Open Source Platforms, Applications and Tools for Software-Defined Networking and 5G Research

Lauri Suomalainen, Emad Nikkhouy, Aaron Yi Ding, Sasu Tarkoma

Technical Report C-2014-2
University of Helsinki
Department of Computer Science

Helsinki, August 15, 2014

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Lauri Suomalainen, Emad Nikkhouy, Aaron Yi Ding, Sasu Tarkoma

Työn nimi — Arbetets titel — Title
Open Source Platforms, Applications and Tools for Software-Defined Networking and 5G Research

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Technical Report C-2014-2	August 15, 2014	29

Tiivistelmä — Referat — Abstract

Software-Defined Networking (SDN) is a novel solution to network configuration and management. Its openness and programmability features have greatly motivated the open source communities where numerous applications and tools are developed for various R&D purposes. For the strength of SDN, the upcoming 5th Generation mobile networks (5G) can also benefit from the modular and open design to innovate the network architecture and services. In this report, we present a survey of existing open source platforms, applications and tools for SDN and 5G research. We discuss the potential directions and share our perspectives in this domain.

Avainsanat — Nyckelord — Keywords
SDN, Open Source, 5G Mobile Networks

Säilytyspaikka — Förvaringsställe — Where deposited
University of Helsinki / Kumpulan Kampuskirjasto

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

Software-Defined Networking (SDN) is a novel solution to network configuration and management. Its openness and programmability features have greatly motivated the open source communities where numerous applications and tools are developed for various R&D purposes. For the strength of SDN, the upcoming 5th Generation mobile networks (5G) can also benefit from the modular and open design to innovate the network architecture and services. In this report, we present a survey of existing open source platforms, applications and tools for SDN and 5G research. We discuss the potential directions and share our perspectives in this domain.

2 Open Source SDN Components

Even though the concept of software-defined networking dates back to year 2004 [79] the first widely recognized applications did not follow until several years later. The most notable one is the widespread OpenFlow protocol [45] which made its official debut in 2009 and has since become the most prevalent protocol for SDN. The available applications on the market reflect the pervasiveness of OpenFlow as practically all of them are built to support the protocol. Keeping SDN's basic concepts and architectural solutions in mind it does not come as a surprise that the network controllers and switch implementations are the most common SDN-related applications. In this section, we present a comprehensive survey of existing open source components for SDN.

2.1 Controllers

Most OpenFlow controllers are organized in the same fashion as shown in Figure 1. Most of the current controllers offer a so-called northbound interface (usually a REST API) to communicate with applications and to offer them services. The southbound interface is for communicating with the actual physical and virtual switches in the network.

Hailed as the first OpenFlow controller, NOX was developed side-by-side with OpenFlow but peculiarly released a year before OpenFlow's official release [26]. NOX is written in C++ and is meant for developing further customized SDN controllers, but it is commonly regarded as complex and cumbersome to use for anything but really performance-critical applications. The developer Murphy McCauley himself recommends using POX [49], the Python version of NOX, for most tasks [69]. There is also a newer version of NOX that according to developers has more streamlined architecture, cleaner codebase and is more efficient. At the time of writing this the latest modification to source code was an over 7 months old bugfix, so NOX's actual viability in current SDN development is questionable.

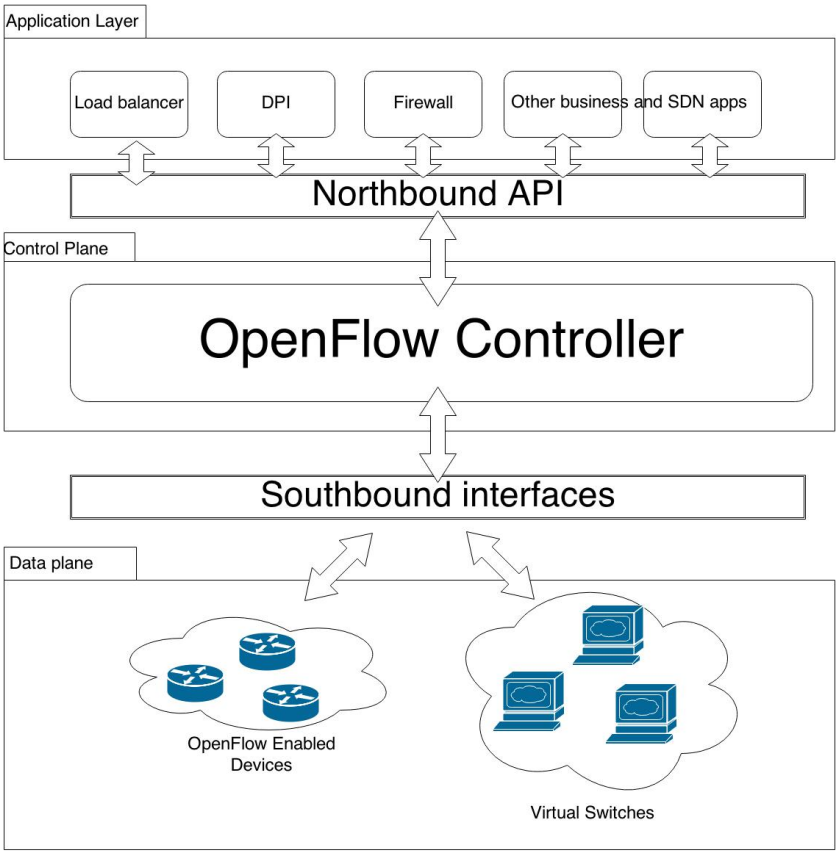

Figure 1: General SDN controller architecture

Beacon is another notable OF controller [4]. It was released in 2010 and it is written in Java. Despite Java's reputation as an inefficient programming language Beacon has fared well performance-wise in comparison with other controllers [70]. Java was chosen to address C and C++ portability problems and to reduce the developers' burden with significantly shorter compilation times and better error logging. Other controllers with similar focus on easier development are for example Python-based Ryu [52] and Ruby-based Trema [62].

One of the features that makes Beacon perhaps more notable than other aforementioned controllers is the fact that it serves as the basis for one of the most popular OpenFlow controllers: Floodlight [7]. Floodlight is developed by Big Switch Networks, a startup founded by networking veterans with notable work history in big companies like Cisco and Juniper. Unlike Beacon, Floodlight does not rely on an OSGi framework and offers more features such as REST API and support for non-OpenFlow domains. The documentation for Floodlight is also generally good and surpasses the Beacon documentation with ease. Big Switch has in the past lobbied for Floodlight to be included

in Open Daylight Project's codebase [66] but the project opted for Cisco's controller implementation.

Open Daylight Project [34], or ODL in short, is commonly considered the leading SDN project with its substantial company backing and large developer community. The project members involved include practically every company relevant to the field. These are for instance Cisco, Microsoft, IBM, Hewlett-Packard, Juniper, Red Hat, VMware and many more. Feature highlights of Open Daylight include a robust REST API and southbound communication which allows using other non-Openflow protocols: OVSDB, NETCONF, LISP, BGP, PCEP and SNMP. Naturally ODL contains components to take advantage of the protocols as well as other components not found in any other SDN projects e.g. DDOS detection and counter-measures. The basic architecture is rather similar to other controllers, but different core components, applications built on top of them and protocol plugins that offer different services and use different interfaces complicate the organization a bit. Figure 2 shows the architecture and some of the components in the bundle. For prototyping applications that only make use of OpenFlow the sheer number of different components in ODP may prove intimidating and distracting, but generally the ODP community offers decent documentation and guides though the quality varies from one component to another. In comparison with Floodlight, ODP is likely to be more viable in a real business environment because of the features it offers and its fast evolving nature. However, when developing quick prototypes or SDN applications that only make use of OpenFlow, Floodlight may be a better choice due to being simpler, smaller and at the moment better documented.

Other lesser known controllers and controller frameworks are for example a very bare bones Libfluid [18], Java-based Maestro [20], On.Lab's FlowVisor [10] which is meant to be used with other On.Lab SDN components, FLOWer [8] which is written with Erlang, the event-driven Resonance[50], Node.js -based NodeFlow [25], the lesser-known KulCloud Open MUL and SNAC [16, 57], flowforwarding.org's Warp [64], network virtualization bundle Open VNet [46], IRIS which uses Floodlight and Beacon components to offer horizontal scalability [15] and Juniper's OpenContrail focusing on SDN and Big Data [38]. Generally these pieces of software are either not in active development or not as well received or otherwise as notable as the controllers mentioned earlier. Table 1 shows comparison of some of the well known SDN controllers.

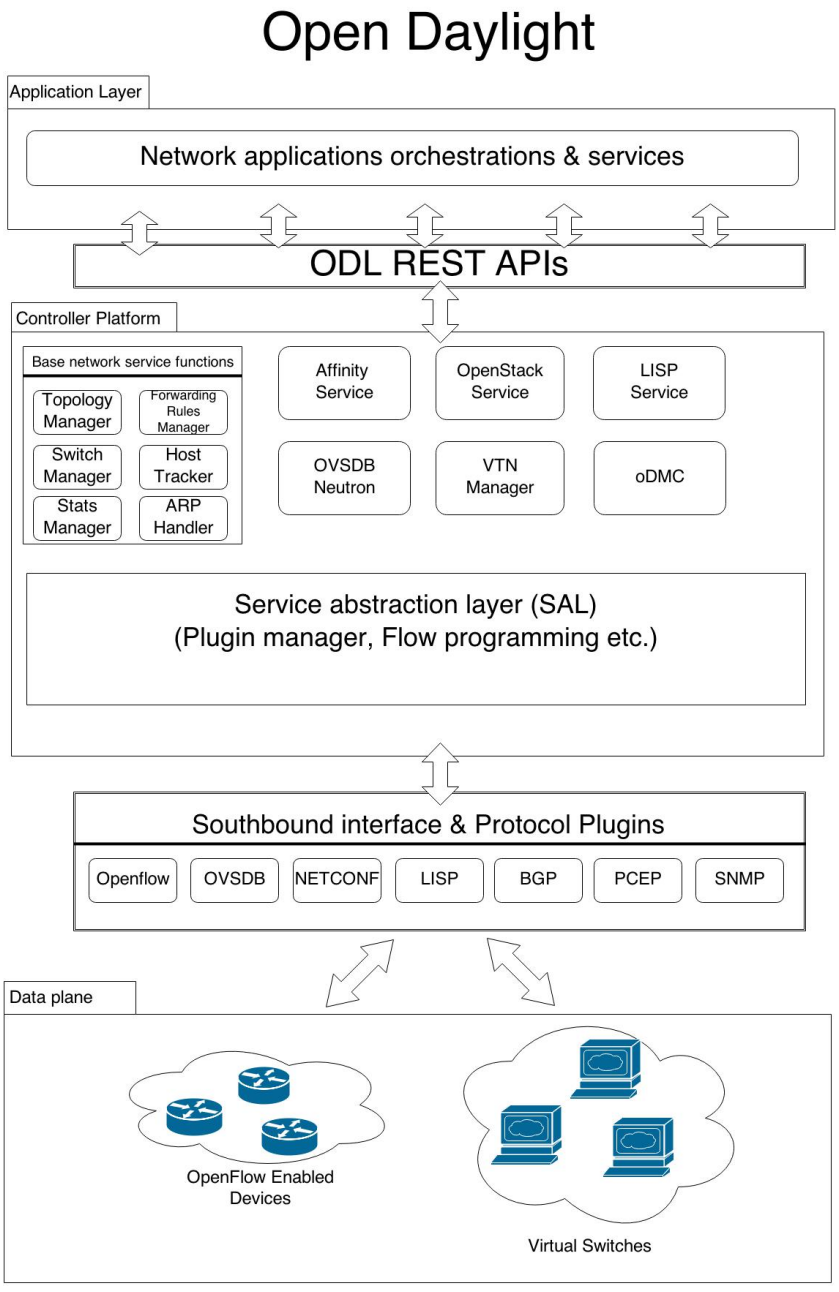

Figure 2: Open Daylight architecture

Table 1: SDN Controllers Comparison

Name	**Pox**	**Beacon**	**Floodlight**	**OpenDaylight**	**Ryu**
Language	Python	Java	Java,Python	Java	Python
User interface	Python + QT4 and Web	Web	Web and Java	Web	GUI Patch
Supports hosts with multiple attachment points	No	No	Yes	Yes	No
Compatible OS(s)	Linux Mac Windows	Linux Android	Linux Mac Windows	Linux Mac Windows	Linux
Development community	Nicira Networks	David Ericson Stanford University	Big Switch Networks	Industry Supported	NTT Communications
Actively developed	Yes	Maintained	Yes	Yes	Yes
Open source	Yes	Yes	Yes	Yes	Yes
Open source license	GNU	BSD	Apache	EPL	Apache
Supports REST API	Yes (Limited)	Yes	Yes	Yes	Yes
Supports OpenStack Quantum	No	No	Yes	Yes	Yes
Provides abstraction layer above southbound protocols	No	No	No	Yes	No
Supports topologies with loops	Yes via spanning tree	No	Yes	Yes	Yes via spanning tree [73]
Supports non-OF island connections	No	No	Yes	Yes	No
Supports OF island connections with loops	No	No	No	Yes	No

2.2 Switches

OpenFlow relies fundamentally on switches: there has to be at least one in the network to connect to a controller, otherwise using OpenFlow is not possible. OpenFlow being nearly synonymous to SDN and network functions virtualisation and NFV being a hot topic for conversation along with SDN there is a demand and supply for standardized, vendor-agnostic virtual and physical switches. With SDN and network virtualization it is possible to eliminate the need for specialized switches, routers, middle boxes etc. and utilize the network elements on ordinary X86 servers.

VMWare's Open VSwitch is undoubtedly the most well known of virtual switches [37]. It has been ported to multiple virtualization platforms such as VirtualBox and KVM, is included in few Linux kernels and bundled with many SDN projects and is also integrated to various virtual management systems like OpenStack. Open VSwitch offers many features like several QoS and security components and it supports a few different protocols.

Other software switches include Indigo Virtual Switch meant to be used with Floodlight and Indigo framework [14], CPqD's OFSoftswitch [31], LINC-switch [19], Snabb.co's virtualized ethernet switch called Snabb Switch [56] and protocol oblivious POFSwitch [48]. The previously mentioned open VNet also includes a virtual switch [46].

Other switch-related projects are Centec's Lantern [17] which includes a modified Open VSwitch but is mainly designed for implementing custom hardware-based SDN switches with the accompanying software bundle. Other partly similar software is Pantou [83]. It can be used to turn normal commercial routers to OpenFlow-enabled switches. Pantou will be discussed more in depth later in the report.

2.3 Testing and Simulating

Like in any other field in computer science, fast and professional development and research is supported by tools for testing and quality assurance. SDN technology is not an exception. The tools fall roughly into two categories: Some are used for simulating networks and network events while others focus on performance benchmarking and monitoring.

The most well known tool for simulating networks is Mininet [21]. Mininet could be considered essential for conducting research on SDN and testing software in a realistic network environment. Mininet supports custom network topologies which can be easily created with its Python API and its switches support OpenFlow. Connecting to real networks should be quite straightforward though a single Linux OS can support over four thousand switches and hosts simultaneously [21].

A slightly more specific tool is STS which stands for SDN Troubleshooting System [53]. It can be used to simulate and troubleshoot specific devices

on the network. A bit similar program is ns-3 which is used for simulating network events [27]. For performance testing and benchmarking there are OFLOPS and PerfSonar [29, 47] and for testing SDN software itself one can utilize NICE [24] for controller applications, OFTest for compliance testing [32] and TestOn for testing automatization [59]. It is also worth to mention HyperGlance by Real Status [12]. It is a network visualization tool which supports Open Daylight. It could most likely be used for debugging as in addition to network topology it displays traffic, too. Unlike the other pieces of software mentioned HyperGlance is not an open source project or free software, but at the time of writing it was in open beta phase and acquiring the software required only registration to the site.

2.4 Other SDN projects

Apart from controllers, switches, testing and simulation tools there are many notable projects that do not straightforwardly fall into these categories.

There has not been as much talk about security with SDN as there has been on for example network virtualization, but the centralized network control introduces fundamental vulnerabilities that need to be addressed. Roy Chua of SDNCentral.com points out that the SDN controller should always be closely controlled and also protected from an outside attack like DDOS because without it SDN networks' functionality is crippled [55]. Other aspects to be aware of are protecting communication throughout the network and maintaining and monitoring network performance and function. Open Daylight for example includes a component for protection from Denial of Service attacks and OpenFlowSec.org offers multiple security solutions for SDN ranging from malware protection to security policy enforcement.

Other projects aim to enhance the performance of the network. InCntre's Flowscale [9] is a network load balancer, but the development has stopped after version 0.6. This may be because of many controller applications including a load balancer of their own. Both the BIRD by CZ.NIC Labs and CPqD's RouteFlow offer IP routing [60, 51].

Some components are made to enhance SDN development. These include domain-specific languages like Frenetic and Pyretic developed in Princeton University [11] and Haskell-based Nettle [23] from Yale University. In the same vein FlowForwarding.org offers protocol libraries for NetConf and OpenFlow written in Erlang [5, 30], Midokura has one for OpenFlow written in Python [44] and Ericsson provides a library made with Node.js [28]. They are named enetconf, of_protocol, OpenFaucet and Oflib-Node, respectively.

Other various projects include Big Switch's Indigo [13], a hardware abstraction layer and configuration layer meant for communication between Floodlight and the physical layer, MirageOS [22], used for constructing unikernels for network applications, wakame-vdc and Open Virtex for data center virtualization [63, 36] and for wireless communication there is nwEPC

- EPC SAE Gateway by Thomas Batia [68] and Open Source IMS Core and OpenEPC from Fraunhofer Fokus [35, 43] of which the latter is not an open source project but otherwise worth mentioning in this context.

2.5 Future developments

SDN development seems to be thriving and especially different companies have taken great interest in the concept. Open Daylight, being the biggest company-backed SDN project to date, is evolving continuously. During this year as many as 15 new project proposals have been accepted to be included in Open Daylight, among those a dedicated Service Function Chaining component, an Authentication, Authorization and Accounting component and an application policy plugin to name a few [41]. While Floodlight is not evolving with as rigorous a pace, the active development and moderate company involvement is likely to keep Floodlight Open Daylight's main competition. However, the two big controllers might get a new rival in the future. Currently the top trending project in sdncentral.com is On.Lab's upcoming network operating system ONOS [54, 33]. ONOS is to complete On.Lab's SDN stack consisting of ONOS, FlowVisor and Mininet. It is to feature a controller function as well as horizontal co-ordination and communication among multiple ONOS instances allowing easier scalability for networks. The first prototype was demonstrated at Open Network Summit 2013, and according to On.Lab ONOS will be released this year but no definitive date has been given.

2.6 Ongoing EU Projects

There are numerous projects, which are operating under Framework Programme for Research (FP7) in Europe. In this section we are going to introduce some of these projects briefly.

2.6.1 FELIX

The main goal of the FELIX project (FEderated Test-beds for Large-scale Infrastruction eXperiments) is to build a common ground for users, in order to monitor, request, and manage a slice of distributed Internet network in Japan and Europe. In FELIX, new technologies, which are rising and Software Defined Networking control frameworks are investigated (such as OFELIA OCF, and Open Grid Forum) in order to assess their applicability in the project. [2]

Therefore FELIX, in order to succeed in its goal, proposes a novel SDN-oriented service design, which has the ability to engage heterogeneous high-end FI facilities (such as JGN-X RISE and OFELIA). Thus, in order to satisfy the needs of Japanese and European research groups, high capable NSI-

enabled networks are used that offer building the experimental infrastructure in a dynamic and seamless way.

2.6.2 SPARC

SPARC (Split architecture for carrier-grade networks) aims to examine dividing the traditional IP router architecture into several control and forwarding planes. Thus, in order to achieve this, SPARC will implement a prototype based on the OpenFlow concept. Upon achievement, they are planning to open the doors for new players in the market by reducing the obstacles that exist in the complexity of each single element. Therefore, the starting point for SPARC would be OpenFlow and GMPLS, and examining how with mixing these two, current network quality can be improved by moving into simpler and standardized hardware. [58]

2.6.3 OFELIA

OFELIA gives the ability to researchers to do experiments on a multi-layer and multi-technology test network, and also gives permission to control the network itself accurately and dynamically. It creates different tools with various options and runs networks and services on top of it, in order to bring innovation to the future Internet. Therefore, if a researcher has an idea, by logging in to the OFELIA webpage, she or he can configure the network slice and run experiments on it. OFELIA supports information centric networking (ICN), thus researchers can deploy and test an ICN system. [65]

2.6.4 ALIEN

ALIEN aims to build a solid infrastructure for software defined networking by abstracting the network mechanism and interoperability of heterogeneous network components. ALIEN uses Network Operating System (NOS), which is a distributed system operating on top of a heterogeneous network structure. NOS gives the ability to view the whole network components and their functionalities. NOS in the ALIEN project will be supported by the management and control framework of OFELIA FIRE facility. [1]

There are different network components that are unfamiliar with OpenFlow technology such as network processors, switches, optical network components, and programmable hardware. ALIEN expands OpenFlow control of OFELIA and its design in order to abstract network information of the above mentioned devices. NOS in the ALIEN project uses a new hardware description language and also functional abstraction method in order to have uniform demonstration of network components and their functionalities that is not compatible with OpenFlow. This language will describe input and output format, topology information, and different functionalities of the hardware in order to show the pipelining of actions on a particular network

device. In short, ALIEN is a Hardware Abstraction Layer (HAL) in order to make non-OpenFlow network components have seamless migration to SDN.

2.6.5 OFERTIE

OFERTIE (Open-Flow Experiment in Real-Time Internet Edutainment) is a 24-month project, which is founded by the EC FP7 programme. The goal of OFERTIE is to use SDN in order to improve the way distributed applications are delivered in the future Internet, known as Real-Time Online Interactive Applications (ROIA). In this project, programmable networking techniques are going to be discovered in order to handle networking bottlenecks that restrict ROIA applications for scalability and quality of experience. OFERTIE with the help of the OFELIA testbed is going to run various experiments to discover how programmable networks can be helpful for technical solutions such as QoS and multicast, and what business models can benefit from these solutions in an economically sustainable manner. [3]

2.6.6 FIRE

Fire (Future Internet Research and Experimentation) is launched in 2007 as a member of Framework Programme 7 (FP7). One of the goals of FIRE is to promote experiments, on new instances, networking concepts and architectures for the future Internet. The other goal of FIRE is to build large-scale experimentation facilities in order to help medium as well as long-term research on networks and services. FIRE currently has twelve projects, eight of which are research-focused and experimentally-driven, whereas the other four projects are “facility projects” to build empirical platforms for future internet researchers. [6]

3 SDN-based Service Chaining for Mobile Networks

SDN (software-defined networking) has revolutionized designing and managing networks over the past few years. Routers and switches operate with complex software, which is closed source and dedicated to the private network companies. Moreover, each of these devices come with their own configuration interfaces that are different between vendors, which makes the work of network administrators exhausting to configure each of these individual network devices. [71]

SDN changes the way networks are managed and designed by defining two characteristics. First, SDN detaches the control plane from the data plane. Second, SDN unifies the control plane, thus a single software program (controller) can control numerous data plane elements. The SDN controller has direct access to the data plane through a well-defined API such as OpenFlow. An OpenFlow switch has a packet-handling table, where different rules can be defined in the table. Therefore, based on different rules that match with a subset of traffic, different actions (such as dropping, forwarding and header modification) can be taken. Thus, based on different rules installed on OpenFlow by the controller, an OpenFlow switch can act as a router, firewall, network address translator, switch or something in between.

As discussed earlier, network instruments are closed and proprietary, which is a barrier to openness. Technical innovation can break down the barrier caused by industry and government in order to breed openness. The future perspective of wireless mobile networks is that any mobile device can connect to any network and move from one network to another seamlessly and freely. Thus, handheld devices can connect to any network regardless of what radio technology it uses and who owns the network. Figure 3 illustrates an overview of OpenFlow Wireless. [82]

Openness and an open architecture can provide improved features in the network delivered by new industry suppliers, which can help the open source community to flourish. OpenFlow is a protocol and also southbound API for the SDN controller, which is installed on top of a router, switch, or access point. Despite the fact that current networking devices such as switches and routers do not have a common external interface, OpenFlow gives the ability to the controller software to control the OpenFlow enabled device.

Kok-Kiong Yap et al., introduced a mobility manager for the network users who move around. In this method, the mobility manager is aware of every application flow in the network and can choose the route for a specific flow. Therefore, when the user travels from one point to another, the mobility manager becomes aware of the user's movement and can decide to reroute the flow. Due to the independency of OpenFlow from the physical layer, vertical handoff between various radio networks is seamless and easy.

To conduct several simultaneous experiments, slicing (or virtualizing) the network is used in order to make a service permit its users to move across

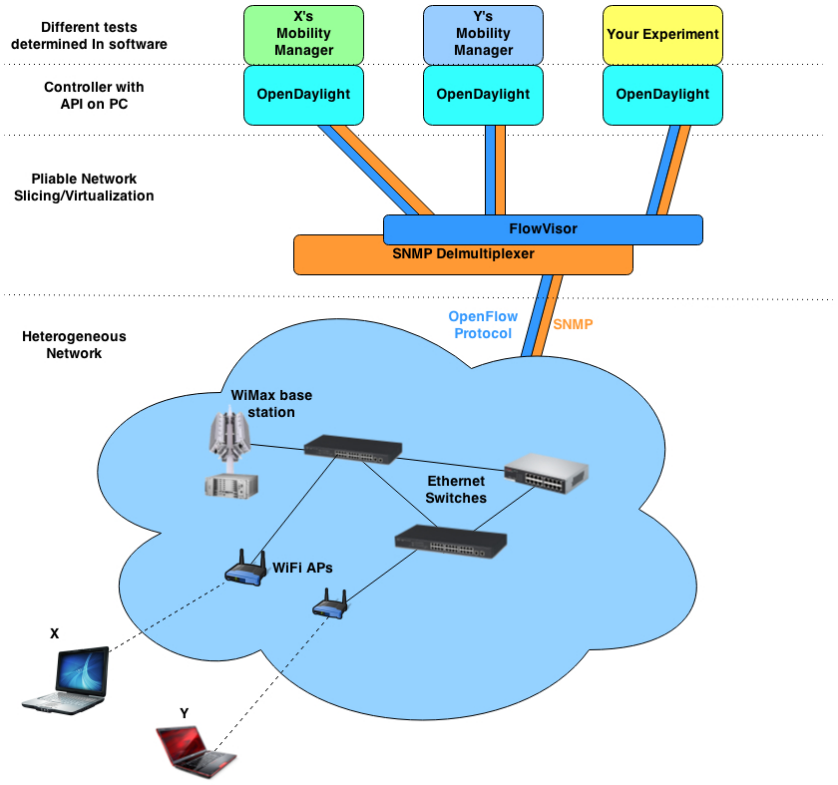

Figure 3: OpenRoads Architecture [81]

multiple physical networks freely. So with slicing, multiple controllers can cooperate together, where each one is responsible for controlling its own slice of network. A slice may comprise one network or many networks; one user or many users; one subset of traffic or all traffic. FlowVisor is an open source application, which can be used to slice OpenFlow networks.

FlowVisor is an extra layer added between the controllers and the datapath. It slices the network and assigns control of various flows to different controllers. Since FlowVisor communicates to the datapath and controllers using the OpenFlow protocol, datapaths think that they are controlled by a single controller, and the controllers believe that they are controlling their own private network, but actually they are controlling a virtual network. Here the method is to classify the flows based on a policy, which is defined by the network manager, then FlowVisor can determine which flow belongs to each slice, and transmit that to the corresponding controller for that slice. For instance, if controller 'Z' is in charge of all John's traffic, then FlowVisor transmits all John's traffic to controller 'Z'.

Slicing makes experiments in the production network simple. Thus, flowspace and topology define each experiment to be assigned to its own slice, which is implemented by the FlowVisor. Moreover, slicing or network

virtualization enables the production network for the versioning, where new features can be introduced in the production slice. Various slices can be assigned with different versions. So in this method, new features can be distributed and tested rapidly, and afterward made available to the whole network, or even shared with other network owners.

Finally, slicing allows decomposition, so network administrators can decompose a network to further slices with the aid of FlowVisor in order to allocate more flow space to them. Repetition of slicing is logical when there is a hierarchy of control in the network. For instance, the network manager can dedicate a slice to a research group in the computer science department, and then that research group can slice it among different experiments.

OpenFlow does not have the ability to control the datapath elements, such as enable or disable interfaces, set power levels or assign channels. Controlling the datapath elements is usually done with NetConf, command line interface, or SNMP. This job is difficult when there are numerous slices of network and each slice is suppose to be configured independently. SNMPVisor is a good tool to configure the datapath, which works alongside FlowVisor. In order to slice the configuration, SNMPVisor observes the SNMP control messages, and forwards them to the correct datapath element. However, sometimes it is infeasible to slice the configuration. For instance, assigning different power levels for various slices on current existing WiFi Access Points is impossible.

3.1 Pantou and OpenWRT

Pantou transforms an access point or wireless router into an OpenFlow enabled device. So, basically in Pantou, OpenFlow operates on top of OpenWRT as an application [83]. OpenWRT is an open source extendable operating system for the router, which is fully customizable for the needs of users and developers. It competes with other existing solutions in performance, robustness, extensibility, design and stability. OpenWRT is highly customizable, so it is not intended only for professional high-end users, rather other users who are seeking for high customisability can also benefit from it. [75]

OpenWRT is designed in such a way to be user friendly, therefore users can choose their desired packages, configure them, and build their own custom firmware [75]. Depending on the model of router that the user is using, there are numerous projects for Pantou that can be used to build custom OpenWRT firmware with OpenFlow on top of it. For instance, Backfire, Attitude Adjustment, and Trunk are some of the Pantou projects that can be used to generate firmware. [61]

By employing OpenWRT, wireless freedom can be achieved. OpenWRT, due to its open architecture, enables the user to use features that can be added to OpenWRT. These features are, intrusion detection, stateful packet inspection, and many other features that normally should be paid to physical

devices, cost thousands of dollars to do the job effectively. The OpenWRT community goal is to keep this open source firmware generic and always up to date alongside with the advancement of technology.

3.2 Network Service Chaining

The traditional network infrastructure has made network service providers (NSPs) to struggle with the user increment and high traffic demands. While users enjoy decrement of "price per bit", NSPs are in battle with operational cost (OPEX) and operator investment (CAPEX), which are increasing due to the complexity of current network infrastructure. Current network infrastructure in NSPs, cannot fulfill their needs, and this is because of inflexibility in their network. [72]

Conventionally, advanced services such as, firewall, deep packet inspection (DPI), intrusion detection and prevention systems, caching, etc., are fixed inside the network, which make the network inflexible and static. Moreover, due to this inflexibility, configuration between these network services suffers from lack of automation. Therefore, troubleshooting in the network might consume a lot of time varying from hours to even weeks depending on the size of the network.

Due to inter-dependencies and the low power of automatic configuration that middleboxes (services) have, if there is a new service to be introduced or removed from the network platform, they need careful engineering and customization. By increasing the automation, the cost of OPEX and CAPEX can be reduced. Decreasing the network touch points, consequently reduces the possible configuration mistakes, which reduces the cost of OPEX. In addition, optimizing and refactoring the use of existing resources by virtualizing certain network functions can reduce the CAPEX cost.

Today, network operators cannot reuse middleboxes, since they are configured to provide only one service. So, if one box is removed from the network, then the whole chain will break. With current issues that network platforms are facing, the need for dynamic service chaining in SDN is gradually increasing. Dynamic service chaining can bring high availability and rapid failure restoration with reliable testing abilities, to the network platform.

Figure 4 illustrates the capabilities of network service chaining (NSC), which can introduce dynamic, upgradable and configurable software to the network. NSC enables data to flow without any interference caused by middleboxes residing at different nodes. Therefore, different services in the network can be utilized only if needed, and can be omitted whenever their service is unnecessary. NSC takes advantage of virtualization, so regardless of the layer (physical or higher layers) where middleboxes are implemented, they can be integrated seamlessly.

Dynamic NSC, introduces more intelligent traffic steering to the network, which accelerates the performance of the network. When there is a single

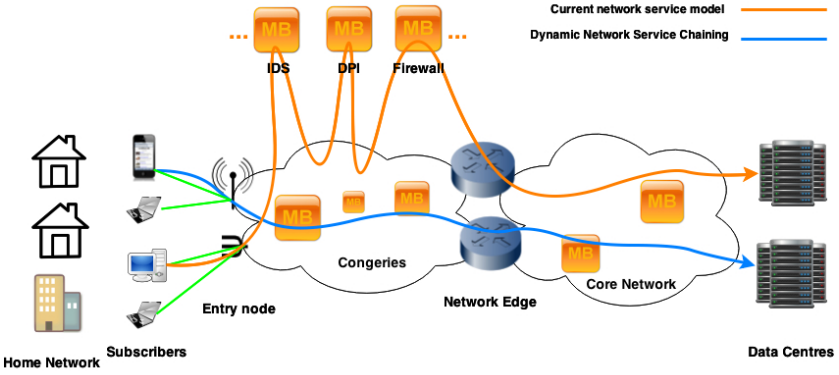

Figure 4: NSC vs. Current network service model [72]

network domain, middleboxes such as load balancers, traffic schedulers, security gateways, etc., provide better fairness and security. However, when it comes to multiple network domains, further operator investment is needed, either in terms of software or hardware. Thus, this extra operator investment causes serious decline in terms of delay, which depending on the number of policy elements that the data is passing and the cross-domain network load, this deferment varies. However, when NSC is implemented in the edge of a network, multimedia flows and sensitive data can steer through different network domains in predictable and reliable manner. Figure 5 depicts this benefit of dynamic NSC.

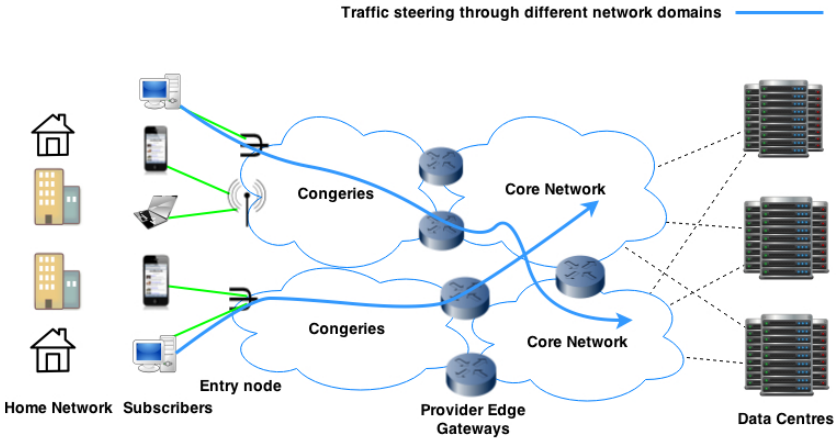

Figure 5: Traffic steering in dynamic NSC [72]

3.3 YANG Model and its usage in OpenDaylight

YANG is a data modeling language, which is used to model the data for network configuration protocol (NETCONF). YANG uses an XML tree to model hierarchical data, where each node has a name, value or series of child nodes. Moreover, the format of event notifications, which are published by the network elements can be specified by YANG. In the YANG modeling language, the signature of a remote procedure call (RPC) can be specified by data modelers, which can be invoked on network components through the NETCONF protocol. [67]

In OpenDaylight there are different projects such as Netconf Client (NCC) and Model Driven Service Abstraction Layer (MD-SAL) controller, which use YANG for their modeling language [40] [39]. In these projects, YANG permits to model the structure of XML data, to describe the semantic components and their connection, and to model all the elements as a unified system [40].

The YANG data model uses XML, which makes the data self-explanatory. Moreover, it is easy for the data to be utilized in applications and controller components that use controller's northbound API. The YANG data model facilitates the development of applications and controller components. Therefore, the risk of bad interpretation of the data structure can be reduced through a defined pattern, which creates easier, statically typed API for developing a module with specific functionality.

3.4 LISP and its usage in OpenDaylight

Conventionally, IP addresses operate as both network and device identifier. LISP (Locator/ID Separation Protocol) divides the address functionality into two distinctive roles. Endpoint identifier (EID) identifies a network device in a unique manner and routing locator (RLOC) is the routing address point for endpoint identifiers on the network. One of the advantages of LISP is that EID and RLOC can be utilized in the current IPv4 and IPv6 address structure. In LISP, due to aggregation of EID addresses by more than only one RLOC, scalability in the network deployment is feasible. Moreover, because of this separation in the architecture, device mobility in the network becomes possible, therefore an EID can be moved freely from one point in the network to another by changing the RLOC without requiring modification of the configuration of the device. [77]

Figure 6 illustrates the order of sending a packet from the client to the server using the LISP infrastructure. First, the packet is forwarded to the LISP router at the client side, which is called egress tunnel router (ETR), then ETR tries to find the next hop router that can reach the destination EID, by sending a MAP request to the MapServer. Next, MapServer tries to find information binded between EIDs and RLOCs and then replies the routing point address of destination EID to the ETR. ETR encapsulates the

packet after receiving the routing point address (RLOC) from the MapServer, and then forwards it to the router that has the corresponding RLOC. The router which has the RLOC, is called ingress tunnel router (ITR), the job of which is to decapsulate the received packet and forward it to the chosen sever.

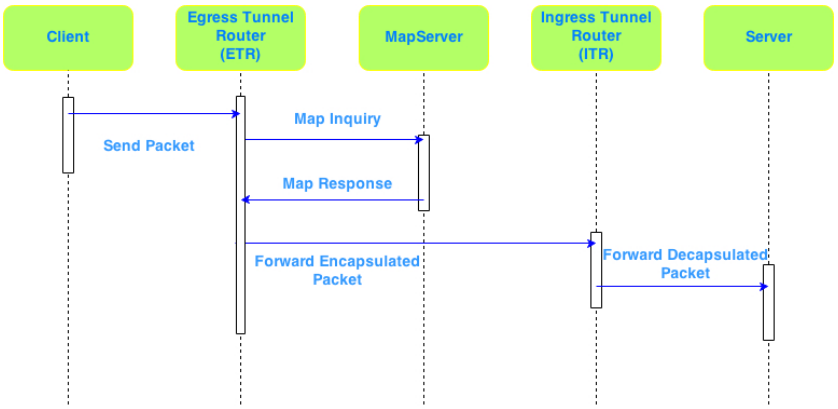

Figure 6: Packet forwarding between LISP routers [77]

In a network, which is virtualized, EID can play the role of virtual address space and RLOC can play the role of physical network address. In software defined networking, the control plane is separated from the data plane. Therefore, LISP in regards to the data plane, defines how encapsulation of virtualized network addresses should be done in addresses from the underlying network. In addition, the control plane stores the binding information between EIDs and RLOCs, and associated forwarding policies, therefore the data plane can fetch this information whenever it receives new flows. [42]

LISP Flow Mapping project in OpenDaylight utilizes LISP infrastructure, which provides mapping system services. This project, consists of the LISP Map Resolver service and LISP Map Server service in order to store and retrieve the mapping data to ODL applications and also data plane nodes. Mapping data can contain different routing policies including load balancing and traffic engineering. Moreover, it can also contain mapping of virtual addresses (EIDs) to physical network addresses (RLOCs), in order to access virtual nodes. In the LISP mapping service, mappings and policies can be specified through the ODL northbound REST API, in order to utilize this service. In addition, data plane devices, which are compatible with the LISP control protocol can utilize this service via a southbound LISP plugin through the LISP control protocol. Figure 7 illustrates the above-mentioned components.

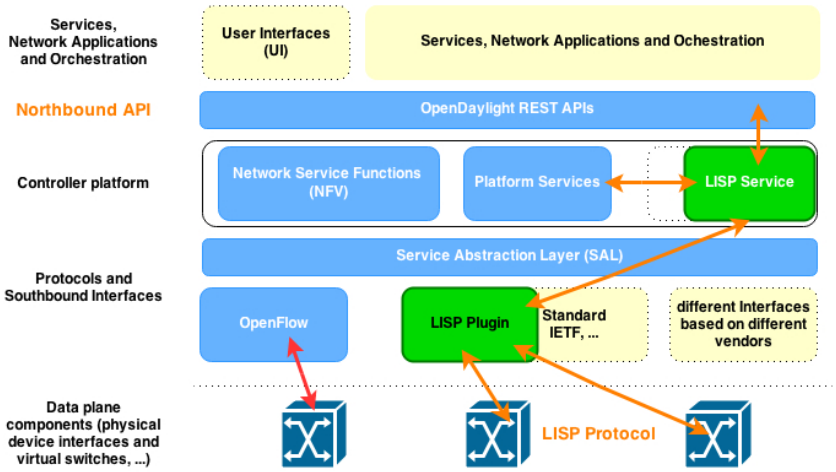

Figure 7: LISP in OpenDaylight [42]

3.5 OpenEPC and nwEPC

Testbeds are good platforms in order to understand and take advantage of new technologies in short time, and in a realistic operator network environment. Current network functionality is extremely complex and this is because of a wide variety of protocols, components and interfaces that have to work simultaneously, which can impact their behavior. There are numerous challenges in order to have a reliable and cost-efficient testbed that can test applications before the final product development. OpenEPC tries to minimize these challenges by offering a novel approach. [43]

OpenEPC entirely simulates the operator core network, by providing a good tool for demonstrations and a profound study of IP communication devices, such as radio access networks, mobile devices and core network up to the service platforms. Due to its openness, its source code can be used to access 3GPP (3rd Generation Partnership Project) standard components. Therefore, developers who are interested in LTE (Long Term Evolution) and EPC (Evolved Packet Core) environments can update these standard components in a shorter period of time.

OpenEPC is inspired by 3GPP EPC architecture, therefore it provides a realistic testbed by binding different standard radio technologies, where it has absolute control over the operator environment. Moreover, it contains all the elements and main functionalities of 3GPP EPC standards, in addition to its own features, which makes it enables all kinds of IP communications such as LTE, EDGE, HSPA, and even non-3GPP accesses such as WiFi.

OpenEPC comes with different functionalities, some of which we are going to discuss briefly. Core network mobility management is one of the OpenEPC functionalities, where it has some necessary features for establishing the user plane. These features include the implementation of PMIP (Proxy Mobile IP)

and GTP (GPRS Tunneling Protocol) mobility, zero packet loss handovers and multiple APN (Access Point Name) support.

Integration of 3GPP access networks is another functionality of OpenEPC. It incorporates with RAN (Radio Access Network) components, which makes it enable control over wireless connectivity of devices to connect them to the testbed. Therefore, it makes the experiments have realistic radio conditions with complete support of PS (Packet Switch) and partial support of CS (Circuit Switch). Moreover, aside from cost efficient components that are integrated in OpenEPC, it comes with its own radio emulation nodes, which can be engaged in a totally virtualized environment.

OpenEPC has a mobility management module along with network ANDSF (Access Network Discovery and Selection Function) on mobile devices, and also takes advantage of location-based handovers and radio conditions, which makes the client choose the suitable access networks in order to prevent packet loss. In addition to above-mentioned functionalities, policy and charging control, harmonized AAA and subscriber management, accounting and charging, distribution and user plane realization are other features of OpenEPC.

OpenEPC is highly modular with configurable architecture, which makes it suitable for any research development in the field of applications, networks and services. It can simulate a number of different deployment scenarios, which varies from the core network in a single box, to medium-scale distributed testbeds. Figure 8 depicts how OpenEPC virtualizes a real network operator environment.

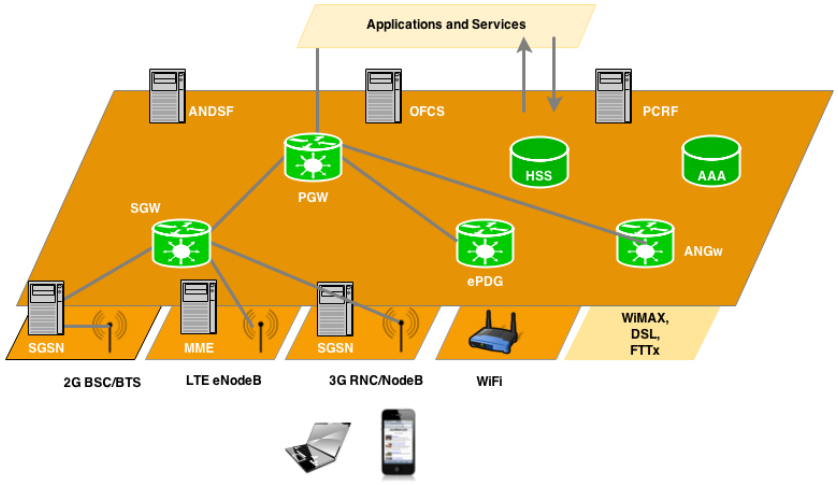

Figure 8: OpenEPC Testbed [43]

The nw-EPC is an open source software package, which is an implementation of the SAE (System Architecture Evolution) gateway. It is written completely in the C programming language and runs as a single-threaded

UNIX process, where the data plane and control plane are both running in the same address space. The goal of nw-EPC is to set up a framework, which implements LTE SGW (Serving Gateway) and PGW (Packet Data Network Gateway) [68]. Figure 9 illustrates how in SAE, different technologies such as 3GPP WLAN, GPRS, Evolved RAN, and non-3GPP networks are transparently unified [76].

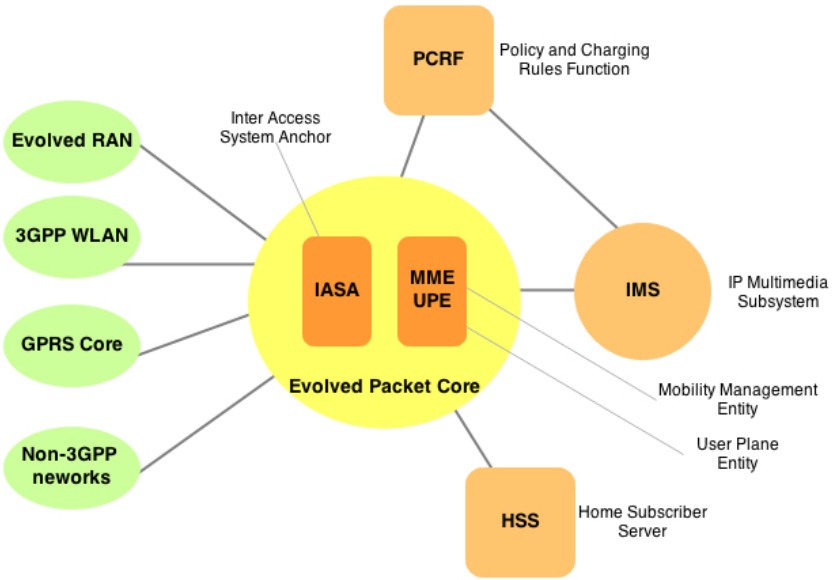

Figure 9: System Architecture Evolution [76]

The control plane in nw-EPC incorporates the SAE gateway behaviour with the help of expandable FSM framework. This enables the nw-EPC node gateway to function as both SAE, and PGW gateways and also other gateway functions, such as WiMAX ASN-GW or GGSN. The data plane in nw-EPC supports IPv4 packet classification, and basic GTP (GPRS Tunneling Protocol) tunneling/detunneling. However, there is lack of support for IPv6, DPI, QOS, traffic shaping and policing.

4 Discussions

4.1 Security concerns

There is a lot of theoretical discussion around SDN, but widely accepted commercial products are still on their way [80]. Most of the discussion has been about the SDN architecture and organization, but while SDN is by no means more vulnerable than traditional networks, the threats exist and are partly different and specific to SDN [74]. The most glaring issue comes with one of the cornerstones of SDN. The centralization of control means that a successful attack on a SDN controller could easily compromise the functionality of the whole network. These attacks could be carried out as Distributed Denial of Service (DDoS) attacks, attacking controller directly, modifying switch behaviour (forged requests, cloning of flows etc.), exploiting administrative stations (gaining access to controller servers themselves) or exploiting a controller with a malicious network applications [74]. The network should also be able to successfully recover from a problem or an incident and hence correct and safe logging and tracing of network states are likely to be required.

Kreutz et al. suggest other general solutions to counter possible security threats. These are controller replication and systems diversity, dynamic associations between network devices and controllers (e.g. if a controller fails, switches associate with a new one), self-healing and software update and patching mechanisms, establishing trust between network devices and controllers and network applications and controllers, establishing security domains (e.g. sandboxing) and using secure components. In conclusion, SDN security is still a relatively unexplored area and for commercial uses of SDN acknowledging and addressing the issues is required.

4.2 Service Chaining Obstacles

In Section 3, we discussed network service chaining and its benefits, however NSC comes with its own challenges. The current network service structure is tight with the network topology, which makes introducing new services or removing them from the chain troublesome. If more services are needed in the chain, the topology should be changed alongside with the new service structure. Due to this coupling between network topology and the service chain, service selection or changing the service transit order based on flow direction is not feasible. [78]

A direct impact of topological dependency is that configuration of a network becomes complex. If a new service needs to be introduced in the service chain, the whole network should be configured again. Therefore once the topology is installed, the network operator tries to avoid any changes in order to not encounter any misconfiguration and consequently breaking down the network, which can constrain the high availability of network.

Because of the topological nature of the current network deployment, there is lack of traffic selection. So, all the traffic on a specific section should go through the whole service function chain, no matter whether the traffic needs to be processed by a particular service or not. Therefore, forwarding technology forces how the data should traverse among services, which causes inflexibility in the network. In addition, classification happens at every service that the traffic passes through, which service functions cannot leverage the classification results from other service functions.

Lastly, End-to-end service visibility is limited in current network deployment. Therefore, when the service function chain is expanded across administrative boundaries, the troubleshooting becomes tough and complex. Moreover, different topologies are deployed on physical and virtual environments, and this topological variance introduces more challenges to the network.

References

[1] *About alien.* http://www.fp7-alien.eu/?page_id=12, visited on 13-08-2014.

[2] *About felix.* http://www.ict-felix.eu/?page_id=25, visited on 12-08-2014.

[3] *About ofertie.* http://www.ofertie.org/, visited on 13-08-2014.

[4] *Beacon controller.* https://openflow.stanford.edu/display/Beacon/Home, visited on 07-08-2014.

[5] *enetconf.* https://github.com/FlowForwarding/enetconf, visited on 08-08-2014.

[6] *The fire landscape.* http://www.ict-fire.eu/home/the-fire-landscape.html, visited on 13-08-2014.

[7] *Floodlight project.* http://www.projectfloodlight.org/floodlight/, visited on 07-08-2014.

[8] *Flower.* https://github.com/travelping/flower, visited on 07-08-2014.

[9] *Flowscale.* http://real-status.com/product/sdn, visited on 08-08-2014.

[10] *Flowvisor.* https://openflow.stanford.edu/display/DOCS/Flowvisor, visited on 07-08-2014.

[11] *Frenetic and pyretic.* http://www.frenetic-lang.org/, visited on 08-08-2014.

[12] *Hyperglance.* http://real-status.com/product/sdn, visited on 08-08-2014.

[13] *Indigo.* http://www.projectfloodlight.org/indigo/, visited on 08-08-2014.

[14] *Indigo virtual switch.* http://www.projectfloodlight.org/indigo-virtual-switch/, visited on 08-08-2014.

[15] *Iris.* http://openiris.etri.re.kr/, visited on 07-08-2014.

[16] *Kulcloud open mul.* http://sourceforge.net/projects/mul/, visited on 07-08-2014.

[17] *Lantern.* https://github.com/CentecNetworks/Lantern, visited on 08-08-2014.

[18] *Libfluid*. http://opennetworkingfoundation.github.io/libfluid/, visited on 07-08-2014.

[19] *Linc-switch*. https://github.com/FlowForwarding/LINC-Switch, visited on 08-08-2014.

[20] *Maestro-platform*. https://code.google.com/p/maestro-platform/, visited on 07-08-2014.

[21] *Mininet overview*. http://mininet.org/overview/, visited on 05-08-2014.

[22] *Mirageos*. http://www.openmirage.org/, visited on 08-08-2014.

[23] *Nettle*. http://hackage.haskell.org/package/nettle-openflow-0.2.0, visited on 08-08-2014.

[24] *Nice*. https://code.google.com/p/nice-of/, visited on 08-08-2014.

[25] *Nodeflow*. http://garyberger.net/?p=537, visited on 07-08-2014.

[26] *Nox controller*. http://www.noxrepo.org/nox/about-nox/, visited on 07-08-2014.

[27] *ns-3*. http://www.nsnam.org/, visited on 08-08-2014.

[28] *Oflib-node*. https://github.com/TrafficLab/oflib-node, visited on 08-08-2014.

[29] *Oflops*. http://archive.openflow.org/wk/index.php/Oflops, visited on 08-08-2014.

[30] *of_protocol*. https://github.com/FlowForwarding/of_protocol, visited on 08-08-2014.

[31] *Ofsoftswitch*. https://github.com/CPqD/ofsoftswitch13, visited on 08-08-2014.

[32] *Oftest*. http://www.projectfloodlight.org/oftest/, visited on 08-08-2014.

[33] *Onos*. http://onlab.us/tools/onos.html, visited on 11-08-2014.

[34] *Open daylight project*. http://www.opendaylight.org/, visited on 07-08-2014.

[35] *Open source ims core*. http://www.openimscore.org/, visited on 08-08-2014.

[36] *Open virtex*. http://ovx.onlab.us/, visited on 11-08-2014.

[37] *Open vswitch.* http://openvswitch.org/, visited on 08-08-2014.

[38] *Opencontrail.* https://github.com/Juniper/contrail-build, visited on 07-08-2014.

[39] *Opendaylight controller:config:examples:netconf.* https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Architecture, visited on 04-08-2014.

[40] *Opendaylight controller:md-sal:architecture.* https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Architecture, visited on 04-08-2014.

[41] *Opendaylight controller:project proposals.* https://wiki.opendaylight.org/view/Project_Proposals:Main, visited on 11-08-2014.

[42] *Opendaylight lisp flow mappin:architecture.* https://wiki.opendaylight.org/view/OpenDaylight_Lisp_Flow_Mapping:Architecture, visited on 6-08-2014.

[43] *OpenEPC, building your own complete mobile broadband operator network testbed.* White paper, Fraunhofer Institute for Open Communication Systems FOKUS. http://www.openepc.net/_docs/OpenEPC-Whitepaper_nov2012.pdf.

[44] *Openfaucet.* https://github.com/rlenglet/openfaucet, visited on 08-08-2014.

[45] *The openflow switch specification.* http://OpenFlowSwitch.org, http://OpenFlowSwitch.org.

[46] *Openvnet.* http://openvnet.com/, visited on 07-08-2014.

[47] *Perfsonar.* http://fasterdata.es.net/performance-testing/perfsonar/, visited on 08-08-2014.

[48] *Pofswitch.* http://www.poforwarding.org/, visited on 08-08-2014.

[49] *Pox controller.* http://www.noxrepo.org/pox/about-pox/, visited on 07-08-2014.

[50] *Resonance.* http://resonance.noise.gatech.edu/, visited on 07-08-2014.

[51] *Routeflow.* http://routeflow.github.io/RouteFlow/, visited on 08-08-2014.

[52] *Ryu controller.* http://osrg.github.io/ryu/, visited on 07-08-2014.

[53] *Sdn troubleshooting system.* http://ucb-sts.github.io/sts/, visited on 08-08-2014.

[54] *Sdncentral.* http://www.sdncentral.com/, visited on 11-08-2014.

[55] *Security challenges in SDN.* http://www.sdncentral.com/security-challenges-sdn-software-defined-networks/, visited on 06-08-2014.

[56] *Snabb switch.* https://github.com/SnabbCo/snabbswitch, visited on 08-08-2014.

[57] *Snac.* http://www.openflowhub.org/display/Snac/SNAC+Home, visited on 07-08-2014.

[58] *Software defined networking for next generation internet components.* http://www.fp7-sparc.eu/goals/, visited on 12-08-2014.

[59] *Teston.* https://github.com/Paxterra/TestON/, visited on 08-08-2014.

[60] *the bird.* http://bird.network.cz/, visited on 08-08-2014.

[61] *Tp-link tl-1043nd.* http://wiki.openwrt.org/toh/tp-link/tl-wr1043nd, visited on 30-07-2014.

[62] *Trema controller.* http://trema.github.io/trema/, visited on 07-08-2014.

[63] *wakame-vdc.* https://github.com/axsh/wakame-vdc, visited on 08-08-2014.

[64] *Warp.* http://flowforwarding.github.io/warp/, visited on 07-08-2014.

[65] *What does ofelia?* http://www.fp7-ofelia.eu/assets/Uploads/About-OpenFlow-OFELIA.pdf, visited on 13-08-2014.

[66] Banks, Ethan: *Big Switch leaves OpenDaylight, toutes white-box future.* http://www.networkcomputing.com/networking/big-switch-leaves-opendaylight-touts-white-box-future/a/d-id/1234231?

[67] Bjorklund", "M.: *Yang - a data modeling language for the network configuration protocol (netconf).* Rfc 6020, Internet Engineering Task Force, October 2010. http://tools.ietf.org/html/rfc6020.

[68] Chawre, Amit: *nw-epc.* http://web.archive.org/web/20120622233936/http://www.amitchawre.net/nw-epc.html, visited on 8-08-2014.

[69] Chua, Roy: *NOX, POX and controllers galore – Murphy McCauley interview*, 2012. http://www.sdncentral.com/sdn-nfv-open-source/nox-pox-controllers-murphy-mccauley/2012/09/.

[70] Erickson, David: *The Beacon Openflow controller*. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-2178-5. http://doi.acm.org/10.1145/2491185.2491189.

[71] Feamster, Nick, Rexford, Jennifer, and Zegura, Ellen: *The road to sdn*. Queue, 11(12), dec 2013. http://doi.acm.org/10.1145/2559899.2560327.

[72] John, W., Pentikousis, K., Agapiou, G., Jacob, E., Kind, M., Manzalini, A, Risso, F., Staessens, D., Steinert, R., and Meirosu, C.: *Research directions in network service chaining*. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, Nov 2013.

[73] KazuyaOkada: *Topologydiscoverywithryu*. http://www.necoma-project.jp/ja/blog/ool-sdn-hackathon14/CloudSDN_Hackathon_20140625_kazuya.pdf, visited on 11-08-2014.

[74] Kreutz, Diego, Ramos, Fernando M.V., and Verissimo, Paulo: *Towards secure and dependable software-defined networks*. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA, 2013. ACM, ISBN 978-1-4503-2178-5. http://doi.acm.org/10.1145/2491185.2491199.

[75] Lorema: *about openwrt*, 2014. http://wiki.openwrt.org/about/start, visited on 30-07-2014.

[76] Marius Iulian Corici, Fabricio Carvalho de Gouveia, Thomas Magedanz: *A network controlled qos model over the 3gpp system architecture evolution*. In *Wireless Broadband and Ultra Wideband Communications, 2007. AusWireless 2007. The 2nd International Conference on*, Aug 2007.

[77] Okada, Kazuya, Hazeyama, Hiroaki, and Kadobayashi, Youki: *Oblivious ddos mitigation with locator/id separation protocol*. In *Proceedings of The Ninth International Conference on Future Internet Technologies*, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2619287.2619291.

[78] P. Quinn, T. Nadeau: *Service function chaining problem statement*. Technical report, August 2014. http://tools.ietf.org/html/draft-ietf-sfc-problem-statement-09.

[79] Robert, B.I.I. and Carman, D.Z.: *System for regulating access to and distributing content in a network*, feb 2012. http://www.google.com/patents/US8122128, US Patent 8,122,128.

[80] Sorensen, Sarah: *Security implications of software-defined networks*. Whitepaper, SDNCentral, 2012. http://www.sdncentral.com/download-sdn-security-whitepaper/.

[81] Yap, Kok Kiong, Kobayashi, Masayoshi, Sherwood, Rob, Huang, Te Yuan, Chan, Michael, Handigol, Nikhil, and McKeown, Nick: *Openroads: Empowering research in mobile networks*. SIGCOMM Comput. Commun. Rev., 40(1), jan 2010, ISSN 0146-4833. http://doi.acm.org/10.1145/1672308.1672331.

[82] Yap, Kok Kiong, Sherwood, Rob, Kobayashi, Masayoshi, Huang, Te Yuan, Chan, Michael, Handigol, Nikhil, McKeown, Nick, and Parulkar, Guru: *Blueprint for introducing innovation into wireless mobile networks*. In *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '10, New York, NY, USA, 2010. ACM, ISBN 978-1-4503-0199-2. http://doi.acm.org/10.1145/1851399.1851404.

[83] Yiakoumis, Yiannis: *Pantou : Openflow 1.0 for openwrt*, 2004. http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT, visited on 29-07-2014.

A Open Source Projects

Table 2: Controllers and related

Name	Usage	Developers & License	Link	Notes
Beacon	OpenFlow controller that supports event-based and threaded operation	Open Source, David Erickson/Stanford University, License: BSD License	https://openflow.stanford.edu/display/Beacon/Home	
Floodlight	Alternative OpenFlow Controller	Open Source, sponsored by Big Switch	http://www.projectfloodlight.org/floodlight/	
FlowER	Erlang OpenFlow controller & development platform	Open Source, TravelPing, License: MIT	https://github.com/travelping/flower	
FlowForward Warp	Parses and encodes OF messages & handles connections with OF datapathelements.	Open Source, InfoBlox Inc., License: Apache 2.0	http://flowforwarding.github.io/warp/	
FlowVisor	Controller that acts as a hypervisor/proxy between OpenFlow switches	Open Source, On.Lab	http://onlab.us/flowvisor.html	Part of ON.LAB's Open SDN Stack.
IRIS	Recursive SDN controller	Open Source, ETRI, License: Apache 2.0	http://openiris.etri.re.kr/	
KulCloud Open MUL	OpenFlow controller	Open Source, KulCloud, License: GPLv2	http://sourceforge.net/projects/mul/	
libfluid	Basic library to implement a controller upon.	Open Source, Open Networking Foundation/CPqD, License: Apache 2.0	http://opennetworkingfoundation.github.io/libfluid/	
Maestro	Control platform with support for OpenFlow switches	Open Source, Zheng Cai/Rice University, License: GNU Lesser GPL	https://code.google.com/p/maestro-platform/	
NodeFlow	Node.js based OpenFlow controller	Open Source, Gary Berger/(Cisco?), License:MIT License	http://garyberger.net/?p=537	
NOX	First OpenFlow controller	Open Source, NoxRepo, License: Apache 2.0	http://www.noxrepo.org/nox/about-nox/	
OpenContrail	Virtual Network Controller, includes analytics engine to handle Big Data.	Open Source, Juniper, License: Apache 2.0	https://github.com/Juniper/contrail-build	
OpenVNet	Provides network virtualization along with components such as SDN controller, virtualswitch and virtual router.	Open Source, Axsh.co, License: LGPL3	http://openvnet.com/	
POX	General SDN controller	Open Source, NoxRepo, License: Apache 2.0	http://www.noxrepo.org/pox/about-pox/	
Resonance	Event-driven network controller	Open Source, Georgia institute of technology	http://resonance.noise.gatech.edu/	
Ryu	SDN Framework written in Python	Open Source, License: Apache 2.0	http://osrg.github.io/ryu/	
SNAC	OpenFlow controller	Open Source, SNAC team, License: GPL	http://www.openflowhub.org/display/Snac/SNAC+Home	
Trema	OpenFlow controller framework	Open Source, NEC, License: GPLv2	https://github.com/trema/trema	

Table 3: Switches

Name	Usage	Developers & License	Link	Notes
Indigo Virtual Switch	A virtual switch	Open Source, sponsored by Big Switch. License: Eclipse Public License version 1	http://www.projectfloodlight.org/indigo-virtual-switch/	Uses Indigo Framework and LoxiGen
Lantern	Software bundle for implementing OpenFlow switches. Includes e.g. SDK, modified OpenVSwitch and more.	Open Source, Centec, License: Apache 2.0	https://github.com/CentecNetworks/Lantern	
LINC-switch	Software Switch	Open Source, FlowForwarding.org, License: Apache 2.0	https://github.com/FlowForwarding/LINC-Switch	
OFSoftswitch	OF protocol 1.3 compatible software switch	Open Source, CPqD, License: BSD	https://github.com/CPqD/ofsoftswitch13	
Open vSwitch	A virtual switch	Open Source, VMware, License: Apache 2.0	http://openvswitch.org/download/	
OpenVNet	Provides network virtualization along with components such as SDN controller, virtual switch and virtual router.	Open Source, Axsh.co, License: LGPL3	http://openvnet.com/	
Pantou	Turns commercial wireless routers/Access Points to OpenFlow-enabled switches	Yiannis Yiakoumis	http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT#Contact	
POFSwitch	Software Switch for POF	Open Source, Huawei, License: BSD	http://www.poforwarding.org/	
Snabb Switch	Virtualized Ethernet networking stack	Open Source, Snabb.co, License: Apache 2.0	https://github.com/SnabbCo/snabbswitch	

Table 4: Testing & Simulating

Name	Usage	Developers & License	Link	Notes
HyperGlance	Network modeling and visualization tool. Supports ODL, OpenStack and AmazonWeb Services.	Real Status, Free beta sign-up	http://real-status.com/product/sdn	
Mininet	Used for creating virtual networks. Useful for creating SDN testbeds.	Open Source, On.Lab, License: BSD Open Source	http://mininet.org/	Part of ON.LAB's Open SDN Stack.
NICE	For testing controller applications	Open Source, NICE, License: BSD	https://code.google.com/p/nice-of/	
ns-3	discrete-event network simulator	Open Source, License: LGPlv2	http://www.nsnam.org/	Has OpenFlow support
OFLOPS	Benchmarking OF switches	Open Source, Rob Sherwood	http://archive.openflow.org/wk/index.php/Oflops	
OFTest	Testing framework and suite for OF compliance testing.	Open Source, sponsored by Big Switch. License: Open Flow Software License	http://www.projectfloodlight.org/oftest/	
PerfSonar	For monitoring network performance	Open Source, EsNet	http://fasterdata.es.net/performance-testing/perfsonar/	
STS	Troubleshooting Simulator	Open Source, STS team, License: Apache 2.0	http://ucb-sts.github.io/sts/	Depends on POX
TestOn	For automating OpenFlow/SDN components	Open Source, Paxterra	https://github.com/Paxterra/TestON	

Table 5: Other projects

Name	Usage	Developers & License	Link	Notes
enetconf	NetConf library in Erlang	Open Source, FlowForwarding.org,License: Apache 2.0	https://github.com/FlowForwarding/enetconf	
Flowscale	**Traffic balancing**	Open source, INCntre	https://github.com/InCNTRE/FlowScale	
Frenetic/Pyretic	DSLs for SDN development. Includes a runtime system	Open Source, Princeton University, License: GNU General Public License version 3	http://www.frenetic-lang.org/	
Indigo	HAL and configuration abstraction for running OpenFlow in hybrid mode.	Open Source, sponsored by Big Switch. License: Eclipse Public License version 1	http://www.projectfloodlight.org/indigo/	Uses LoxiGen
MirageOS	Unikernel for network application development	Open Source, Mirage team, License LGPlv2	http://www.openmirage.org/	Supports OpenFlow
Nettle	DSL written in Haskell for controlling OFswitches	Open Source, Yale University, License: BSD3	http://hackage.haskell.org/package /nettle-openflow-0.2.0	
nwEPC - EPC SAE Gateway	EPC SAE Gateway	Open Source, Thomas Bhatia License: BSD	https://github.com/ thomasbhatia/nwEPC—EPC-SAE-Gateway	
OESS	Controlling VLAN with OF enabled switches	Open Source, NDDI, License: Apache 2.0	https://code.google.com/p/nddi/wiki/README	
of_protocol	OpenFlow Protocol Library in Erlang	Open Source, FlowForwarding.org,License: Apache 2.0	https://github.com/FlowForwarding/of_protocol	
Oflib-Node	OF protocol library for converting OF wireprotocol messages and Javascript	Open Source, Ericsson, License: Custom	https://github.com/TrafficLab/oflib-node	
ONOS	Network OS. NOT YET RELEASED	Open Source, On.Lab	http://onlab.us/tools.html#os	Part of ON.LAB's Open SDN Stack.
Open Daylight	Full SDN stack.	Open Source, Linux Foundation Collaborative Projects, License: Eclipse Public License	http://www.opendaylight.org/	
Open Source IMS Core	IMS - provides CSCFs and HSS	Open Source, Fraunhofer FOKUS	http://www.openimscore.org/	
OpenEPC	Full evolved packet core	Fraunhofer FOKUS, uses open sourcec omponents. Licensing plans available to customers.	http://www.openepc.net/	
OpenFaucet	Python implementation of OpenFlow 1.0.0 protocol	Open Source, Midokura, License: Apache 2.0	https://github.com/rlenglet/openfaucet	
OpenFlowSec utilities	SRI international provides a plethora of security plugins and extensions.	Open Source, SRI International	http://www.openflowsec.org/	
OpenVirtex (OVX)	Network hypervisor for virtual networks	Open Source, On.Lab, License: Apache 2.0	http://ovx.onlab.us/	
RouteFlow	Virtual IP Routing	Open Source, CPqD	http://routeflow.github.io/RouteFlow/	Uses many other components: POX, OpenFlow, Open vSwitch, Quagga, MongoDB, jQuery and JIT.
The BIRD	Internet routing daemon	Open Source, CZ.NIC Labs, License: GPL	http://bird.network.cz/	
Wakame-vdc	Used for data center virtualization.	Open Source, Axsh.co, License: LGPL 3.0 and Apache 2.0	https://github.com/axsh/wakame-vdc	