ISSN: 2321-7782 (Online)
Volume 2, Issue 4, April 2014
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Paper / Case Study
Available online at: www.ijarcsms.com

A Survey of Past, Present and Future of Software Defined Networking

Pritesh Ranjan[1]
Computer Department
MIT college of Engineering
Pune – India

Pankaj Pande[2]
Computer Department
MIT college of Engineering
Pune – India

Ramesh Oswal[3]
Computer Department
MIT college of Engineering
Pune – India

Zainab Qurani[4]
Computer Department
MIT college of Engineering
Pune – India

Rajneeshkaur Bedi[5]
Associate Professor
Computer Department
MIT college of Engineering
Pune – India

Abstract: ***Software Defined Networking is the most recent advancement, extending the idea of programmable networks. SDN framework decouples the network service from underlying implementation providing a novel approach for application to configure, operate and communicate with the network. This paper presents a study of programmable networks with emphasis on the motives and challenges of SDN. Following it we give an overview of SDN architecture its current applications and future applications.***

Keywords: Control plane, Data plane, OpenFlow, Programmable Networks, Software Defined Networking

I. INTRODUCTION

Internet is a critical infrastructure for today's world just like transportation and electricity. The large deployment base of internet has made it quite difficult to evolve in terms of physical infrastructure, protocols and performance. With current demands on an exponential increase there is an urgent need for renovation of the infrastructure. Furthermore the plethora of network devices and middleboxes need to be manually configured with limited tools making it error-prone and challenging.

A few more pains with today's network architecture is – the network devices and middleboxes are vertically integrated i.e. the hardware and software is provided by the manufacturer and can't be customized at will. New software may not be installed because of incompatible hardware, or the currently available software couldn't leverage all the hardware capabilities. Another is the inability to have a global view of the network. Today's routers communicate with each other and are unable to select path from a global view. These problems are one of the few challenges that have motivated the researchers to look for some radical new ideas in networking.

Software Defined Networking is a result of such necessities. SDN as an example of "programmable networks" talks of network evolution. The basic driving idea of SDN is the decoupling of data plane from control plane. Control Plane is all the logic that decides what is to be done and instructs the data (or forwarding) plane to implement the decision. Control plane has the logic of controlling and forwarding behavior like tracking topology changes, install forwarding rules, computing routes etc. Data plane on the other hand forwards traffic based on rules as dictated by control plane logic like forward, filter, buffer, rate-limit and measure packets. SDN advocate the centralization of control plane and distributed data plane. The all powerful

centralized control plane called the controller will control all the data planes and can be implemented completely in software in software and installed on commodity hardware.

The immediate benefit of this separation is the global network view – the controller can see the status of all routes and switches quickly deciding the best route. It also helps in selecting the best egress point in an autonomous system for different flows. Another advantage is the horizontal integration in networking devices which allows for separate and independent growth of hardware and software. It also allows rapid innovation of software because many new players can work on developing controller application as long as they have a well defined API to communicate with the hardware. This approach promises more flexibility in choosing the hardware and software by the customer.

Unfortunately, SDN has to deal with a few challenges as well. The first one being the reliability concerns. In traditional networks if one router fails others update their routing table but with centralized control, the failure of one controller will leave the network without a guardian and with the expiry of table rules in switches will soon stop forwarding. Another concern is scalability –today's hierarchical networks are very scalable to any load, the concern with single controller is whether it can scale to large autonomous systems and handle large volume of traffic and not become a bottleneck in the performance of network. Securing the controller is also a major issue since a compromised controller will take the whole network down. In [25] authors argue that the concerns voiced for SDN are neither caused nor fundamentally unique to SDN and the issues can be addressed without losing the benefits of SDN.

The field of SDN is quite recent but is growing at a very fast pace with wide support and attention from academia and industry. Open Networking Foundation [1] (ONF) has been created by a group of service providers, network operators and vendors to promote SDN and standardize the OpenFlow [8] protocol. OpenFlow Network Research Center [2] has been created with a focus on SDN research. The rest of this paper is organized as follows: we provide a study of early programmable networks followed by SDN architecture, current applications and future trends.

II. EARLY PROGRAMMABLE NETWORKS

The idea of programmable networks has been doing the rounds for many years. We present here a concise list of such early approaches which laid the foundation for SDN.

A. Active Networking

In the mid 1990s, a programmable network infrastructure was proposed for customized services under Active Networking [16] initiative by the DARPA community which focused on two main approaches – user-programmable switches and capsules. The capsules are program fragments that would be carried in user messages and could be interpreted and executed by routers. Active networking never gathered enough momentum for industry use due to security and performance concerns.

B. 4D Project

The 4D project [17] (2004) emphasized on separation between the routing decision logic protocols which govern the interaction among network elements. The "decision" plane would have a global view of the network which would call upon the services of "dissemination" and "discovery" plane, for controlling a "data" plane for forwarding traffic. The "discovery" gives information about what resources are available to network controller. "Dissemination" tells how to detect network topology. Later works like "NOX" extract direct inspiration from these ideas.

C. ForCES

ForCES[19] is under active development by IETF Forwarding and Control Element Separation (ForCES) working group since 2003. The ForCES Network Element (ForCES NE) is separated into forwarding elements (FE) and control elements (CE) whereas the ForCES protocol is used to communicate between the two. In contrast to SDN architecture, this approach still presents the combined entity (FE and CE) as a single network element (ForCES NE) to the outside world.

D. Ethane

ETHANE [18] most closely resembles the SDN architecture; in fact ETHANE laid the foundation stone for SDN. It proposed a centralized controller to manage and policy and security in a network. The controller element was to decide to decide the policy for packet handling and a ethane switch consisting of a flow table and a secure channel to the controller.

III. SDN ARCHITECTURE

The Open Networking Foundation (ONF) [1] is a leading organization responsible for standardization and advancements in SDN. The open networking paradigms mentioned earlier encouraged the development of SDN. The most common components in a network are clients who request for a service, a server which promises to deliver the service and a forwarding device responsible for carrying out the communication. The huge deployment base of internet and its rigidity is referred to as "Internet Ossification" and any major changes to it are almost impossible to implement. In fact the traditional networks are very repellent toward innovation; for example the issue of switching from IPv4 to IPv6 which despite being developed for production level performance lags behind old IPv4 in popularity.

Today's network devices come preinstalled with software from manufacturer and leave a very little customization option for the customers. This vertical integration in traditional networks i.e. tight coupling of forwarding plane and control plane, hinder innovation. The SDN architecture splits the networking devices into data plane and control plane. Data plane is not intelligent but can perform forwarding packets very fast and efficiently. It physically carries data packets from one port to another by following rules that are programmed into the device hardware. Control plane has the responsibility of adding behavior into the device; it decides the logic to program the data plane. These two planes communicate over a secure channel over a standard protocol called OpenFlow. [8]

Decoupling control plane from data plane makes control plane programmable thereby enabling abstraction of underlying network device from application and service layers which in turn treats them as virtual entity. Fig 1 shows the logical organization of SDN.

As depicted in figure 1 the decoupling of controller from data forwarding plane made enterprises to gain vendor independent control over network which in turn simplified the design, operation and maintenance. SDN devices no longer need to store the logic telling what to do when packets arrive; instead they rely on the controller to do this work for them.

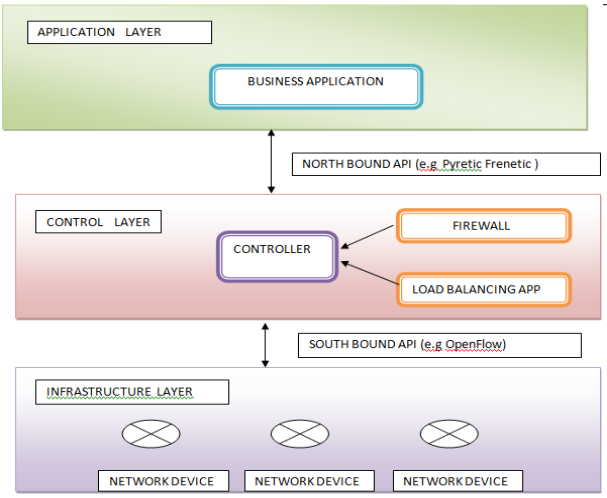

Fig 1: Software Defined Networking Architecture [2]

As shown in **figure 1** many devices from infrastructure layer can be connected to a single centralized control plane which enable controller to have a network wide view of topology hence providing flexibility for traffic engineers to develop and deploy applications like routing and security at one place instead of deploying on each and every device thereby reducing the time consumed by the network to adapt to changes in policy of controlling the underlying devices.

OpenFlow provides an open API between control layer and application layer. This provides an abstraction for business applications to use facility provided by control layer without going into the details of their implementation. In the remainder of this section we will examine the component of the architecture namely: the infrastructure layer (switch/router), the control layer (controller), southbound API and northbound API.

Infrastructure Layer: Switch

The switches can be either purely OpenFlow compliant which do not have onboard control logic, they completely rely on the controller or they can be of hybrid type [26] which support operations supported by traditional switches in addition to OpenFlow protocol.

Virtual switches i.e. switches implemented in software are available which can be used for developing services over SDN or to test an application. Table 1 lists the switches with implementation language, version supported.

Some vendors like HP, Juniper, Pronto, IBM, and PICA8 have made commodity networking hardware available which support OpenFlow. Table 2 lists a brief list of such switches along with manufacturer and OpenFlow version supported.

Table 1: Current software switch implementationz

Software Switch	Implementation Language	OF version
Open vSwitch	C/Python	V1.0
Pantou.OpenWRT	C	V1.0
ofsoftswitch13	C/C++	V1.3
Indigo	C	V1.0

When a packet arrives at the switch the header fields are extracted and matched for the flow entries installed in the switch, if a match is found, the corresponding action in flow entry is performed. A default action for packet handling should be provided -like packet discard, forwarding to another flow table or to the controller- which will be executed in case of table miss.

Table 2: A few currently available commodity switches supporting OpenFlow

Manufacturer	Switch Model	version
HP	8200zl, 6600, 6200zl, 5400zl, 3500/3500yl	V 1.0
Brocade	NetIron CES 2000 series	V 1.0
IBM	RackSwitch G8264	V 1.0

NEC	PF5240 PF5820	V 1.0
Juniper	Junos MX-series	V 1.0
Pronto	3290 and 3780	V 1.0
Pica8	P-3290, P-3295, P-3780, P-3920	V 1.2

The Control Layer: Controllers

The underlying infrastructure devices are connected to controller which decides and installs the rules on switches. To cater scalability and reliability issues two approaches have been proposed , *centralized* and *distributed*. With a physically as well as logically centralized controller the switches are heavily dependent on a single controller which can become a single point of failure or bottleneck. To cater the problem of controller failure OpenFlow makes a provision for multiple replica controllers which provide backup for the controller.

Kandoo [20] proposes a hybrid approach which provides the benefits of both centralized and distributed controller by providing another controller on the top of distributed controllers. In this approach not all the requests are sent to global controller but only those are forwarded which require centralized network view. The main purpose of controller is to add flow rule it can accomplish this task in following way.

a) Reactive Approach

In Reactive approach as proposed in Ethane[18] new packet arrives at the switch, the OpenFlow agent software does a lookup in the flow table at switch and forwards according to the rule, in case no match for flow is found the switch creates a *Packet_in* message and sends that to the controller which in turn installs rule for that packet into the flow table. This approach leads to wastage of time for every new kind of flow.

In proactive approach the OpenFlow controller populates the flow tables in advance for all kind of traffic matches that could be encountered by the switch. This can be achieved by wild carding the rule in flow table due to which *packet_in* message is not generated for each flow. Therefore each packet can be send at rate of the link by eliminating latency induced by controller.

Table 3: A few OpenFlow compliant controllers

Controller	Implementation	Developer
Beacon	Java	Stanford
Floodlight	Java	BigSwitch
FlowVisor	C	Stanford/Nicira
NOX	C	Nicira
POX	Python	Nicira
Ryu	Python	NTT, OSRG group

OpenFlow Protocol

The OpenFlow [26] switch specification describes the communication protocol for infrastructure layer and control layer. A controller uses a secure channel to communicate with switch. OpenFlow packets are sent over this channel. Latest version of OpenFlow protocol supports SSL encryption. The types of message supported by OpenFlow [26] are given below.

a) Controller to Switch Message

This type includes *handshake*, *packet_out* message for installing flow entry. It also handles switch configuration, role configuration, setting asynchronous message configuration and many others.

Controller to switch messages are initiated by the controller and sent to switch over a TCP connection established between switch and controller. A message of this type may or may not require a response from switch.

Following are type of controller to switch message

1. Features request/reply message – A *features request* message is sent by controller to get the capabilities of switch. A *features reply* message is sent by switch which consists of capabilities supported by it. Usually the features message is sent at the time when connection is established through OpenFlow channel.

2. Configuration- The controller can set switch's configurations through this message. It can also query switch's configurations via *query message*.

3. Modify-State – Modify state messages are sent by controller to modify state on the switches .They are used to add/delete and modify flows or group tables on switches .

4. Read-State– Read state messages are used to get switch statistics.

5. Packet-out– The controller sends this message to instruct the switch to send a packet out a particular port or enqueue it or simply drop it by not specifying any action.

6. Barrier – Controller sends this message to ensure that the message dependencies have been met. When switch receives this message it has to finish processing all previously received messages before executing any new message after the Barrier Request.

b) Asynchronous Message

This type includes messages like *packet_in* for sending packet to controller in case of flow miss, flow removed, port status or error message.

These messages are sent without the controller soliciting them from the switch.

Following are the types of Asynchronous message

1. Packet-in – This message is used by a switch to forward the received packet to the controller.

2.Flow-Removed – A flow removed message is sent by switch to controller when a flow entry is deleted .Flows can be either removed using flow delete request or it can expire corresponding to idle-time out or hard time-out. To auto generate a flow removed message the OFPP_SEND_FLOW_REM flag should be set in flow modify message.

3. Port-status – The switch sends a port status message whenever physical ports are a added, modified or removed from the switch.

4. Error – The switch notifies the controller of any problem occurred using error messages.

c) Symmetric Message

It includes messages which are usually used for handshaking purpose like *Hello, echo request, echo reply and Experimenter*.

Symmetric messages are sent without solicitation in both the directions.

Types of Symmetric messages –

1. Hello – A hello message is sent by both controller and switch on connection setup.

2. Echo – It can be sent by both switch and controller and requires an echo reply message. It is used to check the liveliness of a connection. It can also be used to check bandwidth latency

3. Experimenter – Experimenter message is used for future revisions in OpenFlow.

NorthBound and SouthBound communication

OpenFlow does make network programmable but doesn't make it easier to program. High level application like traffic monitoring, firewall, and load balancing are very difficult to develop using such low level of abstraction [4].

Northbound communication refers to the communication between application layer and control layer. We can use northbound API to implement and develop vendor independent application for network management and monitoring, load balancing. Another advantage of northbound API is that they are easy modifiable using high level languages like Python, java, C++ etc. At present no accepted standard protocol exists. These have been implemented on ad-hoc basis for particular applications. Pyretic [27] provides such API which makes writing high level policies easier and a runtime which converts them to low level implementation.

Southbound communication refers to the communication between control layer and infrastructure layer. OpenFlow[8], sFlow [] and NetFlow are such protocols which are used for traffic analysis and monitoring in a network.

IV. SDN APPLICATIONS

SDN promises a lot of opportunities in a number of domains related to computer networking. Decoupling of control and data plane allows for customized control, elimination of middleboxes and easier development and deployment of new network services and protocols. In the following subsection we present a few scenarios where SDN solutions have been proposed or implemented.

A. Data Center

Cloud computing environment of data centers run multiple virtual machines on same physical device. Data center operators continually migrate- VMs around, according to changing traffic patterns and demands. Today's data centers have many design requirements a few of them are easy migration of virtual machines, efficient communication among servers, least traffic congestion due to flooding of broadcast messages and minimal configuration of switches and hosts. Traditional data centers employ layer 3 routers- to connect the core with internet, layer 2 switches (access switches) to connect with servers at edges and aggregation switches to connect the core and access switches. This topology might lead to oversubscribed core and core as a central point of failure.

To overcome these problems data centers can be designed as multi-rooted Fat-tree [ ] with increasing capacity towards the core. Such designs act as a single large L2 topology, which is easily configurable but looses scalability and might get flooded by broadcast messages. PortLand [21] design introduced logically centralized fabric manager and hierarchical pseudo MAC address. Each host has a custom MAC address according to its position in topology. The fabric manager does the task of intercepting packets, rewriting the source/destination MAC addresses and resolving ARP queries. This design is similar to SDN designs where centralized controller intercepts packets and decides the forwarding rules.

Another application is related to energy usage reduction in data center networks. Hellar et al. propose ElasticTree[28], a network-wide traffic manager to measure traffic statistics and control flow routes. ElasticTree turns off switches at low demands claiming in a 60% energy savings. Their proposal is based upon the finding that an ethernet switch working on full load consumes just 5% more energy than one running at no load. This approach increases latency whenever the flows combine to exactly the link rate, an approach to mitigate it is by adding a safety margin to turn on switches before each link nears full capacity and performance is affected. Furthermore a topology aware heuristics makes the prototype scalable enough for data center networks. They further propose that having switches with switch mode or enabling faster booting may make the proposition realistic.

B. Backbone Networks

To support a plethora of WAN intensive apps Google has two backbone networks, one for carrying user traffic and the other for carrying internal traffic between data centers. The problem Google faced was that cost/bit didn't decrease with network size due to quadratic complexity in pair wise interaction between network devices, manual management and configuration of the network components and the complexity to deal with the non standard vendor configuration APIs.

What came as an output was B4 [22], a deployment of SDN in Google's wide area backbone network. The solution adopted was to manage the WAN as a fabric and not as a collection of individual boxes. Following a centralized traffic engineering approach, a centralized controller allocates the paths between two points satisfying capacity constraints of the network and achieving better convergence. The results bear fruits like better network utilization –due to global view, planned deterministic resource allocation by cutting over provisioning. This approach also makes testing the network easier since a centralized control can use a real production network input to research new ideas and test new implementations.

C. Internet Exchange Points

Today's IXP employ BGP as their inter-domain routing protocol which suffers a few limitations like it can route traffic only on destination IP prefix, it can't influence end-to-end path for a traffic and inability to engage in application specific peering. Developing the inter-domain routing at IXPs can influence thousands of ISPs and make the particular IXP a more attractive place for the client ISPs. Deploying SDN at an IXP promises opportunities like freeing from the constraints of working with today's internet protocols. SDN controllers can directly load balance traffic by rewriting packet headers which is a more straightforward method than by using DNS. IXPs could also involve in application specific peering or redirect specific subsets of traffic to middle-boxes to perform more complex operations like deep-packet inspection, encryption or trans-coding.

An initial design of a software internet exchange or SDX has been proposed [23], having a controller, one or more switches for the ISPs to interconnect. The controller would take as input routes per IP prefix, potentially including other attributes such as price, performance or a selection function specific for a particular AS. Outputs of the controller would be FIB entries (Forwarding Information Base) in the switch satisfying the selection criteria for the AS.

D. Other Domains

A few other domains are in Home/small networks [24] by outsourcing management to a third-party which could remotely control the programmable switches and applying distributed network monitoring to detect possible security caveats. Another application is in granting user Dynamic access control to grant and verify identity for users of the network. Seamless mobility/migration can be achieved by letting devices to connect with multiple networks at a time which also promises better connectivity. Load balancing for content distribution networks or distributed IDS [14][15] offer another opportunity for employing SDN concepts. Other proposals include energy-efficient networking, denial-of-service attack detection, network virtualization etc. SDN -still in its cradle- promises much more in a lot many fields, which is supposed to grow with time.

V. Future Directions

More opportunities and challenges are set to arise with refinement of SDN approaches and protocols like OpenFlow. A few such ideas are mentioned below.

A. Controller and Switch Design

Current controllers can't handle the entire traffic single handedly hence for increased scalability; reliability and integrity controllers can be logically centralized but physically distributed. Kandoo [20] , proposes a framework having two layers of controllers – bottom layer controller to run local control application and top layer to maintain network wide state. Onix [29] also emphasizes on a hierarchical set of controllers. A single super ONIX controller tackles all the sub-controllers for the whole domain.

B. Northbound Communication

At present not much of protocols have been defined for controller-service interactions. It is desired to design an interface, through which applications can access the underlying hardware and communicate with other applications as well without knowing the implementation details. Some ideas have been proposed in Procera[32], Frenetic [30] and Pyretic [31]. Frenetic proposes a simple and reusable high level abstraction for programming SDNs, and a run-time system that automatically generates the low-level rules on switches. Pyretic promises to raise the level of abstraction by introducing an abstract packet model, algebra of high-level policies and network objects. It is a domain specific language which provides a runtime and API for northbound communication.

C. Other Domains

Networking devices are to go through a lot of improvement in terms of protocols, business models and applications by their vendors due to the compelling demands of customers. It is still difficult to couple and compose heterogeneous control programs. Future work will certainly focus on making it easier to compose heterogeneous components for control and to facilitate easier debugging and testing of the applications. In areas of security SDN approaches can be used to prevent data leaks. Current advancements in big data science and machine learning we can design intelligent network controllers by mining the huge amount of data that is logged at various devices. Success of SDN in these fields might propel research for innovation in software defined storage and software defined computing.

VI. Concluding Remarks

In this paper, we provided an overview of programmable networks and, in this context, examined the emerging field of Software-Defined Networking (SDN). We look at the history of programmable networks, from early ideas until recent developments. In particular we described the SDN architecture in detail as well as the OpenFlow standard. We presented current SDN implementations and testing platforms and examined network services and applications that have been developed based on the SDN paradigm. We concluded with a discussion of future directions enabled by SDN ranging from support for heterogeneous networks to Information Centric Networking (ICN).

Acknowledgement

We would like to thanks Mr. Rahul Sharma and GS Labs, Pune for extending their support.

References

1. Open networking foundation- https://www.opennetworking.org/about.
2. Open Networking Research Center (ONRC)- http://onrc.net.
3. POX- http://www.noxrepo.org/pox/about-pox/. https://openflow.stanford.edu/display/ONL/POX+Wiki
4. Georgia Tech. Course on SDN at Coursera https://www.coursera.org/course/sdn, 2013.
5. R. Wang, D. Butnariu, and J. Rexford, “Openflow-based server load balancing gone wild,” In Hot-ICE, 2011.
6. Floodlight OpenFlow Controller- http://floodlight.openflowhub.org/.
7. MININET- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
8. OpenFlow white paper- http://archive.openflow.org/documents/openflow-wp-latest.pdf
9. OPENVSWITCH - http://openvswitch.org/
10. MAUSEZAHN-http://www.perihel.at/sec/mz/mzguide.html
11. “CBENCH : Evaluating OpenFlow Controller Paradigms”, Marcial P. Fernandez Universidade Estadual do Cear´a (UECE) Av. Paranjana 1700 Fortaleza - CE - Brazil marcial@larces.uece.br
12. M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford. “A nice way to test openflow applications”. NSDI, Apr, 2012
13. N. Handigol, B. Heller, V. Jeyakumar, D. Mazifieres, and N. McKeown. “Where is the debugger for my software-defined network”. In Proceedings of the first workshop on Hot topics in software defined networks, Hot SDN '12, pages 55-60, New York, NY, USA, 2012. ACM.
14. N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. “Plug-n-serve: Load-balancing web traffic using openflow”. ACM SIGCOMM Demo, 2009.

15. D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden. "A survey of active network research". Communications Magazine, IEEE, 35(1):80-86, 1997.
16. A. Greenberg, G. Hjalmtysson, D. 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 Communication Review, 35(5):41{54, 2005
17. M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. "Ethane: Taking control of the enterprise". ACM SIGCOMM Computer Communication Review, 37(4):1-12-2007.
18. A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. "Forwarding and Control Element Separation (ForCES) Protocol Specification". RFC 5810 (Proposed Standard), Mar. 2010.
19. S.Hassas Yeganeh and Y. Ganjali. "Kandoo: a framework for efficient and scalable offloading of control applications". In Proceedings of the first workshop on Hot topics in software defined networks, HotSDN '12, pages 19-24, New York, NY, USA, 2012. ACM.
20. Niranjan Mysore, Radhika, et al. "PortLand: a scalable fault-tolerant layer 2 data center network fabric." ACM SIGCOMM Computer Communication Review. Vol. 39. No. 4. ACM, 2009.
21. Vahdat, Amin, et al., "B4: Experience with a Globally-Deployed Software Defined WAN." ACM SIGCOMM, August 2013.
22. Feamster, Nick, et al. "SDX: A Software Defined Internet Exchange." Open Network Summit, April 2013.
23. Feamster, Nick. "Outsourcing home network security." Proceedings of the 2010 ACM SIGCOMM workshop on Home networks. ACM, 2010.
24. Feamster, Nick. "Outsourcing home network security." Proceedings of the 2010 ACM SIGCOMM workshop on Home networks. ACM, 2010.
25. Soheil Hassas Yeganeh et. Al. "On Scalability Of Software Defined Networking" in IEEE Communications Magazine – Feb 2013
26. OpenFlow Switch specification- http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf
27. Pyretic - http://frenetic-lang.org/pyretic/
28. B. Heller, S. Seetharaman, P. Mahadevan,Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. "Elastictree: Saving energy in data center networks". In Proceedings of the 7th USENIX conference on Networked systems design and implementation, USENIX Association, 2010.
29. T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al, "Onix: A distributed control platform for large-scale production networks". OSDI, Oct, 2010.
30. N. Foster, R. Harrison, M. J. Freedman,C. Monsanto, J. Rexford, A. Story, and D. Walker. "Frenetic: a network programming Language". In Proceedings of the 16th ACM SIGPLAN international conference on Functional programming, ICFP '11.
31. Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker, "Modular SDN Programming with Pyretic".
32. A.Voellmy, H. Kim, and N. Feamster. "Procera: a Language for high-level reactive network control". In Proceedings of the first workshop on Hot topics in software defined networks, HotSDN '12, pages 43-48, New York, NY, USA, 2012. ACM.
33. http://hal.inria.fr/docs/00/82/50/87/PDF/SDN_survey.pdf "A survey of Software Defined Networking: Past present & future of programmable networks"

AUTHOR(S) PROFILE

Pritesh Ranjan, currently pursuing B.E from Computer Department in Maharashtra Institute of Technology College of Engineering, Pune (MIT-COE) (2010-2014 batch).His subjects of interest includes Software defined Networking and machine learning.

Pankaj Pande, currently pursuing B.E from Computer Department in Maharashtra Institute of Technology College of Engineering, Pune (MIT-COE) (2010-2014 batch). His subjects of interest includes Software defined Networking, and machine learning.

Ramesh Oswal, currently pursuing B.E from Computer Department in Maharashtra Institute of Technology College of Engineering, Pune (MIT-COE) (2010-2014 batch). His subjects of interest includes Software defined Networking, and data science.

Zainab Qurani, currently pursuing B.E from Computer Department in Maharashtra Institute of Technology College of Engineering, Pune (MIT-COE) (2010-2014 batch). His subjects of interest include data mining and information security.

Rajneeshkaur Bedi, has received B. E. degree in Computer Engineering from Amravati University, India in 1997 and M.Tech degree in Computer Engineering from COEP, under Pune University, India in 2005. She is a registered Ph.D student in Computer Department of Amravati University. She is currently working as an Associate Professor in Computer Engineering at MITCOE, Pune, India. She has more than 15 years of teaching experience. Her research interest includes Data mining, Data Privacy, NLP, Information Security, and Machine learning. She has published more than thirty papers in national and international journal and conference. She also has 6 patents to her credit. She is a member of IEEE and life member of ISTE and CSI.

