SDN applications - the intent-based Northbound interface realisation for extended applications

Minh Pham
iNEXT: Centre for Innovation in IT Services and Applications
University of Technology, Sydney
Sydney, Australia
Minh.Pham@student.uts.edu.au

Doan B Hoang
iNEXT: Centre for Innovation in IT Services and Applications
University of Technology, Sydney
Sydney, Australia
Doan.Hoang@uts.edu.au

***Abstract*—The Northbound Interface (NBI) plays a crucial role in promoting the adoption of SDN as it allows developers the freedom of developing their revenue-generating applications without being affected and constrained by the complexities of the underlying networks. To do so the NBI has to allow applications to express their requirements and constraints in their own application specific language, and the SDN controller to translate those requirements into SDN network specific language for provisioning network resources and services to satisfy the application requirements. The intent-based NBI is born from this consideration and the Open Networking Foundation (ONF) provides principles and guidelines to build such an intent-based NBI. However, these principles do not lend themselves readily to the design and practical realization of an intent-based NBI for extended classes of business-like network applications. This paper introduces a solution and its initial implementation in the form of a novel architecture for realizing the intent-based NBI. The new solution exploits the modularized and reuse features of the micro services and service oriented architectures.**

Keywords—intent-based NBI, Software defined network, micro-service architecture, domain driven design, NBI application architecture

I. INTRODUCTION

Software-Defined Network (SDN) is a network technology that separates the control plane from data plane in network devices and centralizes control in the controller. This leads to major benefits: simple network devices, the control of the networks is implemented in the software and via programmability, allowing network virtualization and automation, and the openness of SDN programming interfaces to all service providers and network providers according to Hoang and Pham [1]. SDN architecture consists of three separate layers as shown in Figure 1: application layer, control layer and infrastructure layer or data plane. SDN control layer, which includes one or more controllers, plays an import role in the SDN operations because it provides an abstraction to build network applications to operate on the networks. The controller provides an abstract topology for network devices, device management, etc. It is considered as a network operation system that provides network services via interfaces: the southbound interface (SBI) to network devices and the northbound interface (NBI) to applications [2], [3].

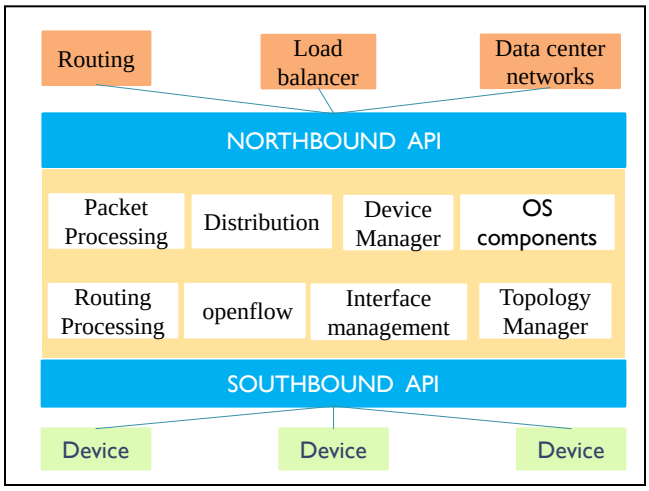

Figure 1: SDN architecture

Despite many benefits of SDN, its adoption depends critically on the ability to support all types of applications via the NBI. Currently, the NBI is yet to be standardized; existing solutions are vendor-specific, ad hoc, and limited in capability. Realizing this, the Open Networking Foundation (ONF) established a Work Group (WG) called NBI WG to manage the development of NBI in controllers. Tadepalli [4] identified two types of NBI usages: prescribed usage, in which users dictate what network services they want in their applications, and intent-based usage, which is completely contrast with the prescribed one. In intent-based NBI model, users describe their requirements of network application in normal conversation language and the controller becomes an intelligent black box that integrate core network services to construct network applications to serve users' requests [5]. Our main concern is in the fulfilling system because the more powerful it is, the more powerful applications it can build. Figure 2 describes the concept of black box controller in intent-based NBI.

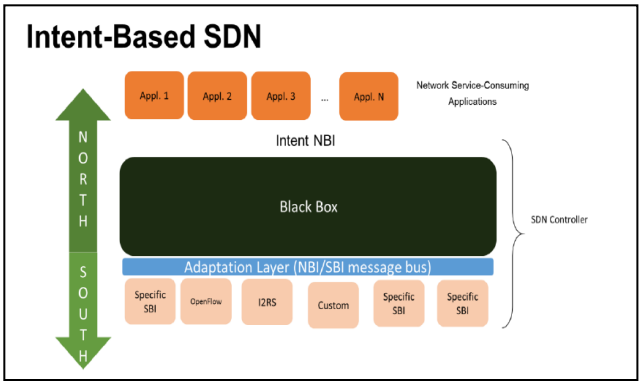

Figure 2: Intent-based NBI, controller as a black box to serve customers' request [5]

Traditional SDN network applications and services are built on top of SDN and mainly concerned with traffic engineering, networking management, access control, network security and data center networking [6]. However, these applications represent only a small portion of network-enabled resources encapsulated and offered as core services through current generation by controllers. The spectrum of SDN applications is much wider than that. Almost all applications require network support to some degree, the real task of an SDN controller is to understand the application requirements and provision required services through the management of its underlying network resources. In increased complexity, Jarraya, et al. [7] and Rao [8] identified many applications through use cases of SDN in cloud computing, information content networking, mobile networks, network virtualization (NV) and network function virtualization (NFV).

In a particular mobile application delivery, over the cloud on a global scale, Paul, et al. [9] identified the task list, which includes service partitioning based on network or service contexts and content, and service composition of detailed components based on requirements. These and other business applications represent a class of emerging applications. Clearly, a well-defined NBI is needed to allow application of all classes to express its requirements and constraints in their own term, and to provide SDN controllers to translate the application requirements to the underlying network requirement and provision the required services. The intent-based NBI is created for this purpose [5], the separation of the fulfilling system enables us to build rich network applications and services as other commercial systems. Currently ad-hoc and application specific NBI are feasible to support traditional applications that mainly use the core services from the controllers. Existing NBIs, however, are not able to support the emerging group of applications without proper definition and well-design intent-based architecture for a unified NBI because of the lack of functionality [9], including service partitioning, network and service context checking, service composition functionalities, etc.

This paper adopts the ONOS NBI intent framework and proposed an intent-based NBI architecture that allows an application to express its objectives, its policies, its requirements and constraints without the need of a network-specific language or the understanding of how the network is being deployed to satisfy its request. Our intent-based NBI is designed to reflect this understanding in order to support a large class of applications. The proposed intent-based NBI architecture is essential to translate diverse application needs, expressed in application-specific language, to the expressions the controllers can understand and function.

Our architecture supports not only traditional and emerging applications but also allows them to create new services and to compose, orchestrate, choreograph or split current services. One of the innovative features of our NBI architecture is its deployment of the micro service and service-oriented architectures with divide-and-conquer domain driven design.

II. RELATED WORK

Following the work of Janz [5], several projects on intent-based NBI were undertook in different controllers. ONF initiated officially Boulder [10] to build a model of intent-based NBI for Open Day Light (ODL) controller. The intent model is similar to the intent framework in ONOS controller. At HP [11] the Intent engine framework (NIC) supported multi applications via detecting and resolving policy conflicts. Project NEMO [12] at ODL concentrated on building a language to translate intents into network processes and services. Cisco designed and built the cloud infrastructure using micro-services [13].

These projects addressed the missing NBI standards raised in [6], [14], and [7] by focusing on the language for translating intents and the framework to handle multiple applications rather than the need of an architecture that is flexible and robust to build extended network applications and services.

ONOS controller provides the application intent framework as its NBI, in which an intent is an abstraction that describes a network connectivity in the format of network policy. While intent-based NBI is a usage that ONF identifies in the controller NBI architecture, in ONOS, application intent framework is a subsystem of the controller that serve applications as NBI. Its purpose is to allow applications specify networks as polices instead of low level rules, then intents are compiled into network devices' flow rules and installed into the devices [15].

The intent framework is organized into an application framework, which is a set of functions that are designed specifically for an application domain so developers do not need to code from scratch to build the application. The framework usually provide access points that users can call or customize, the rest of the framework is considered as a black box and users do not need to know. Developers can customize the application by deriving new classes and overriding required functions of the framework [16].

Figure 3 summarizes the three main use cases of the application intent framework in ONOS controller, the first use case is when an intent is created and submitted via the NBI, after that the framework will compile, store and install intent flow rules on devices via the SBI. The second use case is the result of the changes in the network environment itself, the monitoring system identified the change and raised an event to

recompile the intent. The third use case is when users submit a withdraw intent via the NBI, users need to provide intent id and application id to identify the intent, the intent framework will remove the intent and its flow rules from the store, and from the devices. The application intent framework (below the NB Interface) in figure 3 is invisible to users as these functions operate under the framework's controls.

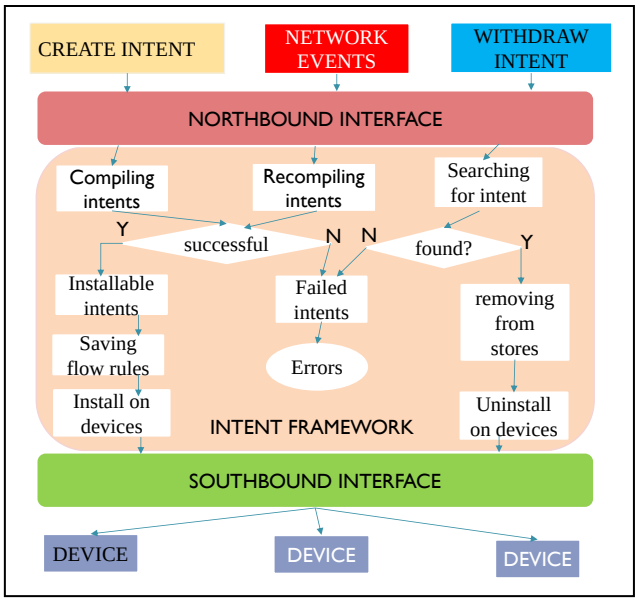

Figure 3: Use cases of the ONOS intent framework

The application intent framework, ODL NBI or Floodlight NBI, and others facilitate application development somewhat but they are inadequate for constructing extended and emerging applications and services because they lack functionality for service partitioning, service composition, service context checking and support for handling complex business logic on top of core network services. In the next section, we will describe our proposed solution architecture to build extended applications and services.

III. THE PROPOSED SOLUTION ARCHITECTURE

The proposed solution must satisfy all requirements of an intent-based NBI architecture that is described in the next subsection.

A. *The requirements of a proposed solution for intent-based NBI realisation*

The extended applications and services need to response to dynamic changes in the application context, and the ability to implement complex service composition based on policy, configuration or performance requirement. The intent-based realisation system needs to handle the interpretation from the intent natural language into the terms that are meaningful to the network core services of the controller. It must be able to handle the intent's composability attribute via the service composition. The architecture should allow creation of new services, reuse of existing services and composition these into new applications. The application should be user-friendly with different user interfaces such as web UI or CLI. The architecture should ensure the separation of application environment from the controller environment so that applications will not interfere with controller. For the architecture, the composability is the main architecturally significant requirement (ASR) [17], because it determines the procedure that allows divide-and-conquer solution approach, enables modular design, and reuse of components to compose the application. Figure 4 visualises the architecture from the user requirement view. Some requirements are linked together for handling complex business rules allowing new service creation, service reuse, and service composition or partitioning.

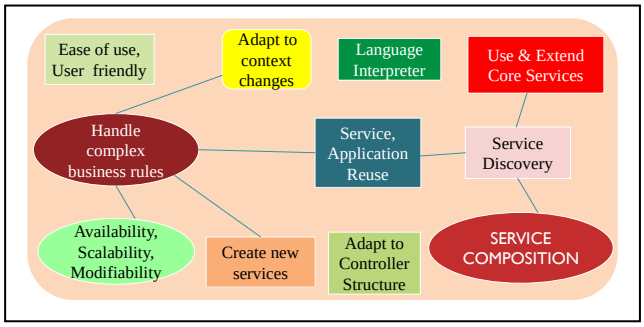

Figure 4: Visualized the requirements of the architecture

B. *The proposed solution architecture*

The proposed architecture is designed based on all requirements identified in the previous section. We adopt the micro service oriented architecture and construct a three-tier application architecture for realizing our NBI solution. In this section we describe components of the solution.

1) Micro Service Architecture (MSA) design

MSA is part of the service-oriented architecture that promotes the service pattern of service provider, service consumer and service discovery [18], [19], [20], and allows the flexibility in managing applications with design principles.

- Data decentralised principle: in our design, each application has its own database management system, to ensure application independence from data perspective.
- Componentised application: web service components constructs are used in the application to promote reuse of existing services and applications.
- Process isolation: In our design, each application runs separately in its own process, so if one application is down, it will not affect other applications and the controller.
- Application design robustness: to ensure is the application robustness, our design is for both successful and failure scenarios.

The proposed architecture takes into account the MSA design pattern to facilitate service discovery and reuse. These include pattern of self-registration whereby an application

registers itself as a service when it starts up and service client lookup whereby client applications use the registry to search for their services.

2) The three-tier application architecture

The proposed architecture is a three-tier architecture for network applications and services as it satisfies the requirements of our solution and serves well the development of rich commercial applications. The three tiers are the database tier, the business logic tier and the presentation tier, and they work cohesively to deliver results.

- The database tier stores application states. It can be relational database, and no SQL database such as Mongo DB, so application states can be stored and retrieved in different contexts.
- The business tier handles requirements' complex business logic via service creation and service composition. It includes the service registry to discover existing services including core network services and other services. The controller's NBI and core services are used as the atomic layer in the service orchestration because the application wants to retrieve abstract network topology provided by the controller. New services are created as atomic service or composite service. The service integration element integrates new and existing services to create the application.
- The presentation tier accepts input from users and provides different interfaces of the application: CLI, the REST interface, and the programming interface. The three tiers work independently to provide the most flexibility: each tier can change its components without effects to other tiers.

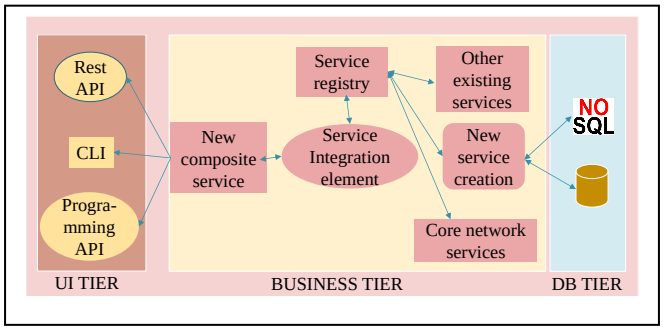

Figure 5: Component view of the proposed architecture

Figure 5 depicts the three-tier pattern architecture and its components. The left most is the user interface tier with Rest API, programming API and Command Line Interface (CLI), business tier is in the middle with new component service creation, core network services, other existing services, service registry, service integration element and new composite service; and database tier is the right most. The new composite service is the application to be returned to the requester.

3) Domain driven design

In the analysis process, the proposed architecture selects domain driven design (DDD) principle over other software designs because DDD matches well with the composition attribute of the intents in the intent-based NBI, and fits in with MSA principles [21]. Applying DDD, the requirements is decomposed into smallest problem subdomains (i.e., a composite intent can be decomposed into based intents), and then solutions of each subdomain are built (base intents composed into a solution intent). The solutions of subdomains can be the core services in the controller or other existing service and new services that NBI builds for the application [22].

Figure 6 demonstrates the DDD process: the problem domain is divided in three sub domains SD1, SD2 and SD3. Using the skill and experience of the domain expert the solution domain is designed with different business contexts that is represented by bounded contexts.

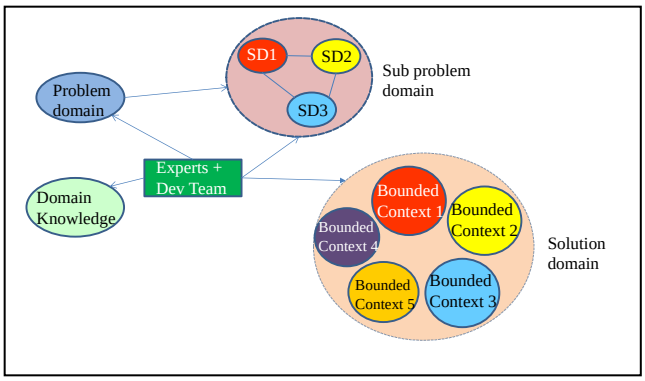

Figure 6: Domain driven design example

4) The controller platform OSGi

Popular controllers written in Java like Beacon, ODL Floodlight, and ONOS are built on the OSGi platform. The core architecture of OSGi technology is a dynamic component system in Java [23]. OSGi has different layers and figure 7 describes its layers.

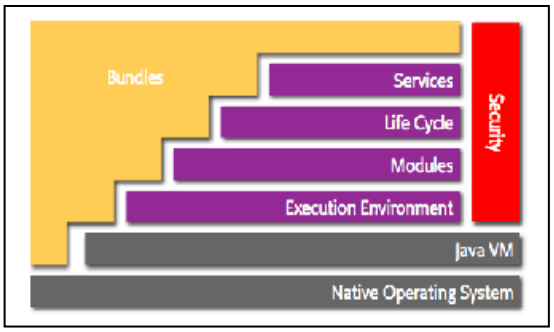

Figure 7: the OSGi platform [23]

Developers divide application into logical components called bundles and develop the bundles using core java language. Each bundle has a manifest file to describe about it. In each bundle, the bundle context is the only interface to connect to the OSGi framework. To resolve the dependencies between bundles, OSGi framework allows bundles to export and import for reference. A bundle can be registered as a

service via the service registry. A bundle retrieves the service from the registry; and then calls its functions. Life cycle is used to install, start, stop, update and uninstall of bundles and services. Module defines how bundles can import and export code for code dependencies. OSGi framework has enterprise OSGi to connect to services and framework developed by Java EE / Spring technology by wrap Java EE function / service in OSGi service. OSGi also provides blueprint that is used for service definition.

In the next section we will describe the prototype of the proposed architecture in ONOS controller to build an application called dynamic resource management [24].

IV. PROTOTYPE

In this section, the prototype to build the DRM application based on the proposed architecture is used to evaluate the hypothesis described in section I. The evaluation information will be gathered in two groups: the steps to build DRM using the proposed architecture and the results of running DRM application in ONOS. We describe the DRM application setup in the next subsection.

A. DRM application setup

Mijumbi, et al. [24] proposed a solution to network virtualisation that emphasized the efficient management of networks' resources. In data centre, users' request for virtual networks (VN) comes randomly, and the mapping to the substrate networks creates the required networks. This gradually depletes switches and links' resources. If resources are not well managed, they will be too fragmented that results in further users' request will be rejected and revenue loss. [24] used SDN to provision virtual networks. It also based on a resource database that records the nodes' and links' resources and the VN cost. And each VN is provisioned based on the least cost path: of the available paths, the one with least ratio of usage resource over available resource is chose. It proved that the dynamic resource management solution improved the request acceptance ratio by 40%.

The DRM application is originally implemented in Floodlight controller. We applied our proposed architecture and design principles presented in section III to implement the DRM application. The three tier application architecture and the DDD were used to build the DRM. The DRM requirements were decomposed into the resource management element and the virtual network creation element.

The resource management used database tables to store usage and available resource of switches and links in the substrate network.

The virtual network creation element used ONOS topology service to search for paths between the two endpoints in input parameters.

The service registry used the built-in OSGi service registry. The service composition element is the program itself. The programming interface and Rest API were used to integrate services, and then the resource management was called to determine the path with the least ratio.

The result path could be empty if it does not satisfy the bandwidth requirements, and if the path exists, the application created an intent for it and returned the result to users.

In ONOS UI views, the intent was displayed and we could view the traffic in the created intent. The resource of switches and links of the substrate network were used as variable for the testing: the virtual network created between two switches was varied when resources of network elements along its path were changed.

B. Network setup

We set up the Geant network in Mininet as in the DRM project. The Geant network includes 40 switches and 122 links between switches across different countries in Europe; it is the main network to promote research and education between involved nations [25]. Based on the Internet topology zoo data in .gml format, we extracted the switches and links between switches, then we created these switches and links in Mininet using python programming language and python' Mininet code libraries. The last step of the setup was deploying the DRM on to ONOS controller running on top of the Geant Mininet network using Mininet commands.

C. Results

The results are divided into two categories: the applying of the proposed architecture and the result of the running DRM on the Geant network based on resources setup in the database as explained in the following sub-sections:

1) Applying the proposed archiecture

We have successfully implemented the prototype. The architecture promotes modularity via the three tier architecture pattern, decentralised data management and componentised application thus enhances the modifiability of the architecture if compared to service oriented architecture. It removed the mystery in intent-based NBI and make the realisation of extended application as straight forward as other business applications. It promotes the economic values of service-based architecture via the maximum reuse of existing components and services that are available in the controller as well as outside. The three-tier architecture promotes a neat architecture and ensures that applications have their own components in three-tier and will not interfere with controller's components, architecture and frameworks. With any type of requirements, the steps in DDD in the architecture can be applied to integrate into the required application. The application is also robust because the design always concerns of success and failed scenarios. Finally, using the controller core service as the atomic layer, it ensures the realisation process always exploit fully the topology abstraction provided in the controller.

2) Running DRM in ONOS controller

When the DRM application was running on the ONOS controller, the available resources were uploaded into the database. Available resources include the available and usage resource for switches and links based on Geant network, they are organised into two separate tables: available resource and usage resource. We used Rest client to run the test cases on the browser and used ONOS UI to check whether the intent was created and its details. The log data was useful to follow the

trace of the program. The actual results were matched with the manual calculations in the expected results and were recorded in the table 1.

Table 1: Test results of running DRM on ONOS

Test case 1 details	Source: switch 09, Destination: switch 04, Bandwidth: 50
Resource setup	ONOS returns two paths between 09–04 Path 1: 09-08-04: average usage / availability ratio: 102/600, Path 2: 09-29-04: average usage / availability ratio: 102/270
Expected result	Path 1 with the least average ratio
Actual result	An intent was created for Path 1
Test case 2 details	Source: switch 00, Destination: switch 31, Bandwidth: 50
Resource setup	ONOS returns two paths between 00–31 Path 1: 00-04-31: average usage / availability ratio: 102/200 Path 2: 00-02-31: average usage / availability ratio: 102/600
Expected result	Path 2 with the least average ratio
Actual result	An intent was created for Path 2
Test case 3 details	Source: switch 04, Destination: switch 12, Bandwidth: 50
Resource setup	ONOS returns one path between04-12 Path 1: 04-29-15-12, all links are >=100
Expected result	Path 1 should be returned
Actual result	An intent was created for Path 1

V. CONCLUSIONS

In SDN network paradigm, NBI plays an important role because it is the base to build innovative network applications, and it lowers the barrier for new comers to enter the network application market to generate more products for consumers. In this paper we studied the NBI architecture and the intent-based NBI guidelines by ONF WG. The intent-based NBI approach is what needed to develop extended and business-like network applications but there existed hardly a practical architecture to realize the NBI. We have proposed an architecture based on micro service and service-oriented design principles and three tier application architecture to realise the intent-based NBI. We deployed ONOS and OSGi platform to build our prototype. We used the prototype to construct the Dynamic Resource Management application. The results demonstrated that the proposed architecture is appropriate, effective and realizable for extended, business-like network applications for SDN with the an intent-based NBI. In this SDN domain, we believe that the proposed architecture is the first of its kind and presents a step forward in promoting modularity, service reuse, robustness and flexibility in managing controller and applications. The proposed solution suggests a practical methodology for building business-like network applications and services in an intent-based NBI. It also provides a business case for the standardization of NBI in SDN controllers.

REFERENCES

[1] D. Hoang and M. Pham, "On Software-defined networking and the design of SDN controllers," presented at the Network of the Future 2015, Montreal, Canada, 2015.

[2] P. Goransson and C. Black, *Software Defined Networks*. USA: Morgan Kaufmann, 2014.

[3] D. Hoang, "Software Defined Networking - Shaping up for the next disruptive step?," *Australian Journal of Telecommunications and Digital Economy*, Vol 3, No 4, pp.46-62, December 2015.

[4] R. Tadepalli, "NBI _ Northbound Interface Framework".

[5] C. Janz, "Intent NBI - Definition and Principles," 2015.

[6] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *IEEE Proceedings*, vol. 103, pp.14-76, Jan 2015

[7] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," *Communications Surveys & Tutorials, IEEE*, vol. 16, pp. 1955-1980, 2014.

[8] S. Rao, "SDN and Its Use-Cases NV and NFV," Nec Technologies India Limited, India2014.

[9] S. Paul, R. Jain, J. Iyer, and D. Oran, "Mobie applications on global clouds using openflow and software-defined networking," in *Network Innovation through Openflow and SDN*, ed: CRC Press, 2014, p. 19.

[10] OpensourceSDN. (2015, 12 January). *Boulder Intent-based NBI*. Available: https://www.google.com.au/search?q=NBi+project+boulder&ie=utf-8&oe=utf-8&gws_rd=cr&ei=31CUVuq5MsOf0gT5kZPICA

[11] ONFSummit. (2015, 12 Jan). *What is the intent anyway* [Video]. Available: https://www.youtube.com/watch?v=QvEK_CFIGik

[12] ODL. (2015, 12 January). *NEMO*. Available: https://wiki.opendaylight.org/view/NEMO:Main

[13] Cisco. (2015, 12 January). *Microservices Infrastructure*. Available: https://github.com/CiscoCloud/microservices-infrastructure

[14] T. Zhang and F. Hu, "Controller Architecture and Performance in Software-Defined Networks," in *network Innovation through OpenFlow and SDN*, ed The United States: CRC Press taylor & Francis Group, 2014.

[15] T. Vachuska. (2014, 18 January). *ONOS overview*. Available: https://www.youtube.com/watch?v=3lya-MY1cZw

[16] P. Arpaia and V. Inglese. (2014). *Flexible Test Automation : A Software Framework for Easily Developing Measurement Applications*. Available: http://UTS.eblib.com.au/patron/FullRecord.aspx?p=1911813

[17] L. Bass, P. Clements, and R. Kazman, *Software Architecture In Practice*. Upper Saddle River, NJ: Addison-Wesley, 2013.

[18] M. L. Fowler, James. (2014). *Microservices*. Available: http://martinfowler.com/articles/microservices.html

[19] J. Lewis. (2012, 11 September 2015). *Micro services - Java the UNIX way*. Available: http://2012.33degree.org/pdf/JamesLewisMicroServices.pdf

[20] C. Richardson, "Microservice Architecture Patterns," 2014.

[21] C. Richardson, "Developing Functional Domain Models with Event Sourcing," presented at the Javazone 2015, Oslo, Norway, 2015.

[22] S. Millett, *Patterns, principles, and practices of domain-driven design*: Wrox, 2015.

[23] OSGi. (2007, 3 December). *Architecture*. Available: https://www.osgi.org/developer/architecture/

[24] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, F. De Turck, and S. Latre, "Dynamic resource management in SDN-based virtualized networks," in *Network and Service Management (CNSM), 2014 10th International Conference on*, 2014, pp. 412-417.

[25] University_of_Adelaide. (2012, 12 January). *The internet topology zoo*. Available: http://www.topology-zoo.