Global Journals LaTeX JournalKaleidoscope™
Artificial Intelligence formulated this projection for compatibility purposes from the original article published at Global Journals. However, this technology is currently in beta. *Therefore, kindly ignore odd layouts, missed formulae, text, tables, or figures.*

Past before Future: A Comprehensive Review on Software Defined Networks Road Map

Windhya Rankothge

Received: 8 December 2018 Accepted: 31 December 2018 Published: 15 January 2019

Abstract

Software Defined Networking (SDN) is a paradigm that moves out the network switch?s control plane (routing protocols) from the switch and leaves only the data plane (user traffic) inside the switch. Since the control plane has been decoupled from hardware and given to a logically centralized software application called a controller; network devices become simple packet forwarding devices that can be programmed via open interfaces. The SDN?s concepts: decoupled control logic and programmable networks provide a range of benefits for management process and has gained significant attention from both academia and industry. Since the SDN field is growing very fast, it is an active research area. This review paper discusses the state of art in SDN, with a historic perspective of the field by describing the SDN paradigm, architecture and deployments in detail.

Index terms — software defined network (SDN), review.

1 Introduction

hree components of the network architecture are control plane, data plane, and management plane [1]. The control plane carries control traffic (routing protocols) and is responsible for maintaining the routing tables. The management plane carries administrative traffic and is considered a subset of the control plane. The data plane bears the user traffic that the network exists to carry. It forwards the user traffic based upon information learned by the control plane. In a conventional network, all these three planes are implemented in the firmware of routers and switches.

Software Defined Networking (SDN) is a new paradigm that moves out the network switch's control plane from the switch and leaves only data plane inside the switch [2]. Since the control plane is decoupled from hardware and given to a logically centralized software application called a controller, network devices become simple packet forwarding devices that can be programmed via open interfaces. The SDN's concepts: decoupled control logic and programmable networks provide a range of benefits for the network management process. They include centralized control, simplified algorithms, commoditizing network hardware, eliminating middle-boxes and enabling the design and deployment of third-party applications.

The promise of SDN has gained significant attention from both academia and industry. The Open Network Foundation (ONF) is an industrial driven organization, founded in the year 2011 by a group of network operators, service providers, and vendors to promote SDN and standardize the OpenFlow protocol **??**3]. Deutsche Telekom, Facebook, Google, Microsoft, Verizon and Yahoo are among the founders. Currently, ONF has around 95 members including several major vendors. The OpenFlow Network Research Center (ONRC) was created by the academia with a focus on SDN research **??**4]. Since the SDN field is growing very fast, it is a very active research area. This review paper discusses the state of art in SDN, with a historic perspective of the field by describing the SDN paradigm, architecture and deployments in detail.

2 II.

3 SDN History

The idea of programmable networks and decoupled control logic has a story of years. The history of SDN goes back to 1980s [5]. This section provides an overview of four technologies which helped SDN to evolve.

4 a) Central network control

In earlier days telephone networks were using in-band signaling where the data (voice) and the control signals are sent over the same channel. The resulting networks were always complex and insecure. In 1980s, AT&T separated data and control planes of their telephone network and introduced the concept of "Network Control Point" (NCP) [6]. The idea was to separate voice and control, and the control resided on NCP. NCP allowed operators to have a central networkwide vantage point and directly observe the networkwide behavior. Elimination of in-band signaling lead to independent evolution of infrastructure, data, and services where new services were able to be introduced to customers easily. So NCP was the origin of the SDN's concept: separating control and data plane, and to have centralized control over the network [5].

5 b) Programmability in networks

In the mid-1990s, DARPA research community introduced "Active Networks" with the idea of a network infrastructure that would be programmable for customized services [7]. There were two main approaches: user programmable switches, with in-band data transfer and out-of-band management channels and capsules, which were program fragments that carried in user messages. Program fragments would be interpreted and executed by routers [8]. A Cambridge project in the year 1998, Tempset developed programmable, virtualizable switches called switchlets [9]. Switchware project of Penn, introduced a programmable switch and a scripting language to support switchlets [10]. Smart Packets, research by BBN was focused on applying the active networks framework to network management process [11]. The Open Signaling project of Columbia, introduced NetScript, a language to provide programmable processing of packet streams [12] [13]. Pro-grammable switches accelerated the innovation of middle-boxes (firewalls and proxies) which are programmed to perform specific functions. Providing programming functions in networks and compose these functions together were the legacy of active networks for SDN [5].

6 c) Network virtualization

Network virtualization is the representation of one or more logical network topologies on top of the same infrastructure. It separates the logical infrastructure from underlying physical infrastructure. There are many different instantiations such as Virtual LANs (VLANs), network testbeds and VMWare. In the Switchlets, the control framework has been separated from the switch and allowed virtualization of the switch [9]. In the year 2006, VINI provided a Virtual Network Infrastructure to support different experiments on virtual topologies using a single infrastructure [14]. VINI used the concept of separating control and data planes, and its control plane was a software routing protocol called XORP, which allowed to run routing protocols on virtual network topologies. VINI's data plane "Click" provided the appearance of the virtual network topologies to experimenters. In the year 2007, CABO, a network infrastructure, separated the infrastructure and services to allow service providers to operate independently [15]. The concepts of separating services from infrastructure, using multiple con-trollers to control a single switch and exposing multiple logical switches on top of a single physical switch were the legacy of network virtualization for SDN [5].

7 d) Control of packet switched networks

With the above evolution of network technologies, the separation of control was needed for rapid innovation of networks. Since the control logic is tied to hardware, it was easier to modify the existing control logics of the telephone network. Having a separate control channel made it possible to have a separate software controller and could easily introduce new services to the telephone network. Software controllers also allowed operators to have a centralized network-wide vantage point and directly observe the network-wide behavior of the telephone network. With these motivations, packet switched networks also tried to separate the control plane from the data plane. There are four main ways that packet switched networks achieved separation of control: separate control channel, in-band protocols, customizing the hardware in the data plane and open Hardware [5].

The first approach of a separate control channel for packet switched network came from the Internet Engineering Task Force (IETF) with the protocol "FORCES" in the year 2003 [16]. The FORCES redefined the network device's internal architecture by separating the control element (CE) from the forwarding elements (FE). The CE executes control and signaling functions and uses the ForCES protocol to instruct FEs on how to forward packets. The FEs forwards packets according to the instructions given by the CE. Each FE has a Logical Function Block in its data plane which enables the CE to control the FEs' configuration and used to process packets. The communication between FEs and CE are achieved by the FORCES protocol. The protocol works based on a master-slave model; FEs are slaves and CE is the master. Even though the FORCES architecture separated the control plane from the data plane, both the planes were kept in the same network device and was represented as a single entity. However, the FORCES required standardization, adoption and deployment of new hardware.

The second approach was to use existing protocols as control channels to send control messages to FEs, and it was called in-band protocols. With the Routing Control Platform (RCP) in the year 2004, each autonomous system in the network had a controller in the form of an RCP [17]. An RCP computed the routes on behalf of routers and, it used existing routing protocols to communicate routes to routers. The limitation with this approach was, the control process was constrained by what the existing protocols can support.

Customizing the hardware in the data plane, supported a wide range of applications in the control plane. In the year 2007, Ethane presented a network architecture for enterprise networks, which used a centralized controller to manage policies and security in a network [18]. Ethane directly enforced a single, network policy at an element called "Domain Controller." A Domain controller computes the flow table entries that should be installed in each of the enterprise switches based on access control policies defined at the Domain Controller. OpenWrt, NetFPGA, and Linux built custom switches to sup-port the Ethane protocol. However, they required new hardware deployments that support Ethane protocol.

The solution was the last approach, to use a method that can operate on existing routing protocols, and did not require customized hardware [19]. It is called open hardware and in the year 2008, the OpenFlow project started with this concept [20] [21]. OpenFlow took the capabilities of existing hardware and opened those capabilities, such that standard control protocols could control the behavior of that hardware.

8 e) OpenFlow

The OpenFlow network has been deployed in academic campus networks initially [20] [21] and today more than nine universities in the US have deployed OpenFlow networks ??22]. OpenFlow has gained significant attention from both academia and industry as a strategy to increase the functionality of the network, but at the same time reducing costs and hardware complexity. The OpenFlow architecture consists of three modules: a Flow Table in each switch, a Secure Channel that connects the switch to a remote control process (called the controller) and the OpenFlow Protocol [20] [21] as shown in Figure 1.

The forwarding device (OpenFlow enabled switch/router) has one or more flow tables. A flow table consists of flow entries, each of which determines how packets belonging to a flow will be processed and forwarded. Flow entries are stored according to their priorities. A flow table entry consists of three main fields [23] and shown in Figure 2.

? Match fields (information found in the packet header): used to match incoming packets ? Counters: used to collect statistics for the particular flow (number of received packets, number of bytes and duration of the flow) A set of instructions, or actions, to be applied upon a match; they dictate how to handle matching packets. The actions include dropping the packet, continuing the matching process on the next flow table, or for-ward the packet to the controller over the OpenFlow channel. An OpenFlow enabled switch/router has the capability of forwarding packets according to the rules defined in the flow table. Figure 3 shows a high-level description of how an OpenFlow enabled switch/router processes a packet. Internally, a switch uses Ternary Content Addressable Memory (TCAM) and Random Access Memory (RAM) to process each packet [24]. When a packet arrives at the OpenFlow enabled switch/router, packet header fields are extracted and matched against the matching fields of the first flow table entries. If a matching entry is found, the switch applies the appropriate set of instructions associated with the matched flow entry. If a matching entry is not found, depends on the instructions defined by the tablemiss flow entry, the switch will take action. To handle table misses, every flow table must contain a table-miss entry which specifies a set of actions to be performed when no match is found for an incoming packet ??23]. Figure 4 shows a low-level description of how an OpenFlow switch processes a packet. The metadata field acts as a register which can be used to pass information between the tables as the packet traverses through them. The Multi-Protocol Label Switching (MPLS) fields are included to support MPLS tagging. Since there are multiple flow tables available in the switch, the processing of a packet entering the switch is changed. The flow tables in the switch are linked together using a process called "pipeline processing." When the packet first enters the switch, it is sent to the first flow table to look for the flow entry to be matched. If there is a match, the packet gets processed there. If there is another flow table that the particular flow entry points to, the packet is then sent to that flow table. The process is repeated until a particular flow entry does not point to any other flow table. The flow entries in the flow tables can also point to the group table. The group table is specially designed to perform operations that are common across multiple flows. The OpenFlow 1.1.0 also replaced actions with instructions. In OpenFlow 1.0.0 an action could be to forward the packet or to drop it, as well as processing it normally as it would be in a regular switch. Instructions are more complex and they include modifying a packet, updating an action set or updating the metadata.

The OpenFlow 1.2.0 specification was released in De-cember 2011 and it included support to IPv6 addressing. Matching could be done using the IPv6 source and destination addresses. With OpenFlow 1.2.0 specifications, a switch could be connected to multiple controllers concurrently. The switch maintains connections with all the controllers. Controllers can communicate with each other. Having multiple controllers facilitated load balancing and faster recovery during a failure. The OpenFlow 1.3.0 specification was released in June 2012. It included features to (1) control the rate of packets through per flow meters, (2) have auxiliary connections between the switch and the controller and (3) add cookies to the packets sent from the switch to the controller. Table I shows a summarization of OpenFlow specifications.

9 SDN Architecture

In SDN, the control plane is decoupled from the hard-ware data plane and given to a software application called a controller. The controller is the core of an SDN network and it lies between network devices and applications ??25] [26]. This section gives a brief introduction to the SDN architecture. SDN architecture is shown in figure 6 and it includes: SDN Controllers, Southbound Interfaces, and Northbound Interfaces [25]. point to the network (network operating system) [27]. While a computer operating system provides read and write access to various resources, a network operating system provides the ability to observe and control a network. The network operating system which is referred to as the controller here after, does not manage the network, but it provides a programmatic interface which can be used to implement applications to perform the actual management tasks. SDN controllers presents two possible behaviors: reactive and proactive [28].

When the controller behaves reactively, it listens to switches passively and configures routes on-demand. The first packet of each new flow, received by a switch (flow request) triggers the controller to insert flow entries in each switch of the network [28]. Every new flow introduces a small delay because of the additional setup time. Also with the hard dependency of the controller, if a switch losses the connection to the controller, the switch will not be able to forward packets of new flows. When the controller behaves pro-actively, it prepopulates a flow table for each switch. So it has zero additional flow set-up time because the forwarding rules are already defined [28]. With this approach, if the switch loss the connection with the controller, it will not disrupt traffic. However, the proactive approach requires the controller to know the traffic flows in advanced to configure the paths before it is used. Current controllers are implemented to facilitates both approaches. The Controller behaves reactively in the initial state of the network and, after getting to know the network it starts to behave pro-actively.

10 b) Southbound Interfaces

The southbound interfaces allow switches to communicate with the controller. The OpenFlow protocol is the most popular implementation of the southbound interface. OpenFlow 1.3.0 and above provide optional support for encrypted Transport Layer Security (TLS) communication and a certificate exchange between the switches and the controller for secure communication ??23]. The OpenFlow protocol consists of three types of messages. 2) Asynchronous messages: Sent by the switch: The Packet-in messages are used to inform the controller about a packet that does not match an existing flow. The Flow Removed messages are used to inform the controller that a flow has been removed because of its time to live parameter or inactivity timer has expired. Finally, the Port status messages are used to inform the controller of a change in port status or that an error has occurred on the switch. 3) Symmetric messages: Sent by both the switch or the con-troller: The Hello messages exchanged between the controller and switch on startup, and the Echo messages are used to determine the latency of the controller-to-switch connection and to verify that the controller-to-switch connection is still operative. The Error messages are used to notify the other side of the connection of problems. Finally, the Experimenter messages are used to provide a path for future extensions to OpenFlow technology.

The Border Gateway Protocol (BGP), a wellknown core Internet routing protocol is used by Juniper Network's in their SDNs [29]. The controller uses BGP as a control plane protocol and leverage NETCONF (an IETF network management protocol) as a management plane protocol to interact with physical routers, switches and networking services like firewalls. This approach enables SDN to exist in a multi vendor environment without requiring infrastructure upgrades. OpenFlow does not address the issue of the controller interoperability and requires physical changes to the network, so Juniper is introducing BGP to be the standard of the SDN. Extensible Messaging and Presence Protocol (XMPP) which was originally developed for instant messaging and online presence detection is also emerging as an alternative SDN protocol ??30]. XMPP can be used by the controller to distribute control plane information to the server endpoints because XMPP manages information at all levels of abstraction down to the flow, not only to network devices.

11 c) Northbound APIs

The southbound interfaces allowed controllerswitches communication and provided basic operations to access the network system. But they could not retrieve complex information from the switches and therefore programming the network to perform high-level tasks (load balancing, implementing security policies) was difficult. Also, it was difficult to perform multiple independent tasks (routing, access control) concurrently using the south bound interfaces. So the northbound interface, a programming interface that allows applications to program the network with higher level abstraction [25] [26] was introduced. Developers can use the northbound interface to extract information about the underlying network and to implement complex applications such as path computation, loop avoidance, routing, and security. Additionally, northbound interface can be used by controllers to communicate with each other to share resources and synchronize policies. The North-bound interface offers vendor in-dependability and ability to modify or customize control through popular programming languages. Unlike southbound interfaces, there is no currently accepted standard for northbound interfaces and they are more likely to be implemented depending on the application requirements.

IV.

12 SDN Development Tools and Frameworks

The concept of decoupling control plane from the data plane allows SDN to facilitate network evolution and innovation by introducing new services and protocols easily. This section gives an overview of currently available tools and environments for developing services and protocols with SDN.

13 a) SDN controller platforms

Many controller implementations are available for SDNs and a suitable controller can be selected by considering the programming language and performances of the controller [31] [32] [33]. The popular controller platforms include ovs [23], NOX [27],

POX [34], Beacon [31], Maestro [35], Trema [36] Ryu [37] and Floodlight **??**38]. Table II shows a comparison of the SDN controller platforms according to their general details and Figure 7 (taken from [31]) shows a comparison of the performances of SDN controller platforms.

The current standard for evaluating SDN controller performance is Cbench. The Cbench simulates OpenFlow switches and operates in either throughput or latency mode. In through-put mode, each of 64 emulated switches constantly sends as many Packet In messages as possible to the controller, ensuring that the controller always has messages to process. Evaluation tests have been run on Amazon's Elastic Computer Cloud using a Cluster Compute Eight Extra Large instance, containing 16 physical cores from 2 x Intel Xeon E5-2670 processors, 60.5GB of RAM, using a 64-bit Ubuntu 11.10 VM image. Figure 7 shows Cbench throughput mode results using controllers with a single thread. Beacon shows the highest throughput at 1.35 million responses per second, followed by NOX with 828,000, Maestro with 420,000, Beacon Queue with 206,000, Floodlight with 135,000, and Beacon Immediate with 118,000. Both Python-based controllers run significantly slower, POX serving 35,000 responses per second and Ryu with 20,000. b) SDN software switch platforms With SDN, the switch architecture has become very simple, because it is left only with the data plane. It has reduced functions of switches and introduced concepts of software switch implementation and switch virtualization. The result was rapid innovations in software switch platforms. The software switch platforms can be used to replace the firmware of physical switches that do not support SDN. The popular software switch platforms include Open vSwitch [23], Pantou/OpenWRT [39] and ofsoftswitch13 [40]. Table III shows a comparison of the SDN software switch platforms.

14 c) Native SDN switches

As explained at the beginning of the paper, the promise of SDN has gained significant attention from many network de-vices vendors. One clear evidence of industry strong commitment to SDN is the availability of OpenFlow enabled commodity network hardware. Hewlett-Packard, Brocade, IBM, NEC, Pronto, Juniper, and Pica8 have introduced many OpenFlow enabled switch models. Table IV shows a partial list of native SDN switches.

15 d) SDN languages

SDN programming languages are used for higher level abstraction of programming for network management. They consist of high-level abstractions for querying network state, defining forwarding policies and updating policies in a consistent way [41]. SDN languages is an area of very active research and several languages have been proposed and are still under development. Table V shows a classification of different SDN languages.

The FatTire [42] allows programmers to declaratively specify sets of legal paths through the network and fault tolerance requirements for those paths. The FatTire compiler takes programs specified regarding paths and translates them to OpenFlow switch configurations. Since the backup paths are configured with those programs, responding to link failures can be done automatically without controller intervention.

The Nettle [43] was originally designed for programming OpenFlow networks. Using the discrete nature of Functional Reactive Programming, Nettle can capture control messages to and from OpenFlow switches as streams of Nettle events. The Nettle model messages from switches with a data type SwitchMessage and commands to switches with a data type SwitchCommand. A Nettle program is a signal function (SF) having an input carrying switch messages from all switches in the network and output carrying switch commands to any switches in the network, SF (Event SwitchMessage) (Event SwitchCommand).

The Flow-based Management Language (FML) [44] comes with high-level built-in policy operators that allow or deny certain flows flowing through a firewall or provide quality of service. If network forwarding policy falls into the space of policies that can be described by an FML program, the code for implementing the policy is easy. But adding new policy operators to the system requires coding outside the FML language. Moreover, a resulting policy decision applies equally to all packets within the same flow and it is not possible to move or redirect a flow as it is processed. So, even though FML provides network operators with a very useful set of SDN abstractions, the programming model, is inflexible.

The Procera [45] is an extension to Nettle, which has been designed to incorporate events that originated from sources other than OpenFlow switches. It supports policies that react to conditions such as user authentications, time of day, bandwidth use and server load. Procera is expressive and extensible, so users can easily extend the language by adding new constructs. The input to the main Procera signal function is a world signal whose instantaneous values have the abstract World type. The output of a Procera program is a signal carrying flow constraint functions. A flow constraint function determines the constraints that are applied to a flow: allow or deny. The Frenetic language is embedded in Python and comprises two integrated sub-languages: a declarative network query language and a network policy management library. The results of such queries may be used for security monitoring and for decisions about the forwarding policy.

The Flog [46] combines features of both FML and in Frenetic. From FML, Flog uses logic programming as the central paradigm for controlling SDNs. Logic programming fits the SDN domain because SDN programming is table driven collection and processing of network statistics. From Frenetic, Flog uses the concept that controller programs may be factored into three key components: a mechanism for querying network state, a mechanism for processing data learned from queries and a component for generating packet forwarding policies. Flog is designed as an event-driven and forward chaining logic programming language. Each time a networking event occurs, the logic program executes. It can have two effects: generates a packet forwarding policy that is compiled and deployed on switches and generates a state that is used to help the logic program to be executed when the next network event is processed.

The Pyretic system [47] enables programmers to specify network policies, compose them together and execute them on abstract network topologies. The Pyretic's static policy lan-network), and policy combinators, which are used to mix primitive actions, predicates, and queries together to craft so-phisticated policies from simple components. The policies can be composed together in two ways: parallel and sequential. In parallel composition, multiple policies operate concurrently on separate copies of the same packets. In sequential composition, one module operates on the packets produced by another.

16 e) SDN debugging tools

The emergence of SDN enables adding new network functionalities easily, at the risk of programming errors. Even though the centralized programming model has reduced the likelihood of bugs, the ultimate success of SDN depends on having effective ways to test applications in pursuit of avoiding bugs. There are many SDN debugging tools have been developed and they can be divided into four categories based on the layers they are working with. Table VI shows a classification of different debugging tools according to the layers they are working with.

The NICE [48] is an automated testing tool that can be used to identify bugs in OpenFlow programs though model checking and symbolic execution. It automatically generates streams of packets under possible events and tests unmodified controller programs. The programmer must supply the controller program and the specification of a topology with switches and hosts, to use with NICE. NICE can be instructed by the programmer to check for generic correctness properties (no forwarding loops or no black holes), and optionally application-specific correctness properties. NICE is developed to explores the space of possible system behaviors systematically and checks them against the desired correctness properties. As the output, NICE reports property violations with the traces to deterministically reproduce them.

Anteater [49] is the first design and implementation of a data plane analysis system which can be used to find bugs in real networks. The system detects problems by analysing the contents of forwarding tables in routers, switches, firewalls and other networking equipment. **??**t The ndb [50] is a prototype network debugger inspired by gdb (a popular debugger for software programs). It implements two primitives useful for debugging a SDN control plane: breakpoints and packet back-traces. A packet back-trace in ndb allows the user to define a packet breakpoint (an un-forwarded packet or a packet filter). Then it shows the sequence of for-warding actions seen by that packet leading to the breakpoint.

OFRewind [51] allows SDN control plane traffic to be recorded at different granularities. Later they can be replayed to reproduce a specific scenario, giving the opportunity to localize and troubleshoot the events that caused the network anomaly. It records flow table state via a proxy and logs packet traces and aids debugging via scenario re-creation. The VeriFlow [52] is a SDN debugging tool which finds faulty rules issued by SDN applications and prevents them from reaching the network and causing anomalous network behavior.

VeriFlow operates as a layer between the controller and the devices, and checks the validity of invariants as each rule is inserted. To ensure a real-time response, VeriFlow introduces new algorithms to search for potential violation of key network invariants: availability of a path to the destination, absence of routing loops, access control policies or isolation between virtual networks.

Other than the SDN debugging tools which were described earlier, there are two SDN troubleshooting simulators: STS (SDN Troubleshooting Simulator) [53] and OpenSketch [54]. STS [53] is a SDN troubleshooting simulator which is written in python and depends on POX controller [34]. It simulates the devices of the network to allow operators to easily generate test cases, examine the state of the network interactively and find the exact inputs that are responsible for triggering a given ment architecture, which separates the measurement data plane from the control plane. In the data plane, OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting). They can be implemented with commodity switch components and support many measurement tasks. In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resources for different measurement tasks.

17 f) SDN emulation and simulation tools

The Mininet [55], the Emulab and the ns-3 [56] are popular emulation and simulation Tools used with SDN. Mininet [55] is an emulation environment which creates a complete network of hosts, links, and switches on a single machine. It creates virtual networks using process-based virtualization and network namespaces (features available in Linux kernels). In Mininet, hosts are emulated as bash processes running in a network namespace. So any code that would run on a Linux server can be run within a Mininet "Host". The Mininet "Host" has its private network interface and can only see its own processes. Switches in Mininet are software- The Emulab [57] is a network emulation testbed which includes a network facility and a software system. Emulab is widely used by computer science researchers in the fields of networking and distributed systems and it support OpenFlow. So currently it is used also used for SDN research works. The primary Emulab installation is

18 g) SDN virtualization tools

The OpenFlow has opened the control of a network for innovation, but only one network administrator can do experiments on the network at a time. If there is a way to divide, slice or replicate network resources, more than one network administrator can use them in parallel to do experiments. Actions in one slice or replication should not negatively affect other, even if they share the same underlying physical hardware. SDN Virtualization concepts have been introduced to achieve these goals.

The FlowVisor [58] is a special purpose OpenFlow controller that allows multiple researchers to run experiments independently on the same production OpenFlow network. It uses a new approach to switch virtualization, in which the same hardware forwarding plane is shared among multiple logical networks, each with distinct forwarding logic. FlowVisor acts as a middle layer between the underlying physical hardware and the software that controls it. It is implemented as an OpenFlow proxy that intercepts messages between OpenFlow switches and OpenFlow controllers. The AutoSlice [59] devel-ops a transparent virtualization layer (SDN hypervisor) which automates the deployment and operation of vSDN topologies. In contrast to FlowVisor, AutoSlice focuses on the scalability aspects of the hypervisor design. AutoSlice monitors flow level traffic statistics to optimize the resource utilization and to mitigate flow-table limitations. With the distributed hypervisor architecture, Autoslice can handle large numbers of flow table control messages from multiple tenants.

In a virtual machine environment, moving applications from one location to another without a disruption in service is called Live virtual machine (VM) migration. SDN applications can reside and rely on multiple VMs. So migrating individual SDN VMs, one by one, may disrupt the SDN applications. So the LIME [60] design migrate an ensemble: the VMs, the network, and the management system to a different set of physical resources at the same time. LIME uses the SDN concept of separation between the controller and the data plane state in the switches. LIME clones the data plane state to a new set of switches, transparent to the application running on the controller. And then incrementally migrates the traffic sources.

The RouteFlow [61] provides virtualized IP routing over OpenFlow capable hardware. It is composed with a OpenFlow Controller application, a server, and a virtual network environ-ment. The virtual network environment rebuild the connectivity of the physical infrastructure and runs IP routing engines. The routing engines generate the forwarding information base (FIB) according to the routing protocols configured. An ex-tension of RouteFlow [62], discusses incorporating RCPs [17] in the context of OpenFlow and SDN. It proposes a controller centric networking model with a prototype implementation of an autonomous system-wide abstract BGP routing service.

V.

19 Final Remarks

SDNs have emerged in the last decade as a very active research domain, gaining significant attention from both academia and industry. This survey discussed the state of art in SDN, with a historic perspective of the field by describing the SDN paradigm, architecture and deployments in detail.

We first introduced the concepts and definitions that enable a clear understanding of SDNs. The idea of programmable networks and decoupled control logic has been around for many years and the history of SDN goes back to the early 1980s. Central network control, programmability in networks, network virtualization and control of packet switched networks were the four main supporting technologies which helped SDN to evolve. The survey was extended by exploring the OpenFlow project and the standardized SDN architecture. Standard SDN three tier architecture includes: SDN controller, southbound APIs and northbound APIs. For a broader scope, the pa-per detailed the tools and frameworks associated with SDN development in the categories of SDN controller platforms, SDN software switch platforms, native SDN switches, SDN languages, SDN debugging tools, SDN emulation/simulation tools and SDN virtualization tools. Year 2 019 ( ) C [1] [2]

[1] ( ) C © 2019 Global Journals Past before Future: A Comprehensive Review on Software Defined Networks Road Map

[2] © 2019 Global JournalsPast before Future: A Comprehensive Review on Software Defined Networks Road Map

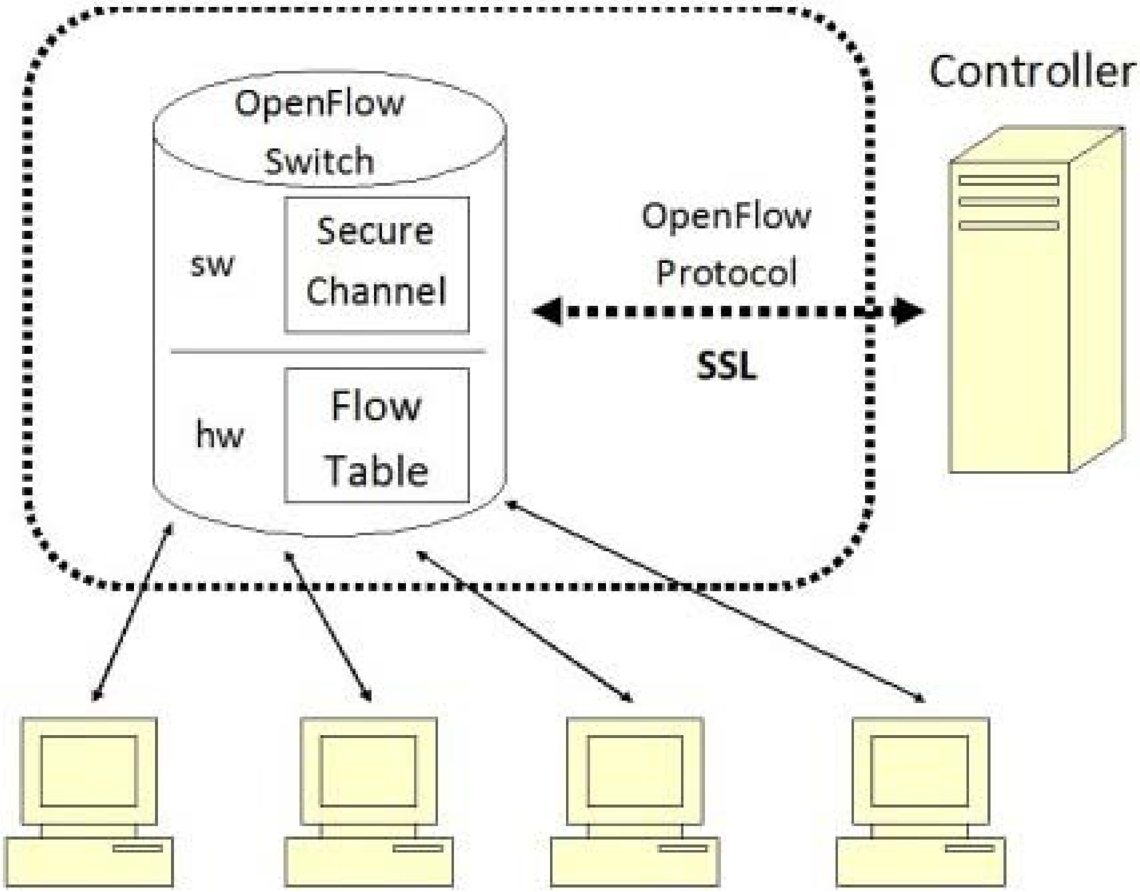
1 

Figure 1: Fig. 1 :

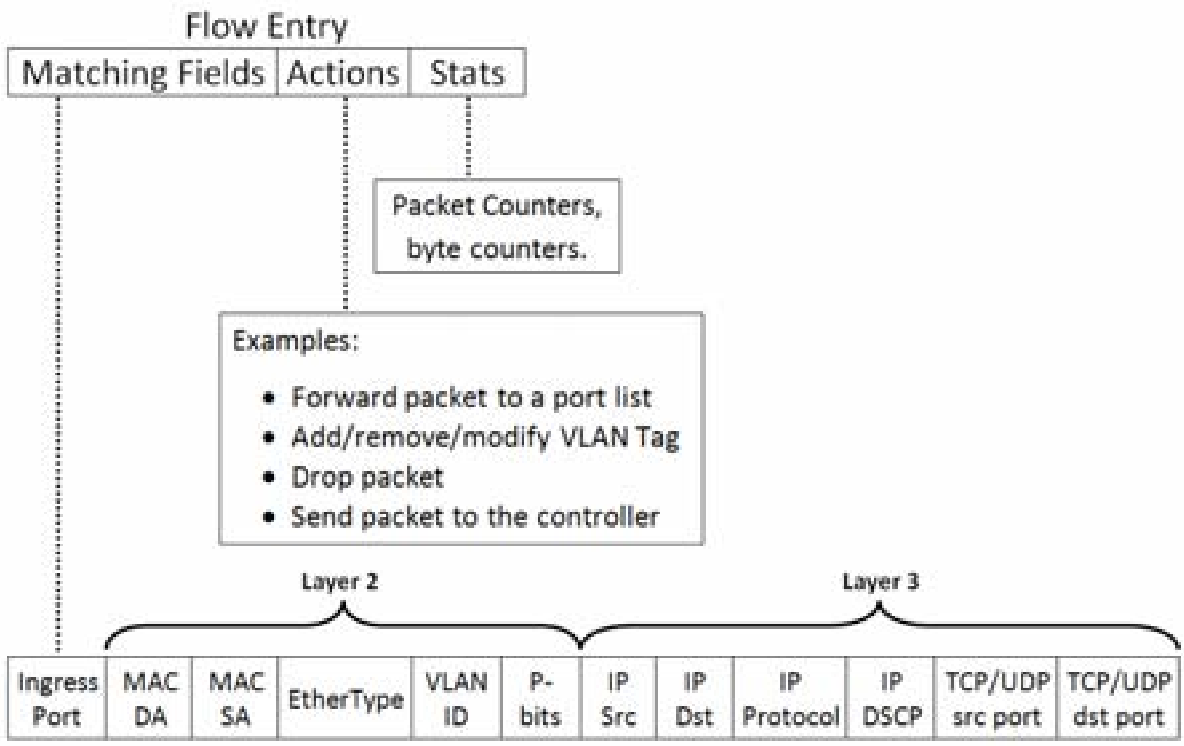
2 

Figure 2: Fig. 2 :

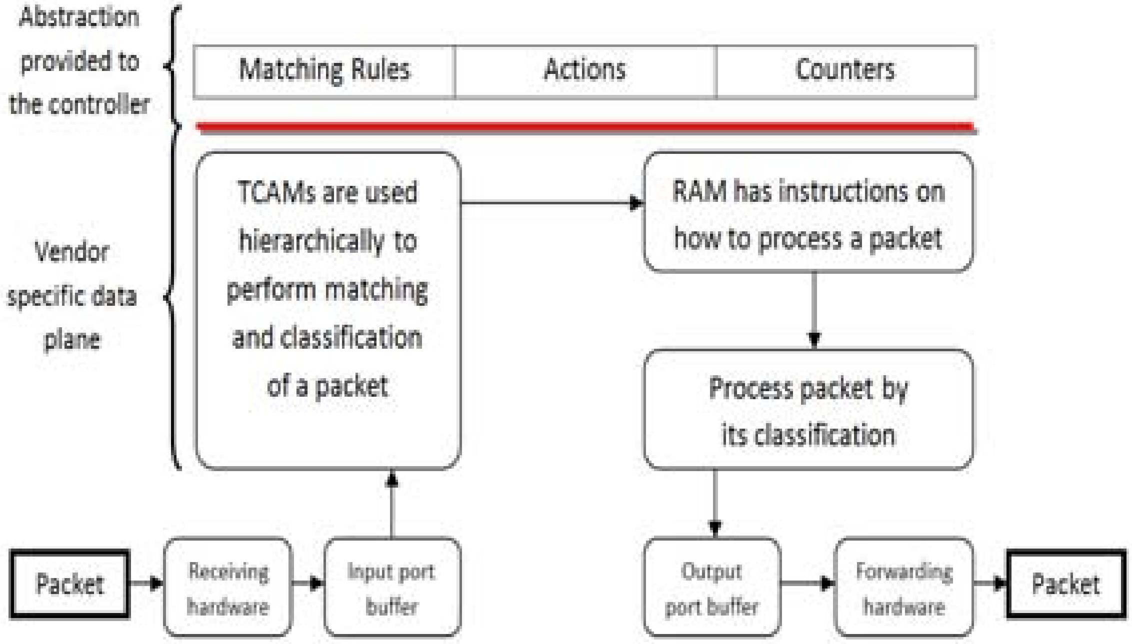
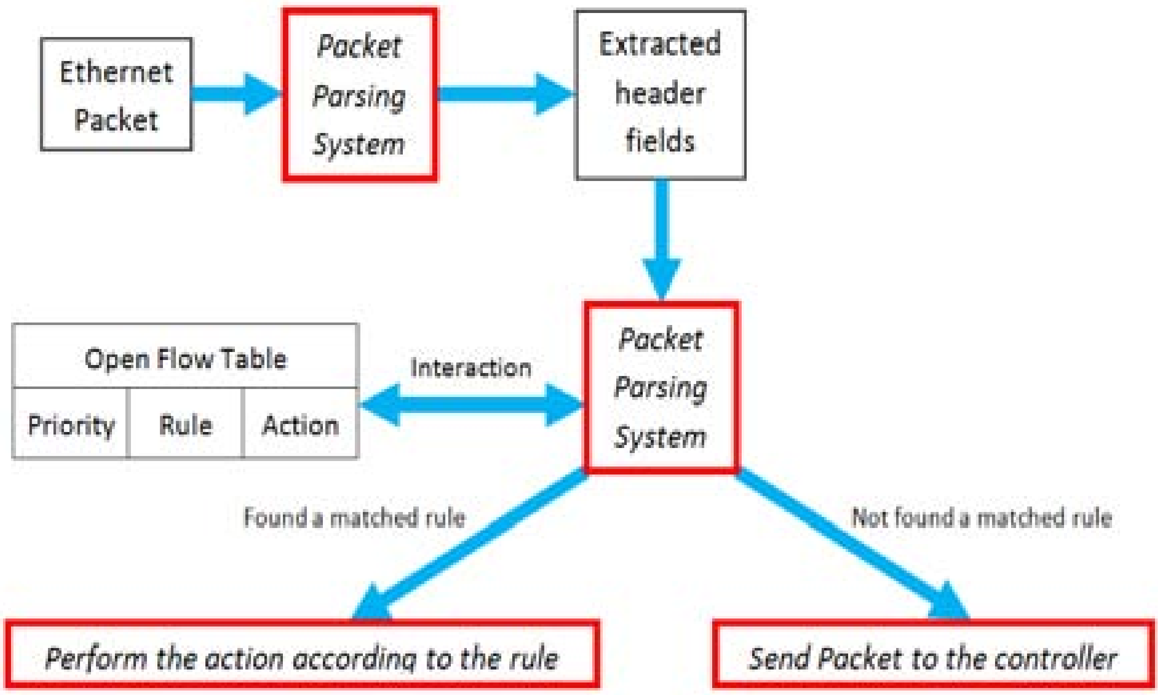
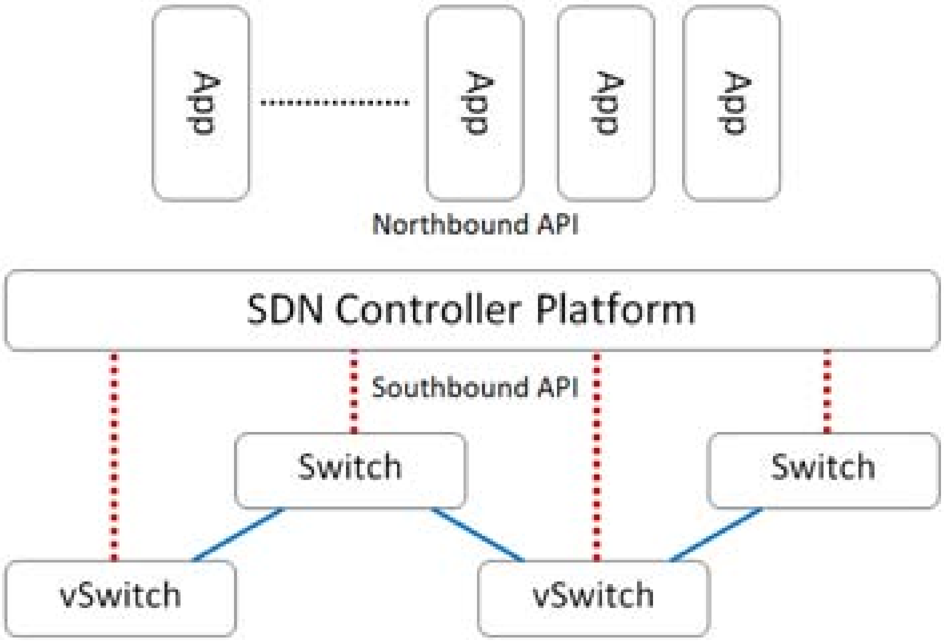


Figure 3:



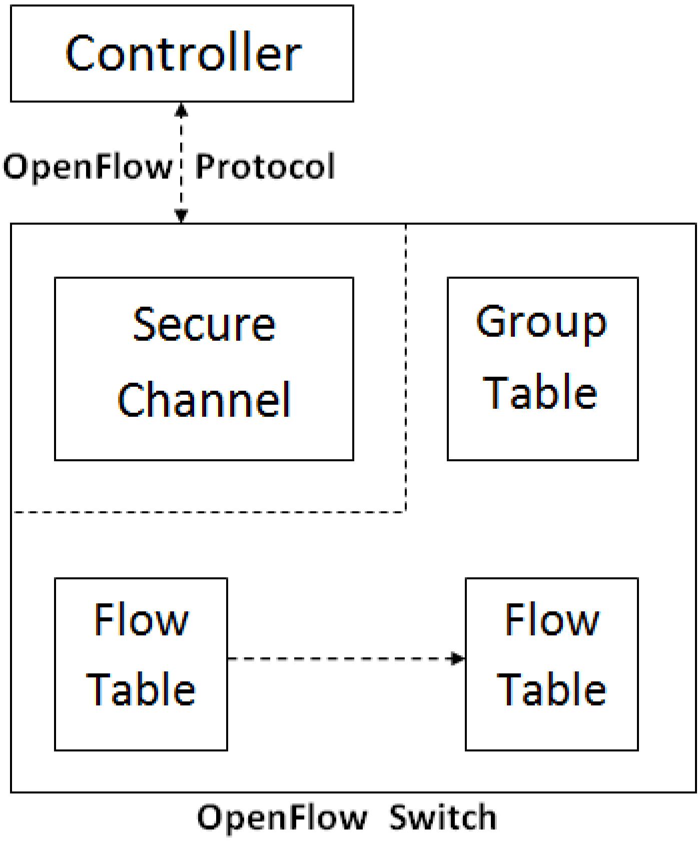
3

Figure 4: Fig. 3 :



4

Figure 5: Fig. 4 :



5

Figure 6: Fig. 5 :

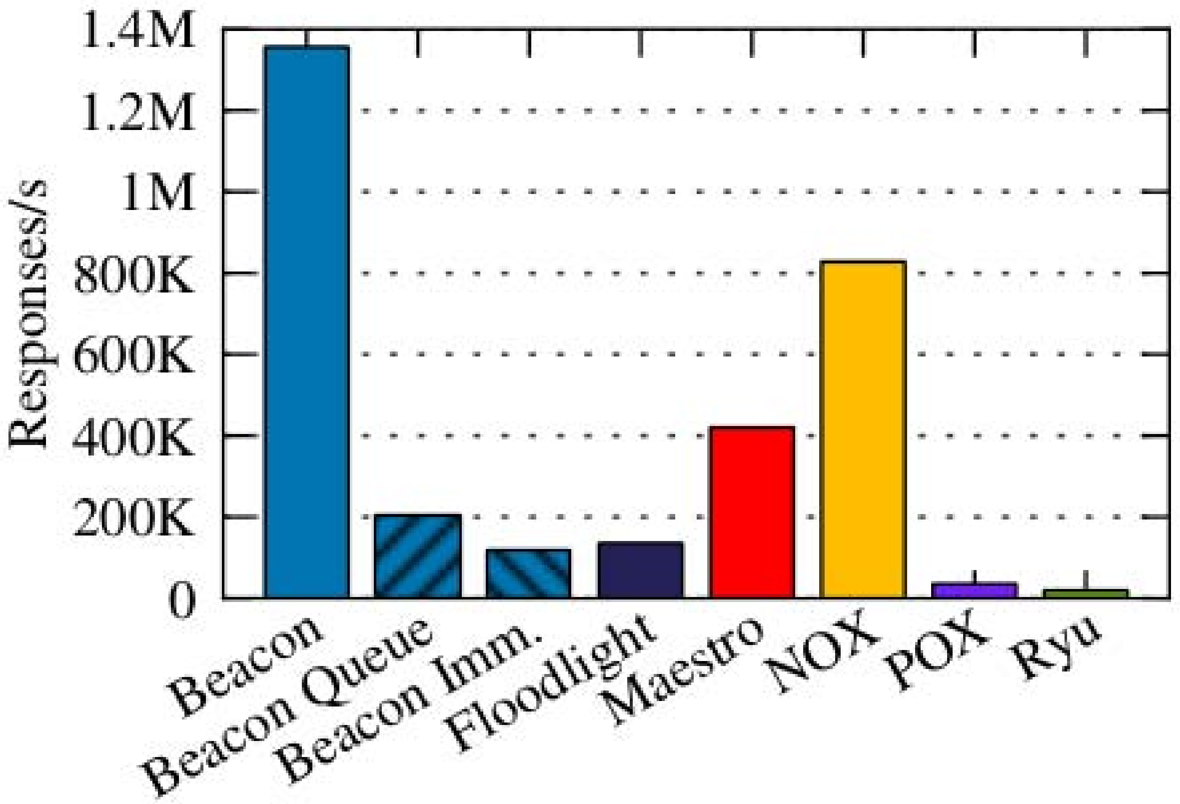


Figure 7:

I

Specification	1.0.0	1.1.0	1.2.0	1.3.0
Widely deployed	Yes	No	No	No
Flow tables	One	Multiple	Multiple	Multiple
Group tables	No	Yes	Yes	Yes
MPLS matching	No	Yes	Yes	Yes
Group tables	No	Yes	Yes	Yes
IPV6 Support	No	No	Yes	Yes
Simultaneous communication	No	No	Yes	Yes
III.				

Figure 8: Table I :

II

Name	Language	License	Original authors	Can Extend	Currently active	Notes
Ovs	C	OpenFlow license	Stanford/ Nicira	No	No	A reference controller, act as a learning switch
NOX	C++	GPL	Nicira	Yes	Yes	Event-based
POX	Python	GPL	Nicira	Yes	Yes	Event-based
Beacon	Java	GPL	Stanford	Yes	Yes	Web Interface, Regression test framework, Event based and Multi-thread based
Maestro	Java	LGPL	Rice	Yes	No	Multi-thread based
Trema	Ruby, C	GPL	NEC	Yes	No	Emulator and Regression test framework
Floodlight	Java	Apache	Big switch	Yes	Yes	REST APIs, Supports multi-tenant clouds

Figure 9: Table II :

III

Year 2 019
14

) (C

Software switch	Language	OpenFlow Version	Notes
OpenVSwitch	C, Python	V 1.0	Implements a switch platform in a virtua ized server environment. Supports standard Et ernet switching with VLANs and access contr lists. Provides interfaces for managing configur tion state and a method to remotely manipula the forwarding path.
Pantou/ OpenWRT	C	V 1.0	
ofsoftswitch13	C, C++	V 1.3	A user space software switch implementation. Th code is based on the Ericsson's Traffic Lab 1.1 soft switch implementation.

© 2019 Global Journals

Figure 10: Table III :

IV

Figure 11: Table IV :

V

Switch Company	Series	
Cisco	Cisco cat6k, catalyst 3750,6500 series	
Juniper	Juniper MX-240,T-640	
HP	HP pro-curve 5400zl,8200zl,6200zl,3500zl,6600	
NEC	NEC IP8800	
Pronto	Pronto 3240, 3290	
Dell Toroki Ciena	Dell Z9000 and S4810 Toroki Light switch 4810 Ciena Core-director running firmware version 6.1.1	Year 2 019
Quanta	Quanta LB4G	

Table VI: Classification of SDN debugging tools according to the layers they are working with

( ) C

(connectivity or consistency) that exist in the data plane. Violations of these invariants are considered as a bug in the network. Anteater translates the detected high-level network invariants into instances of boolean satisfiability problems (SAT). Then checks them against network state using an SAT solver. And finally, if violations have been found, it reports counter examples.

Figure 12: Table V :

Language	Supports	Type	Based on	Used for
FatTire	Only OpenFlow	-	Regular expressions	Fault tolerant programming
Nettle	Only OpenFlow	Functional	Functional Reactive Programming	Load balancing programming
FML	Only OpenFlow	Logical	datalog	Policy implementation programming
Procera	Any type of hard-ware	Functional	Functional Reactive Programming	General programming
Flog	Any type of hard-ware	Logical	datalog	General programming
Frenetic	Any type of hard-ware	Logical	Query language	General programming
Pyretic	Any type of hard-ware	Logical	Query language	General programming

Year 2 019

Layer	Tools
Application layer	NICE
Data Plane	Anteater
Control Plane	ndb, OFrewind
A new layer between Data Plane and Control Plane	VeriFlow

) (C

Figure 13:

.1 Acknowledgment

I would like to thank Prof. Jorge Lobo, Prof. A. Russo, Dr. Frank Le, Dr. C. Makaya and Prof. H. Ramalhino, who collaborated in all my SDN related research work [63], [64], [65], [66], [67], [68].

run by the Flux Group, part of the School of Computing at the University of Utah. The ns-3 [56] is a discrete event network simulator for internet systems. It is based on C++ and Python and widely used for research and educational use. Since ns-3 provides support for OpenFlow, it can be used to emulate SDNs.

[Rothenberg and Nascimento] , C E Rothenberg , M R Nascimento . (Revisiting routing)

[Ma et al.] *A compre-hensive study on load balancers for vnf chains horizontal scaling*, J Ma , W Rankothge , C Makaya , C Morales . arXiv:1810.03238.

[Lantz et al. ()] 'A network in a laptop: rapid prototyping for software-defined networks'. B Lantz , B Heller , N Mckeown . *ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[Canini and Venzano ()] 'A nice way to test openflow applications'. M Canini , D Venzano . *USENIX conference on Networked Systems Design and Implementation*, 2012.

[Tennenhouse and Smith ()] 'A survey of active network research'. D L Tennenhouse , J M Smith . *IEEE Communications Magazine* 1997.

[D ()] 'Active network vision and reality: lessons from a capsule-based system'. D . *ACM Operating Systems Review* 1999.

[Ma et al. ()] 'An overview of a load balancer architecture for vnf chains horizontal scaling'. J Ma , W Rankothge , C Makaya , C Morales , F Le , J Lobo . *IEEE IFIP CNSM*, 2018.

[Wang and Tsou] *Analysis of comparisons between openflow and forces draft-wang-forces-compareopenflowforces*, H J S X Wang , Z Tsou , T , YX . `http://tools.ietf.org/html/draft-wang-forces-compare-openflow-forces-01`

[Bozakov and Papadimitriou] 'Autoslice: automated and scalable slicing for software-defined networks'. Z Bozakov , P Papadimitriou . *CoNEXT Student 12*,

[Available] `http://https://www.coursera.org/course/sdn` *Available*,

[Sherwood and Chan ()] 'Carving research slices out of your production networks with openflow'. R Sherwood , M Chan . *ACM SIGCOMM* 2008.

[Menga ()] *Ccnp practical studies: Layer 3 switching*, J Menga . `http://www.ciscopress.com/articles/article.asp?p=102093` 2003.

[Santhanam ()] *Cisco support community: Cam vs tcam*, T Santhanam . `https://supportforums.cisco.com/docs/DOC-15833` 2011.

[Fernandez] 'Comparing openflow controller paradigms scalability: Reactive and proactive'. F M Fernandez . *AINA 2013*,

[Monsanto and Reich ()] 'Composing software defined networks'. C Monsanto , J Reich . *USENIX Conference on Networked Systems Design and Implementation*, 2013.

[Controller performance comparisons ()] *Controller performance comparisons*, `http://www.noxrepo.org/pox/about-pox/` 2011. 2012. (About pox)

[Rankothge et al.] *Data modelling for the eval-uation of virtualized network functions resource allocation algorithms*, W Rankothge , F Le , J Lobo . arXiv:1702.00369.

[Mai and Khurshid ()] 'Debugging the data plane with anteater'. H Mai , A Khurshid . *ACM SIGCOMM*, 2011.

[Caesar and Caldwell ()] *Design and implementation of a routing control platform*, M Caesar , D Caldwell . 2005.

[Emulab -network emulation testbed home ()] *Emulab -network emulation testbed home*, `http://www.emulab.net/` 2013.

[Casado ()] 'Ethane: Taking control of the enterprise'. M Casado , MJ F . *Computer Communication Review (ACM SIGCOMM)* 2007.

[Rankothge et al. ()] 'Experimental results on the use of genetic algorithms for scaling virtualized network functions'. W Rankothge , F Le , A Russo , J Lobo . *IEEE SDN/NFV*, 2015.

[Reitblatt and Canini ()] *Fattire: Declarative fault tolerance for software-defined networks*, M Reitblatt , M Canini . 2013.

[Johnson ()] *feature/Border-Gateway-Protocol-as-a-hybrid-SDN-protocol 30*, S Johnson . `http://searchsdn.southbound-SDN-protocol` 2012. 2012. (The role for xmpp as a southbound sdn protocol)

[Doria and Salim ()] *Forwarding and control element separation (forces) protocol specification*, A Doria , J H Salim . RFC 5810. 2010.

[Feamster et al. ()] 'How to lease the internet in your spare time'. N Feamster , L Gao , J Rexford . *ACM SIGCOMM*, 2007.

[Kim and Feamster ()] *Improving network management with software defined networking*, H Kim , N Feamster . 2011. (Open networking foundation)

[Bavier and Feamster ()] 'In vini veritas: Realistic and con-trolled network experimentation'. A Bavier , N Feamster . *Computer Communication Review* 2006.

[NF ()] 'Languages for software-defined networks'. NF . *IEEE Communications Magazine* 2013.

[Keller and Arora ()] *Live migration of an entire network and its hosts*, E Keller , D Arora . 2012. (in HotNets-XI)

[Kaia et al. ()] 'Logic programming for software-defined networks: Flog'. N P Kaia , J Rexford , D Walker . *ACM SIGPLAN Workshop on Cross-model Language Design and Implementation*, 2012.

[Cai and Cox ()] 'Maestro: A system for scalable openflow control'. Z Cai , A L Cox . *Tech. Rep* 2010.

[Voellmy and Hudak ()] 'Nettle: Functional reactive programming of openflow networks'. A Voellmy , P Hudak . *International Conference on Practical aspects of declarative languages*, 2011.

[Henderson et al. ()] 'Network simulations with the ns-3 simulator'. T R Henderson , M Lacage , G F Riley . *ACM SIGCOMM Workshop on Hot Topics in Networks*, 2008.

[Gude and Koponen ()] 'Nox: towards an operating system for networks'. N Gude , T Koponen . *ACM SIGCOMM* 2008.

[Wundsam and Levin ()] 'Of rewind: enabling record and re-play troubleshooting for networks'. A Wundsam , D Levin . *USENIX conference on USENIX annual technical conference*, 2011.

[Erickson ; A. Tootoonchian and Gorbunov ()] *On controller performance in softwaredefined networks*, D Erickson ; A. Tootoonchian , S Gorbunov . 2013. 2012. (The beacon openflow controller. in Hot-ICE)

[Open networking research center (onrc) ()] `http://onrc.net` *Open networking research center (onrc)*, 2011.

[Campbell and Katzela ()] 'Open signaling for atm, internet and mobile networks'. A T Campbell , I Katzela . *Computer Communication Review (ACM SIGCOMM)* 1998.

[Limoncelli ()] 'Openflow: a radical new idea in networking'. T A Limoncelli . `http://www.openflow.org/wp/current-deployments/23` *Open Networking Foundation, Tech. Rep* 2012. 2012. 2013. 22. (ACM Queue -Performance)

[Mckeown and Anderson ()] 'Openflow: enabling innovation in campus networks'. N Mckeown , T Anderson . *ACM SIGCOMM*, 2008.

[Rankothge and Le ()] 'Optimizing resources allocation for virtualized network functions in a cloud center using genetic algorithms'. W Rankothge , F Le . *IEEE TNSM*, 2017.

[Hinrichs and Gude ()] 'Practical declarative network management'. T L Hinrichs , N S Gude . *Proceedings of the 1st ACM Workshop on Research on enterprise networking*, (the 1st ACM Workshop on Research on enterprise networking) 2009.

[Voellmy et al. ()] *Procera: A language for high-level reactive network control*, A Voellmy , H Kim , N Feamster . 2012.

[Ryu : a component-based software-defined networking framework ()] *Ryu : a component-based software-defined networking framework*, `http://www.openflow.org/wk/index.php/Open-Flow1.0forOpenWRT39` 2012. 2012. 2012. 2011. (Pantou: openflow 1.0 for openwrt. ofsoftswitch13," 2011. [Online)

[Feamster ()] *Sdn course*, N Feamster . 2013.

[Sdn troubleshooting simulator (sts) ()] *Sdn troubleshooting simulator (sts)*, `http://ucb-sts.github.com/sts/` 2013.

[Schwartz and Jackson ()] 'Smart packets: applying active networks to network management'. B Schwartz , A W Jackson . *ACM Transactions on Computer Systems* 2000.

[Yu et al. ()] 'Software defined traffic measurement with opensketch'. M Yu , L Jose , R Miao . *USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[Shin et al.] *Softwaredefined networking (sdn): A reference architecture and open apis*, M.-K Shin , H.-J Kim , K.-H Nam . ICTC 2012.

[Lawser ()] 'Stored program controlled network: Generic network plan'. J J Lawser , Lecronier . *Bell System Technical Journal* 1982.

[Smith and Farber ()] 'Switchware: Accelerating network evolution'. J M Smith , D J Farber . *Tech. Rep* 1996.

[Jonathan and David ()] 'The open sdn architecture -big switch networks'. M S Jonathan , J F David . *Tech. Rep* 2011.

[Van ()] *The tempest: A practical framework for network programmability*, J E Van . 1998. IEEE Networks Magazine.

[Rankothge et al. ()] ‘Towards making network function virtualization a cloud computing service’. W Rankothge , J Ma , F Le , A Russo , J Lobo . *IEEE IM*, 2015.

[Yemini and Silva ()] ‘Towards programmable networks’. Y Yemini , S D Silva . *IEEE International Workshop on Distributed Systems: Operations and Management*, 1996.

[Khurshid and Zhou ()] *Veriflow: verifying network-wide invariants in real time*, A Khurshid , W Zhou . 2011.

[Nascimento and Rothenberg ()] ‘Virtual routers as a service: the routeflow approach leveraging software-defined networks’. M R Nascimento , C E Rothenberg . *Future Internet Technologies*, 2011.

[Handigol and Heller ()] *Where is the debugger for my software-defined network?*, N Handigol , B Heller . 2012. (in HotSDN)