Software Defined Networking concepts and challenges

Conference Paper · December 2016

DOI: 10.1109/ICCES.2016.7821979

3 authors, including:

Mohammad Mousa
Ain Shams University
2 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Ayman Bahaa-Eldin
Ain Shams University
62 PUBLICATIONS **255** CITATIONS
SEE PROFILE

Software Defined Networking Concepts and Challenges

Mohammad Mousa
Computers and Systems Eng. Dept.
Faculty of Engineering
Ain Shams University
mohammad.mousa@ieee.org

Ayman Bahaa-Eldin
Computers and Systems Eng. Dept.
Faculty of Engineering
Ain Shams University
ayman.bahaa@eng.asu.eg

Mohamed Sobh
Computers and Systems Eng. Dept.
Faculty of Engineering
Ain Shams University
mohamed.sobh@eng.asu.eg

***Abstract*—Software Defined Networking (SDN) is an emerging networking paradigm that greatly simplifies network management tasks. In addition, it opens the door for network innovation through a programmable flexible interface controlling the behavior of the entire network. In the opposite side, for decades traditional IP networks were very hard to manage, error prone and hard to introduce new functionalities. In this paper, we introduce the concepts & applications of SDN with a focus on the open research challenges in this new technology.**

I. Introduction

Traditional IP network protocols were designed to adopt a distributed control architecture where network devices should communicate with each other through a large set of network protocols to negotiate the exact network behavior based on the configuration of every individual device. Network devices are sold as closed components & network administrators are only able to change the parameters of different network protocols. Network administrators should translate high level network policies into low level scripts written for every individual device commonly known as the "Configuration Language". Also, every equipment vendor has its own configuration language with its own compliance to the large set of network standards leading to several inter-operability issues for the integration of equipment of different vendors. Also, each time a new functionality is needed (as a load balancer or a firewall), a new device is integrated to the network or perhaps one of the devices is upgraded to perform the new functionality. In this manner, traditional IP networks are neither flexible nor programmable by any means. In such a distributed, multi-vendor, multi-protocol & human-dependent environment, service creation & troubleshooting became a very hard task.

SDN has introduced a paradigm shift in the networking industry. Instead of having a distributed control architecture, it consolidates all the control in a single node called the "Network Controller" which is simply a software running on a commercial server platform. Network forwarding devices no longer participate in the network control & only forward packets based on the set of rules installed from the network controller. The network controller programs the forwarding rules of the forwarding devices through the "Openflow protocol" & hence the network forwarding devices are called "Openflow switches". Openflow is a standard protocol that's vendor independent & hence no specific knowledge of equipment vendor is needed to control the forwarding behavior. In order to implement a new network functionality only a new application is needed to be installed over the network controller while no change is needed from the forwarding devices side. Figure 1 explains the architecture difference between traditional network & a Software-Defined Network [4].

SDN has provided a very flexible interface for the creation of new services. Network programmers need to write their own network policies & services through a high level programming language (one example of network policies is load-balancing the traffic to a certain destination over multiple paths to avoid the congestion of a certain path). The network controller should be able to translate these high level programs into low level forwarding rules over individual forwarding devices. By the use of centralized high level programs to control the network behavior, network administrators are able to automate network tasks using programs written in high level general purpose programming languages like C++, java & python.

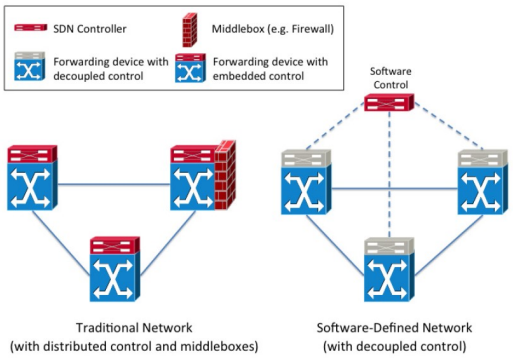

Figure 1 Architecture Difference between Traditional IP networks and Software-Defined Network [4]

II. History of Network Programmability

Openflow protocol was developed late 2008 by a group of researchers from different universities including Stanford University, University of Washington, MIT, Princeton University & other universities **[1].** As explained by the authors, the main target of the project was to enable researchers to test experimental network protocols in their campus networks. Later on, the idea was used in other domains such as datacenters with the emergence of cloud

computing & the need for a flexible programmable network. In 2012, SDN was adopted by Google to interconnect its datacenters spread around the world due to the great flexibility when using SDN for inter-data center traffic engineering.

Also SDN has emerged since only few years, many of the concepts adopted by SDN were developed over the last 2 decades. SDN is built around 3 main concepts which are: **Programmable Networks**, **Centralized Network Control** and **Control & Data planes separation**. In this section, we brief some of the previous work done in each area.

A. Programmable Networks

Programmable Networks is the concept to deploy new functionalities in network nodes by the use of a programming interface. In this networking paradigm, network nodes are not sold as closed "as-is" components. **Active Networks [2]** represents a very early trial for network programmability appearing in the 90's of the last century. It exposed the resources of individual network nodes (as processors, memory & packet queues) to be programmed for creating a new functionality for a specific pattern of packets. The code controlling the active nodes could be carried over the data packet itself sent form the end user (the Capsule Mode) or it could be sent from a separate dedicated management interface (the programmable router/switch model).

Active networks faced many critics related to network security as end users would be able to play with the network nodes supporting in-band active programming. Also, although active networks proposed a flexible interface for network innovation, it didn't find a compelling industry problem to solve perhaps due to the limited spread of the internet that time.

B. Centralized Network Control

Centralized network control is related to delegating a certain network function to a central node communicating with network nodes through a standard protocol. Below we introduce SNMP & NETCONF as a sample of the work done in this area.

SNMP: [3] SNMP stands for Simple Network Management Protocol. It's an IETF protocol proposed thirty years ago to have a unified interface for the management of network nodes including statistics collection, alarms collection & network configuration through a remote SNMP agent. SNMP is widely used for statistics & alarms collection while it's rarely used for configuration management due to having many limitations for the modeling of configuration information and other reliability & security issues. The protocol tried to overcome these issues in subsequent recent versions.

NETCONF: [4] NETCONF is an IETF protocol proposed in 2006 to automate the configuration of network devices through a standard Application Programming Interface (API). It's based on a data modeling language called YANG that overcomes the limitations found in SNMP. NETCONF is not an alternative to Openflow protocol as it only allows network administrators to configure their networks through a programmable interface but all functionalities & logic should be implemented first at the network device itself before being configured by NETCONF & hence it doesn't offer the capability to develop new functionalities as Openflow offers.

C. Control & Data planes Separation

Network devices are involved into two major tasks. First, the devices communicate with each other through standard network protocols to agree on the traffic paths (as the case of dynamic routing protocols) or to prevent network loops (as the case of the STP Protocol) or to perform any other functionalities depending on the used protocol. This task is commonly known as the **"Control Plane"** or the signaling plane. The output of this task is the data structures used to handle the traffic as the routing table found in network routers. Second, the devices start to forward the traffic based on the negotiations done in the first step. The action of forwarding the end user traffic (or even to block or any other action directly affecting the end user traffic) is known as the **"Data Plane"** or the forwarding plane.

Control & Data planes separation is the concept of having the control plane negotiations done in a separate node other than the node actually handling the end user traffic or the data plane. Of course both types of nodes will be communicating through a standard protocol. In this section, we introduce some of the work done in this area.

ForCES: [5] ForCES stands for Forwarding & Control Element Separation. It's an IETF protocol proposed since 2004 to separate the Control & Data plane elements within the network device internal infrastructure in order to allow the control element of a device to communicate with a third party forwarding element. Regardless of the separation, the network device will appear as a single entity to other network devices.

PCE: [6] PCE stands for Path Computing Element. It's an element proposed by the IETF in 2006 to delegate the function of calculating the path that traffic should follow in an MPLS network in order to comply with a set of user defined policies such as QoS assurance, loadbalancing or minimization of the WAN cost. Instead of having these calculations done & negotiated between edge routers, the function could be completely delegated to a standalone separate node.

Ethane: [7] Ethane project was the predecessor of openflow. The work was published in 2006 with the main target of building a flexible access control frame work for enterprise network. The frame work composed of two main components: the network controller where the access policies exist & Ethane switches controlled by the network controller. The controller compiles the access policies into flow table entries over the different switches to control either to forward or to block a certain traffic flow.

III. SDN ARCHITECTURE

SDN architecture could be divided into four layers as shown in Figure 2. In this section, we explain the first three layers while layer four (SDN Applications) is introduced in section IV due to the large set of available applications. We explain the functions & interfaces of each layer together with a sample of the ongoing research work.

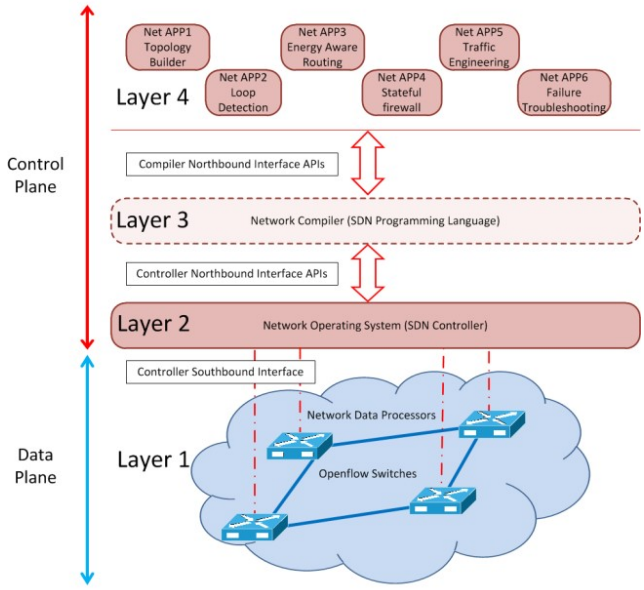

Figure 2 Layered architecture of an SDN network

A. Layer 1: Network Data Processors

In this layer data processing devices exist. Data packets are handled based on the actions installed from the network controller in every individual device. Openflow is the standard protocol that is used for the communication between the network controller & forwarding devices in order to install the data processing rules. As shown in Figure 3, an openflow rule consists of four main parts: priority, matching condition, action & related active counters. Priority field is used to define the order of matching a data packet against the set of installed rules & hence once a higher priority rule is matched other lower priority rules are ignored. Matching condition consists of any combination of several IP/Ethernet headers (such as source IP, destination IP, port numbers & VLAN id). The action field defines the action that should be taken upon receiving the packet such as forwarding the packet to an outgoing interface or to modify some of the header fields before forwarding or to drop the packet. Finally the counters field defines the associated counters to this rule such as the count of the number of matching packets in order to send the values to the controller to get some statistics about the data plane. Openflow has passed many revisions with the addition of many new matching conditions & actions in order to satisfy more complicated use cases. Major changes between different revisions are shown in Table 1. In the following, we highlight some of the open research challenges in this layer.

1) Software Switching: Software switching is related to using a software running over a linux OS for openflow switching. It's commonly used in cloud computing to connect multiple VMs in the same host to the same external L2 network. **Open vSwitch [9]** is an open-source multi-layer switch for software switching in a virtualized environment. It shows increased flow rate of 25% compared to a linux kernel-based software switch. Suggested future work is related to supporting new functionalities such as Network-Address Translation (NAT), Deep Packet Inspection (DPI) and stateful flow processing. Also, another research direction is related to decreasing the software switching delay.

Figure 3 Structure of an Openflow rule

OpenFlow Version	Year	Introduced Match Fields	Statistics
v 1.0	2009	Ingress Port	Per table statistics
		Ethernet: Source, Destination, Type, VLAN ID	Per flow statistics
		IP: Source, Destination, Protocol, ToS	Per port statistics
		TCP/UDP: Source Port, Destination Port	Per queue statistics
v 1.1	2011	MetaData, SCTP	Group statistics
		MPLS: Lablel, traffic Class	Action bucket statistics
v 1.2	2011	Openflow extensible match	
		IPv6: Source, Destination, flow label	
v 1.3	2012	IPv6 extension headers, Provider Backbone Bridging tagging	Per-flow meter
v 1.4	2013		Optical Port Properties
v 1.5	2014	TCP flags	Extensible flow entry statistics

Table 1 A Comparison between different versions of openflow protocol (reproduced from [8] with changes)

2) Southbound Interface Protocols: While openflow is the most widely used southbound interface protocol, other protocols already exist as OVSDB. **OVSDB [10]** stands for Open vSwitch Database Management Protocol. It's intended to provide a programmable interface to Open vSwitch to support advanced features that are not currently supported by openflow while openflow is still used for other operations. Compared to Openflow, OVSDB management operations occur on a relatively long timescale. Examples include: the creation of virtual bridges where each bridge is considered as a separate virtual switch controlled by a separate openflow controller, the configuration of quality of service policies over interfaces with the associated traffic queues, the configuration of tunnels for the communication between virtual machines residing in separate hosts & finally statistics collection. Suggested future work is related to the cooperation between OVSDB & openflow for better network control.

3) Network Slicing: Network slicing is related to slice the forwarding infrastructure to allow multiple logical networks over the same physical forwarding devices. **FlowVisor [11]** is one of the early technologies for SDN slicing by acting as a

proxy between the controller & the openflow switches allowing each slice to be controlled by a separate controller. **libNetVirt [12]** is another network virtualization effort. It's written as a network virtualization library similar to the libvirt library in computer virtualization. It has the advantage of being able to communicate with other southbound interfaces other than openflow (as MPLS) for providing network slicing in a legacy network by creating agents to communicate with libNetvirt driver & use CLI scripts or NETCONF to configure the legacy element. Suggested future work is related to creating business applications depending of the benefits offered by network slicing.

B. Layer 2: Network Operating System

In this layer, the network control exists and hence it's called the **"Network Controller"** or "The Network Brain". A controller communicates with the data forwarding devices to install, update & remove openflow rules based on the logic defined by the running applications. As the operating system of a normal computer provides functions such as resources management & file system management, the SDN controller provides similar functions to the SDN applications. There exist many open source SDN controllers supporting different openflow versions, programming interfaces & finally providing different services for the running applications. Due to the centralized nature of the controller, it suffers from many reliability, scalability, security and performance issues. It's noted that there's no standards defining the interfaces or the services offered by the SDN controller to the network applications & hence many challenges exist in this layer. In the following, we highlight some of the open research challenges in this layer.

1) Controller Placement Problem: Controller placement plays an important role due imposing a limitation over the processing speed of the new flows (as it's common that the switch consults the controller for every new flow to avoid overwhelming the switch memory with unused rules). In **[13]** the authors tried to find the optimum number of controllers & their locations in order to minimize the communication latency between the controller & the openflow switches. They have shown the results when optimizing for two different performance metrics: the average latency time & the worst case latency time. Simulation results showed that that random placement is around double the delay compared to the optimal placement. Also, it's shown that a single controller is sufficient to satisfy the response time requirements for medium-size topologies.

2) Centralized vs. Distributed Controller: A single controller represents a single point of failure & a performance bottleneck due to the increased delay for the communication with remote switches. To satisfy high-availability & fault tolerant requirements, a distributed controller model is proposed. In the opposite side, a distributed controller will suffer from distributed systems issues such as state distribution, consistency & increased programming complexity. **Onix [14]** proposed the design of a logically centralized controller over a distributed physical controllers taking care of the state distribution between individual controllers. **HyperFlow [15]** provides advanced features for distributed controllers such as the localization of flow decisions to individual controllers & hence it minimizes the communication latency. Also, it provides resilience to network partitioning due to individual controller failures & finally it enables interconnecting individually managed openflow networks.

3) Controller Services & Application portability: Traditional operating systems provides abstractions in the form of APIs (Application Programming Interface) to provide access to shared resources (as CPU scheduling) and allow the communication with external devices using the appropriate driver. Network Operating Systems provides a similar function to network applications. While there're standards for the APIs & services provided by traditional operating systems such as the POSIX family of standards, no similar function exists in Network operating Systems. Every controller provides its own set of APIs & SDN applications are written for a specific network controller. **NOSIX [16]** proposed a lightweight SDN application portability layer. However, NOSIX is not a generic northbound interface abstraction layer, but rather a higher layer abstraction of the flow tables of openflow switches. In order to achieve more maturity in this point, more applications should be developed & being used commercially to settle on a widely accepted set of controller services. We propose the use of SDN programming languages & their associated run-time environment as a solution for these inter-operability issues as discussed in the following section. In Table 2, a comparison is shown between many available network controllers with their supported features.

C. Layer 3: Network Compiler

In this layer, the network applications are written using SDN specific programming languages & associated compilers are used to translate the application code to the correct API supported by the used network controller or SDN operating system. While using this layer is not common (as most applications are written directly using the controller APIs), **we suggest using this layer for the following reasons:**

1) Application portability: Due to the variety of network controllers & associated APIs, applications portability is not allowed with current SDN networks. Using a network programming language, only using the correct network compiler would be sufficient in order to guarantee application portability between different controllers. Of course this imply that a compiler software should exists for the target network operating system but it's only a one-time job & would be done by the compiler authors for most of the widely used network operating systems.

2) High level of network abstraction: Using a high level programming language implies that the developer should take care of the programming logic while leaving low level operations to the programming language. This implies that the developer should specify high level policies while the compiler should be able to analyze these policies & generate related openflow rules to be installed over individual switches without bothering the developer with this level of details. In this way, network innovation is becoming much easier.

3) Code reusability: While most controllers offer a direct programming interface, they don't offer a conflict free execution of composite code due to the low level nature of the exposed programming APIs. As suggested in **Frenetic [17],** the sequential execution of 2 simple programs over a NOX controller (one program forwards the packets from an interface to another interface while the other program measures the web

traffic entering the same interface) will lead to completely incorrect results & a third program should be developed to combine the executions of the two programs. The result is expected due to having both programs installing overlapping rules, while the switch will execute only the higher priority matching rule. As a result a programming language is needed to allow the reuse of developed applications & libraries without the need to adjust the code for correct execution.

In the following, **we survey some of the challenges related to this layer:**

1) Programming Language Design: Many features are desired to exist in an SDN programming language. **Frenetic [17]** introduced a collection of high-level compositional operators for querying and transforming streams of network traffic. A run-time system handles all of the details related to installing and uninstalling low-level rules. **FlowLog [18]** provided extra features such as a stateful rule-based language to write middle box applications as stateful firewalls & application loadbalancers.

SDN Controller	Architecture	Prog. Lang.	Supported openflow version	License
OpenDayLight	Distributed	Java	v1.0 & v1.3	EPL v1.0
Hyperflow	Distributed	C++	v1.0	No License
ONOS	Distributed	Java	v1.0	No License
Onix	Distributed	Python,C	v1.0	Commercial
Ryu NOS	Centralized, Multi-Threaded	Python	v1.0, v1.2 & v1.3	Apache 2.0
Floodlight	Centralized, Multi-Threaded	Java	v1.1	Apache
Beacon	Centralized, Multi-Threaded	Java	v1.0	GPLv2
NOX-MT	Centralized, Multi-Threaded	C++	v1.0	GPLv3
NOX	Centralized	C++	v1.0	GPLv3
POX	Centralized	Python	v1.0	GPLv3

Table 2 A Comparison between different SDN controllers (reproduced from [8] with changes)

2) Big Switch abstraction & Rule Placement problem: Rule placement problem is related to simplifying the design of SDN applications by the abstraction of the whole switching network as a single big switch. It's the role of the SDN Compiler to efficiently distribute the rules over the switches along the path between the source & the destination in order to decrease the number of total installed rules & according to the rule capacity of individual switches. Apparently, it's an optimization problem. **One Big Switch [19]** proposed a solution for this problem by dividing the network into paths and get the union of all rules over each path. It then divides the total rule capacity between paths according to the number of path policies. Finally, it tries to find a feasible solution for each path according to the granted rule capacity.

D. Cross Layer Challenges

1) Testing, Prototyping & Simulation: Testing & simulation represents a basic stage in the software development process. Many tools were proposed to simulate the behavior of openflow switches to simulate the forwarding plane. **Mininet [20]** is a system for rapidly prototyping large SDN networks on the limited resources of a single laptop. Using OS-level virtualization features, allows the virtualization to scale to hundreds of SDN switching nodes. **fs-sdn [21]** is another tools that performs simulation using flowlet instead of a packet (a flowlet refers to the data volume emitted by an interface in a certain time period for example 100 ms). It could simulate large networks of switches with more accurate results compared to mininet. Controller simulation is a much simpler step as a controller is a pure piece of software & could run over a personal computer as long as it meets its minimum hardware & software requirements.

2) Fault Troubleshooting: Currently network troubleshooting is done in a very primitive way. Very elementary tools (such as ping, traceroute & SNMP statistics) are used to discover network forwarding faults making fault discovery a very hard problem for large networks specially when the problem is due to a mal-functioning node or a configuration mistake. Many of the following proposals could be used to troubleshoot SDN & legacy networks forwarding faults.

XTrace [22] suggested a solution to this problem by using a generic tracing functionality over multiple technologies (Routing, encrypted tunnels, VPNs, NATing & middles boxes) by inserting its own metadata in all network layers along the path. The main problem with this technique is related to degraded performance of network nodes due to the processing & propagation of the metadata. Interestingly, the technique is quite similar to software debuggers used in programming languages.

NetReplay [23] proposed another technique for discovering routers causing packet loss by observing TCP retransmits. A packet is tagged with the ID of the first router processed this packet while routers don't tag packets if they have processed the packet in the near past. As a result, a retransmitted packet will be tagged with the router ID of the router dropped it the first time. Concerns related to this technique are related to being limited to packet loss while failing to find problems related to out-of-order delivery, also not being able to find routing problems & finally not being able to find abnormalities related to UDP traffic such as VOIP traffic (where there's no retransmits).

NetCheck [24] proposed an algorithm to detect network applications issues (such as skype bad performance) through the analysis of the application system calls from the relevant end hosts. By the correlation of the events from end hosts, it could discover issues such as MTU size mismatch, IP changes due to passing through a NAT device & UDP packets truncation due to using default buffer size settings. Limitations of this algorithm are related to not being able to detect faults

not impacting application system calls. Also, detection engine could be enhanced using machine learning by comparing application system calls pattern with previously observed patterns of correct & incorrect behaviors.

3) SDN software Debugging: In addition to the complexity of troubleshooting forwarding problems (which is common with traditional IP networks), new challenges arise in SDN due to the need to debug network applications software. **Ndb [25]** proposes a debugger for SDN applications to debug network applications exactly as the case of normal software. The debugger utility is able to record the network state changes & to trace the sequence of events leading to exceptions to find their root causes. The ndb debugger lets the user define a packet breakpoint (such as unforwarded packet or a packet hitting a certain rule) & provides the user with the set of forwarding actions seen by the packet leading to the breakpoint. The authors showed how their debugger could be used to detect errors related to race conditions, controller logic & switch implementation. Current implementation is not able to find network performance abnormalities such as non-line-rate forwarding or traffic in-balance.

NICE [26] is an automated debugging tool to discover SDN software bugs through investigating the state space of the entire system through model checking (i.e. to explore all transitions between all states & if the new state satisfies the defined correctness properties). Examples of correctness properties are loop-free & back-hole-free forwarding. Due to the large state space, state transition input (i.e. network packets those trigger certain state transition) is reduced using symbolic execution of the packet event handlers (symbolic execution is used to get which packets are going to change the code execution flow & trigger different execution paths or in other words will trigger certain state transition).

4) SDN Forwarding Validation: Anteater [27] focused on discovering forwarding bugs by modeling the network state (i.e. installed forwarding rules) together with the network bugs under investigations (such as routing loops, forwarding inconsistency & network cuts) as a Boolean satisfiability problem. The tool is going to use an on-the-shelf SAT solver to solve the problem & report if any of the bugs are applicable or not to the network under investigation. Also, the tool provides one of the feasible solutions to the satisfiability problem (i.e. the network state that's triggering the forwarding problem) to help the administrator to check the problem reason. The main problem with this technique is related to the long time needed to check the correctness properties of a certain network state.

Veriflow [28] proposed another technique for real-time forwarding bugs discovery. Veriflow lies in the middle between the network controller & the openflow switch to check all rule installation or modification APIs in real time. Veriflow is going to send the rule to the switch only if the rule doesn't trigger any violation to the forwarding correctness properties. Veriflow uses Equivalent Classes to classify packets according to their forwarding path. For each Equivalent Class, a forwarding graph is build showing the sequence of traversed nodes. Finally, all forwarding paths are checked versus stored network bugs (such as connectivity issues, asymmetric routing or routing backholes). Whenever, a rule is received, Veriflow checks the modified equivalent class of the matching packets & only repeat the previous algorithm if the equivalent class has not passed the validation process before. Due to such incremental nature, Veriflow is used for real time forwarding validation.

IV. SDN Applications

In this section, we survey some of the applications developed in five major SDN applications categories which are: Hybrid Network applications, Traffic Engineering applications, Data Center networking applications, Mobility and Wireless applications & finally Network Secuirty applications.

A. Hybrid Network Applications

While SDN networking offers a lot of benefits for network operators in terms of flexibility & programmability, migration to SDN is still in a slow pace. It's observed that only Huge Service Providers such as Google **[29]** has migrated their globally deployed network to be SDN based. The main reason for this slow deployment rate is related to the high CapEx needed to swap traditional network equipments with openflow enabled switches. Also, the lake of experience with the new technology & the need to train network administrators with a new set of skills (such as network programming) represents another challenge. One of the most prominent solutions to this problem is the incremental migration strategy. While it's very logical to swap out-of-support nodes with SDN nodes, such hybrid network comes with its own set of new challenges.

In **[30]** the authors suggested the classification of networks into 2 types: SDN networks (SDN) & Traditional or COTS-based Networks (CN) where network components are sold as COTS products (Component On The Shelf). Hybrid SDN was suggested to have four forms according to the integration way between both types of networks. Topology-based integration was proposed where both types of networks exist in different geographical areas. Service-based integration was proposed where both types of networks exist together in the same network devices & each network handles a different type of services. Class-based integration was proposed where both types of networks implement all the services but for a certain traffic class such as using SDN for the routing of a delay sensitive application & relaying on CN for the routing of other applications. Finally, Integrated SDN where the network forwarding devices are kept using the current routing protocols & the SDN controller uses current legacy protocols to control the forwarding behavior of the traditional networking devices. Proposed research points are related to the identification & implementation of services through the cooperation between these paradigms & tools to solve the added complexity of such hybrid paradigms. Figure 4 summarizes the four integration strategies.

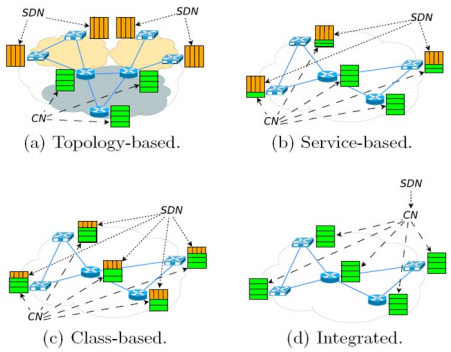

Figure 4 Figure 4 Types of hybrid SDN networks [30]

Fibbing [31] suggested the use of augmented topology to control the routing process in IGP routing protocols as OSPF & ISIS by inserting fake entries into the topology database of the routing protocol. This technique was shown to achieve results which are not possible using pure IGP protocols such as the case of traffic distribution over unequal cost paths (as an IGP always chooses the lowest cost path). Figure 5 shows the concept of topology augmentation. The algorithm paths through three steps: First, Path Compilation into a DAG (Directed Acyclic Graph), then Topology Augmentation to find the fake topology that satisfies the path requirements & finally the Topology Optimization to optimize the augmented topology in terms of the number of augmented routers. Figure 6 explains the main stages of the proposed algorithm. More work is suggested in the cooperation between openflow & legacy devices in a hybrid SDN deployment.

OpenRouteFlow [32] proposed to perform software upgrade to legacy routers operating system in order to support openflow while keeping the same hardware. While it seems a good suggestion for the migration scenarios, such assumption is not guaranteed to work properly with no performance degradation. Also, the paper has proposed a practical use case of how to control the network behavior using this hybrid control model.

Exodus [33] is a tool for automatic migration of enterprise networks to SDN architecture. The tool takes the configuration files of the legacy nodes as its input & produces the controller application of the equivalent SDN network based on flowlog network programming language (briefly explained in section III). The tool supports a group of non-trivial configuration items such as static & dynamic routing, ACL, NAT & VLAN. Proposed future work is related to optimizing the auto generated application by adopting the One Big Switch abstraction (briefly explained in section III). Also, the concept of auto migration could be generalized to include Service Provider network protocols such as MPLS & BGP.

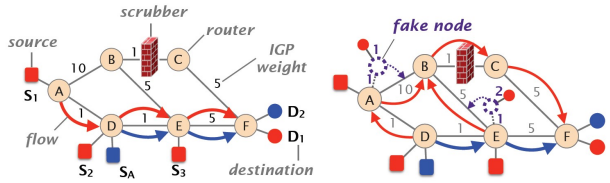

Figure 5 The concept of augmented topology [31]

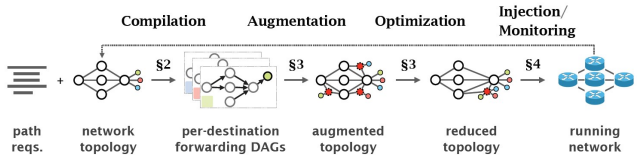

Figure 6 Fibbing algorithm [31]

ClosedFlow [34] proposed to provide openflow-like functionalities while using legacy switches & routers. The authors proposed to use a central controller that communicates with legacy switches & routers through vendor defined CLI shell. They have successfully implemented 4 types of openflow capabilities over CISCO switches which are: connecting the controller to the switches, automatic discovery of the network topology, allowing the controller to add or to modify the flow table entries of the switches (using cisco route map & access-list data structures to specify the flow matching conditions & associated action) & finally allowing the send-to-controller action whenever a packet is received without matching any of the specified flow table entries. The work could be extended by allowing other openflow functionalities such as statistics reporting by using the well-known SNMP protocol. Also, there should be a compiler that compiles openflow APIs to vendor specific CLI commands. Finally, there should be some benchmarking through the use of a real openflow application & comparing the performance of the Closedflow with native openflow switches.

Panopticon [35] proposed a framework for deploying & operating a network consisting of traditional & SDN switches while benefiting from the simplified management & flexibility of SDN. The framework proposes a logical SDN network over a partially upgraded traditional network & hence extending SDN benefits to the entire network. Based on the performance evaluation, the authors were able to operate the entire network as a single SDN network while upgrading 30% to 40% of the network traditional distribution switches. Panopticon uses a mechanism called Solitary Confinement Tree (SCT) which uses VLANs over legacy switches to ensure that traffic passes through at least one SDN switch where traffic end-to-end policies are deployed. Network is divided into cell blocks to allow VLAN reuse in different cell blocks. Cell blocks are divided by SDN switches (called Frontier). It also uses an Inter-Switch Fabric (ISF) which is a virtual tunnel for logically connecting two SDN switches. Using the previous technique, each access port of a legacy switch is controlled by a certain SDN switch & seen as being connected to the SDN switch from the network logical view. Figure 7 shows the proposed architecture. Panopticon adopts the "Big Switch Abstraction" **[19]** concept explained previously in section III. Network administrators are supposed to specify their end-to-end traffic policies (such as traffic path, load balancing or dropping certain type of traffic) & Panopticon will deploy the needed SDN application with the associated VLAN configuration over the legacy switches. Future work is related to the development of network applications utilizing the proposed framework. Also, due to the increased complexity of the network topology, there's a need for a troubleshooting application on the top of the proposed framework.

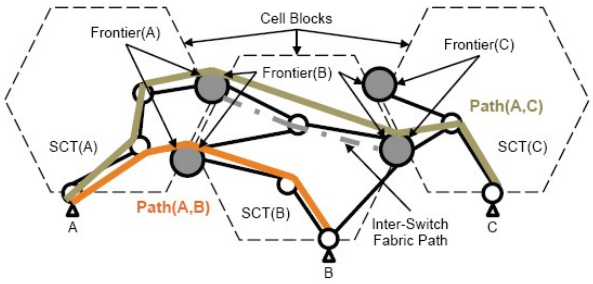

Figure 7 Panopticon proposed logical SDN network [35]

LegacyFlow [36] proposed another topology where openflow enabled switches are used on the edge of the network while using traditional switches at the core of the network. Edge openflow switches will be connected to each other through virtual tunnels & rely on LegacyFlow to make necessary tunnel configurations over the traditional core switches.

Tunnels could be as simple as a VLAN or using more advanced technologies as MPLS L2 VPN. The main advantage over Panopticon is the simplified network architecture with the restriction to completely swap all edge devices to be openflow switches leading to less flexibility in the migration plan.

RFCP [37] or RouteFlow Control Platform is a frame work for the inter-domain routing control of a service provider hybrid SDN network. A control node referred to as RFCP acts as a gateway between traditional iBGP route reflectors & opneflow controllers controlling the openflow forwarding devices. Using this hybrid model the authors were able to demonstrate new use cases those are hard to achieve using only iBGP such as calculating best BGP paths from the point of view of each ingress point, installing redundant paths using openflow multi forwarding table for fast recovery after path failure & finally providing advanced distributed security services through using the flexibility of openflow switches. Figure 8 shows the RFCP architecture.

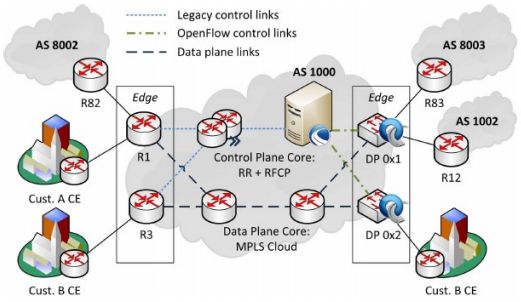

Figure 8 RouteFlow Control Platform Architecture [37]

B. Traffic Engineering Applications

Traffic engineering is related to the dynamic monitoring & control of the network in order to achieve high level design objectives such as satisfying differentiated service delay requirements, fast failure recovery & maximizing the traffic that could be served by the network. Using business terminology, traffic engineering or TE is related to service continuity & cost optimization. TE has emerged since the early days of ATM networks, passing through routing protocol TE (such as OSPF loadbalancing between equal cost paths) & currently MPLS TE is the standard TE solution in a Service Provider network. MPLS based TE sufferers from many limitations such as complexity of configuration (specially the need to define backup paths or LSPs manually), scalability & robustness limitations **[38]**. Due to the inherited flexibility & programmability, SDN is regarded as the future TE solution. Figure 9 shows the evolution of Traffic Engineering techniques.

Google [29] proposed the design of advanced traffic engineering mechanism through using SDN for the routing between the company data centers distributed around the globe. The proposed SDN controller is a modified version of the ONIX controller**.** Through the proposed design, the company were able to drive links to near 100% utilization for extended period (during WAN failures) while distributing the traffic among several paths & taking care of the bandwidth allocation to meet different application requirements. Quagga was used as an open source BGP/ISIS routing application running over the controller to communicate with existing routing protocols to support hybrid network deployment. Future proposed work is related to solve bottlenecks for the communication between the controller & the switches.

In **[39]** the authors suggested an algorithm for choosing which legacy routers to be migrated first to SDN paradigm in order to maximize the benefits from traffic engineering point of view. The number of newly introduced alternative paths was used as the target maximization function compared to the least cost paths used by legacy routers running link-state protocols such as OSPF & ISIS. Simulation results showed that proper choice of the first router to upgrade could make capacity savings up to 16% compared to 9% when choosing a random router for the upgrade.

Authors of **[40]** proposed a technique that uses openflow switches as backup paths for the recovery of any single link failure in a legacy network while taking care not to congest the new paths by the extra traffic through dynamic traffic distribution between backup paths. For the proposed design, as SDN controller is not aware by the status of the legacy network a virtual tunnel is used between every router & its backup SDN switch in order to identify the failing link. The work could be enhanced by directly integrating the legacy network with the controller to have a holistic view over the network. Also, a mechanism should be proposed to deal with multiple link failures & associated capacity problem while differentiating between different services.

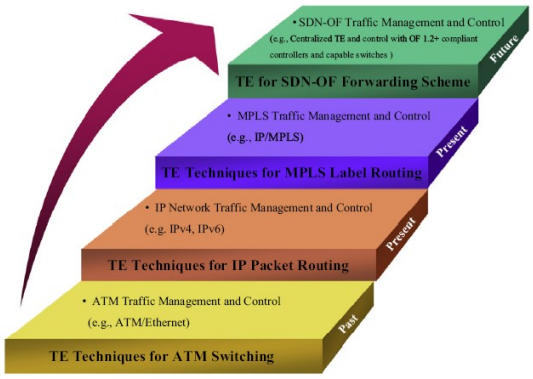

Figure 9 Evolution of Traffic Engineering [38]

In **[41]** the authors proposed a system for the incremental deployment of hybrid SDN network & showed how to use such hybrid system to satisfy a variety of traffic engineering goals such as loadbalancing & fast failure recovery. The system proposes a migration path in order to maximize the traffic engineering benefits due to the initial upgrade steps. Simulation results showed a maximum link utilization reduction to an average of 32% (compared to least cost routing) while upgrading only 20% of the network. The authors went a step further showing that hybrid topology has its own benefits compared to pure SDN deployment showing that previous results required only 41% of the flow table capacity required by a pure SDN deployment. Future work is related to studying the scalability issues of the proposed system, using historical traffic demands to decide the SDN migration plan & finally the effect of frequent rule updates on the network consistency & related traffic engineering performance.

C. Data Center Applications

In **[42]** the authors proposed the design of a large-scale low-cost multi-tenant data center. A multi-tenant data center is a one that's shared between different users while every user is not aware by the other users. Network slicing is commonly used to indicate the same meaning. The designed network uses a combination of openflow switches & low cost ethernet switches in order to decrease the network cost. Dual SDN controllers are used for achieving the needed network resilience. Up to 64K tenant could share the same data center with each tenant can individually assign IP addresses & VLAN IDs to his own virtual machines. The design also supports virtual machine live migration, fast failure recovery, network load balancing & multi-path TCP to provide multiple sub-flows for increased throughput of TCP connections between virtual machines.

In **[43]** the authors tackled the problem of decreasing data center network energy consumption in low traffic periods while sustaining the same network performance. They have developed an energy-aware routing application based on the OpenNaaS network manager which could run over several SDN controllers such as OpenDayLight, RYU & Floodlight controllers.

In **[44]** the authors suggested to use the SDN controller to update the look-up tables of an optical packet switching (OPS) node used for high speed data center switching. They were able to create a virtual data center network which benefits from the wire speed switching of the OPS node with the flexibility of SDN technology. Openflow was extended to support the OPS switching paradigm. The setup was used to test many SDN features such as QoS guarantee for high priority data flows & successful network slicing for intra data center routing.

In **[45]** the authors tackled the problem of uneven traffic distribution over the servers of a data center. They suggested using an SDN controlled loadbalancer between the data center servers & the switches connecting the servers to the external network. SDN is used to collect statistics from servers & switches to learn the traffic distribution & server loads and accordingly to perform load distribution between the servers while guaranteeing QoS for high priority data flows. Even distribution of the traffic has the impact of decreasing the average data latency.

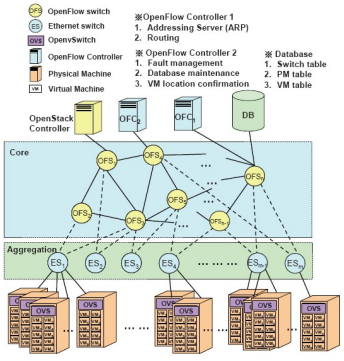

Figure 10 A Multi-tenant Data Center Design as proposed in [42]

D. Mobility & Wireless Applications

In **[46]** the authors investigated the function placement problem of the Gateways (LTE SGW & PGW) of a Mobile operator offering LTE services. As explained by the authors, the function placement problem is either to choose to deploy LTE gateways as a fully virtualized service where both the control & data planes are processed in the same node over a datacenter or as an SDN solution where the data plane is kept at the transport network element which could be programmed through an extended version of openflow supporting LTE data plane functionalities such as GTP tunneling & traffic charging functions. The virtualized solution has the advantage of simplicity with the cost of introducing additional traffic delay due to processing the traffic over the Data Center commodity servers. The SDN solution has the advantage of minimal delay with the cost of increased complexity & introducing additional network load for the openflow communication between the transport network element & the virtualized controller. The authors developed a model that minimizes the additional network traffic (as a result of openflow signaling) for a certain data plane delay budget. Figure 11 shows the different design alternatives of the GW function placement problem where GW-c indicates the control (or the signaling) part of the GW function, GW-u indicates the user data traffic handled by the GW function & NE is the external network entity routing the signaling & the user traffic to the GW.

In **KLEIN [47]** the authors investigated the problem of the re-design of a mobile operator core network with the target of more elasticity with minimal disruption to the existing network. The authors proposed a resource management algorithm that's able to manage thousands of sites while working within the operational constraints of existing 3GPP standards. Combining the algorithm with a distributed mobile core network, the algorithm showed an enhanced network performance, better load distribution & an elastic fault recovery. Figure 12 shows the architecture of the proposed resource management framework.

In **[48]** the authors showed how to use SDN concepts to offload the core of a mobile operator network. Their framework suggests to route peer-to-peer traffic directly between access sites without passing through the core network which improves the user experience by decreasing latency & decreases the load over the mobile core network. Suggested future work is related to supporting other functionalities over SDN switches connecting different access sites such as charging functions & deep packet inspection in order to be able to deploy the framework in a life network.

In **[49]** the authors suggested the use of SDN for WSN (wireless sensor network). They explained several SDN-based WSN use cases such as centralized networking, optimal code deployment & predicting WSN performance. Future work is related to having prototypes of the suggested use cases & to study the trade-offs between using SDN architecture & energy efficiency of the whole network.

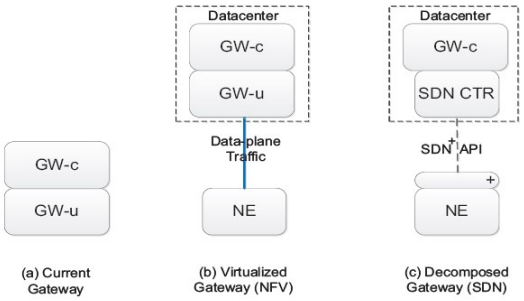

Figure 11 LTE GW function placement alternatives [46]

In **[50]** the authors suggested the use of SDN in vehicular ad-hoc networks (VANETs). They explained several use cases such as safety applications, surveillance applications & wireless network virtualization. They also compared the performance of SDN based routing to the performance of traditional VANET/MANET routing protocols showing that SDN outperforms other Ad-hoc routing protocols mainly due to the knowledge about the whole network state & hence the fast failure recovery. Future work is related to exploring more SDN applications in VANETs & studying more complicated architecture such as allowing the controller-to-sensor (the sensor could be either the Vehicle or the Road Side Unit) communication in a Peer-to-Peer mode. Figure 13 explains the proposed architecture.

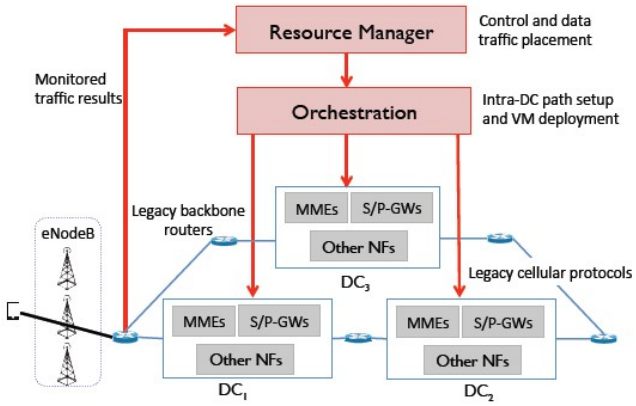

Figure 12 The Mobile Core network orchestration framework proposed in [47]

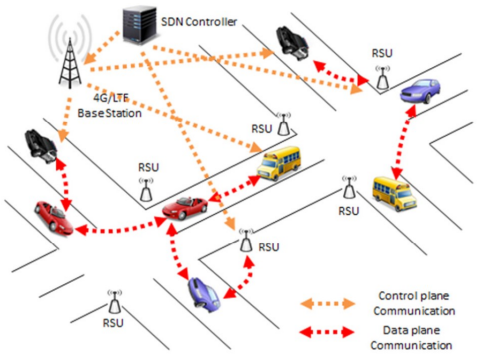

Figure 13 SDN-based VANET architecture as proposed in [50]

E. Security Applications

NICE [51] is a framework for cloud security by preventing vulnerable virtual machines from being compromised in the cloud & off-course securing the cloud against cloud zombies (which would be used in later stages to perform DDOS attacks to the cloud legitimate users). NICE uses SDN to mirror the traffic of a suspicious host for Deep Pack Inspection without interrupting the traffic of other users as in the case of a proxy-based IPS/DPI system. Also, SDN is used to collect flow-based countermeasures (statistics) as requested by the framework for identifying compromised virtual machines. Future work is related to the use of host-based IDS for improving the detection accuracy. Also, decentralized network control & attack detection is proposed as a future improvement to the system. Figure 14 shows the architecture of the proposed SDN-based IDS system.

FlowGuard [52] proposed a software firewall as an application running over an SDN controller. While the idea seems direct & simple (as openflow rules support dropping packets of a certain traffic flow), many challenges were solved to achieve this idea. For example, due to the frequent openflow rules updates, a dynamic policy violations checker is needed to make sure that the security policy is applied all the time. Another challenge is related to having openflow "SET action" which allows an attacker to change the headers of a packet and consequently to bypass the applied security policy. Hence a dependency checking tool was developed to check flow paths in the entire network & related dependencies in the flow tables. Future work is related to the development of a stateful firewall application which is widely used in business environments (i.e. a firewall that relates a packet to a network connection such as a TCP connection instead of dealing with each packet individually) which is a challenging topic due to the stateless nature of openflow rules. Also, the work could be enhanced by the development of a policy visualization tool.

sFlow [53] proposed a design change in order to use SDN for anomaly detection. While flow statistics is a very rich source of information for the detection of network anomalies & attacks, the authors showed that using openflow for flow statistics collection overloads the control plane & introduces scalability limitations. They proposed to separate the data collection process from the control plane using the sflow mechanism while relying into openflow only to take the mitigation action over the detected anomalies. sFlow uses packet sampling for statistics reporting instead of reporting the real matching counters as the case of openflow. sflow requires to deploy the packet sampling mechanism over the openflow switches which is not a standard feature for the time being & hence there's an opportunity for extra work to achieve the same target while using the available openflow standard features.

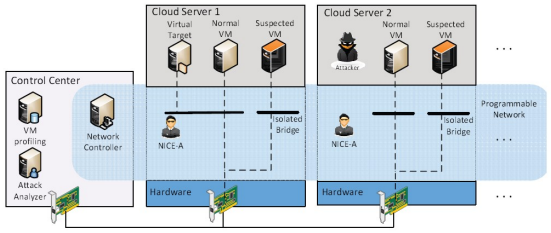

Figure 14 Architecture of SDN-based Network IDS framework as proposed in [51]

In **[54] a Provenance Verification Point component** was proposed to use the network itself as a point of observation in a forensics system using SDN reporting & traffic steering capabilities. The idea is to transform every network link into a reporting tool by programming a distributed set of openflow switches. When an operation is not applicable within the switch itself (such as deep packet inspection), the traffic could be steered to a middlebox or the controller itself for extra processing.

V. CONCLUSION

SDN is an emerging networking paradigm that allows the control of the network behavior through a centralized programming capability. SDN offers simplified & automated network management that meets the demand of increased network complexity & several application domains. We explained the architecture of SDN networking paradigm with the associated open research challenges & surveyed some of the work done in each challenge. While SDN is only a networking paradigm, the benefits of SDN could be achieved through using the correct application. We have surveyed several applications utilizing the benefits of SDN in five major application domains which are: Hybrid network control, Traffic Engineering, Data Center networking, wireless networks & network security applications showing the recommended future work for each application domains. The authors hope that the work would be useful for researchers willing to start a research work in the interesting research area of SDN or for professional engineers willing to extend their knowledge with the benefits & the terminology of the future networking technologies.

REFERENCES

[1] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.

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

[3] Choi, Mi-lung, et al. "XML-based configuration management for IP network devices." Communications Magazine, IEEE 42.7 (2004): 84-91.

[4] Nunes, Bruno AA, et al. "A survey of software-defined networking: Past, present, and future of programmable networks." Communications Surveys & Tutorials, IEEE 16.3 (2014): 1617-1634.

[5] Doria, Avri, et al. "Forwarding and control element separation (ForCES) protocol specification." Internet Requests for Comments, RFC Editor, RFC 5810 (2010).

[6] Farrel, Adrian, Jean-Philippe Vasseur, and Jerry Ash. A path computation element (PCE)-based architecture. RFC 4655, August, 2006.

[7] Casado, Martin, et al. "Ethane: taking control of the enterprise." ACM SIGCOMM Computer Communication Review. Vol. 37. No. 4. ACM, 2007.

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

[9] Pfaff, Ben, et al. "The design and implementation of open vswitch." 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). 2015.

[10] Pfaff, B., and B. Davie. The Open vSwitch Database Management Protocol. No. RFC 7047. 2013.

[11] Sherwood, Rob, et al. "Flowvisor: A network virtualization layer." OpenFlow Switch Consortium, Tech. Rep (2009): 1-13.

[12] Turull, Daniel, Markus Hidell, and Peter Sjödin. "libNetVirt: the network virtualization library." Communications (ICC), 2012 IEEE International Conference on. IEEE, 2012.

[13] Heller, Brandon, Rob Sherwood, and Nick McKeown. "The controller placement problem." Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[14] Koponen, Teemu, et al. "Onix: A Distributed Control Platform for Large-scale Production Networks." OSDI. Vol. 10. 2010.

[15] Tootoonchian, Amin, and Yashar Ganjali. "HyperFlow: A distributed control plane for OpenFlow." Proceedings of the 2010 internet network management conference on Research on enterprise networking. 2010.

[16] Yu, Minlan, Andreas Wundsam, and Muruganantham Raju. "NOSIX: A lightweight portability layer for the SDN OS." ACM SIGCOMM Computer Communication Review 44.2 (2014): 28-35.

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

[18] Nelson, Tim, et al. "Tierless programming and reasoning for software-defined networks." 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). 2014.

[19] Kang, Nanxi, et al. "Optimizing the one big switch abstraction in software-defined networks." Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. ACM, 2013.

[20] Lantz, Bob, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks." Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010.

[21] Gupta, Mukta, Joel Sommers, and Paul Barford. "Fast, accurate simulation for SDN prototyping." Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013.

[22] Fonseca, Rodrigo, et al. "X-trace: A pervasive network tracing framework." Proceedings of the 4th USENIX conference on Networked systems design & implementation. USENIX Association, 2007.

[23] Anand, Ashok, and Aditya Akella. "Netreplay: a new network primitive." ACM SIGMETRICS Performance Evaluation Review 37.3 (2010): 14-19.

[24] Zhuang, Yanyan, et al. "Netcheck: Network diagnoses from blackbox traces." 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). 2014.

[25] Handigol, Nikhil, et al. "Where is the debugger for my software-defined network?." Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[26] Canini, Marco, et al. "A NICE way to test OpenFlow applications." Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012.

[27] Mai, Haohui, et al. "Debugging the data plane with anteater." ACM SIGCOMM Computer Communication Review 41.4 (2011): 290-301.

[28] Khurshid, Ahmed, et al. "Veriflow: Verifying network-wide invariants in real time." Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013.

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

[30] Vissicchio, Stefano, Laurent Vanbever, and Olivier Bonaventure. "Opportunities and research challenges of hybrid software defined networks." ACM SIGCOMM Computer Communication Review 44.2 (2014): 70-75.

[31] Vissicchio, Stefano, et al. "Central control over distributed routing." Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. ACM, 2015.

[32] Feng, Tao, and Jun Bi. "OpenRouteFlow: Enable Legacy Router as a Software-Defined Routing Service for Hybrid SDN." Computer Communication and Networks (ICCCN), 2015 24th International Conference on. IEEE, 2015.

[33] Nelson, Tim, et al. "Exodus: toward automatic migration of enterprise network configurations to SDNs." Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. ACM, 2015.

[34] Hand, Ryan, and Eric Keller. "ClosedFlow: OpenFlow-like control over proprietary devices." Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014.

[35] Levin, Dan, et al. "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks." 2014 USENIX Annual Technical Conference (USENIX ATC 14). 2014.

[36] Farias, Fernando, et al. "Legacyflow: Bringing openflow to legacy network environments." (2011).

[37] Rothenberg, Christian Esteve, et al. "Revisiting routing control platforms with the eyes and muscles of software-defined networking." Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[38] Akyildiz, Ian F., et al. "A roadmap for traffic engineering in SDN-OpenFlow networks." Computer Networks 71 (2014): 1-30.

[39] Caria, Marcel, Admela Jukan, and Marco Hoffmann. "A performance study of network migration to SDN-enabled traffic engineering." Global Communications Conference (GLOBECOM), 2013 IEEE. IEEE, 2013.

[40] Chu, Cing-Yu, et al. "Congestion-aware single link failure recovery in hybrid SDN networks." Computer Communications (INFOCOM), 2015 IEEE Conference on. IEEE, 2015.

[41] Hong, David Ke, et al. "Incremental Deployment of SDN in Hybrid Enterprise and ISP Networks." Proceedings of the 2nd ACM SIGCOMM Symposium on Software Defined Networking Research(SORS 16). ACM, 2016.

[42] Lee, Steven SW, et al. "Design of SDN based large multi-tenant data center networks." Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on. IEEE, 2015.

[43] Zhu, Hao, et al. "Joint flow routing-scheduling for energy efficient software defined data center networks: A prototype of energy-aware network management platform." Journal of Network and Computer Applications 63 (2016): 110-124.

[44] Miao, Wang, et al. "SDN-enabled OPS with QoS guarantee for reconfigurable virtual data center networks." Journal of Optical Communications and Networking 7.7 (2015): 634-643.

[45] Tu, Renlong, et al. "Design of a load-balancing middlebox based on SDN for data centers." Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on. IEEE, 2015.

[46] Basta, Arsany, et al. "Applying NFV and SDN to LTE mobile core gateways, the functions placement problem." Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges. ACM, 2014.

[47] Qazi, Zafar Ayyub, et al. "KLEIN: A Minimally Disruptive Design for an Elastic Cellular Core." Proceedings of the 2nd ACM SIGCOMM Symposium on Software Defined Networking Research(SORS 16). ACM, 2016.

[48] Saunders, Ryan, et al. "P2P Offloading in Mobile Networks using SDN." Proceedings of the 2nd ACM SIGCOMM Symposium on Software Defined Networking Research(SORS 16). ACM, 2016.

[49] Jacobsson, Martin, and Charalampos Orfanidis. "Using software-defined networking principles for wireless sensor networks." 11th Swedish National Computer Networking Workshop (SNCNW), May 28-29, 2015, Karlstad, Sweden. 2015.

[50] Ku, Ian, et al. "Towards software-defined VANET: Architecture and services." Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual Mediterranean. IEEE, 2014.

[51] Chung, Chun-Jen, et al. "NICE: Network intrusion detection and countermeasure selection in virtual network systems." Dependable and Secure Computing, IEEE Transactions on 10.4 (2013): 198-211.

[52] Hu, Hongxin, et al. "FLOWGUARD: building robust firewalls for software-defined networks." Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014.

[53] Giotis, Kostas, et al. "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments." Computer Networks 62 (2014): 122-136. Elsevier,2014.

[54] Bates, Adam, et al. "Let SDN be your eyes: Secure forensics in data center networks." Proceedings of the NDSS workshop on security of emerging network technologies (SENT'14). 2014.