ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

GUI Tool for Network Designing Using SDN

P.B.Arun Prasad, Varun, Vasu Dev, Sureshkumar

Assistant Professor Department of Computer Science and Engineering, Saranathan College of Engineering, Tamil Nadu, India

B.E. Final year Students, Department of Computer Science and Engineering, Saranathan College of Engineering, Tamil Nadu, India

ABSTRACT: In day to life internet is essential for all the fields, so number of internet users growing rapidly every day. The creation of network topology for n number of nodes in the network and performance monitoring is major issues for network administrators. The virtual network is one of the solutions for creating virtual topology and monitoring overall network, in this paper, the virtual network is created by using the GUI tool which is designed special for overall network management and it is supported by SDN. Software Defined Network (SDN) is a new approach to manage and maintain the network infrastructure. Mininet is the most popular emulator to understand and experience how SDN works. In GUI tool, node and switch can be created; it is connected with wired channel. The bandwidth and latency given as input to the nodes. The required topology can be created with nodes and maximum traffic can be induced in this tool to identify the performance of network, by this way of creating virtual networks to avoid the topology failures and improve the network performance. The final output as performance evaluation sheet is generated based on traffic occurrences in network; it gives which node is affected by maximum traffic, node failures, link failures, switch failures and topology failure is analysis and it can redesign the network based on the requirements. This GUI tool helps to avoid the failure of topology design.

KEYWORDS: Topology, GUI tool, SDN, Mininet, Switch, Traffic, Network Traffic.

I. INTRODUCTION TO SOFTWARE DEFINED NETWORKING

The Open Networking Foundation (ONF) is the group that is most associated with the development and standardization of SDN. According to the ONF1, “Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today’s applications. This architecture decouples the network control and forwarding functions enabling the network control 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.”
According to the ONF, the SDN architecture is:
Directly programmable: Network control is directly programmable because it is decoupled from forwarding functions.
Agile: Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
Centrally managed: Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
Programmatically configured: SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
Open standards-based and vendor-neutral: When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

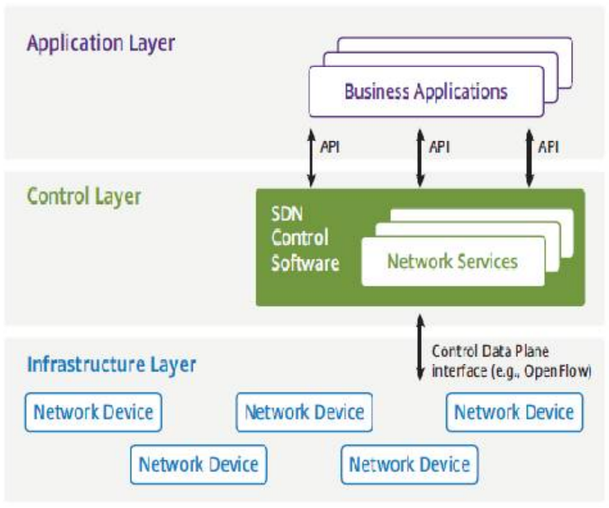


Fig 1.1: The SDN System Architecture

Below is a description of some of the key concepts that are part of the SDN system architecture that is shown in Figure 1.1

Business Applications: This refers to applications that are directly consumable by end users. Possibilities include video conferencing, supply chain management and customer relationship management.

Network & Security Services: This refers to functionality that enables business applications to perform efficiently and securely. Possibilities include a wide range of L4 – L7 functionality including ADCs, WOCs and security capabilities such as firewalls, IDS/IPS and DDoS protection.

Pure SDN Switch: In a pure SDN switch, all of the control functions of a traditional switch (i.e., routing protocols that are used to build forwarding information bases) are run in the central controller. The functionality in the switch is restricted entirely to the data plane.

Hybrid Switch: In a hybrid switch, SDN technologies and traditional switching protocols run simultaneously. A network manager can configure the SDN controller to discover and control certain traffic flows while traditional, distributed networking protocols continue to direct the rest of the traffic on the network.

Network Virtualization: Network virtualization isn't a new topic as network organizations have a long history implementing techniques such as virtual LANs (VLANs), virtual routing and forwarding (VRF) and virtual private networks (VPNs). However, throughout this white paper, the phrase network virtualization refers to the capability shown in the right half of Figure 1. 2. In particular, network virtualization refers to the ability to provide end-to-end networking that is abstracted away from the details of the underlying physical network in a manner similar to how server virtualization provides computer resources that are abstracted away from the details of the underlying x86 based servers.

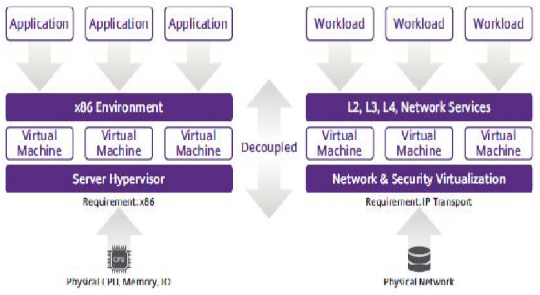


Fig 1.2 Network Virtualization

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

One way to implement network virtualization is as an application that runs on a SDN controller, leverages the Open Flow protocol and defines virtual networks based on policies that map flows to the appropriate virtual network using the L1-L4 portions of the header. This approach is often referred to as fabric-based network virtualization.

II. INTRODUCTION TO MININET TOOL

Mininet is a *network emulator*, or perhaps more precisely a *network emulation orchestration system*. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; you can ssh into it (if you start up sshd and bridge the network to your host) and run arbitrary programs (including anything that is installed on the underlying Linux system.) The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middle box, with a given amount of queuing. When two programs, like an iperf client and server, communicate through Mininet, the measured performance should match that of two (slower) native machines.
In short, Mininet's virtual hosts, switches, links, and controllers are the real thing – they are just created using software rather than hardware – and for the most part their behavior is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform.

A Working with mininet tool

Creating Topologies

Mininet supports *parametrized topologies*. With a few lines of Python code, you can create a flexible topology which can be configured based on the parameters you pass into it, and reused for multiple experiments.
For example, here is a simple network topology (based on mininet/topo.py:SingleSwitchTopo) which consists of a specified number of hosts (h1 through hN) connected to a single switch (s1):

B Setting Performance Parameters

In addition to basic behavioral networking, Mininet provides performance limiting and isolation features, through the CPULimitedHost and TCLink classes. There are multiple ways that these classes may be used, but one simple way is to specify them as the default host and link classes/constructors to Mininet(), and then to specify the appropriate parameters in the topology.
Important methods and parameters:
self.addHost(name, cpu=f):This allows you to specify a fraction of overall system CPU resources which will be allocated to the virtual host. self.addLink(node1, node2, bw=10, delay='5ms', max_queue_size=1000, loss=10, use_htb=True): adds a bidirectional link with bandwidth, delay and loss characteristics, with a maximum queue size of 1000 packets using the Hierarchical Token Bucket rate limiter and netem delay/loss emulator. The parameter bw is expressed as a number in Mbit; delay is expressed as a string with units in place (e.g. '5ms', '100us', '1s'); loss is expressed as a percentage (between 0 and 100); and max_queue_size is expressed in packets.

C Measuring Performance

These are recommended, though you're free to use any tool you're familiar with.

- Bandwidth (bwm-ng, ethstats)
- Latency (use ping)
- Queues (use tc included in monitor.py)
- TCP CWND statistics (tcp_probe, maybe we should add it to monitor.py)
- CPU usage (global: top, or per-container cpuacct)

D OpenFlow and Custom Routing

One of Mininet's most powerful and useful features is that it uses **Software Defined Networking**. Using the OpenFlow protocol and related tools, you can program switches to do almost anything you want with the packets that enter them. OpenFlow makes emulators like Mininet much more useful, since network system designs, including custom packet forwarding with OpenFlow, can easily be transferred to hardware OpenFlow switches for line-rate operation. A tutorial for creating a simple learning switch using Mininet and OpenFlow may be found at:

E Open Flow Controllers

If you run the mn command without specifying a controller, it will use the ovsc controller, ovs-controller, by default. This is equivalent to $ sudo mn --controller ovsc. This controller implements a simple Ethernet learning switch, and supports up to 16 individual switches.

If the Mininet() invoke, constructor in your script without specifying a controller class, by default it will use the Controller() class to create an instance of the Stanford/OpenFlow reference controller, controller. Like ovs-controller, it turns your switches into simple learning switches, but if you have installed controller using Mininet's install.sh -f script, the patched version of controller should support a large number of switches (up to 4096 in theory, but you'll probably max out your computing resources much earlier.) You can also select the reference controller for mn by specifying --controller ref.

If you want to use your own controller, you can easily create a custom subclass of Controller () and pass it into Mininet. An example can be seen in mininet.controller.NOX(), which invokes NOX classic with a set of modules passed in as options.

It's important to remember that Ethernet bridges (also known as learning switches) will flood packets that miss in their MAC tables. They will also flood broadcasts like ARP and DHCP requests. This means that if your network has loops or multiple paths in it, it **will not work** with the default ovs-controller and controller controllers, nor NOX's pyswitch, nor POX's l2_learning, which all act as learning switches/Ethernet bridges.

If you are building a fat-tree like topology, you may wish to take a look at RipLPOX, a basic datacenter controller implemented using POX. You may be able to use it as a starting point for your own custom multipath routing. You may also wish to implement a custom Controller() subclass to invoke RipLPOX for convenience.

III. OVERALL DESCRIPTION

A Traditional Network:

The following graphic is roughly how a network device functions. Network Devices have a control plane that provides information used to build a forwarding table. They also consist of a data plane that consults the forwarding table. The forwarding table is used by the network device to make a decision on where to send frames or packets entering the device. Both of these planes exist directly on the networking device.

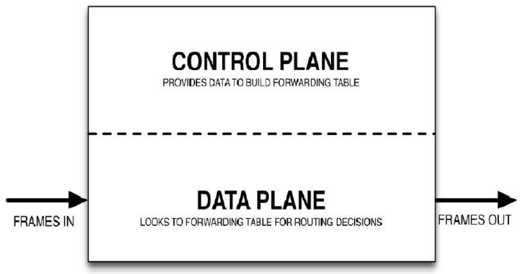

CONTROL PLANE
PROVIDES DATA TO BUILD FORWARDING TABLE
DATA PLANE
LOOKS TO FORWARDING TABLE FOR ROUTING DECISIONS
FRAMES IN
FRAMES OUT

Fig 3.1.1 Software Definied Network

B Software Defined Networking:

Software Defined Networking abstracts this concept, and places the Control Plane functions on an SDN controller Fig 3.1.1. The SDN controller can be a server running SDN software. The Controller communicates with a physical or virtual switch Data Plane through a protocol called OpenFlow. OpenFlow conveys the instructions to the data plane on how to forward data. The network device must run the OpenFlow protocol for this to be possible. What is the point of doing something like this? SDN creates a dynamic and flexible network architecture that can be change as the business requirements change. There are some additional benefits while using a centralized control and there are also benefits in the form of Network Automation. By using an API, controller can implement network commands to multiple devices without the need to learn the command line syntax of multiple vendor products. These are very few of the benefits of controlling network with SDN.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

IV. SYSTEM ENVIRONMENT

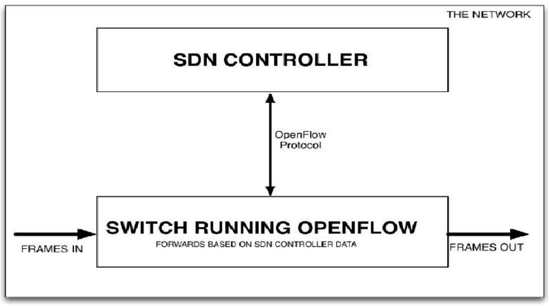


Fig 3.1.2 System Environments

A Product Functions

Our proposed project the following problems.

iii) Development of GUI for creating custom topologies in a user friendly manner without explicitly writing any programs. The users can even specify the different link parameters like Bandwidth, Latency etc. through GUI and create the network topology of their choice to test their innovative ideas and protocols.

iv) Our next focus is to collect the information from the SDN controller, preserve it for historical analysis and present via a GUI.

B Monitoring and Measurement

SDN provides the network and ability to perform in certain network monitoring operations for measurements without any additional equipment or overhead. The concept was introduced by Yu etal and is based on the fact that an SDN inherently collects information about the network to maintain a global network state at the logically centralized controller. This information can then be processed in software to obtain a subset of monitoring parameters. Furthermore, active measurements are enabled by selectively mirroring specific production traffic flows to the control plane or an external measurement device without the need of introducing artificial and potentially disruptive measurement probe traffic into the network. For example, by mirroring the traffic for a phone call at ingress and egress point of the network, the network administrator can determine the delay and quality of service for a particular call at a certain time.

C Network Management

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 once set policy is seldom modified. This leads to an often very inefficient network operation. The fact that traffic patterns continually change cannot be taken into account this way.

In order to change this, the network needs to be able to adapt policies dynamically and automatically based on a range of information. This calls for a more general specification of network policies that are subsequently translated into specific rules

for each device in the network using a policy engine. The logically centralized control plane of SDN offers itself as a very suitable way to enable such an approach as it has all information about the network available. For example, a high level network policy dictates the prioritization of VoIP traffic inside an Enterprise network. The SDN controller can then identify corresponding network flows and assign them to a high priority level in each device. This is dynamic on the one hand as VoIP flows are set up and terminated with each phone call and on the other hand it is automated as the devices are configured without the need for physical access and any human intervention. In fact the administrator does not have to know the topology of the network or the devices involved in order to achieve the policy's goal. Such an approach has been implemented prototypically by Kim et al.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

V. SYSTEM FEATURES

A The entire system consists of our scenarios.
Developing a GUI tool: By using Java, GUI Tool is developed for creating custom topology in user friendly manner in Fig 4.1.

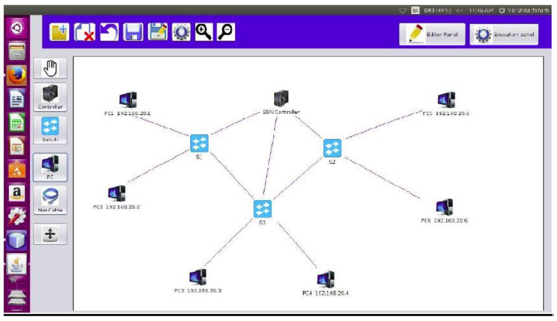
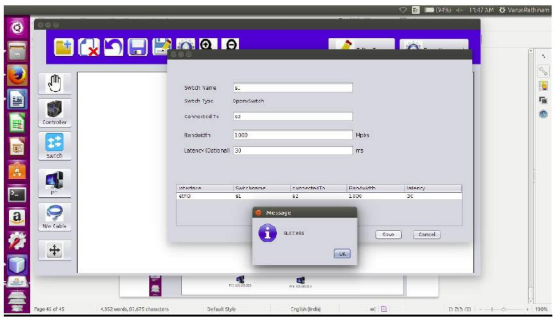


Fig: 4.1GUI Panel Design

B Creation of the Virtualization network: By running the mininet in Ubuntu, we are going to create a network by using several commands. By this way the virtualization can be achieved in fig 4.2.



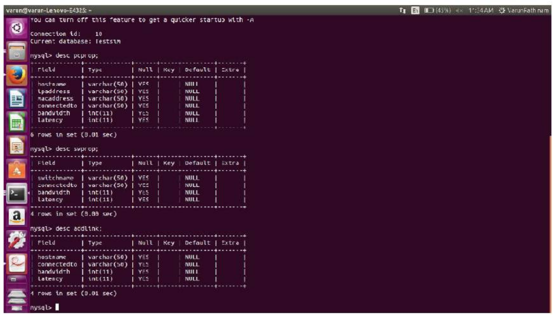
Swtch Name: s1
Swtch Type: OpenvSwitch
Connected To: s2
Bandwidth: 1000 Mpbs
Latency (Optional): 30 ms

interface	Switchname	connectedTo	Bandwidth	latency
eth0	s1	s2	1000	30

Message
SUCCESS

Fig: 4.2 Creation of Virtual network

C Data monitoring: The data and its information such as bandwidth, latency etc. are recorded in fig 4.3.



```
varun@varun-Lenovo-E4325: ~
You can turn off this feature to get a quicker startup with -A

Connection id:    10
Current database: Testsim

mysql> desc pcprop;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| hostname    | varchar(50) | YES  |     | NULL    |       |
| ipaddress   | varchar(50) | YES  |     | NULL    |       |
| macaddress  | varchar(50) | YES  |     | NULL    |       |
| connectedto | varchar(50) | YES  |     | NULL    |       |
| bandwidth   | int(11)     | YES  |     | NULL    |       |
| latency     | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)

mysql> desc swprop;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| switchname  | varchar(50) | YES  |     | NULL    |       |
| connectedto | varchar(50) | YES  |     | NULL    |       |
| bandwidth   | int(11)     | YES  |     | NULL    |       |
| latency     | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> desc addlink;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| hostname    | varchar(50) | YES  |     | NULL    |       |
| connectedto | varchar(50) | YES  |     | NULL    |       |
| bandwidth   | int(11)     | YES  |     | NULL    |       |
| latency     | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql> █
```

Fig: 4.3Database Maintenance

D Generating report: The historical and present information about the network are presented in the form of report and which can be used for the future analysis.

VI. CONCLUSION

The SDN controller and the Mininet emulator can be used to replicate the current day operational networks in a user friendly manner with the help of a GUI. The aggregated information about the network can be gathered for the analysis of the network resources. It is also possible to optimize the resource utilization by considering the application needs and the load on the network .SDN controller can be used to locate different flows in the network. By gathering the information about the response time, it can identify the paths that are heavily loaded. If a particular flow(path) is having more response time, the central controller can locate such paths and reallocate some of the flows to different paths which are lightly loaded and reduce the load on the network. This can be achieved by updating the forwarding tables of some of the switches by the controller which may help in improving the performance of the network.

REFERENCES

[1] Blueprint for Introducing Innovation into the Wireless Networks.Technical Report openflow-TR-1-4, 2009.
[2] G. Vijay Teja, Karishma Sureka, Sai Gopal and Ram P. Rustagi. Application Development in *SDN* - Reduction of Broadcast Traffic in Data Centers. Project Report, 2014.
[3] Msahil M, Pujolle G, Serhrenchni A, Fadlallah A, Guenane F. “Openflow and on demand Networks.” Third international conference on “Network of the Future”. Nov. 2012.
[4] Nick Mckeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Peterson, Scott Shenker, Jonathan Turner. "Openflow: Enabling Innovation in Campus Networks." ACM SIGGCOMM Computer Communication Review 38, number 2 (April 2008).
[5] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz and Nick McKeown. “Reproducible Network Experiments Using Container-Based emulation.” CoNEX*T.* Dec. 2012.
[6] P.Fonseca, R. Bennesby, E. Mota and A, Passito. "A replication component for resilient openflow-based networking." *NOMS.* 2012.
[7] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. “PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric.” SIGCOMM. Aug. 2009.