Open Source Platforms, Applications and Tools for SDN and 5G research

TBD

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

Helsinki, August 11, 2014

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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

Tekijä — Författare — Author

TBD

Työn nimi — Arbetets titel — Title

Open Source Platforms, Applications and Tools for SDN 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-X	August 11, 2014	19

Tiivistelmä — Referat — Abstract

Software-Defined Networking (SDN), a novel solution to network configuration and management, has shown great potential to simplify the existing complex and inflexible network infrastructure. For the design strength, SDN is gaining various investment from both industry and academia in terms of experimental implementation and deployment. In this report, we present. Our focus is on . We discuss the potential research 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

2 Open Source SDN Components

Even though the concept of software defined networking dates back to year 2004 [41] the first widely recognised applications followed only several years later. The most notable one is the widespread OpenFlow protocol [22] 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 doesn't 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 [13]. NOX is written in C++ and is meant for developing further customised 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 [24], the Python version of NOX, for most tasks [35]. 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.

Beacon is another ntremaotable OF controller [1]. It was released in 2010 and it's written in Java. Despite Java's reputation as an inefficient programming language Beacon has fared well performance-wise in comparison with other controllers [36]. Java was chosen to address C's and C++'s portability problems and to reduce developer's burden with significantly shorter compilation times and better error logging. Other controllers with similar focus on easier development are for example Python-based Ryu [26] and Ruby-based Trema [30].

One of the features that makes Beacon perhaps more notable than other aforementioned controllers is the fact that it serves as basis for one of the most

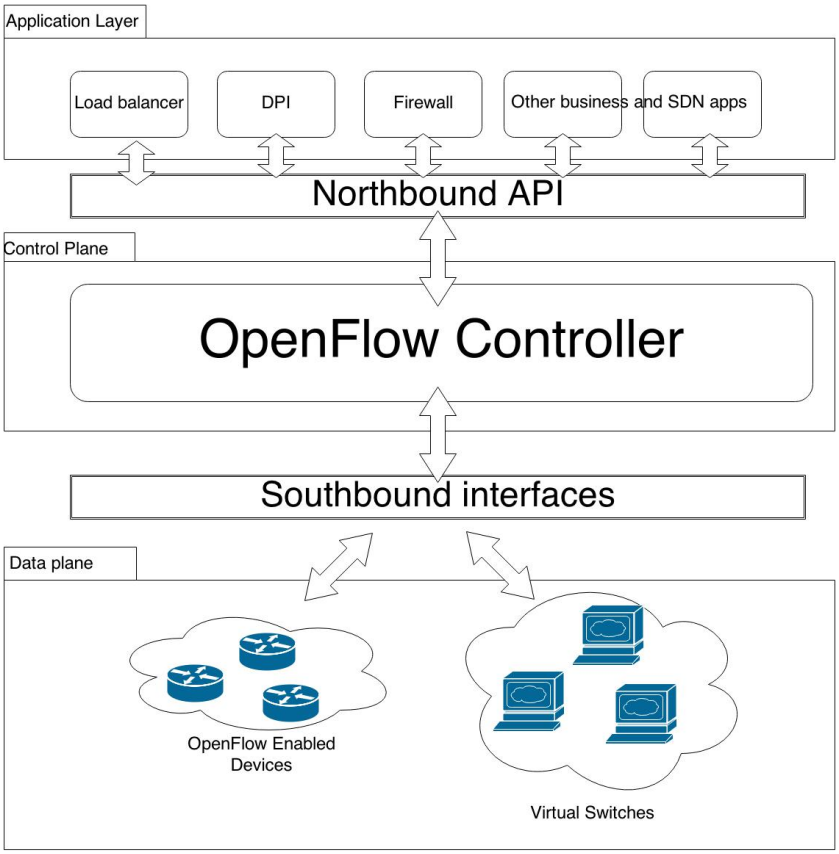

Figure 1: General SDN controller architecture

popular OpenFlow controllers: Floodlight [2]. Floodlight is developed by Big Switch Networks, a startup founded by networking veterans. Unlike Beacon, Floodlight doesn't rely on 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 [32] but the project opted for Cisco's controller implementation.

Open Daylight Project [15], or ODP in shorthand, is commonly considered the leading SDN project with it's 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 and VMware and many more. Feature highlights of Open Daylight include a robust REST API and southbound communication which allows for using other non-Openflow protocols: OVSDB, NETCONF, LISP, BGP, PCEP and SNMP. Naturally ODP 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 interface 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 be intimidating and distracting, but generally ODP community offers decent documentation and guides though quality varies from one component to another. In comparison with Floodlight ODP is likely to be more viable in real business environment because of the features it offers and fast evolving nature but when developing quick prototypes or SDN applications that only make use of OpenFlow, Floodlight may be a better choice due to it being simpler, smaller and at the moment better documented.

Other lesser known controllers and controller frameworks are for example a very bare bones Libfluid [8], Java-based Maestro [10], On.Lab's Flowvisor [4] which is meant to be used with other On.Lab's SDN components, FLOWer [3] which is written with Erlang, the event-driven Resonance[25], Node.js -based NodeFlow [12], lesser-known KulCloud Open MUL and SNAC citeMUL, SNAC, flowforwarding.org's Warp [31], network virtualization bundle Open VNet [23], IRIS which uses Floodlight and Beacon components to offer horizontal scalability [6] and Juniper's OpenContrail focusing on SDN and Big Data [17]. 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.

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 near synonymous to SDN and network functions virtualisation or NFV being a hot topic for conversation along with SDN there's a demand and supply for standardised, vendor-agnostic virtual and physical switches. With SDN and network virtualization it is possible to eliminate the need for specialised switches, routers, middleboxes etc. and utilise the network elements on ordinary X86 servers.

VMWare's Open VSwitch is undoubtedly the most well known of virtual switches [16]. 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 few different protocols.

Other software switches include Indigo Virtual Switch meant to be used with Floodlight and Indigo framework [5], CPqD's OFSoftswitch [14], LINC-

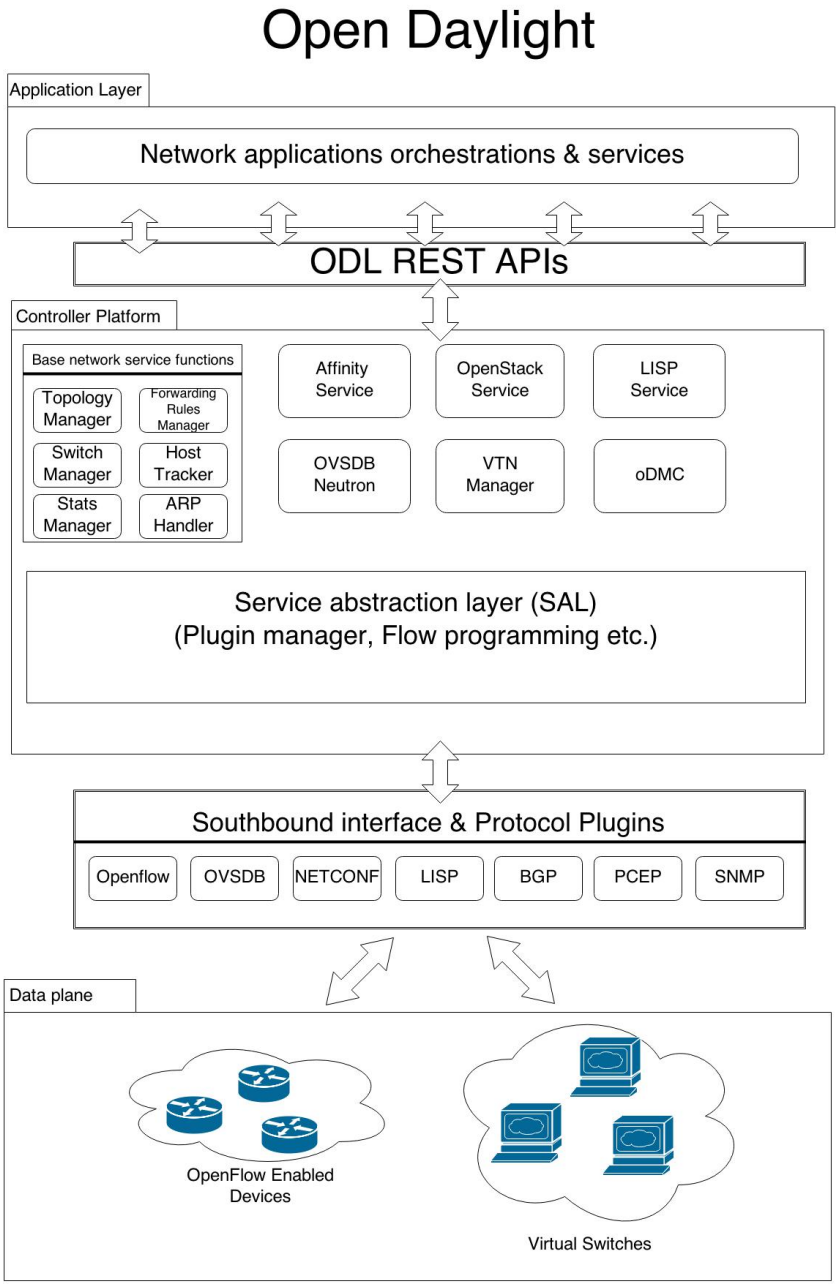

Figure 2: Open Daylight architecture

switch [9], Snabb.co's virtualized ethernet switch called Snabb Switch [28] and protocol oblivious POFSwitch [**?**]. The previously mentioned open VNet also includes a virtual switch [23].

Other switch related projects are Centec's Lantern [7] which includes a modified Open VSwitch but is mainly design for implementing custom

hardware based SDN switches with the accompanying software bundle. Other partly similar software is Pantou [44]. 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. Tools fall roughly in 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. 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 [11].

A little bit more specific tool is STS which stands for SDN Troubleshooting System. 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. For performance testing and benchmarking there are OFLOPS and PerfSonar and for testing SDN software itself one can utilize NICE for controller applications, OFTest for compliance testing and TestOn for testing automatization. It's also worth to mention HyperGlance by Real Status. It is a network visualization tool which supports Open Daylight. It could most likely to be used for debugging as in addition to network topology it displays traffic too. Unlike other pieces of software mentioned HyperGlance isn't 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 don't straightforwardly fall into these categories.

There hasn't been as much talk about security with SDN as there have 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 network's functionality is crippled [27] . 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 is a network load balancer, but the development has stopped after version 0.6. This may be because of many controller applications including load balancer of their own. Both the BIRD by CZ.NIC Labs and CPqD's RouteFlow offer IP routing.

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

Other various project include Big Switch's Indigo, a hardware abstraction layer and configuration layer meant for communication between floodlight and the physical layer, MirageOS, used for constructing unikernels for network applications, wakame-vdc for data center virtualization and for wireless communication there is nwEPC - EPC SAE Gateway by Thomas Batia and Open Source IMS Core from Fraunhofer Fokus and OpenEPC of which latter is not an open source project but otherwise worth mentioning in this context.

EU has various research projects under moniker Seventh Framework Programme (FP7) which has produced both research and open source software. Some individual projects in FP7 like FELIX and OFELIA and FIRE research test-beds for use and frameworks for using them (OFELIA). Other projects focus more straightforwardly to OpenFlow. ALIEN project for instance offers an OpenFlow enabling hardware abstraction layer for non-OpenFlow enabled devices and integration with OFELIA whereas OFERTIE aims to improve delivery of real-time interactive applications on the networks. SPARC project aims to enhance standard SDN architecture by splitting the control and forwarding functions by utilizing MPLS.

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 is different between vendors, which makes work of network administrators exhausting to configure each of these individual network devices. [37]

SDN changes the way networks are managed and designed by defining two characteristics. First, SDN detaches control plane from the data plane. Second, SDN unifies the control plane, thus a single software program (controller) can control numerous data plane elements. SDN controller has direct access over the data plane through a well-defined API such as OpenFlow. An OpenFlow switch has packet-handling table, which 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. [43]

Openness and an open architecture can provide improved features in the network delivered by new industry suppliers, which can support 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 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, mobility manager is aware of every application flow in the network and can choose the route for specific flow. Therefore, when the user travels from one point to another, mobility manager become aware of the user's movement and can decide to reroute the flow. Due to 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

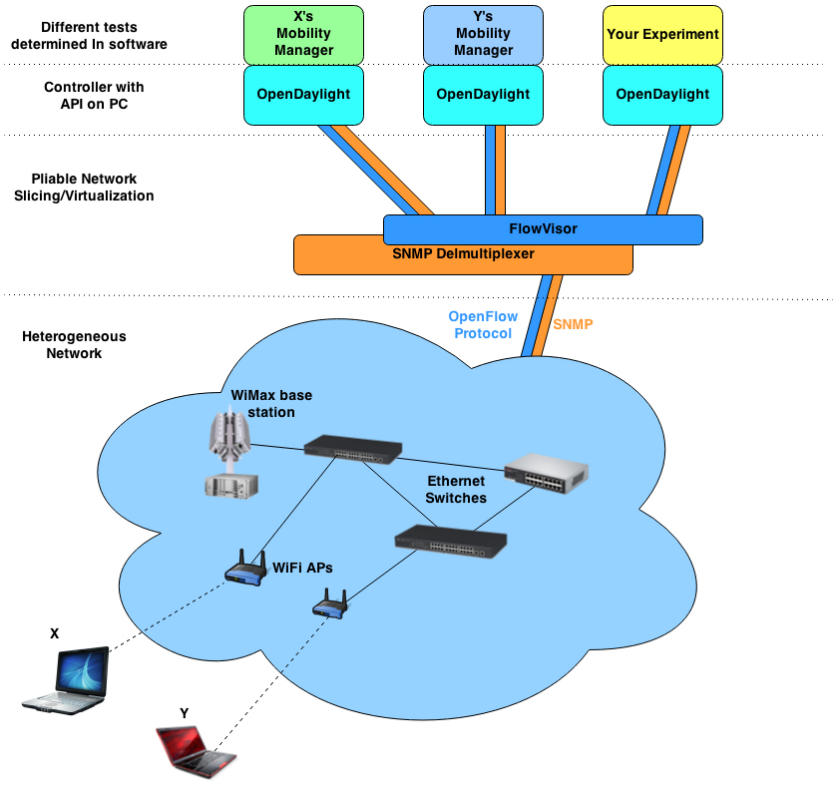

Figure 3: OpenRoad's Architecture [42]

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 communicate to the datapath and controllers using OpenFlow protocol, the datapath think that they are controlled by single controller, and 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 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 the John's traffic, then FlowVisor transmits all the 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, afterward made available to the whole network, or even shared with other network owners.

Finally, slicing allows decomposition, so network administrators can decompose 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 hierarchy of control in the network. For instance, network manager can dedicate a slice to a research group in the computer science department, and then that research group 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 with FlowVisor. In order to slice the configuration, SNMPVisor observes the SNMP control messages, and forward 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 access point or wireless router into OpenFlow enabled device. So, basically in Pantou, OpenFlow operates on top of OpenWRT as an application [44]. 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. [39]

OpenWRT is designed in such way to be user friendly, therefore users can choose their desired packages, configure them, and build their own custom firmware [39]. Depending on the model of router that 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. [29]

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 remain this open source firmware, generic, and always up to date alongside with the advancement of technology.

3.2 Network Service Chaining

Traditional network infrastructure, has brought 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 complexity of current network infrastructure. Current network infrastructure in NSPs, cannot fulfill their needs, and this is because of inflexibility in their network. [38]

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 varies from hours to even weeks depending to the size of network.

Due to inter-dependencies and low power of automatic configuration that middleboxes (services) have, if there is a new service want to be introduced or removed from the network platform, they need careful engineering and customization. By increasing the automation, 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 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) that middleboxes are implemented, they can be integrated seamlessly.

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

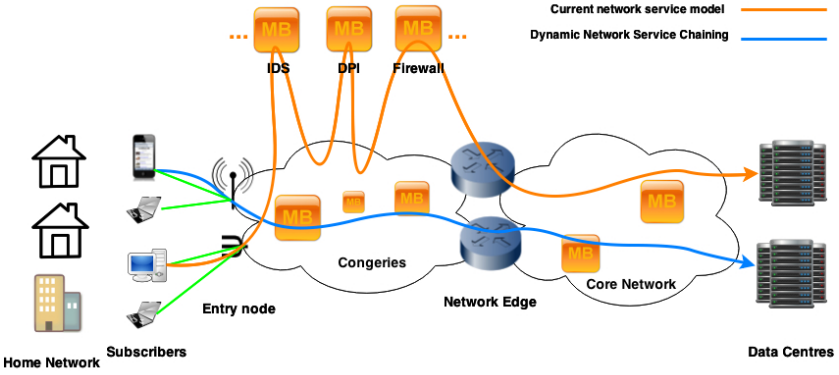

Figure 4: NSC vs. Current network service model [38]

domain, middleboxes such as load balancers, traffic schedulers, security gateways, etc, provide better fairness and security. However, when it comes to multiple network domain, 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 to the number of policy elements that data is passing and cross-domain network load, this deferment varies. But, when NSC is implemented in the edge of 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.

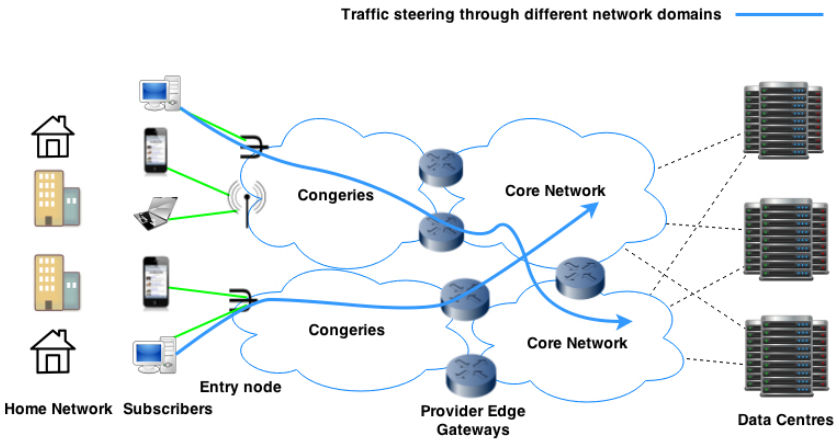

Figure 5: Traffic steering in dynamic NSC [38]

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 XML tree to model hierarchical data, where each node has a name, value or series of child nodes. Moreover, format of event notifications, which are published by the network elements can be specified by YANG. In YANG modeling language, signature of remote procedure call (RPC) can be specified by data modelers, which can be invoked on network components through NETCONF protocol. [33]

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 [19] [18]. In these projects, YANG permits to model structure of XML data, describe the semantic components and their connection, and to model all the elements as a unified system [19].

YANG data model uses XML, which makes the data to be self explanatory. Moreover, it is easy for the data to be utilized in applications and controller components that use controller's northbound API. Using YANG data model, facilitates the development of applications and controller components. Therefore, the risk of bad interpretation of 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 routing address point for endpoint identifiers on the network. One of 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 to modify the configuration of the device. [40]

Figure 6 illustrates the order of sending packet from client to the server using 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 to the destination EID, by sending 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), which its job is to decapsulate the received packet and forward it to chosen sever.

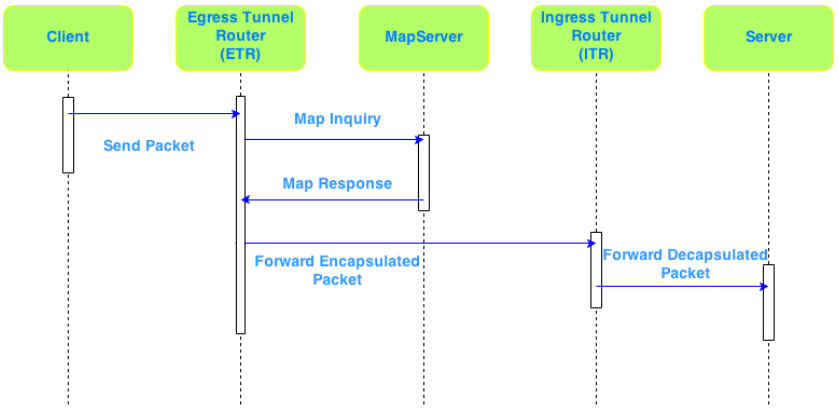

Figure 6: Packet forwarding between LISP routers [40]

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, control plane is separated from data plane. Therefore, LISP in regards to data plane, defines how encapsulation of virtualized network addresses should be done in addresses from underlying network. In addition, control plane stores the binding information between EIDs and RLOCs, and associated forwarding policies, therefore data plane can fetch these information whenever it receives new flows. [20]

LISP Flow Mapping project in OpenDaylight, utilizes LISP infrastructure, which provides mapping system services. This project, consists of 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 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 LISP control protocol can utilize this service via southbound LISP plugin through the LISP control protocol. Figure 7, illustrates the above mentioned components.

3.5 OpenEPC and nw-EPC

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

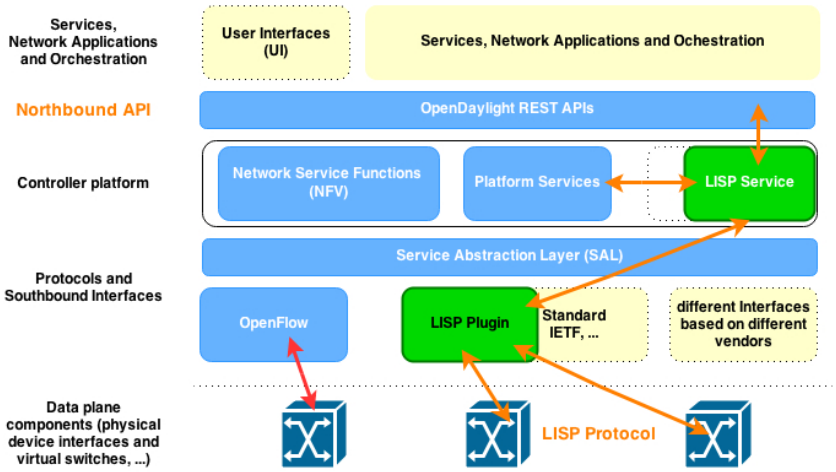

Figure 7: LISP in OpenDaylight [20]

of wide variety of protocols, components and interfaces that have to work simultaneously, which can impact on one another's behavior. There are numerous challenges in order to have reliable and cost-efficient testbed which can test applications before the final product development. OpenEPC tries to minimize these challenges with offering a novel approach. [21]

OpenEPC entirely simulates the operator core network, with providing a good tool for demonstrations and 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 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 operator environment. Moreover, it contains all the elements and main functionalities of 3GPP EPC standards, in addition to its own features, which makes it enable for all kind of IP communications such as LTE, EDGE, HSPA, and even non-3GPP accesses such as WiFi.

OpenEPC comes with different functionalities, which we are going to discuss some of them briefly. Core network mobility management is one of OpenEPC functionalities, where it has some of necessary features for establishing the user plane. These features are including 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 to have control over wireless connectivity of devices to connect them to the testbed. Therefore, it makes the experiments to 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 totally virtualized environment.

OpenEPC has 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 to 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 field of applications, networks and services. It can simulate different number of deployment scenarios, which varies from core network in a single box, to medium scale distributed testbeds.

The nw-EPC is an open source software package, which is implementation of SAE (System Architecture Evolution) gateway. It is written completely in C programming language and runs as single-threaded UNIX process, where 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 is implementing LTE SGW (Serving Gateway) and PGW (Packet Data Network Gateway). [34]

The control plane in nw-EPC incorporates the SAE gateway behaviour with the help of expandable FSM framework. This enables 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.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] *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.

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

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

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

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

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

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

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

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

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

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

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

[33] 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?

[34] 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.

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

[36] 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/.

[37] 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.

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

[39] 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.

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

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.