Society for Science and Education
United Kingdom

TNC Transactions on Networks and Communications

Volume 3, Issue 5
ISSN: 2054 -7420

# Network Flexibility and Policy making in Software Defined Networks

**Abhinav Sharma and Manu Sood**
*Department of Computer Science, Himachal Pradesh University, Shimla*
aasvi2006@gmail.com, soodm_67@yahoo.com

**ABSTRACT**

Today's computer networks are complex which is very challenging to manage as well as these traditional networks struggle to scale to the requirements of some of today's environment. SDN gives solution to all these requirements with new dynamic networking features that enhance server value and user services. SDN supplements traditional networking by offering much flexibility and software centric control creating a more policy based process for adding intelligence into today's networks. Traditionally tweaking a policy/network configuration, network administrators typically rely on a combination of manual intervention and ad-hoc scripts. In this paper we have made an attempt to show how SDN is more robust and provides users flexibility to program the network according to their needs and requirements. We have used KINETIC – a domain specific language and SDN controller to write our priority based switching application.

***Keywords***: Network Flexibility, Green Networking, Policies, SDN, Northbound APIs

## 1 Introduction

SDN is a way to manage networks that separates control plane from the forwarding plane. SDN offers a centralized view of the network giving a SDN controller the ability to act as the brain of the network. The SDN controller relays information to switch and routers via southbound APIs and to the applications with Northbound APIs [2] [6].

SDN is an additive technology that enables network administrators to solve problems that are difficult to solve with traditional methods. The goal of SDN is to solve inefficiencies in existing networks by making them more automated, dynamic and easier to adjust to changing condition. SDN separates the control plane (which decides how to handle the traffic) from the data plane (which forwards traffic according to decisions that the control plane makes). Moreover, an SDN consolidates the control plane, so that a single software control program controls multiple data-plane elements [2] [6]. Evolution of SDN dates back to 1980s when AT&T developed a Network Control Point for telephone networks, which gave operators freedom to independent evolution of infrastructure, data and services. In 1990s came the concept of Active networks, where switch perform custom computations on packets. Active networks approach was quite similar to what we are seeing for SDN today but still this technology at that time didn't took off as at that time hardware support was not cheap and there was not the concept of data-centers [1] [5]. After this the major supporting technology for SDN comes in the form of Network Virtualization. The basis of

**DOI:** 10.14738/tnc.35.1517
**Publication Date:** 16th October 2015
**URL:** http://dx.doi.org/10.14738/tnc.35.1517

SDN is virtualization, which in its most simplistic form allows software to run separately from the underlying hardware. We can apply the idea of virtualization to the network as well, separating the function of traffic control from the network hardware, resulting in SDN.

## 2 A Southbound Interface for SDN: OpenFlow

OpenFlow is a communication interface defined between the controls and forwarding layers of SDN architecture, which allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual.

OpenFlow-based SDN technologies enable networks to address the high-bandwidth, dynamic nature of today's applications, adapts the network to ever-changing needs, and significantly reduces operations and management complexity [13].

OpenFlow uses the concept of flows to identify network traffic based on pre-defined match rules that can be statically or dynamically programmed by the SDN control software. Using OpenFlow, we can define how traffic should flow through network devices based on parameters such as usage patterns, application and cloud resources.

By removing the control-processing load from the switches, OpenFlow lets the switches focus on moving traffic as fast as possible. Moreover, by virtualizing network management, OpenFlow enables network that are less costly to construct and run [61]. It enables network operators to implement the features they want in software they control thereby promoting rapid service introduction through customization [13]. OpenFlow lets administrators prioritize different types of traffic and develop policies for how the network handles congestion and equipment problems.

OpenFlow Switches maintain a flow table containing flow entries consisting of a match condition, a list of forwarding actions, expiration settings and flow statistics as shown in fig below:

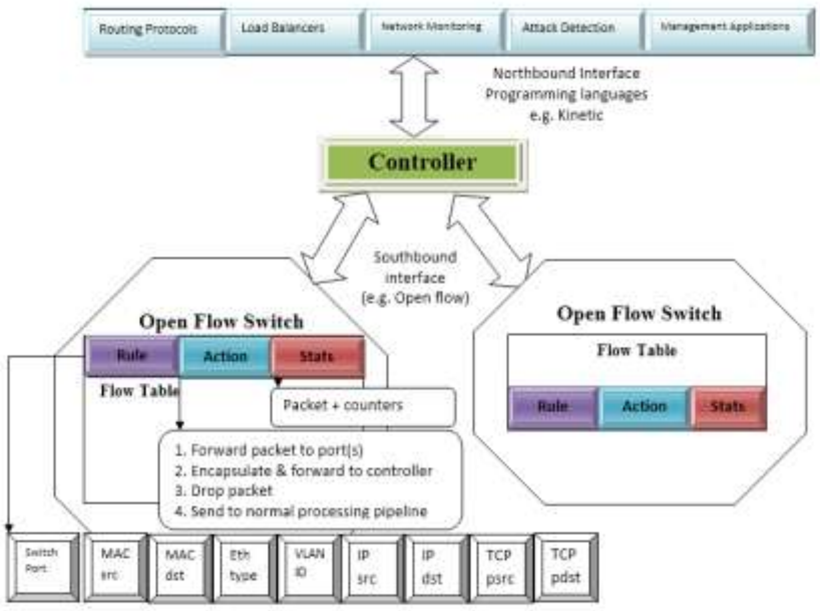


Figure 1 Open Flow-Environment: Flow-based Switching [13] [14] [15]

## 3 Towards Network Flexibility

### 3.1 Growth of Internet:

As per the estimates of internet world stats about 42.4% of the world's population uses the Internet [29]. According to ITU, almost half of the world's population will be online by the end of 2015.

There will be almost five billion things connected by the end of 2015 and three for every person on the planet – by the end of 2020.This implies Internet traffic is going to grow exponentially.

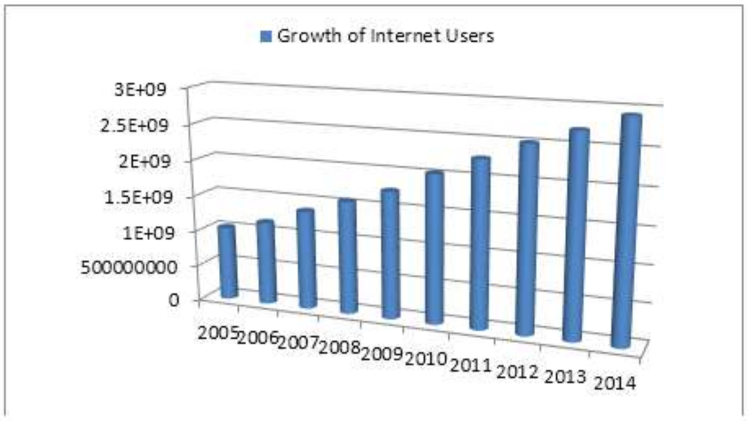


Figure 2: Growth of Internet Users [29]

The chart above shows how growth of internet users has increased in recent years. Here "Internet User" is an individual who can access the Internet, via computer or mobile device, within the home where the individual lives.

Thus network continue to increase and becoming more complex, yet configuring it remains primitive and error prone. Network operators need a dynamic and programmable approach to effectively and efficiently manage their networks to fulfil customer's requirements and assure the end-to-end service qualities.

### 3.2 Green Networking:

Second factor which is now becoming a major concern globally is energy consumption. Green Networking is the practice of selecting an energy-efficient networking technologies and products, and minimizing resource use whenever possible. Internet consumes between 107 and 307 GW and network devices consume around 7% of the total power consumed by Information and Communication Technology (ICT) [30] [31].

As shown in Figure 2, as the internet user base is growing exponentially, so there will be an increase in power consumption too.

In order to efficiently reduce the power consumption, there is a need of activity-adaptive internet architecture [32].

One other method to reduce energy consumption which in turn reduces operating costs of the network is to adopt technology which provides centralized management and monitoring of network and powered devices. We can make routers and switches more energy aware using network policies like link rate adaption during periods of low traffic and sleeping during no traffic. Software Defined Networking, with one of its main feature having centralized controller makes it easy to implement such policies (activity-adaptive) thereby decreasing energy consumption too.

**Table 1: Breakdown of power consumed by a router [32] [33]**

| | **Percentage of Total Power** | |
|---|---|---|
| | Single Chassis | Multi-chassis |
| Supply loss and blowers | 35% | 33% |
| Forwarding Engine | 33.5% | 32% |
| Switch fabric | 10% | 14.5% |
| **Control Plane** | **11%** | **10.5%** |
| I/O (O/E/O) | 7% | 6.5% |
| Buffers | 3.5% | 3.5% |
| **Total** | **100%** | **100%** |

SDN has enabled an increase in link utilization to almost 100% [12] which is hardly possible in traditional networks. Moreover as from the table (i) above 11% of total power consumption in traditional switches is because of control plane but SDN by decoupling the control plane from switches is making a direct reduction of 11% in total power consumption by switches/routers.

## 4 Network Policies in SDN environment

In [9] the authors state that one of the major drivers for SDN is simplification. In traditional networking, to express high level network policies network operators need to configure each individual network device separately using low-level and vendor specific commands. Moreover vertical integration of control and data planes in traditional network makes it hard to deploy new networking features like routing algorithm very difficult, since it would require modification of control plane of each individual device. Thus, new networking features are commonly introduced via expensive, specialized and hard to configure equipment such as load balancers or firewalls [2]. Today's network management policies are usually decided upon by the network operator and then configured once in each network element by an administrator. The larger the network, the greater the required configuration effort becomes. Hence a one set policy is seldom modified.

In contrast to traditional networking, SDN provides a better way in form of Northbound APIs to implement or develop new networking features. One of the main aspects of SDN is centralization of controller, which with global knowledge of the network simplifies the development of network functions, services and applications. Using a centralized controller and feeding rules in this centralized controller network device can be instructed to act like a router, repeater, switch, and firewall or perform other roles as per the need of networking environment [4].

Meter, match and act are the three steps SDN undertakes to execute tasks in a policy-driven network [23]. SDN enables the metering of traffic conditions, application and user behaviour to match those conditions against a set of pre-defined criteria and then to act on the match according to a policy. Part of a policy framework is to pre-set conditions that are metered against.

With the Northbound-API of the SDN controller, the application itself can inform the network about its properties and state. This way the network controller can direct traffic flows to complement rather than disrupt each other [24] [25].

## 4.1 Northbound APIs

The northbound API presents a network abstraction interface to the application and management systems at the top of the SDN stack [28]. The northbound and southbound interface allows a particular network component to communicate with higher-or lower-level network component, respectively.

Northbound APIs enable basic network functions like path computation, loop avoidance, routing, security, dynamic load management, bandwidth calendaring etc.

In traditional networks, all network applications must come directly from the equipment vendors, which make it hard to change the network features as per ones need. Using Northbound APIs provided by SDN architecture network administrators can quickly modify or customize their network. One of the main advantages of Northbound APIs is that it doesn't require one to dig into different data plane devices because it's abstracted and normalized by the controller through the northbound API. Northbound API puts applications in control of the network thereby eliminating the need of tweaking an adjusting infrastructure repeatedly to get an application or service running correctly.

## 4.2 Programming Languages for SDN

### 4.2.1 Frenetic Project:

Frenetic uses SQL like query language to control the information using a collection of high-level operators for classifying, filtering, transforming and aggregating the stream of packets traversing the network [34]. It makes use of primitive predicates and set-theoretic operators. Frenetic allows parallel and sequential composition of network policies. Frenetic run time system installs rules on switches using a reactive micro-flow based strategy.

Frenetic is embedded in Python. Frenetic provides a functional reactive combinatory library for describing high-level packet forwarding policies. It supports a see-every packet abstraction which guarantees that every packet is available for analysis.

### 4.2.2 Pyretic:

Pyretic is member of the Frenetic family of SDN programming languages. Pyretic helps programmers to focus on how to specify a network policy at a high level of abstraction, rather than how to implement it using low-level OpenFlow mechanisms [21].Using pyretic networking policies are specified for the entire network once rather than implementing that policy in individual switches.

***Some features of Pyretic [21]:***

1. Pyretic hides low level details by allowing programmers to express network policy as a function that takes a packet as input and return a set of new packets.
2. Pyretic allows programmer to write policies which matches packets based on Boolean predicates rather than bit patterns.
3. One of the main drawback before Pyretic was that how controller is going to do multiple tasks without interfering with other modules e.g. routing and server load balancing. To solve this Pyretic provides two composition operator viz. Sequential Composition (>>) and Parallel Composition (+).

#### 4.2.3 Nettle:

Nettle is based on the principle of functional reactive programming (FRP) which expresses languages as an electric circuit. In nettle, event based system is implemented declaratively where message streams is taken as a whole [36].

Nettle program work in terms of low-level OpenFlow concepts such as switch-level rules, priorities and timeouts. It allow composition of two independent modules but they are hard to implement and thus susceptible to network race conditions [35].

Nettle actually substitutes for the network controller i.e. Nettle is for general purpose programming of a network controller.

#### 4.2.4 Procera:

Procera is a controller architecture and high level network control language that allows operators to express network policies without resorting to general purpose programming of a network controller. In Procera, policy layer acts as a policy engine, which provides guidance and directives to the network controller and this layers sits on top of the network controller. Procera applies the principles of functional reactive programming (FRP). Procera is an embedded domain-specific language (EDSL) in Haskell [37].

#### 4.2.5 Hierarchical Flow Tables:

HFT allows high-level, network-wide policies that do not require knowledge of network topology [38]. Network policies in HFT are organized as trees of policy nodes which contain set of policy atoms. A policy atom is a (match, action) pair. HFT supports conflict resolution operators which are user-defined to resolve conflicting decisions. HFT translates policy trees to Network Flow Tables and uses Network Information Base to configure distributed network of switches. HFT also enables Participatory networking, in which end-users and their application propose changes to the network configuration [38]. HFT doesn't allow writing dynamic policies e.g. when topology changes or automatic reconfiguration. HFT have been used in PANE system.

#### 4.2.6 Corybantic:

Corybantic makes use of the concept of modularity, wherein different independent modules manage different aspects of the network. Corybantic represents the physical topology of the network as a graph of resources including switches and links and modules are expressed in terms of virtual subset topologies of the underlying network topology [39]. Modules are written in Python.

Corybantic uses two search approaches to avoid problem of local optima, one is inspired by search heuristics used in genetic algorithm while second approach is about carefully defining a convex objective function for different modules. Corybantic used a multi-phase iterative approach to constantly adapt to new customer demands.

#### 4.2.7 NetEgg:

Emphasis in NetEgg tools have been given to network operators who are actually not real programmers and thus they find it hard to program various network policies in Domain Specific Languages. NetEgg tool allows network operators to specify network policies using example behaviours [40]. In NetEgg, network operators specify policies using scenarios and it generates a policy table, multiple state tables and a controller program. NetEgg approach is based on synthesizing an implementation automatically from examples.

#### 4.2.8 Merlin:

Merlin provides a collection of high-level programming constructs for classifying packets, controlling forwarding paths, specifying packet-processing functions and provisioning bandwidth in terms of maximum limits and maximum guarantees [41]. Merlin allows dynamic modifications of policies using small run time components known as negotiation. Merlin uses regular expressions to specify the set of allowed forwarding paths through the network.

#### 4.2.9 Kinetic:

Kinetic is a domain specific language (DSL) and SDN controller that enables writing network control program that capture responses to changing network conditions in a concise, intuitive and verifiable language [22].

Kinetic represents network policies in terms of a Finite State Machine (FSM). Different states correspond to distinct networking behaviour.

Kinetic uses Computation Tree Logic (CTL) and has the ability to automatically verify policies with the NuSMV model checker. Both of these empower network administrators to verify the dynamic behaviour of the controller before the control program are ever run.

Kinetic is build on top of Pyretic, an SDN programming language embedded in Python.

## 5 Policy Implementation with Kinetic

We have written a policy based application for the above topology in SDN, where networking path followed by packets will be defined based on priority level. Through this example we have made an attempt to show how SDN is making network flexible and how easily one can write network policies as per one’s requirement.

When;

- Priority=1, path will be: h1->s1->s2->s3->s4->h2
- Priority=2, path will be: h1->s3->s4->h2
- Priority=3, path will be: h1->s4->h2

### 5.1 Topology Used

We have used two host, four switches and 1 controller topology for our kinetic application as shown in figure 3.

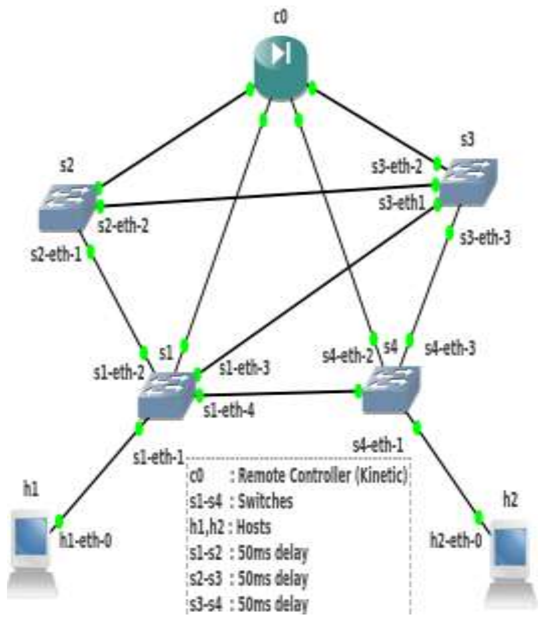


**Figure 3: Topology for Kinetic Example**

## 5.2 Ovs-ofctl-

Ovs-ofctl is the utility that comes with Openswitch and enables visibilty and control over a single switch's flow table. It is especially useful for debugging by viewing flow states and flow counters. e.g.

dump-flows will output the following:

```
PC:~$ sudo ovs-ofctl dump-flows s1

NXST_FLOW reply (xid=0x4):
```

i.e. it return an empty flow table. This is because we have not yet started any controller for our topology.

So, if we will run ping command in mininet:

```
mininet> pingall

*** Ping: testing ping reachability

h1 -> X

h2 -> X

*** Results: 100% dropped (0/2 received)
```

Using ovs-ofctl we can also add-flows manually for switches e.g.

```
PC:~$ sudo ovs-ofctl add-flow s1 in_port=1,actions=output:4

PC:~$ sudo ovs-ofctl add-flow s1 in_port=4,actions=output:1

PC:~$ sudo ovs-ofctl add-flow s4 in_port=2,actions=output:1

PC:~$ sudo ovs-ofctl add-flow s4 in_port=1,actions=output:2
```

```
PC:~$ sudo ovs-ofctl dump-flows s1

NXST_FLOW reply (xid=0x4):

cookie=0x0, duration=49.184s, table=0, n_packets=0, n_bytes=0, idle_age=49, in_port=1 actions=output:4

cookie=0x0, duration=37.332s, table=0, n_packets=30, n_bytes=6271, idle_age=0, in_port=4 actions=output:1

PC:~$ sudo ovs-ofctl dump-flows s4

NXST_FLOW reply (xid=0x4):

 cookie=0x0, duration=12.618s, table=0, n_packets=0, n_bytes=0, idle_age=12, in_port=1 actions=output:2

 cookie=0x0, duration=21.419s, table=0, n_packets=3, n_bytes=557, idle_age=4, in_port=2 actions=output:1
```

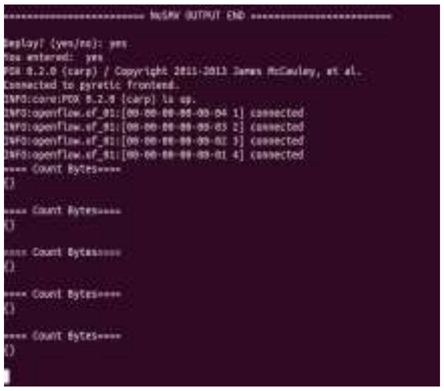
In the above we have manually added flows in switches s1 and s4 using ovs-ofctl as:

**h1**(inport=eth-0)->(inport=1)**s1**(outport=4)->(inport=2)**s4**(outport=1)->(inport=eth-0)**h2**

Now after adding flows when we will run ping command in mininet :

```
mininet> h1 ping -c 2 h2

PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.540 ms

64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.070 ms

--- 10.0.0.2 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 999ms

rtt min/avg/max/mdev = 0.070/0.305/0.540/0.235 ms

mininet>
```

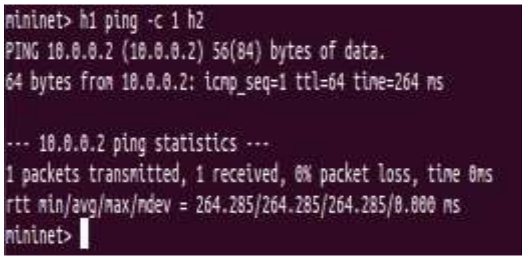
Now when we will start our priority application from the terminal, the controller will be up as show in fig below and we can ping host h1,h2.



```
======================== NetSMV OUTPUT END ========================

Deploy? (yes/no): yes
You entered:  yes
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-04 1] connected
INFO:openflow.of_01:[00-00-00-00-00-03 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
==== Count Bytes====
{}

==== Count Bytes====
{}

==== Count Bytes====
{}

==== Count Bytes====
{}

==== Count Bytes====
{}
```

Figure 4: Kinetic Controller running priority based switching application

By default policy_1 is running:

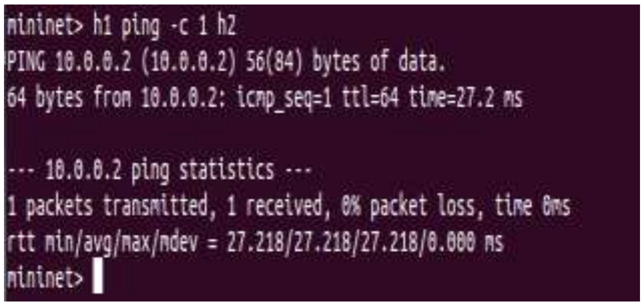


```
mininet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=264 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 264.285/264.285/264.285/0.000 ms
mininet>
```

Now to change data flow to higher priority level i.e. priority_3, we can use json_sender method to tell controller externally that now data has to go on high priority path.

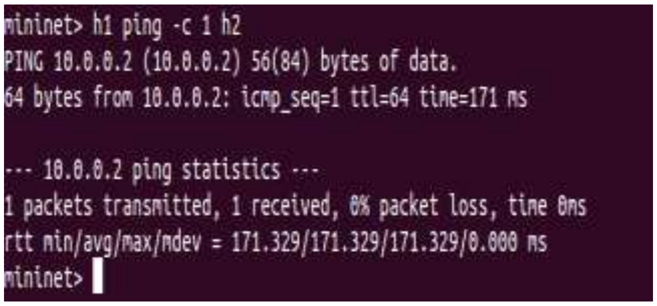
So we will send following event:

$python json_sender.py –n level –l 3 –flow=”{srcip=10.0.0.1,destip=10.0.0.2}” –a 127.0.0.1 –p 50001



```
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_seq=1 ttl=64 time=27.2 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 27.218/27.218/27.218/0.000 ms
mininet>
```

Similarly output for policy_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_seq=1 ttl=64 time=171 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 171.329/171.329/171.329/0.000 ms
mininet>
```

We have defined inter-switch functions as follows:

```
def interswitchs2():
        return if_(match(inport=2),fwd(1),fwd(2))
 def interswitchs3():
        match_switch  =intersection([match(inport=2),match
(inport=3)])
        return if_(match_switch,fwd(1),fwd(3))
```

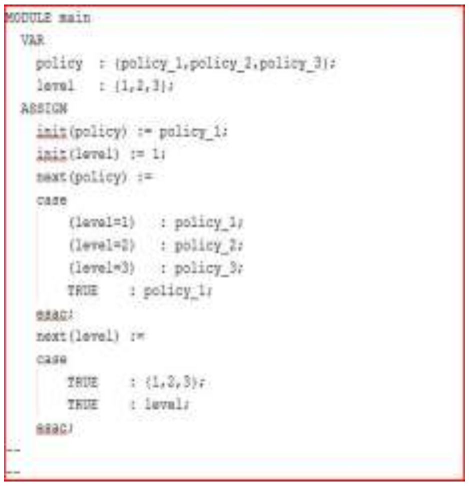
Here:

| **Syntax** | **Summary** |
|---|---|
| fwd(a) | modify (port=a) |
| identity | returns original packet |
| match(f=v) | that a field f matches an abstract value v e.g. match(inport=2) i.e. packets comes at inport 2 of the switch |

FSM description for our application is:

```
self.fsm_def = FSMDef(
          level=FSMVar(type=Type(int,set(levels)),
                    init=1,
                    trans=level),
policy=FSMVar(type=Type(Policy,set([level_rate_policy(i+1) for i in levels ])),
                     init=level_rate_policy(2),
                     trans=policy))
```

NuMSV FSM model for priority application is as shown below:



```
MODULE main
  VAR
    policy  : {policy_1,policy_2,policy_3};
    level   : {1,2,3};
  ASSIGN
    init(policy) := policy_1;
    init(level) := 1;
    next(policy) :=
    case
        (level=1)    : policy_1;
        (level=2)    : policy_2;
        (level=3)    : policy_3;
        TRUE     : policy_1;
    esac;
    next(level) :=
    case
        TRUE     : {1,2,3};
        TRUE     : level;
    esac;
--
--
```

Finite state diagram for priority based switching is as shown below:

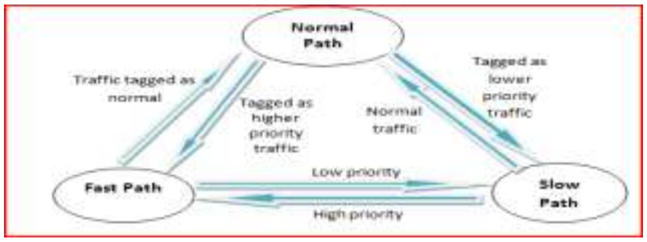


Figure 5 below shows how ping behavior varies as the priority levels for the host changes.

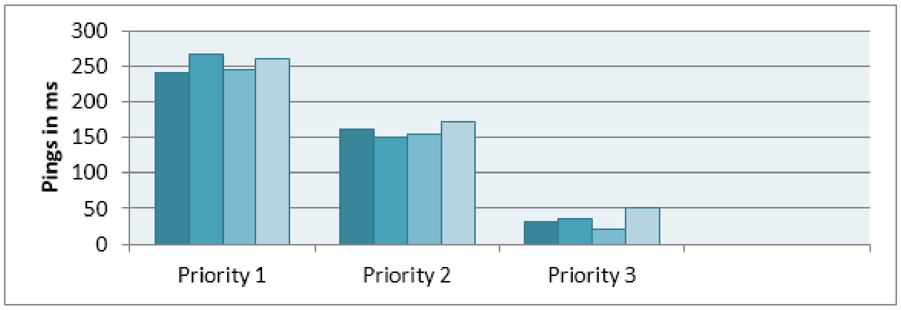


Figure 5: Ping behavior according to priority

## 6 SDN and Its Application Areas

### 6.1 SDN and Data Centers

The roots of SDN are in data centers. In general, SDN birth was mainly because of the needs of data centers. SDN has attracted many data centers operators towards it and Google has deployed SDN approach into one of its backbone WAN. The SDN deployed network has been in operation at Google and has offered benefits including higher resources utilization, faster failure handling and faster up gradation. Google has identified a number of benefits that are associated with its G-scale WAN backbone including that Google can run the network at utilization levels up to 95% [12][10].

### 6.2 SDN and Campus Networking

With new advent of Northbound APIs for SDN Controller like Kinetic, SDN has opened a new path to campus networking. After data centers, campus networking is the field where we require dynamic policy and network behaviour features. While with traditional network achieving a flexible network is a cumbersome process, SDN has made it an easy work. With a little line of codes, a network administrator can easily control the behaviour of network.

PROCERA a network control framework that helps operators express event-driven policies that react to various types of events [4]. In campus network, PRCOERA supports four control domains viz. Time (like peak traffic hours or session start), data usage (limiting data rate), Status (varying network speed, delay etc based on user priorities or groups) and flow (i.e. depending on to where data is going)

### 6.3 SDN, White-Fi and Rural Connectivity

#### 6.3.1 SDN from India’s Perspective

McKinsey Research into the internet economy has shown that internet contributes 3.4 % to GDP [44]. Further internet accounts for 21% of GDP growth over the last five years among developed countries. Moreover there was 10% increase in productivity for small and medium businesses from Internet Usage. According to 2011 Census of India, 833.5 million live in rural areas which is more than two-third of the total population and there are 111.76 million internet/ broadband subscribers[42][43]. Moreover rural internet subscribers stand at 12.89 per 100 populations. The data from the report shows that India still lacks in rural connectivity and McKinsey report [44] has revealed that how important internet is becoming for economic growth of nations. Rural areas are often ignored by network companies because the profit margins are small and it is difficult to update the required hardware [17].Step-aside infrastructure cost, the other hindrances’ in rural connectivity are network configuration, maintenance and poor road connectivity which makes it harder for ISPs to make rural internet a profitable business by implementing specific network policies for rural areas.

With the flexibility of SDN, internet can become more widespread than it already is. The main issues with rural areas include sparse population, small profit margins and resource constraints. However recent innovations like concept of White-Fi and SDN might help to alleviate these issues. SDN can reduce Capital expenditure (CapEx) as well as Operating expenditure (OpEx) by automating network functions, technology-agnostic connections, network aware applications and software-based functionality [4][8][17][45].

The separation of network construction and network configuration allows companies to decrease start-up costs in rural environments thus making rural networking a profitable business.

***White-Fi technology:***

1. White-Fi technology uses the unused spectrum in frequencies utilized for broadcasting of television signals and uses it for the internet. These unused spaces are called white spaces [26][27].
2. The 200-300 MHz spectrum in the white space can reach up to 10 km as compared to current Wi-Fi technology that allows a range of only about 100 meters.
3. It can be run on solar power and thus overcome a key hindrance that currently impedes ISPs namely the high cost of installation equipment.

Thus with the advent of White-Fi technology and flexibility of SDN, rural connectivity can become more profitable and more companies will be willing to give access to more and more rural areas.

## 7 Conclusion

With the exponential growth of Internet user base and need for green networking, network flexibility has become an important aspect of networking field. SDN has successfully managed to pave the way towards a next generation networking, which is flexible and provides network operators to implement various kinds of network policies in a simple programming fashion without digging into specific low-level details of network devices. SDN has appeared to be a success not only in data center networking or cloud networking but also in other fields like campus networking or rural networking etc. SDN is more cost-effective, more performance oriented as well as flexible. SDN gives network administrators freedom to implement policy driven mechanisms in campus or data centers as well as allows companies to decrease start-up costs in rural areas thereby making rural networking a profitable business. Moreover with the usage of White Fi technology as communication medium and SDN as network architecture, Rural Connectivity will become more economical feasible and can easily be optimized. With high level abstraction of Northbound APIs like Kinetic and Pyretic SDN has certainly change the policy making field which is rigid and vendor specific in traditional networks. This has also opened various opportunities for implementation of green networking. For example our priority based application has shown that how easily network administrator can implement network policy configurations as per requirement which is not possible in traditional networks.

## REFERENCES

[1] Feamster, Nick, Jennifer Rexford, and Ellen Zegura. "The road to sdn: an intellectual history of programmable networks." ACM SIGCOMM Computer Communication Review 44.2 (2014): 87-98.

[2] Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey. “Proceedings of the IEEE 103.1 (2015): 14-76.

[3] Open Networking Foundation. SDN Architecture Overview, Version 1.0 December 12,2013

[4] [4]. Kim, Hyojoon, and Nick Feamster. "Improving network management with software defined networking." Communications Magazine, IEEE 51.2 (2013): 114-119.

[5] Nunes, B., et al. "A survey of software-defined networking: Past, present, and future of programmable networks." (2014): 1-18.

[6] Jarraya, Yosr, Taous Madi, and Mourad Debbabi. "A survey and a layered taxonomy of software-defined networking." (2014): 1-1.

[7] Agarwal, Sugam, Murali Kodialam, and T. V. Lakshman. "Traffic engineering in software defined networks." INFOCOM, 2013 Proceedings IEEE. IEEE, 2013.

[8] Zinner, Thomas, et al. A Compass Through SDN Networks. Tech. Rep. 488, University of Würzburg, 2013.

[9] Shenker S. The future of networking and the past of protocols, Open Networking Summit, Palo Alto, CA, USA: Stanford University; October 2011

[10] Nirmalan Arumugam; The Thinking Network-Software Defined Networks will provide the intelligence the network needs to keep up in a cloud centric world. Available online: [www.wipro.com/documents/insights/the-thinking-network.pdf]

[11] Gautam Khetrapal and Saurabh Kumar Sharma; Demystifying Routing Services in Software-Defined Networking; Available online: [http://www.aricent.com/sites/default/files/pdfs/Aricent-Demystifying-Routing-Services-SDN-Whitepaper.pdf]

[12] Jain, Sushant, et al. "B4: Experience with a globally-deployed software defined WAN." ACM SIGCOMM Computer Communication Review. Vol. 43. No. 4. ACM, 2013.

[13] Open Networking Foundation Available online: [https://www.opennetworking.org/sdn-resources/openflow]

[14] Foundation, Open Networking. "Software-defined networking: The new norm for networks." ONF White Paper (2012).

[15] HP OpenFlow Protocol Overview; Technical Solution Guide Version: 1 September 2013.

[16] Specification, OpenFlow Switch. "Version 1.1. 0 Implemented (Wire Protocol 0x02)." Available online: [http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf]

[17] Hasan, Shaddi, et al. Enabling Rural Connectivity with SDN. Technical Report UCB/EECS-2012-201, University of California at Berkeley, 2012.

[18] Limoncelli, Thomas A. "Openflow: a radical new idea in networking." Queue10.6 (2012): 40.

[19] [Vaughan-Nichols, Steven J. "OpenFlow: The next generation of the network?"Computer 44.8 (2011): 13-15.

[20] Stallings, William. "Software-defined networks and openflow." The Internet Protocol Journal 16.1 (2013): 2-14.

[21] Reich, Joshua, et al. "Modular sdn programming with pyretic." Technical Reprot of USENIX (2013).

[22] Kim, Hyojoon, et al. "Kinetic: Verifiable Dynamic Network Control."

[23] Steve Garrsion Emerging Use Cases for SDN; February 2015. Available online: [https://www.sdxcentral.com/articles/contributed/emerging-use-cases-for-sdn-steve-garrison/2015/02/]

[24] Qazi, Zafar Ayyub, et al. "Application-awareness in SDN." ACM SIGCOMM Computer Communication Review. Vol. 43. No. 4. ACM, 2013.

[25] Jarschel, Michael, et al. "Sdn-based application-aware networking on the example of youtube video streaming." Software Defined Networks (EWSDN), 2013 Second European Workshop on. IEEE, 2013.

[26] http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11af-white-fi-tv-space.php

[27] Bahl, Paramvir, et al. "White space networking with wi-fi like connectivity." ACM SIGCOMM Computer Communication Review 39.4 (2009): 27-38.

[28] http://searchsdn.techtarget.com/guides/Northbound-API-guide-The-rise-of-the-network-applications

[29] Internet Live Stats http://www.internetlivestats.com/

[30] Raghavan, Barath, and Justin Ma. "The energy and emergy of the internet."Proceedings of the 10th ACM Workshop on Hot Topics in Networks. ACM, 2011.

[31] Kaup, Fabian, Sergej Melnikowitsch, and David Hausheer. "Measuring and modeling the power consumption of OpenFlow switches." Network and Service Management (CNSM), 2014 10th International Conference on. IEEE, 2014.

[32] Imaizumi, Hideaki, and Hiroyuki Morikawa. "Directions towards future green Internet." Towards Green Ict 9 (2010): 37.

[33] Baliga, J., et al. "Photonic switching and the energy bottleneck." Photonics in Switching. Vol. 2007. 2007.

[34] Foster, Nate, et al. "Languages for software-defined networks."Communications Magazine, IEEE 51.2 (2013): 128-134

[35] Foster, Nate, et al. "Frenetic: A network programming language." ACM SIGPLAN Notices. Vol. 46. No. 9. ACM, 2011.

[36] Voellmy, Andreas, Ashish Agarwal, and Paul Hudak. Nettle: Functional reactive programming for openflow networks. No. YALEU/DCS/RR-1431. YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE, 2010.

[37] Voellmy, Andreas, Hyojoon Kim, and Nick Feamster. "Procera: a language for high-level reactive network control." Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[38] Ferguson, Andrew D., et al. "Hierarchical policies for software defined networks." Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[39] Mogul, Jeffrey C., et al. "Corybantic: Towards the modular composition of sdn control programs." Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks. ACM, 2013.

[40] Yuan, Yifei, Rajeev Alur, and Boon Thau Loo. "NetEgg: Programming Network Policies by Examples." Proceedings of the 13th ACM Workshop on Hot Topics in Networks. ACM, 2014.

[41] Soulé, Robert, et al. "Merlin: A language for provisioning network resources."Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. ACM, 2014.

[42] Indian Telecom Services Performance Indicator Report for the Quarter ending March, 2015

[43] India Census 2011: Census of India: Census Data Online censusindia.gov.in/2011-common/censusdataonline.html

[44] Manyika, James, et al. Internet matters: The Net's sweeping impact on growth, jobs, and prosperity. McKinsey & Company, 2011.

[45] Fujitsu; Software-Defined Networking for the Utilities and Energy Sector Available online: [http://www.fujitsu.com/us/Images/SDN-for-Utilities.pdf]