Volume: 6 Issue: 2 Pages: 2233-2239 (2014) ISSN : 0975-0290

Software Defined Networking: A Concept and Related Issues

Deepak Kumar
Department of Computer Science, Himachal Pradesh University, Shimla
Email: deepak.cs339@gmail.com
Manu Sood
Department of Computer Science, Himachal Pradesh University, Shimla
Email: soodm_67@yahoo.com

----------**ABSTRACT**----------

SDN (Software Defined Networking) is the networking architecture that has gained attention of researchers in recent past. It is the future of programmable networks. Traditional networks were very complex and difficult to manage. SDN is going to change this by offering a standard interface (OpenFlow) between the control plane and the networking devices (data plane). Its implementation is fully supported by software so that we can control the behavior of networking devices through programmatic control. This programmatic control provides various new ways to find breakpoints and failures in networking devices. Today SDN has become an important part of networking, so it is important to emulate its behavior. SDN support virtualization which makes it scalable and flexible. Data traffic resides in the data plane. The main function of intelligent controller is to decide the routing policy and manage the traffic in data plane. So effectively SDN emerges as a networking architecture that has the ability to solve all problems those were found in traditional architecture In this paper the authors discussed historical perspective of SDN, languages that support SDN, emulation tools, security issues with SDN and advantages that makes SDN suitable choice for today's network.

Keywords - **OpenFlow, Security, Software Defined Networks, Virtualization.**

Date of Submission: September 01, 2014 Date of Acceptance: October 10,2014

1. INTRODUCTION

The Software Defined Networking (SDN) is a new networking approach. The traditional networking architectures are not capable of fulfilling the changing needs and requirements of networking organizations and individual users. In traditional network it is very difficult to manage the network because network consists of heterogeneous hardware devices that support different protocols. SDN paradigm helps us to solve these problems by enabling the separation of the control plane from data plane. The concept of decoupling the control and data plane is not new, it has taken from early researches during early 2000[1].

SDN is a new emerging networking architecture and provides the platform for the development or innovation of new services. The advantage of separating the control plane is that we can develop the control plane in software and can change the functionality of the hardware devices whenever needed. Early network used the 4D [2] and ethane [3] approach for the network control and management. To make the scalable network the architecture must be able to provide local controller. The main function of the local controllers is to run the applications close to underlying switches [4].

To make any network scalable it must support virtualization and the SDN is a natural platform to support for virtualization [5]. We have various available controllers for SDN such as Beacon, NOX, POX, Onix, and Floodlight [6, 7, 8, 9, and 10]. The choice of particular controller depends upon the experience and knowledge of networks operators. SDN is fully programmable and whole intelligence lies in the centralized controller. There are number of available programming languages like Frenetic, FML, Procera, Flog and Pyretic to code for the intelligent controllers. The centralized intelligent controller is abstracted from the hardware devices that work on the data plane.

The main function of the centralized controller is to manage the hardware devices through programmatic control. Through controller we can modify the behavior of hardware devices by changing the logic code of centralized controller.

It is not possible for switches to pre-install all rules, because we use only one rule at a time and also switches do not have enough memory space. As controller decides particular routing policy, it implements that policy by instructing the switches to install that policy [12]. These rules and policies are based on the HFT (Hierarchical Flow Table) [11]. These policies help us to determine whether to forward the packet or to drop the packet by comparing the packet based on the values stored in the flow table.

Volume: 6 Issue: 2 Pages: 2233-2239 (2014) ISSN : 0975-0290

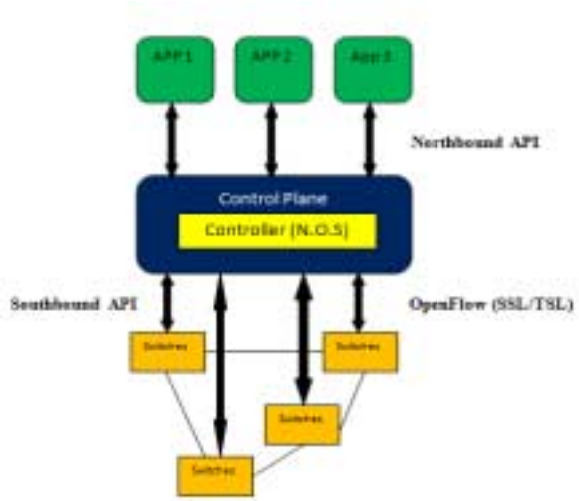

Figure 1: Software Defined Networking Architecture

There are also some issues regarding how many controllers are needed for network architecture to make it more scalable and flexible. OpenFlow [12] is important component of SDN architecture. OpenFlow is a protocol which enables the separation of control and data plane, which makes SDN more flexible. OpenFlow acts as an interface between the control and physical layer that allows the controller to configure the hardware devices programmatically. This paper includes Section 2 which describe how SDN evolves and its history. Section 3 includes virtualization and SDN. Section 4 describes languages for SDN to make it programmable. Section 5 describes scalability and SDN. Section 6 describes tools for SDN. Section 7 describes advantages of SDN. Section 8 describes issues with SDN. Finally we conclude in Section 9.

2. EVOLUTION OF SDN

The concept of programmable networks is not new, it resembles the early research. It has emerged from ideas that were used in early telephony networks. As network grows the traffic across networks also grows. In early traditional network it was difficult task to manage the network performance, reliability etc. So there were needs of approaches that can provide better network management (e.g. how to manage the traffic across each node in a network).

Also traditional network involves the coupling of control and data plane, which makes network management difficult. So it gives researchers an idea of separating the control and data plane, so that we can develop flexible, scalable, reliable networks. The same concept we have follows in today's SDN architecture. OpenFlow is a new approach which helps SDN to change the way how network were previously operated. In this section we discuss the previous networking researches that lead to scalable networking architecture known as SDN.

2.1. 4D Projects

It is an architecture which includes four planes: data, discovery, decision and dissemination [2]. It proposed giving the "decision" plane a centralized view of the network and provides network level objective in order to configure the forwarding plane devices. "Dissemination" provides the robust communication across each hardware devices in data plane. "Discovery" provides ability to each hardware devices to find out its own resources and local environment. "Data" plane for forwarding of packet across underlying hardware switches.

2.2. Active Networking

It emerged during 1990s when researchers introduced the new programming concepts in the traditional networks which enable greater chances of innovations [1]. Early network architecture used the programs such as Global Environment for Network Innovations (GENI) [13] and Future Internet Design (NSF FIND) [14].

2.3. Ethane

It allows the network operator to apply network wide policy. Ethane can be implemented in both hardware and software [3]. Ethane provides the communication between two end hosts by allowing the accurate permission. All policies studied in ethane are depends upon flows and header fields defines the flow based on the type of packet.

3. VIRTUALIZATION AND SDN

Virtualization is the key feature for the success of SDN. SDN is a natural platform to support virtualization. Network virtualization provides us the ability to create logical links and virtual networks that are separated from the hardware devices [15]. Today's networks are fully dependent on network virtualization and this is possible only through the decoupling of control and data plane. Network virtualization provides each tenant in networks with its own network topology and control over the flow of packet traffic [5].

Table 1: Controller that support OpenFlow standard

Serial No.	Developer	Controller	Implementation/Platform	Freely Available
1	Stanford	Beacon	java	yes
2	BigSwitch	FloodLight	java	yes
3	Nicira/Stanford	FlowVisor	C	yes
4	Rice University	Maestro	java	yes
5	Nicira	NOX	Python/C++	yes
6	Nicira	POX	Python	yes
7	CPQD	RouteFlow	C++	yes
8	Independent Developer	NodeFlow	JavaScript	yes

We have various controllers that support virtualization such as NOX, FlowN, Maestro [16], Beacon, POX, FloodLight and Onix. FlowN is built as upgraded version of NOX and it make SDN scalable by providing the illusion of running the separate controller for each host. Running separate controller for each host is not possible, it is very costly process. FlowN fully support the container

Volume: 6 Issue: 2 Pages: 2233-2239 (2014) ISSN : 0975-0290

based virtualization and the standard API allows the controller to provide interaction between the physical and virtual networks. SDN allows each tenant to run its own rules and policies on controller rather than hardware devices. The main focus point of network virtualization is to utilize the existing network resources such as several logical instances of networks which are composed of virtual nodes [17] [18]. FlowVisor [19] is one of most popular approach based on SDN to instantiate the virtual links and networks. FlowVisor has the ability to host many guest controllers i.e. one controller for each slice, each of which separated from each other. Each tenant that runs on controller contains many fields in packet header such as source IP address, destination IP address and port number etc. FlowVisor not fully utilizes the virtualization.

Table 2: SDN Controller Functionalities

Serial No.	Controller Functionalities	Benefits
1.	OpenFlow	**(1).**What version? **(2).** Which optional Features? **(3).** What vendors extensions?
2.	Support Virtualization	**(1).** Virtual network topology abstracted from physical network topology. **(2).** Ability to create flexible virtual network to meet wide variety of requirements.
3.	Network Functionality	**(1).** Keep virtual network totally isolated. **(2).** Enable to discover multiple ways and divides the traffic across different links.
4.	Scalability	**(1).** Provides the ability to add no. of devices, and the controller manages all underlying devices as a single device. **(2).** How many hardware devices can controller support. **(3)** The ability to create an SDN that span over multiple sites.
5.	Programmability	**(1).** Ability to redirect traffic-inbound traffic via firewalls. **(2).** Ability to apply sophisticated filters.
6.	Performance	**(1).** Pre-populate the flow table to the possible degree. **(2)** Minimize flow set up time.
7.	Reliability	**(1).** Multiple paths from source to destination. **(2).** Area of server cluster to drive reliability on the top of hardware-software in the control itself.
8.	Security	**(1).** Sophisticated filters keep virtual networks separate. **(2)** Support authentication of users. **(3).** Support security applications such as FRESCO, DDOS protection etc.
9.	Centralized monitoring & virtualization	**(1).** Monitor same class of traffic and not others. **(2).** Visualize the physical network and the virtual networks.

VeRTIGO [20] which is the extension of FlowVisor, it fully covers all the flavors of network virtualization. VeRTIGO interacts with controller and underlying hardware switches through control channel which includes Classifier, Node Virtualizer, Port Mapper, Internal Controller, Storage and VT Planner. Network virtualization is nearly comparable to VMware's virtualization. NVP (Network virtualization Platform) is the first network virtualization platform introduced by Nicira [21] in Feb, 2012. NVP provides us the platform that allows the extensible creation of the virtual network and that is fully abstracted from the hardware devices. In Table 1 we have various controllers from different developers that work on different platform platforms and most of them are open source. In Table 2 we have shown the various controller functionality and their benefits.

4. LANGUAGES FOR SDN

Conventional networks were built of collection of heterogeneous hardware devices, each of which running distributed algorithm and protocol that possess access rules and policies, topology information, routing policies and traffic monitoring services etc. It is very difficult task for the network operators to provide communication among them. Whenever we want to add or remove any device from network, we have to change whole setup and it is very time consuming process.

As traditional networks consisted of coupled control and data plane which limits the flexibility and scalability. With the advent of new networking architecture, i.e. SDN, it provides the platform to develop the programmable interface on which application runs. SDN makes it possible for programmers to control the behavior of the abstracted hardware devices through programmatic control.

Most of the controller such as NOX [7], POX [8], Maestro [16], and Beacon [6] provides the programmable interface that enables the programs to react to network events such arrival of packet, drop of packet, and link status update etc. Early network protocols and overlay networks [22] use logic programming like NDlog [23], Overlog [24].

One of the main logic programming for controller is FML (Flow Management language) [22]. It provides rule based formalism for OpenFlow networks [28]. FML is a datalog based and its main focus is to write rules and policies. It is a domain specific high level language. It consists of various built in rules and policies that are responsible for making decision allow or deny the certain flow of packet. Another language FRENETIC [25] is a high level language for programming heterogeneous collection of network hardware devices.

FRENETIC is a combination of declarative query language, functional stream language, specification language [22]. Pyretic [26] is also new language that allows the programmers to specify networks rules and policies at a high level of abstraction. The main component of Pyretic is the abstract packet model. Flow-Log is a programming language for SDN controllers, it is declarative language. FlowLog [27] programs are simply a collection of concrete table that are comparable to relational database. Flow-Log is refinements of datalog. Flow-Log programs can access the current state of the controller and also have ability to make continuous modification to the packet field. Flow-Log programs are non-recursive datalog.

5. SCALABILITY AND SDN

In SDN the centralized controller provides the global view of the network and has direct control over the hardware devices in data plane. As we know network consists of the underlying switches and the centralized controller. The main function of the switch is to forward the packet based on the rules and policies stored in the flow tables. Centralized controller controls the underlying switches by setting up the rules and policy. Packet processing rules and policies are installed in hardware devices in two ways: one

Volume: 6 Issue: 2 Pages: 2233-2239 (2014) ISSN : 0975-0290

way is that switches decide their routes themselves (known as reactive) and other way, when first packet comes across ingress port. Ingress port send packet to the controller and then controller decides which route switches have to follow (known as proactive). Configuring the hardware devices through programmatic control makes the SDN flexible. But question comes, how we can make SDN scalable.

We have various option of controller like NOX, FlowN, FlowVisor and Beacon etc. One of most commonly used controller is NOX. All of these controllers provide the programmable interface that supports a low-level and event based model in which programs control the task of arrival of packet, to drop the packet, or to forward the packet based on the decision made by the centralized controller. To make control plane scalable we must have better choice of controller such as ElastiCon [28]. Early controllers support the static mapping between the forwarding hardware devices and the intelligent controller where as ElastiCon support the dynamic mapping between the underlying switches and the controller.

We can make SDN scalable by keeping the packet traffic in the data plane [29] that is abstracted from the controller in control plane. In order to efficiently handle the traffic flow, the switches must be able to take decision where to forward the packet. This is possible by having firmware installed in switches with silicon chips and inbuilt buffer storage. These switches are smart enough for making any decisions. When a packet arrives across switch in a network, the rules installed in the switches firmware tell the switch where to forward the packet. These switches are based on ASIC's (Application Specific Integrated Circuits). Firmware is inbuilt software (set of programs) in switches which provides the ability to communicate with centralized controller.
These switches also contain CPU to handle the data plane traffic [30]. The CPU contains the complete forwarding tables. Also other factor that makes SDN scalable is the ability of faster failure recovery. The programming language that support fast failure recovery is FatTire [31]. The most common and basic requirement of any network is the ability to tolerate and recover from failure. Same requirements are needed for SDN. In SDN we automatically recover from failure through centralized controller. Whenever the failure take place or switch stops working SDN is capable of changing the flow of traffic through programmatic control. To recover from failure in seconds the network must support multiple routing paths and must also be able to work with any kind of networking topology [32].

6. TOOLS FOR SDN

SDN is a new paradigm that facilitates the development of networks. In this section we provide introduction to SDN based tools such as emulators.

6.1. Mininet

Mininet [33] is an emulator that allows network to be emulated on single computer system. It configures and runs the same things that we find on real network such as links, switches, servers and packets. The emulated server runs logic codes rather than events. Emulator provides the artificial environment with artificial traffic comparable to the real network environment. Mininet is container based emulator (CBE) and support process level virtualization, which is lighter form of virtualization. Other examples of container based emulators are Netkit [34], CORE [35] and trellis [36].

6.2. DieCast

DieCast [39] is very closer to the behavior of actual network and it uses very less no. of physical resources. Large network includes thousands of devices runs heterogeneous protocols and software configurations that are distributed across hundred of physical networks. DieCast does not focus on scaling of hardware resources like main memory and disk etc.

6.3. ModelNet

ModelNet [37] is a scalable emulator that provides the emulation of the network at a large scale. The main point that makes ModelNet different from other emulator, it focuses on the emulation of large topology of networks. In ModelNet we require lesser resources to emulate network [38]. The ModelNet includes five phases: Create, Distill, Assign, Bind and run. The emulation in ModelNet is based on real time therefore each packet come across the emulated network with equal time interval, with same delay and same rate of loss of packet as the actual network.

7. ADVANTAGES OF SDN

Early networks were not built to meet the today's rapidly changing requirements. Everything is evolving except the networking architecture. SDN has the ability to overcome the problems of conventional networking architecture. SDN networks are easily scalable as network grows because SDN provides us with the programmable interface. Networking hardware devices are increasing day-by-day that results in network growth, so for traditional network it is very tedious and costly task to configure all hardware devices manually. SDN is fully implemented in software and therefore it support programmable control to configure the hardware devices. Following are some of advantages of SDN:

7.1. Global View

SDN provides the global view of the network that helps us to simplify the network configuration, management and makes it available to the user who wants to use it.

7.2. Flexible Traffic

The main functionality of the intelligent controller is to programmatically configure hardware devices which makes the SDN flexible and more agile. Also we can change the functionality of controller according to change in traffic.

7.3. Easily Programmable

In SDN, control plane is easily programmable because it is separated from the data plane. Whenever we have to make

Volume: 6 Issue: 2 Pages: 2233-2239 (2014) ISSN : 0975-0290

changes, we can change the functionality of control plane by changing the logic programming in control plane.

7.4. Programmatically Configurable

As SDN is fully implemented in software and it therefore configures the hardware devices through programmatic control. When the controller decides to implement particular policy, it instructs the hardware devices to install that packet forwarding policy through programmatic control.

7.5. Faster Failure Recovery

SDN support virtualization which provides end-to-end connectivity between links and node across network. So failure handling is much faster, the controller automatically checks for failure and handle them accurately.

7.6. Platform for Innovation

As SDN is a new networking paradigm, the separation of data and control plane which enables the chances for innovation of new applications and services.

8. RESEARCH RELATED ISSUES WITH SDN

SDN is a new concept in networking field. As SDN has advantages, there are also some issues. SDN consists of three layers where each layer requires its own level of protection. Centralized controller acts as the standard interface for the interaction between the applications and switches that works on the top and bottom layer of SDN architecture. The bottom layer includes network virtual device which is the emulation of physical switches. So we need some mechanism to secure the controller and network virtual device [40]. Also emerging networks are facing the problem of how to attain security to protect the enterprise as well as user data. The concept of virtualization has enabled new threats that must be addressed. Therefore the authors next presents account of the issues related to the security of SDN. They have segregated these issues into three categories as detailed below.

8.1. Virtualization Security Issue

Networking industries are stepping forward to support for virtualized applications with different risk factors on a single server. In virtualized networks packet traffic does not flow through physical network [41]. There are no obvious ways to manage or control the data traffic in virtualized networks. There are various services like firewalls, IDS, QOS etc. These services are of no use in virtualized networks. If the attacker in the one virtual network is able to detect the other virtual networks then it must destroy the illusion of separation [41]. The weaker network security makes the virtual network vulnerable. In order to prevent network from an attacker, we have some of solutions as given below.

Solution:

The layer and boundaries that manages and control the virtual network must be well secured. So we need some additional security for virtual networks like virtualized firewall which includes IPS, anti-malware, URL filtering and content blocking to control the threats. In SDN we can make virtual machine secure by providing additional security layer to the hypervisor. Some following schemes can be used to make virtual networks secure.

8.1.1. Worm Detection

Worms are serious threat to the networking infrastructure. They spread across network from one host to another in continuous manner. There are some techniques for the detection of worm and defend against worms. For worm detection we use the behavioral approach, automatic detection of worm is very challenging because it is difficult to predict what would be the next form of worm [42]. Another approach for the worm detection is the multi resolution approach [43] and threshold based mechanisms.

8.1.2. Intruder Detection

One of most important part of network security is the NIDS (Network Intrusion Detection System). It provide us the layer which help us to monitor the data traffic and identify the suspicious activity, whenever found it instruct the administrator to take some suitable action. SNORT [44] is a packet sniffer that is used for NIDS. SNORT helps to identify various attacks like buffer overflows, CGI attacks and Server Message Block probe etc.

8.2. Controller Security Issue

Control plane controller has offered several benefits to configure the networks. But there also some security issues related with controller. SDN centralized controller potentially have risks and threat compared to the conventional network architecture. What if an attacker tries to modify or hack the hardware devices in data plane? SDN is fully dependent on the intelligent controller, so we need some security mechanism to protect it from attacker. The southbound interface between the control plane and data plane is sensitive and without any security services, it affect the performance and integrity of the network [45]. So without security network becomes inefficient.

Solution:

Whenever attacker tries to change the functionality of underlying hardware devices or tries to capture information from the data packet, the centralized controller can restrict attacker by modifying the data path. We can make SDN controller secure using following security approaches:

8.2.1. FortNOX

FortNOX is the policy enforcement kernel. FortNOX [47] extends of NOX [7] controller by providing policy based flow rule enforcement. FortNOX support role based OpenFlow applications authentication through digital signatures. FortNOX enables NOX [7] to check flow rule contradiction in real time. It also uses the conflict resolution policy to accept or discard the rules.

8.2.2. FRESCO

FRESCO is a new security application development framework [46]. The main purpose of the FRESCO is to identify the various issues that speed up the composition of OpenFlow enabled security services. Various security applications can be easily implemented with FRESCO framework. Application layer and the security

Volume: 6 Issue: 2 Pages: 2233-2239 (2014) ISSN : 0975-0290

enforcement kernel are the main components of the FRESCO and both are the integral part of the NOX [7] controller. The basic operating unit in FRESCO framework is known as module. FRESCO application extended through FRESCO API's to provide two developer functions:

- Development environment.
- Resource controller.

8.3. Controller Placement Problem

SDN architecture placed the control logic over the external controller that helps us to configure the packet forwarding devices in data plane. This architecture with separate control plane improves the scalability, reliability and performance which were not possible in traditional architecture. There are many problems that occur in controller placement are as follows:

- How controller placements affect latency?
- How many controllers are needed?

Solution:

The solution to above problems is to have a single centralized controller. One controller location is sufficient to meet the requirements of network topology. As we use the no. of controllers, it becomes difficult to manage them and also it is very costly to have more than one controller. Also controller should be placed so that we have minimum latency that results in efficient networks.

9. CONCLUSION

SDN provides us the new way to implement networking. This paper describes how SDN architecture is the suitable approach to meet the rapidly changing requirements of networking organizations and customers. Fully programmatic functionality of SDN makes it flexible and hence scalable.

In traditional networks it is impossible to achieve scalability and flexibility. The centralized controller can change the functionality of hardware devices by changing the routing policy through programmatic control. We do not need to change the hardware set up with changes in requirements. SDN is the latest concept in networking industries. It enhances the scope for further development in networking. It is the future of networking which allows us to build the cost effective and agile networks. As in SDN the whole dependence lies on the intelligent programmable controller that is on one hand beneficial for management and control purposes but it also has drawbacks, because controller is the main target for the attacks.

Also virtual network in SDN does not have security layer to protect the network where as traditional networks are secure because we have various choices to protect network such as firewall, IDS, worm detector etc. SDN OpenFlow [12] interface is also vulnerable from security point of view. Also it has other issues like whether the functionality will reside in the control plane or both (data or control plane).

REFERENCES

[1] Nick Feamster, Jennifer Rexford, Ellen Zegura, The Road to SDN: An Intellectual History of Programmable Networks. *ACM SIGCOMM, Volume* 44 Issue 2, 2014, 87-98

[2] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, A clean slate 4D Approach to network control and management. *ACM SIGCOMM Computer Communications Review*, 35(5), 2005, 41–54.

[3] Martìn Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, Scott Shenker, Ethane: Taking Control of the Enterprise. *SIGCOMM'07*, 2007, 27–31.

[4] Soheil Hassas Yeganeh, Yashar Ganjali, Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. *HotSDN'12*, 2012.

[5] D. Drutskoy, E. Keller, and J. Rexford, Scalable Network Virtualization in Software-Defined Networks. *IEEE Internet Computing*, 2013

[6] David Erickson, The Beacon OpenFlow controller. *In Proc. HotSDN* 2013.

[7] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casado, Nick McKeown and Scott Shenker, NOX: Towards an Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, Volume 38 Issue 3, 2008.

[8] POX [ONLINE] Available at: http://www.noxrepo.org/pox/about-pox/

[9] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, Scott Shenker, Onix: A Distributed Control Platform for Large Scale Production Networks. *In OSDI* 10, 2010.

[10] Floodlight, [ONLINE] Available at: http://floodlight.openflowhub.org/

[11] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, Shriram Krishnamurthi, Hierarchical Policies for Software Defined Networks. *HotSDN'12*, 2012.

[12] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, OpenFlow: Enabling innovation in campus networks. *SIGCOMM CCR*, 38(2): 2008, 69–74.

[13] GENI (Global Environment for Network Innovations) [ONLINE] Available at: http://www.geni.net/

[14] NSF Future Internet Design [ONLINE] Available at: http://www.nets-find.net/.

[15] [ONLINE] Article available at: http://www.sdncentral.com/whats-network-virtualization/.

[16] Zheng Cai, Alan L. Cox, T. S. Eugene Ng, Maestro :A System for Scalable OpenFlow Control, *Rice University Technical Report TR*10-08, 2010.

[17] E. Keller and J. Rexford, The "Platform as a Service" model for networking. *In Proc. of USENIX INM/WREN*, San Jose, California, 2010

Volume: 6 Issue: 2 Pages: 2233-2239 (2014) ISSN : 0975-0290

[18] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, Virtualizing the network forwarding plane. *In Proc. of ACM, PRESTO*, Philadelphia, USA, 2010.

[19] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown and Guru Parulkar, Can the Production Network Be the Testbed?. *In Operating system Design and Implementation*, 2010.

[20] Roberto Doriguzzi Corin, Matteo Gerola, Roberto Riggio, Francesco De Pellegrini, Elio Salvadori, VeRTIGO: Network Virtualization and Beyond. *In EWSDN*, 2012, 24-29

[21] Nicira, "Network virtualization platform", [ONLINE] Available at: http://nicira.com/en/network-virtualization-platform.

[22] Naga Praveen Katta, Jennifer Rexford, David Walker, Logic programming for Software-defined-Networks. Princeton University.

[23] Boon Thau Loo, Joseph M. Hellerstein, Ion Stoica, and Raghu Ramakrishnan, Declarative routing: Extensible routing with Declarative Queries. *In Proceedings of SIGCOMM '05*

[24] Boon Thau Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica, Implementing Declarative Overlays. *In Proceedings of SOSP*, 2005.

[25] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, Frenetic: A network Programming Language. *In ACM SIGPLAN International Conference on Functional Programming* (ICFP), 2011.

[26] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, David Walker, Princeton Cornell, Composing Software-Defined Networks. *10th USENIX Symposium on Networked Systems Design and Implementation,* 2013.

[27] Tim Nelson, Arjun Guha, Daniel J. Dougherty, Kathi Fisler, Shriram Krishnamurthi, A Balance of Power: Expressive, Analyzable Controller Programming. *HotSDN'13*, 2013.

[28] Advait Dixit, Fang Hao, Sarit Mukherjee, T .V. Lakshman, Ramana Kompella, Towards an Elastic Distributed SDN Controller. *HotSDN'13,* 2013.

[29] Minlan Yu, Jennifer Rexford, Michael J. Freedman, Jia Wang, Scalable Flow-Based Networking with DIFANE. *SIGCOMM' 10*, 2010, New Delhi, India.

[30] Guohan Lu, Rui Miao, Yongqiang Xiong, Chuanxiong Guo, Using CPU as a Traffic Co-processing Unit in Commodity Switches . *HotSDN'12*, 2012.

[31] Mark Reitblatt, Marco Canini, Arjun Goha, Nate Foster, FatTire: Declarative Fault Tolerance for Software-Defined Networks. *HotSDN'13 Proceedings of the second ACM SIGCOMM* workshop on hot topics in Software Defined Networking, 109-114

[32] H. Kim, M. Schlansker, J.R. Santos, J. Tourrilhes, Y. Turner, N. Feamster, CORONET: Fault tolerance for Software Defined Networks. *In Proceedings of ICNP*. 2012, 1-2.

[33] Bob Lantz, Brandon Heller, Nick McKeown, A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. *In HotNets. ACM*, 2010.

[34] M. Pizzonia, M. Rimondini, Netkit: easy emulation of complex networks on inexpensive hardware. *In International Conference on Testbeds and research infrastructures for the development of networks & communities, TridentCom '08*, Pages 7:1{7:10, Brussels, Belgium, 2008. ICST.

[35] J. Ahrenholz, C. Danilov, T.R. Henderson, and J.H. Kim, CORE: A real-time network emulator. *In Military Communications Conference, MILCOM '08*, 1 *IEEE*, 2008

[36] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, J. Rexford , Trellis: a platform for building flexible, fast virtual networks on commodity hardware. *In CoNEXT '08*, pages 72:1{72:6. *ACM*, 2008.

[37] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, David Becker, Scalability and Accuracy in a Large-Scale Network Emulator.

[38] Arjun Roy, Kenneth Yocum, and Alex C. Snoeren, Challenges in the Emulation of Large Scale Software Defined Networks. University of California, San Diego.

[39] Diwaker Gupta, Kashi V. Vishwanath, and Amin Vahdat, DieCast: Testing Distributed Systems with an Accurate Scale Model. *ACM Transactions on Computer Systems 29*, 2011, 4:1-4:48.

[40] Kapil Dhamecha, Bhushan Trivedi, SDN Issues- A Survey. *International Journal of Computer Applications* (0975-8887), 2013, 73-18.

[41] Wang, Anjing et al., Network Virtualization Technologies, Perspectives and Frontiers. Journal of Lightwave Technology 31.4 (2013): 523-537.

[42] Daniel R. Ellis, John G. Aiken, Kira S. Attwood, Scott D.Tenaglia, A Behavioral Approach to Worm Detection. *In Proceedings of WORM*, 2004.

[43] Vyas Sekar, Yinglian Xie, Michael K. Reiter, Hui Zhang, A Multi-Resolution Approach for Worm Detection and Containment.

[44] M. Roesch, Snort - Lightweight Intrusion Detection for Networks. *In Proceedings of USENIX Large Installation System Administration Conference* (LISA), 1999.

[45] Security Implication in Data Center [ONLINE], Available at https://www.opennetworking.org/solution -brief-sdn-security-considerations-in-the-data-center

[46] Shin, Seugwon, et al., FRESCO: Modular Composable Security Services for Software-Defined Networks. *To Appear in the ISOC Network and Distributed System Security Symposium* 2013.

[47] Porras, Philip, et al., "A security enforcement kernel for OpenFlow networks", *Proceedings of the first workshop on Hot Topics in software defined networks. ACM*,2012.