenFlow switch.

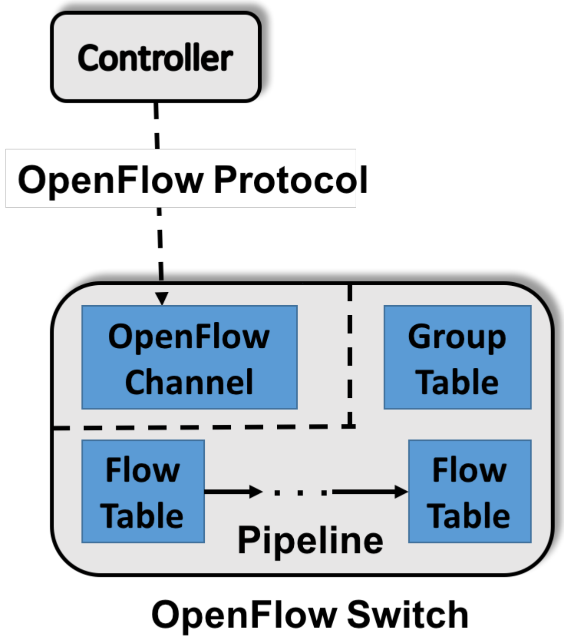
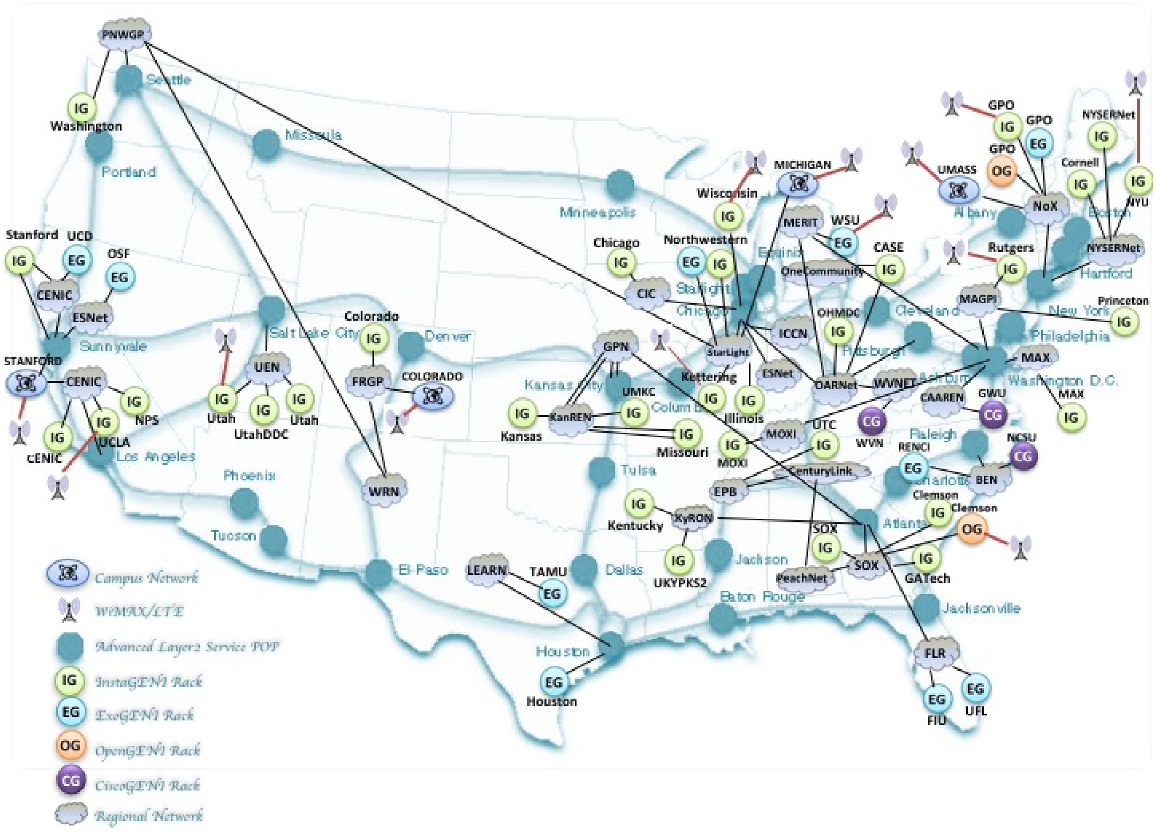


Figure 2.3: The map of GENI compute and netwrok resouces.



Chapter 3

System model

Workflow of GENI

Figure 3.1 shows the workflow of GENI.

Create a slice

In the beginning, the GENI experimenters need to creat 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.

Create a topology

For creating a specific topology, there are three methods to achieve that. Resouece Specification (RSpec) is a XML document in a prescribed format. Experimenters sned to aggregates a request RSpec that describe the rosources they want.

Configure switches and controller

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 roesources 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.

Run the experiment and collect the data

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** [4] is a software that could create the traffic in the networks.

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

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.

Figure 3.1: The workflow of GENI.

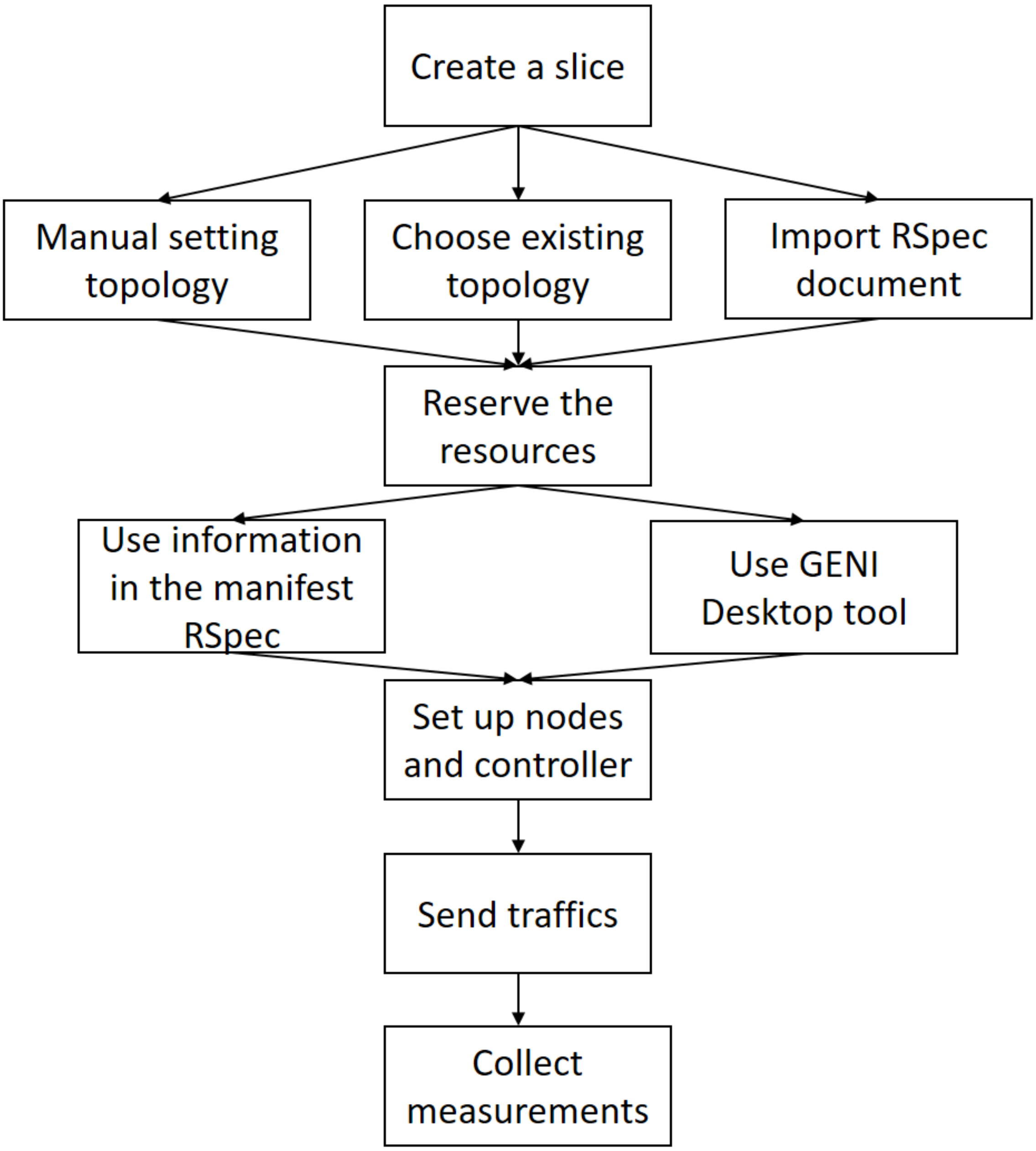
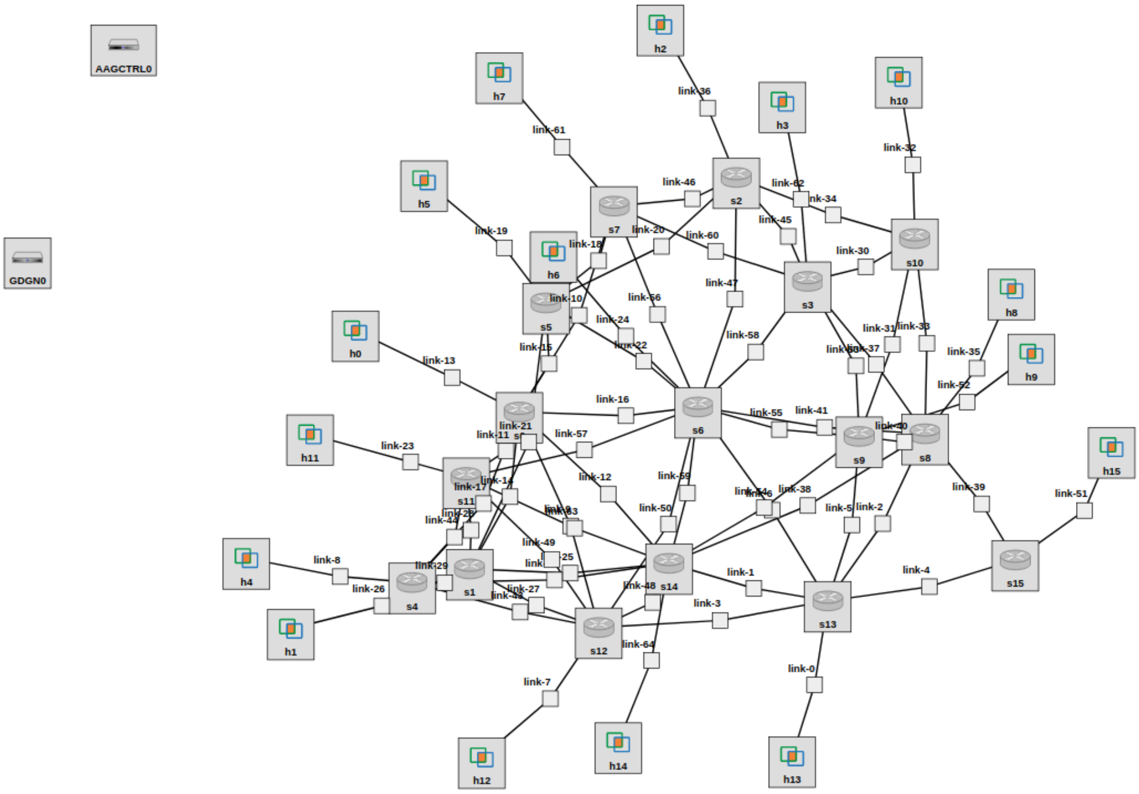


Figure 3.2: The network in GENI.



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 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.

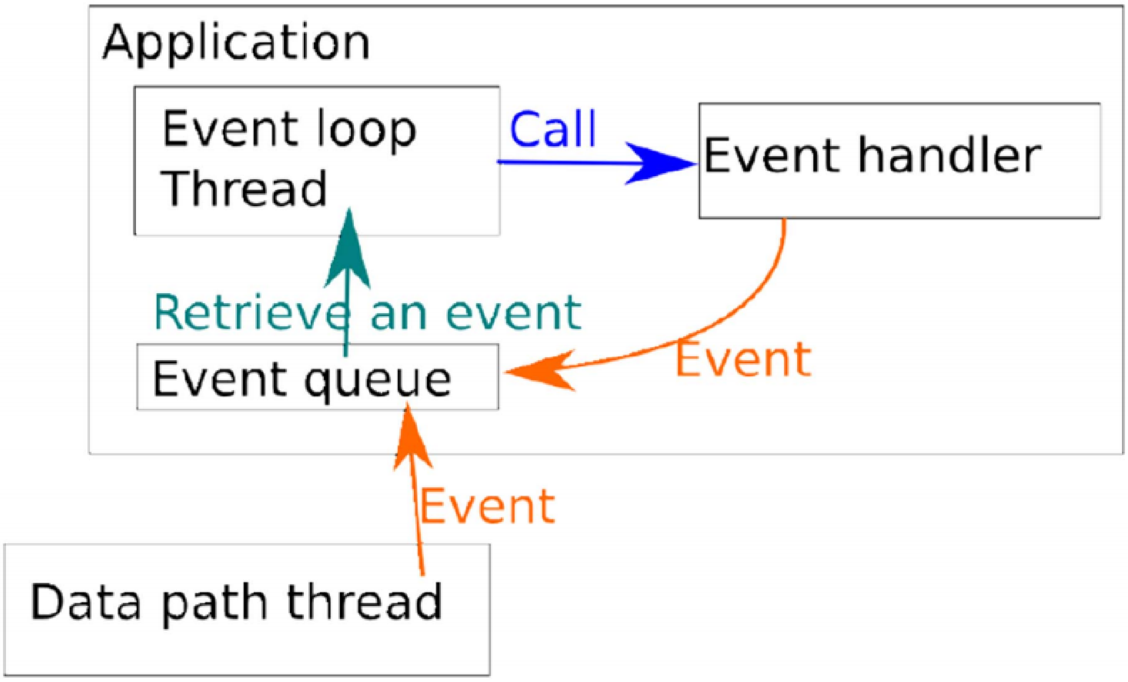


Figure 3.4: The four status in ryu controller

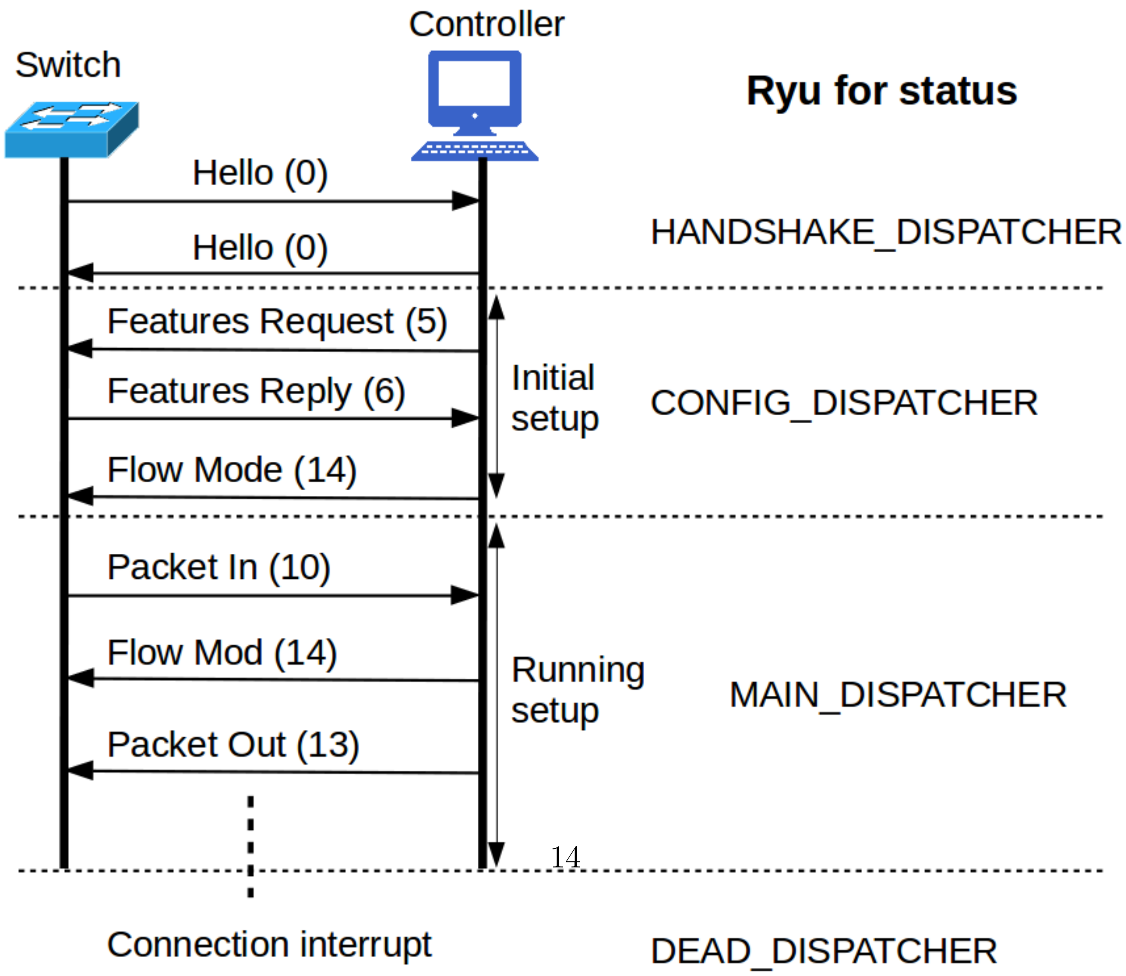


Figure 3.5: The example of the function which handle the packet

```
@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

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 Resouce specification (Rspec) documents. Figure 4.1 to Figure 4.3 show the three methdos 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.

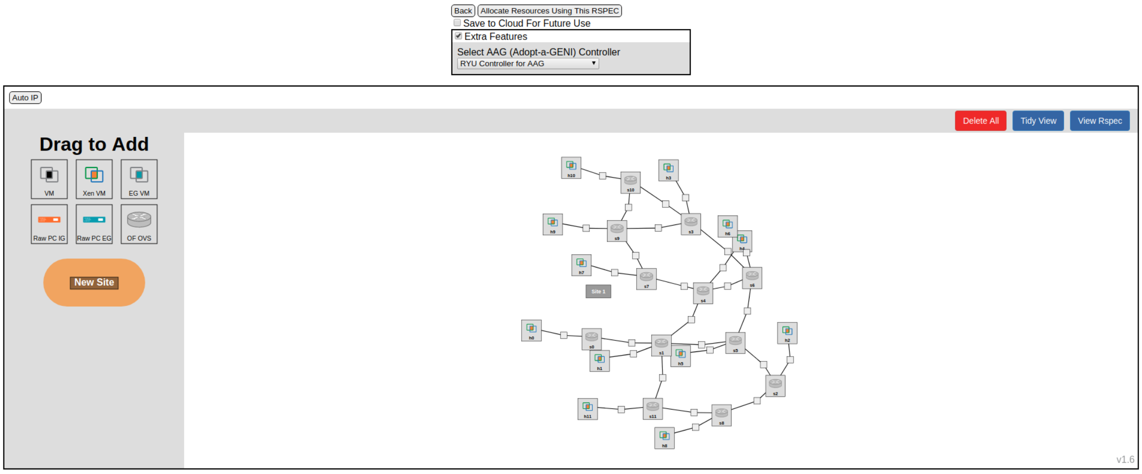


Figure 4.2: (b) Choose existing topology.

Figure 4.3: (c) Import Resouce 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"/>
     <property capacity="100000" dest_id="interface-0" source_id="interface-16"/>
     <site id="undefined"/>
     <link_attribute key="nomac_learning" value="yep"/>
   </link>
  +<link client_id="link-1"></link>
  +<link client_id="link-2"></link>
  +<link client_id="link-3"></link>
  +<link client_id="link-4"></link>
  +<link client_id="link-5"></link>
  +<link client_id="link-6"></link>
  +<link client_id="link-7"></link>
  +<link client_id="link-8"></link>
  +<link client_id="link-9"></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.protoge
  <node xmlns:emulab="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-OVS2.31"/>
    </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
      <ip address="10.10.1.1" netmask="255.255.255.0" 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
      <ip address="10.10.2.1" netmask="255.255.255.0" 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.ceni
      <ip address="10.10.3.1" netmask="255.255.255.0" 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
      <ip address="10.10.4.1" netmask="255.255.255.0" 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
      <ip address="10.10.5.1" netmask="255.255.255.0" 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.ceni
      <ip address="10.10.7.1" netmask="255.255.255.0" 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.ceni
      <ip address="10.10.6.1" netmask="255.255.255.0" type="ipv4"/>
    </interface>
    <sliver_type name="emulab-xen">
      <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops:UBUNTU14-OVS2.31"/>
    </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:GDXEN160218"/>
    </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
      <ip address="10.10.1.2" netmask="255.255.255.0" type="ipv4"/>
    </interface>
    <sliver_type name="emulab-xen">
```


Offline algorithm

create the network graph for the input of routing algorithm

After reserving the resources, copy the network structure document from GENI online website. Fugure 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 netowrk graph for the input of the routing algorithm.

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.

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.

Online algorithm

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. After the controller gets the topology, every time when a packet sends to the controller, we parse the information of the packet. Getting its ip source, ip destination, destination port as the demand's key. In-

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]
```

put this key and topology into the algorithm then get the path output by controller. Sending the messages to the switch to add the flow in the flow table.

Chapter 5

Case study

Maximum Concurrent Flow Problem in MPLS-based Software Defined Networks

In this section, we study the Maximum Concurrent Flow Problem in MPLS-based Software Defined Networks and implement four algorithms in GENI. Because the MPLS is a per-flow routing, the number of flow paths through a node (switch) needs to be bounded due to the limited ternary content addressable memory (TCAM). Therefore, the problem term to be bounded forwarding rule maximum concurrent flow problem (BFRMCF)

Problem definition

Given a network topology $G = (V, E)$ with link link capacity (bandwidth capacity) $c_e \in R^+$ associated with each directed link $e \in E$

and node capacity (forwarding table size) $b_v \in Z^+$ associated with each node (SDN-FE) $v \in V$, and given a set of flows $F = \{1, 2, ..., |F|\}$. Each flow $f \in F$ is associated with a source-destination pair $(s_f, t_f) \subseteq V \times V$ and the transmission demand $d_f \in R^+$. The Bounded Forwarding Rules Maximum Concurrent Flow Problem (BFRMCF) asks for the forwarding paths and the transmission rate on each chosen path such that 1) for each flow f, the rate is at least λd_f,

2) for each flow f, the rate is at most d_f (the demand constraints),

3) for each link e, the taotal rate through e are at most c_e (the link capacity constraints),

4) for each node v, the number of flow paths going through v does not exceed b_v (the node capacity constraints), and

5) λ is maxized.

Let P_f be the set of possible forwarding paths for flow f in the network, and $P = P_1 \cup P_2 \cup ... \cup P_F$. Moreover, let c_p be the capacity of a path $p \in P_f$, i.e., $c_p = min\{min_{e:e \in p}\{c_e\}, d_f\}$. In addition, variable $x\,(p)$ deontes the fraction of the transmission rate on path p to c_p, and variable $y\,(p) = 1$ if and only if p is a chosen forwarding path. The BFRMCF problem is formulated as an integer linear program as below.

$$
\begin{aligned}
\text{maximize} \quad & \lambda && \text{(1a)}\\
\text{subject to} \quad & \sum_{p \in P_f} x(p)c_p \geq \lambda d_f, \forall f \in F && \text{(1b)}\\
& \sum_{p \in P_f} x(p)c_p \leq d_f, \forall f \in F && \text{(1c)}\\
& \sum_{p:e \in p} x(p)c_p \leq c_e, \forall e \in E && \text{(1d)}\\
& \sum_{p:v \in p} y(p) \leq b_v, \forall v \in V && \text{(1e)}\\
& x(p) \leq y(p),\ \forall p \in \mathcal{P} && \text{(1f)}\\
& x(p) \in [0,1] && \text{(1g)}\\
& y(p) \in \{0,1\} && \text{(1h)}
\end{aligned}
$$

Algorithms of BFRMCF

We reivew three offline algorithms for BFRMCF problem. In [5], [6], [7], we implement it on GENI and show the total throughput and the λ in rxperiment results.

Implement method

The ryu controller

In ryu controller we need to parse the packet to get the source of ip, destination of ip and destination of port for pretending as a key. Figure 5.1 is the code in ryu for parsing the packet.

OpenFlw version and flow table size

In BFRMCF problem, because it has path degree constraint, we need to let the Open vSwitch support to 1.3 version and set up the flow limit. Figure 5.2, 5.3 is the code to set up.

iPerf

For setting a target bandwidth, because TCP couldn't fixed in a target bandwidth, so we selected UDP to transport. In figure 5.4, for server side, we user -p parameter to set up the server in a particular listen port, -u let the server listen for UDP packet.

For a client side, -c menas to be a client, -b is to set a target bandwidth. -p is transport to a particular listen port.

Figure 5.1: The code of parse packet

```
@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)
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(array.array('B',msg.data))
    print 'datapath_id %s'%(datapath.id)

    if pkt.get_protocol(ipv4):
        ipv4_pkt = pkt.get_protocol(ipv4)
        #print('src ip: %s, dst ip: %s'%(ipv4_pkt.src, ipv4_pkt.dst))

    if pkt.get_protocol(udp):
        udp_pkt = pkt.get_protocol(udp)
        #print('src port: %d, dst port: %d'%(udp_pkt.src_port, udp_pkt.dst_port))

    key = ('nw_src:{0}, nw_dst:{1}, tp_dst:{2}'.format(ipv4_pkt.src, ipv4_pkt.dst, udp_pkt.dst_port))
```

Figure 5.2: Support to OpenFlow 1.3

```
ovs-vsctl set bridge ovs-br protocols=OpenFlow13
```

Figure 5.3: Change flow table size

```
sudo ovs-vsctl add bridge s1 flow_tables 1=@nam1 -- --id=@nam1 create
flow_table flow_limit=1
```

Figure 5.4: The code of iPerf

```
server: iperf -s -u -p 5566

client: iperf -c 192.168.0.1 -b 10M -p 5566
```

Chapter 6

Experiment results

We tested three algorithms which are Garg's algorithm [6], Gushchin's algorithm [7] and PD's algorithm [5].The network instances were obtained from the 6 real-life traces from SNDlib [8]. Since the capacity of each node is unavailable, we set the switch capacity from the 0.1 number of demands($|K|$) to $1.0|K|$. The following figures show the minimum fraction and total throughput in different topologys.

Figure 6.1: The impact of the switch capacity on the minimum fraction on GENI.

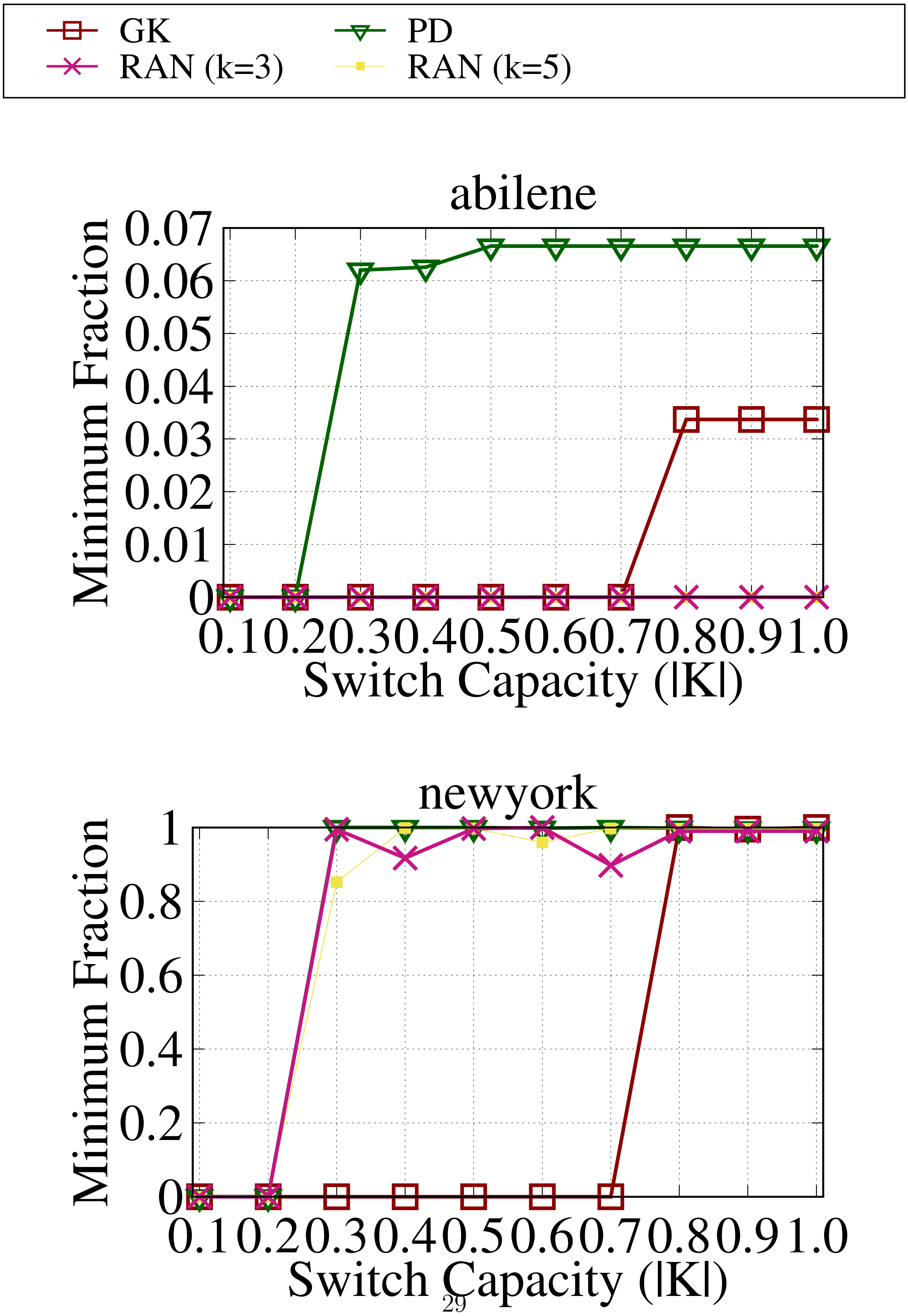


Figure 6.2: The impact of the switch capacity on the minimum fraction on GENI.

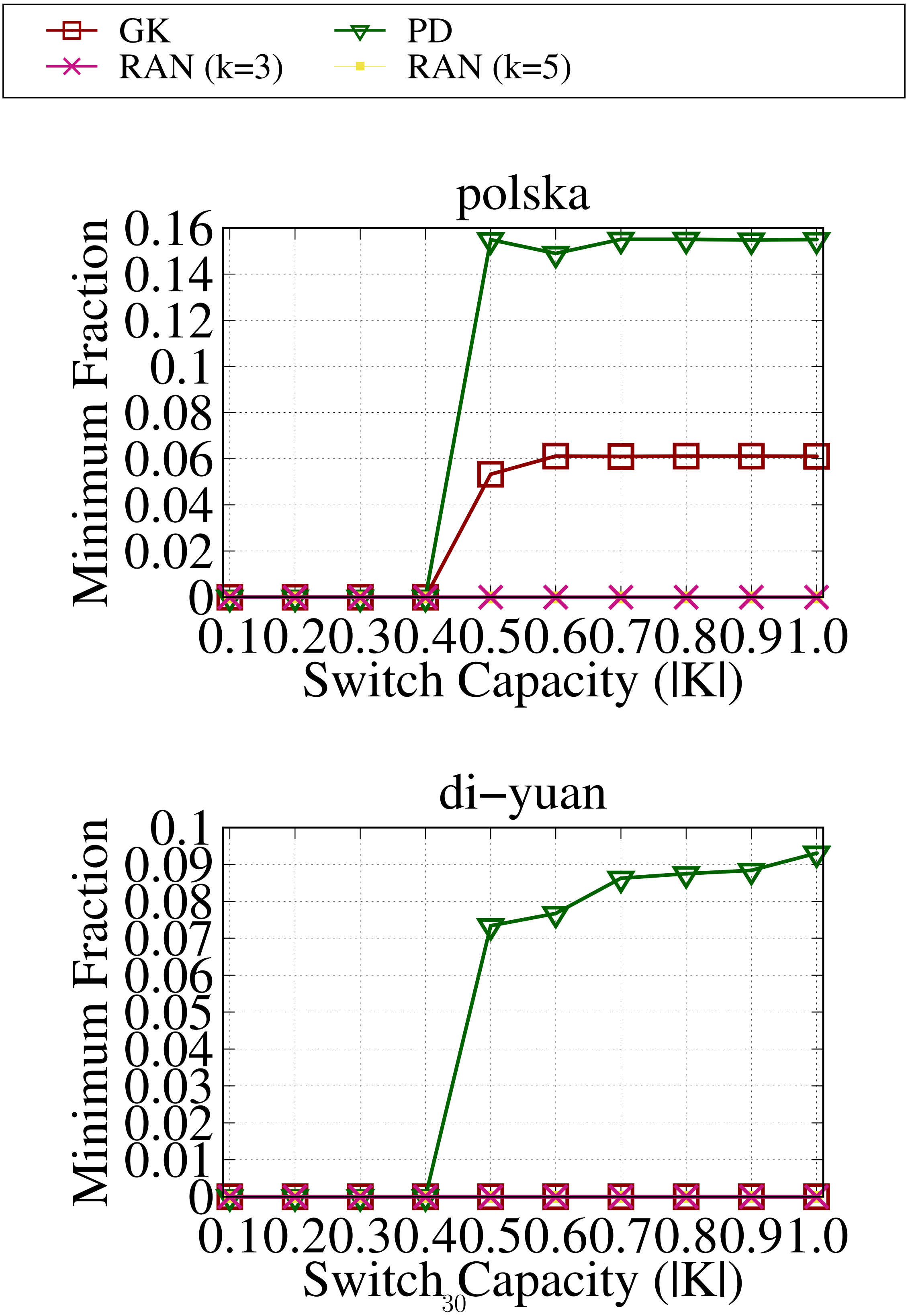


Figure 6.3: The impact of the switch capacity on the minimum fraction on GENI.

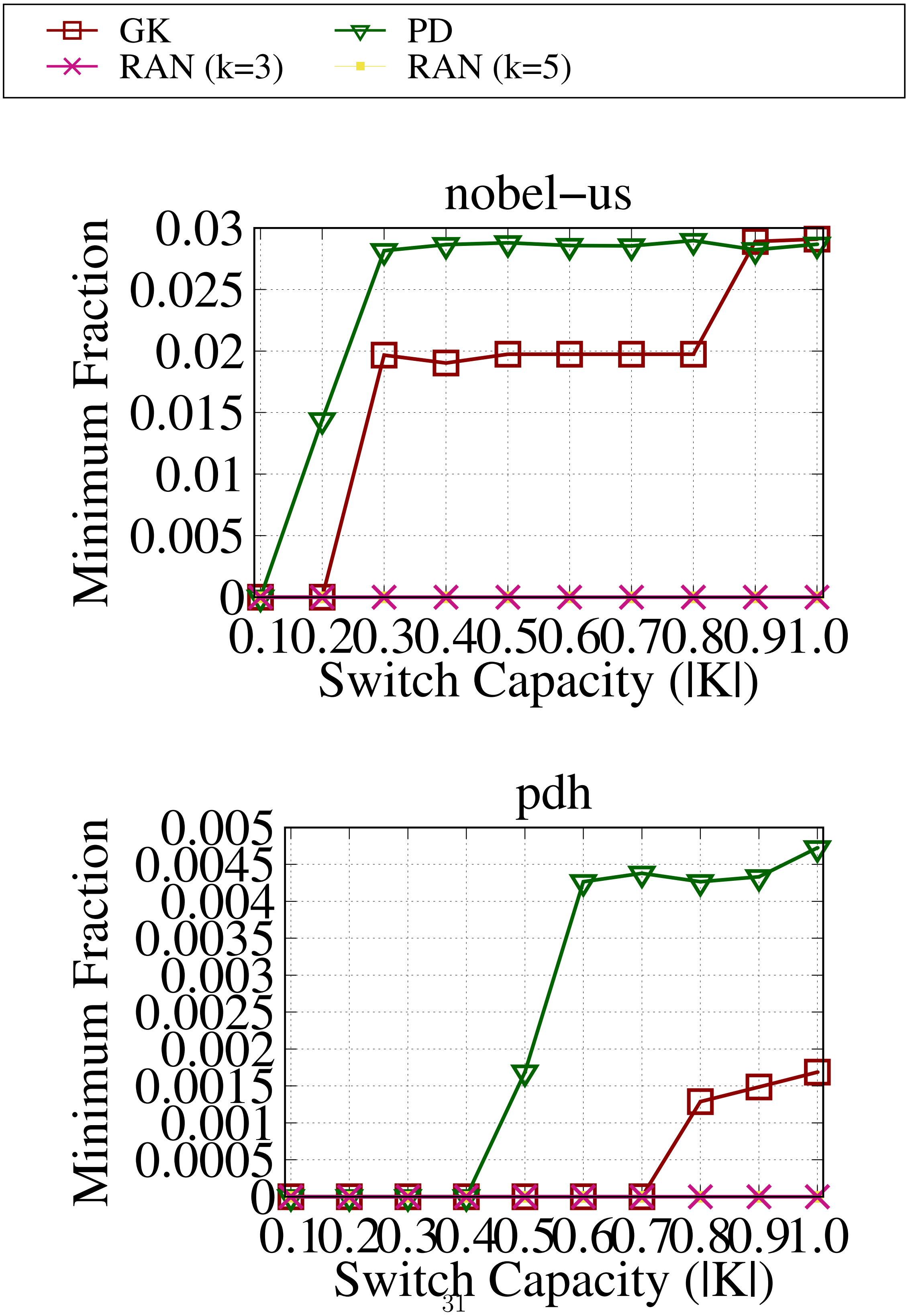


Figure 6.4: The impact of the switch capacity on the total throughput on GENI.

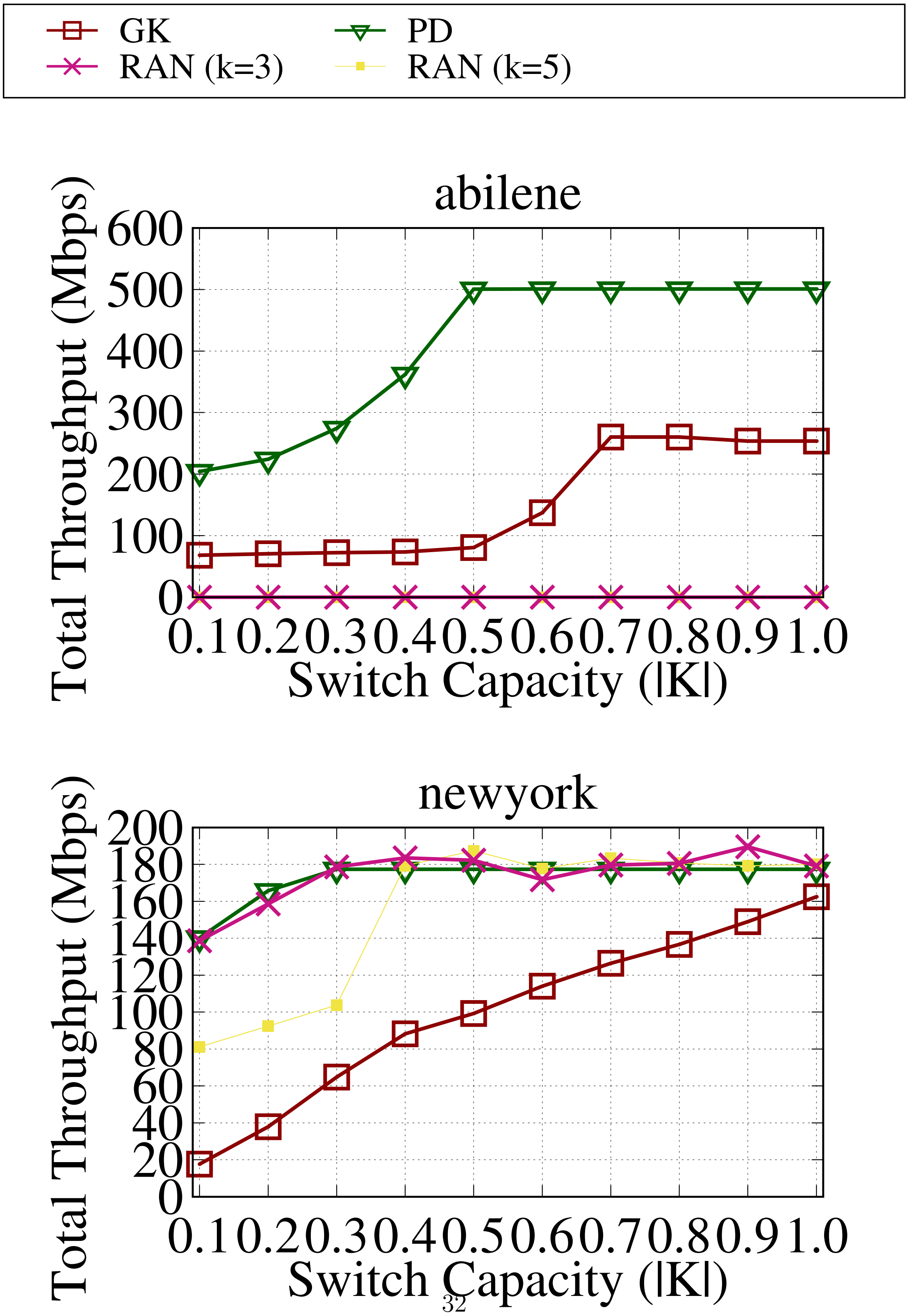


Figure 6.5: The impact of the switch capacity on the total throughput on GENI.

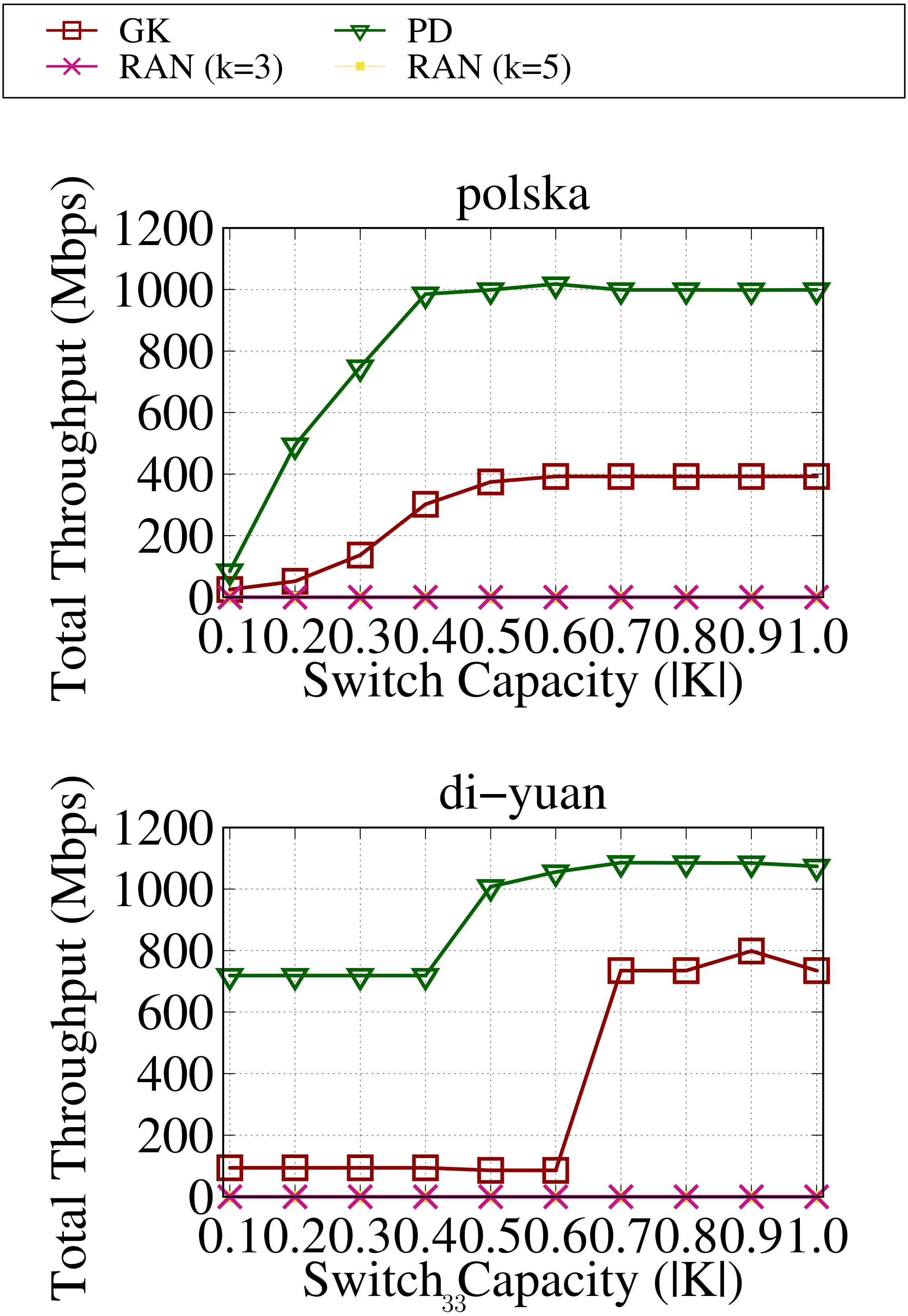
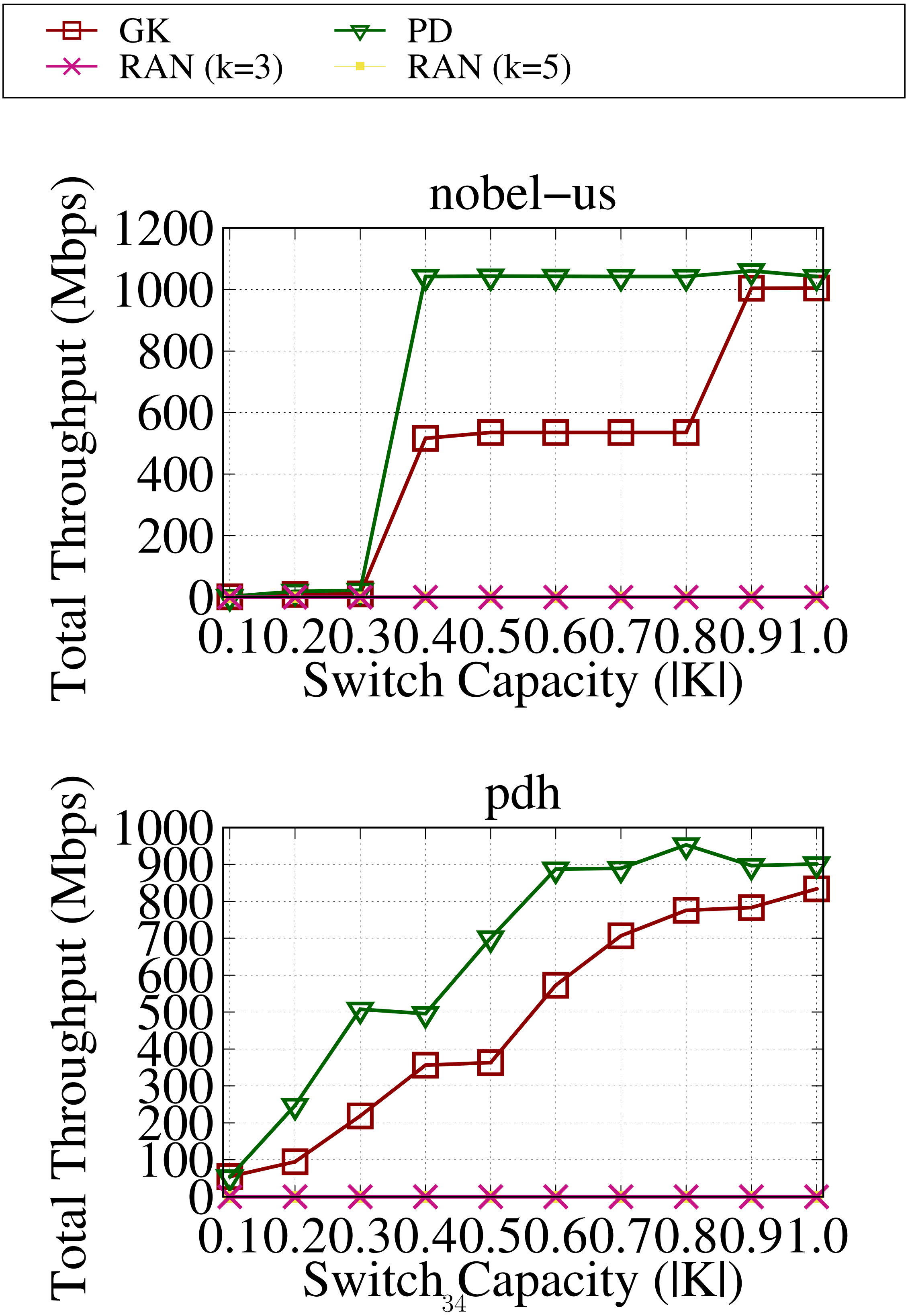


Figure 6.6: The impact of the switch capacity on the total throughput on GENI.



Chapter 7

Conclusion

For implementing a routing algorithm of SDN, GENI is a persuasive environment for experiment compared to simulator like NS-2, mininet, ... In this thesis, we divided the algorithms into online algorithm and offline algorithm. For each part, we gave a full description about how to implement it in GENI. Reader could base on this thesis to get a faster way for building their experiment.

Bibliography

[1] Open-Network-Foundation, https://www.opennetworking.org/.

[2] GENI, http://www.geni.net/.

[3] Open vSwitch, https://www.openvswitch.org/.

[4] iPerf, *https://iperf.fr/.*

[5] Tzu-Wen Chang, Yao-Jen Tang, Ming-Jer Tsai, *Efficient Algorithm for Maximum Concurrent Flow Problem in MPLS-based Software Defined Networks.*

[6] N. Garg and J. Könemann. *"Faster and simpler algorithms for multicom-modity flow and other fractional packing problems,"* SIAM Journal on Computing, 2007.

[7] A. Gushchin, A. Walid, and A. Tang, "Enabling service function chaining through routing optimization in software defined networks," in IEEE Allerton Conference on Communication, Control, and Computing, 2015.

[8] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, “Sndlib 1.0- survivable network design library,” Networks, 2010. [Online]. Available: http://sndlib.zib.de