FINAL YEAR PROJECT DOCUMENT

Implementing VNF (virtual network function) in control management plane (Ligato) for data plane (VPP)

Zahid-Ur-Rehman
Uob: 15026401

Supervisor Name: Dr.Adnan Iqbal

Abstract

Contents

Introduction

Networking in the old days, 1999 was about the speed, because at that time the data which essentially was in use were files. Around 2005, things like YouTube and Skype came out, VoIP became a bigger deal. Now working with these software's, we are now doing Real Time Communication (RTC), so it is different from transferring of files which means that Real Time Communication cares about latency. In 2013, we have encountered a problem where more and more devices are connecting to the network. It was getting to a point that sometimes FTP traffic is more important than VoIP traffic or vice versa. That is where SDN (Software Defined Networking) comes in to play. SDN can dynamically model and shape our traffic depending on the need. Now in 2018 another word is around the corner and that is NFV (Network Function Virtualization). NFV are like SDN because both are a software form of network. With NFVs or VNFs we are virtualizing the functions of a data plane of SDN, for example: when you buy a router it is not in the form of software. The router is embedded in a specific hardware which is made for routers only. Now what if someone gives you a router in a software form? With that, you can run the software on any type of hardware that is what actually what NFV is doing.

This project aim is to implement a plugin in VPP-Agent which is a component of Ligato, a control plane in our case, and will be discussed later in report.

This document covers the details about implementation of a virtual network function (VNF) in Ligato, a control/management plane for VPP which is data plane.

About Project

This project is implementation of plugin inside control/management plane, VPP-Agent for VPP. Overviews to these unfamiliar terms are here in the Introduction chapter, but will be described in more detail in the next chapter.

Ligato is an open source cloud native orchestration platform which is implemented in GO (Programming Language). It compromises of different components; one of them is VPP-Agent. VPP-agent implements plugins which manage all virtual network functions or plugins for VPP like ACL, IPsec, L2, L3 plugins. There are a lot of plugins for virtual network functions of VPP which are yet to be implemented in VPP-agent. Our task is to implement one of these plugins in VPP-agent. Agent would use GoVPP for communication with VPP, it is used because VPP and Ligato are implemented in different languages therefore a language converter is required. In order to communicate with VPP-Agent, applications will use protocol buffers.

VPP is a virtual router, which comprises of different virtual network functions. It is implemented in C language. We call it virtual because it is independent of some specific hardware like in old routers. VPP provides cli; it is used to test VPP plugins. In cloud domain we can say VPP is a data plane.

Motivation

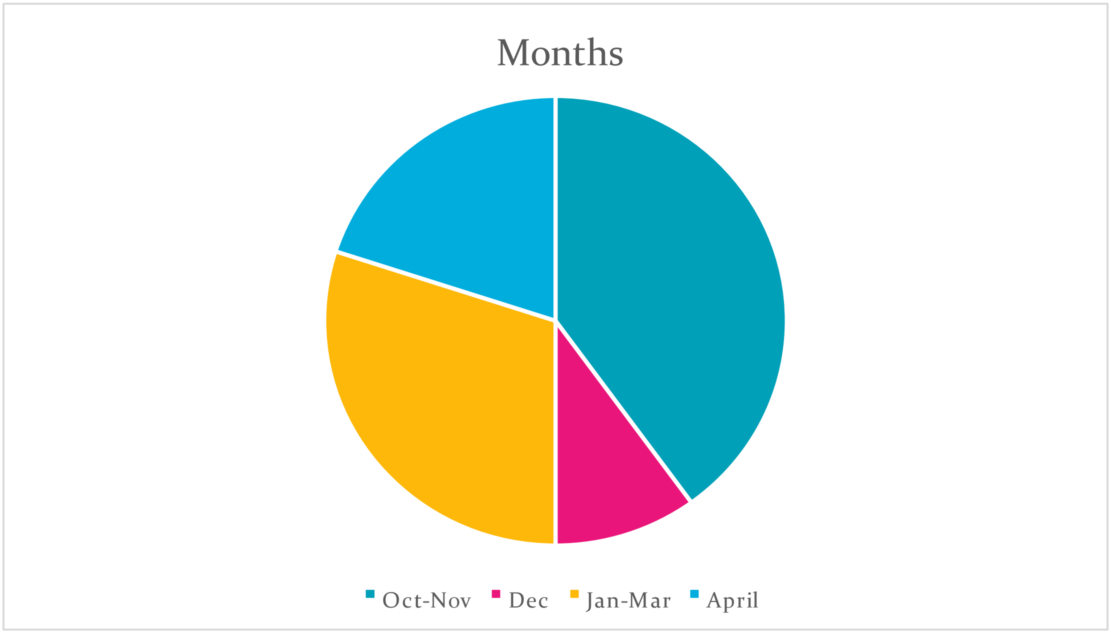
At this time, SDN and NFV have a significant role in networking, and enterprises require different kind of plugins, all in one place. VPP is continuously under development processes and many of networking plugins have been added to it, therefore, these plugins have to be implemented inside VPP-Agent (control/management plane for VPP). This project will implement a management plugin for VPP in VPP-Agent, so if someone wants to use that specific plugin, he can use it.

This project will help in understanding the concepts of cloud computing, and about VPP-Agent, so if VPP-Agent need some kind of optimization, it will be easy to do so.

Next chapter is literature review, in next chapter I will briefly describe the research work which I have studied for doing this project.

Implementation Strategy

As I am implementing VPP based VNF in Ligato, here is the chart which shows my implementation into portions which I will try to achieve on time.



Oct-Nov: Studying about specific VNF, testing and understanding of VPP cli.

Dec: Conversion of VNF VPP api in to golang objects and testing these objects with the goVPP.

Jan-Mar: Implementation of VNF inside VPP-agent.

April: Additional work e.g. integrating VPP-agent with kubernetes.

Literature Review

The networking has now entered in the era of: Micro-service architecture, Cloud computing, Containerization and Cloud native infrastructure, therefore this part will contain overview of these latest networking domain and the tools which are necessary for this project.

Cloud Computing

It is a domain in which we access remote servers and machines by the help of network to store, manage and processes data, rather than a local server or machine [1].

Cloud computing is a domain of computer science, to my understanding whenever we are talking about virtual things or when we want to make the user independent of hardware dependencies we are talking about cloud computing.

Cloud Computing consist of three Service Models which are [2]:

- Software As a Service (SaaS): this Service provides user to access to provider's application hosted on a cloud infrastructure.
- Platform As a Service (PaaS): this Service provides user to deploy user created applications on a cloud infrastructure.
- Infrastructure As a Service (IaaS): this service provides user to deploy and run software's, which include operating systems and applications. The user has control over operating systems, storage and deployed applications.

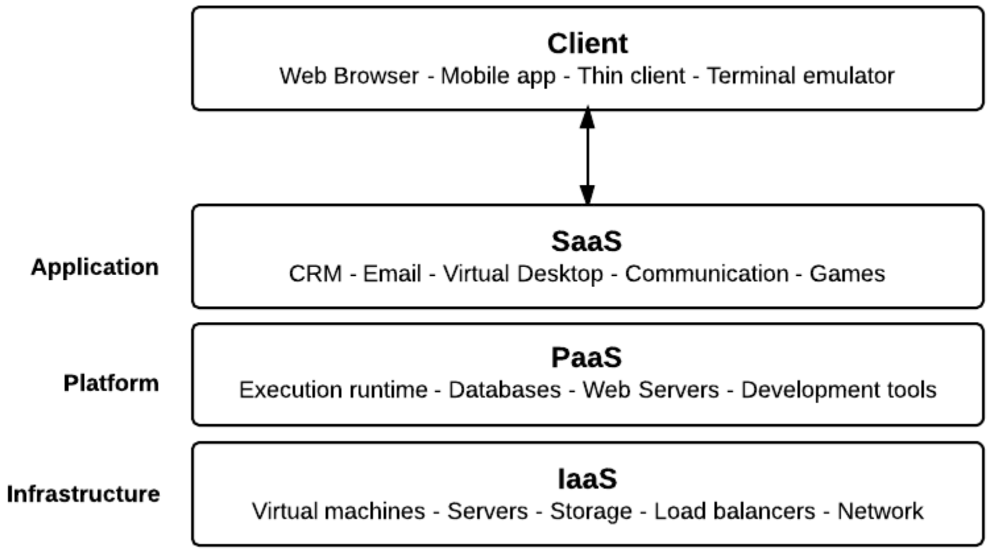

Client
Web Browser - Mobile app - Thin client - Terminal emulator
Application
SaaS
CRM - Email - Virtual Desktop - Communication - Games
Platform
PaaS
Execution runtime - Databases - Web Servers - Development tools
Infrastructure
IaaS
Virtual machines - Servers - Storage - Load balancers - Network

Figure 1: Cloud Service Model[3]

As cloud computing is a hot topic in networking, also this project is related to NFVs (Network Function Virtualization), therefore this section will describe Cloud Computing from network management perspective which include SDN (Software defined Networking) and NFV.

SDN

Software-defined networking (SDN) is an approach of cloud computing that facilitate network management [4].SDN idea is to improve the traditional decentralized network by decoupling the intelligence (routing processes) and forwarding processes of network packets. So essentially what happens is that we separate networking into Application layer, Control layer and Data layer [5].

- Application Layer: it contains network applications, such as firewalls or load balancers.
- Control Layer: It is a set of management servers that communicate with all of the different networking equipment on a data plane and will allow how at a particular instance, data will be prioritized. So control plane works like the brain for a data plane.
- Data Layer: It comprises of switches and routers which can be either physical or virtual that allow packets to go from point A to point B.

The communication between these three layers is done with their respective northbound and southbound APIs.

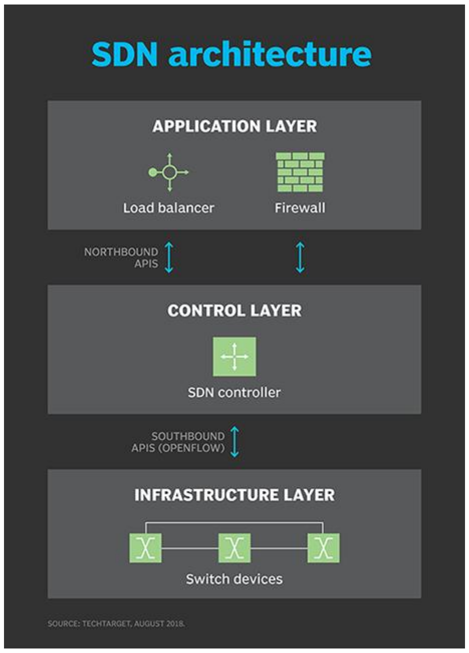


Figure 2: SDN Architecture [5]

From over project perspective the Data layer is VPP (A Set of Cloud-native VNFs), Control layer is VPP-Agent, northbound API is protocol buffers and southbound API is GoVPP.

NFV

It is an initiative to virtualize network services which is currently running on dedicated hardware [6]. One can virtualize different service or plugins of physical router like: routing, firewalls and load balancing and each plugin would be called as a VNFs (Virtual Network Functions), and these VNFs are necessary component of NFV design.

The benefit of NFV is that it eliminates specific hardware dependency, so network administrators can add, move or change network functions at server level [6].

There is another term known as Cloud-native VNFs (CNFs). A Cloud-native VNFs is designed for emerging cloud environments, it runs in a container and managed by container orchestration system such as Docker SWARM or Kubernetes.

It is not just theoretical, the reflection of CNFs can be seen in VPP, and it is a high-performance data plane. VPP consists of set of VNFs.

VPP (High Performance Data Plane and NFV)

VPP (Vector Packet processing) provides switch/router functionality. Vpp is implemented in “C” Language. VPP is made by set of VNFs. VPP is open source project, it is used in diverse applications , which includes Virtual Switch (SW) in enterprises to support virtual machines and containers, as well as in 4G/5G technology and in security. VPP can traverse up to 12 million packets per second (Mpps) with our general purpose CPU. VPP runs in user space rather than kernel space, this is because with recent improvements in transmission speed and network card capabilities, a general purpose kernel stack is too slow for processing packets at wire speed. Therefore, bypassing operating system kernel and bringing hardware abstraction directly to user space is done by netmap and DPDK (Intel Data Plane Development Kit) [7].

VPP process packets in a ”vectorized” fashion rather than scalar forwarding. Instead of letting each packet traverse the whole forwarding graph in a scalar manner, each node processes all packets in a batch, it improve performance [7].

Figure 3 illustrate how packets are traversed in VPP [7]:

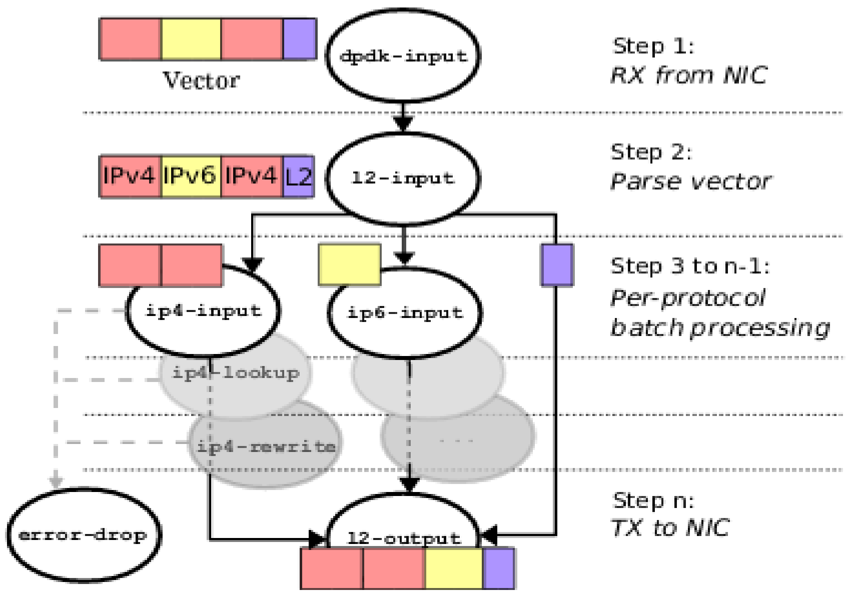


Figure 3: Vectorized Packet Processing [7]

Step 1: dpdk-input is used for packet reception.

Step 2: Full vector is passed to next node for vector parsing.

Step 3: vector is split to multiple protocols and drop decision is applied at every step.

Step n: finally output packet is passed to NIC.

Comparison with Click (Virtual Router)

To improve the flexibility in network is also done by the Click Virtual Router: the main idea of click router is to move the network functionalities performed by specific hardware to software form [8]. Similar to VPP, Click also lets user arrange functions as a processing graph, providing full stack of network functions [7]. In Click each packet is traversed by whole tree, while in VPP each traversed node processes all packets which improve the overall throughput of VPP.

Click runs as kernel thread [8]. The kernel thread runs the router driver s; however VPP runs in user space [7]. One can say that while VPP running in user space can be overhead, because, it includes system calls but as we know that DPDK and netmap are low-level building blocks for kernel bypass referred as KBnetes (Kernel-bypass networks).

Containerization

Cloud relies on virtualization techniques to achieve elasticity of large scale shared resources [9]. Virtual machines (VMs) being the back bone and providing hardware allocation and management, and VMs are fulfilling IaaS (Infrastructure as a Service) requirements. Containers are similar, more light-weight virtualization technique, achieve PaaS (Platform as a service) requirements [9].

Different containers technologies are [9]:

- Linux: Docker, LXC Linux containers, OpenVZ, and others for variants such as BSD, HP-UX and Solaris.
- Windows: Sandboxie
- Cloud PaaS: Warden/Garden (in Cloud Found- ry), LXC (in Openshift)

In this project, Docker containers are in use.

Docker

It is software that performs operating-system-level virtualization, also known as "containerization" [10]. Containers of Docker are run by a single kernel and thus are more light-weight then VMs. Containers are created by Docker Images, where Docker Image are readable layer and Containers are writable layer.

Cloud-Native Micro-services

With emergence of cloud computing, many enterprises are considering cloud as target for migration [11].Application scalability is one of the important aspect and it is possible with scalable architecture. Cloud-native architectures like micro-services have these characteristics: availability and scalability [11]. Micro-service architecture is replacing monolithic architecture; it contains application logic with deployable unit. For small system, monolithic approach is still better but when system grows, problems like understanding of code, increased deployment time and scalability of intensive load will occur [11].This is where micro-services is the solution providing small services which can be deployed and scaled independently also they are easy to understand.

In this project, Ligato CN-Infra (Cloud-Native Infrastructure) is a platform for building these micro-services.

Ligato

Ligato is an open source project that provides a platform for development of cloud-native VNFs.it has two main components: CN-Infra (Cloud-Native Infrastructure) and VPP-agent.

CN-Infra

CN-Infra is a component of Ligato implemented in Go Language, and used for development of cloud native micro-services [12]. It was first intended to use for developing control/management plane for cloud-native VNFs. Each app built on top of CN-Infra is a "Plugin", which provide a very specific functionality.

The architecture of CN-Infra is shown in figure:

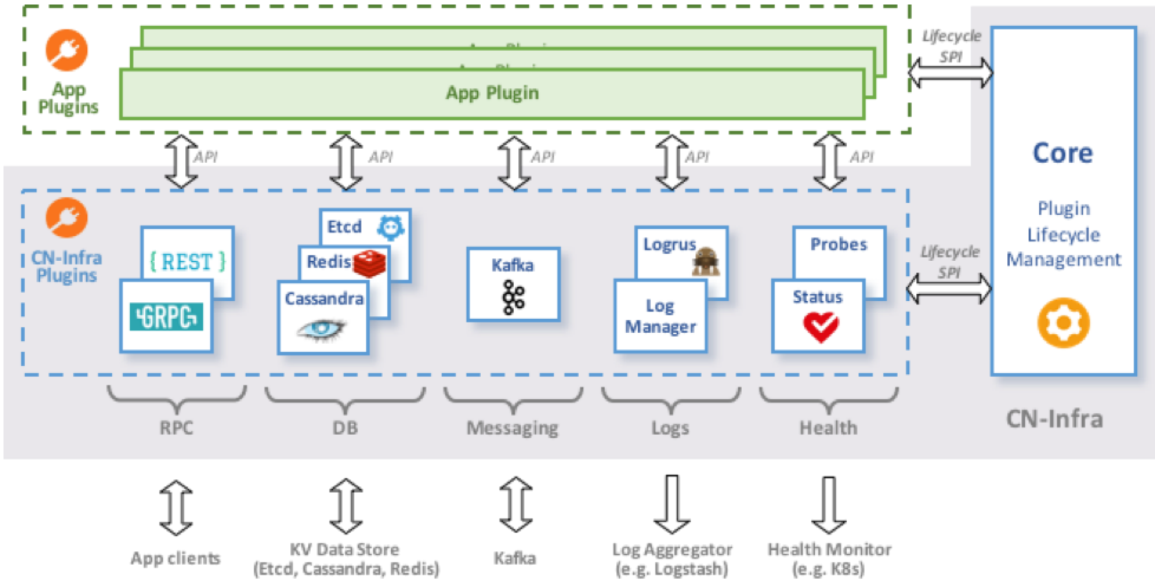


Figure 4: CN-Infra Architecture [12]

The Core manages the plugin lifecycle. App can also use APIs provided by CN-Infra plugins for communication purpose.CN-Infra is modular and extensible. New functionality can be easily added to existing CN-Infra platform. In this project, the important micro-service which will be used is Etcd (Key-Value data store). The apps which will use VPP-Agent will dump there configuration in Etcd data store and VPP-Agent will read configuration from Etcd data store.

VPP-Agent as Control/Management Plane

It is a component of Ligato which is used as control/management plane for VPP based cloud-native VNFs [13].VPP-Agent is implemented in Go Language. VPP-Agent consists of VPP-Specific plugins that use the CN-Infra platform to interact with other micro-services in cloud. VPP-Agent exposes functionality of VPP to client apps by the help of higher-level modern-driven API.

VPP-Agent architecture is shown in following figure:

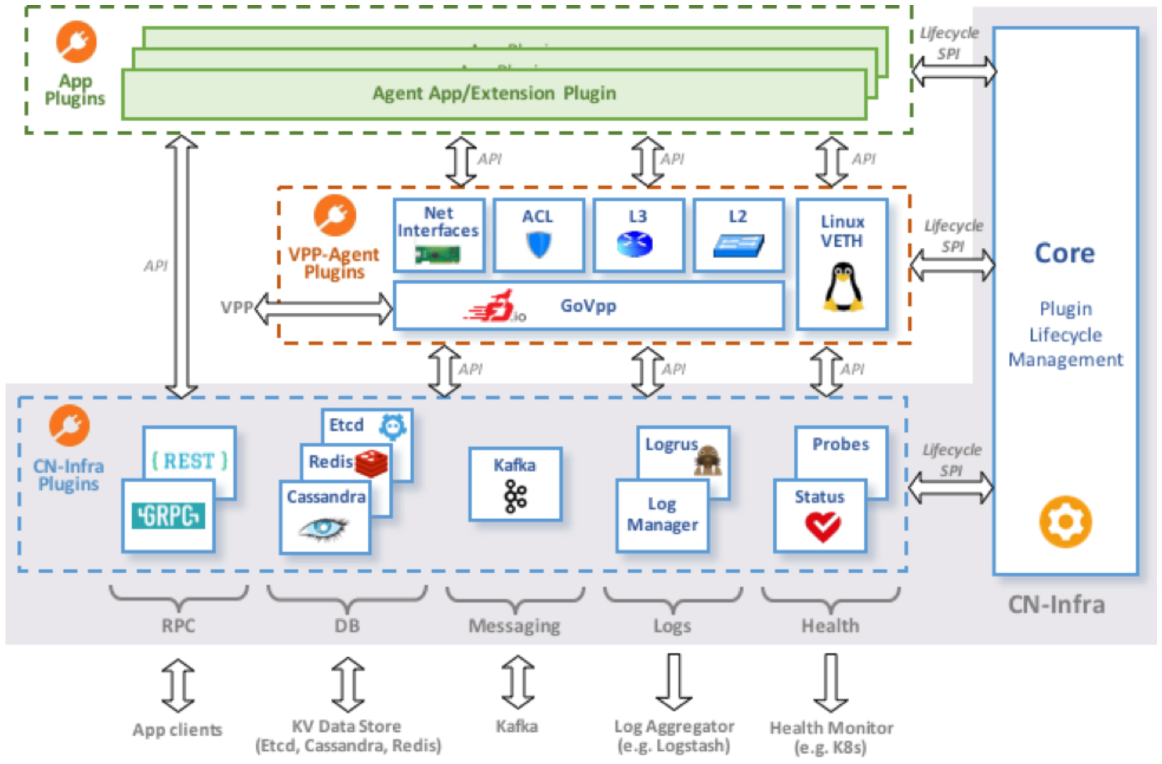


Figure 5: VPP-Agent Architecture [13]

VPP-Agent use Protocol Buffers as Northbound APIs and GoVPP library as Southbound APIs [13].

Protocol Buffers are used for serializing structured data [14]. The main aim of Protocol Buffers is to highlight simplicity and performance, it is faster and smaller then XML. Protocol Buffers serves as basis for remote procedural call (RPC) that is used for inter machine communication at Google [14].

Design

Design of this project includes several components, which are working together. As project is about implementation of plugin inside VPP-Agent, the Agent will act as control/management plane and will use northbound and southbound APIs in order to work as a full service. The plugin which will be implemented in VPP-Agent will manage plugin of VPP which is our data plane. GoVPP will be used as southbound API and Protocol Buffers will be used as northbound API.

The whole design of project will look like this:

App → Protocol Buffers → Etcd →VPP-Agent → GoVPP → VPP

App user will pass configuration to plugin using Protocol Buffers as northbound API and these configurations will be dumped in Etcd. VPP-Agent will sync new configurations from Etcd and will pass it to plugin, the configuration will be then mapped to plugin logic and then VPP-Agent will send these configurations to GoVPP, it will then map the Go Language to C Language and will pass it to VPP.

In first part following components are integrated together and tested:

Components	**Description**	**Results**
GoVPP → VPP	VPP binary APIs were used to generate Go code of particular plugin. The Go code was then used in GoVPP to pass some specific configurations to VPP to check whether APIs are running correctly or not.	VPP cli was used to see the result, and it was satisfactory. The configurations passed from GoVPP were viewable in VPP cli.
Protocol Buffers→ GoVPP → VPP	Protocol Buffers (just like XML for front-end purpose) will be used to define the model of plugin or what specific information will user pass for particular plugin and then it will be passed directly to GoVPP and rest of work is same as above description.	VPP cli was used to see the result, and it was satisfactory. The configurations passed from Protocol Buffers to GoVPP were viewable in VPP cli.

References

[1] Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A. and Stoica, I., 2009. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, *28*(13), p.2009. [Accessed 10 Jan. 2019].

[2] Mell, P. and Grance, T., 2011. The NIST definition of cloud computing. [Accessed 10 Jan. 2019].

[3]Different Types of Cloud Computing Service Models .https://www.bluepiit.com/blog/different-types-of-cloud-computing-service-models/.[Accessed 10 Jan. 2019].

[4] Benzekki, K., El Fergougui, A. and Elbelrhiti Elalaoui, A., 2016. Software-defined networking (SDN): a survey. Security and communication networks, 9(18), pp.5803-5833. [Accessed 10 Jan. 2019].

[5] software-defined networking (SDN). https://searchnetworking.techtarget.com/definition/software-defined-networking-SDN

. [Accessed 10 Jan. 2019].

[6] network functions virtualization (NFV) . https://searchnetworking.techtarget.com/definition/network-functions-virtualization-NFV . [Accessed 10 Jan. 2019].

[7] Linguaglossa, L., Rossi, D., Pontarelli, S., Barach, D., Marjon, D. and Pfister, P., 2017. High-speed Software Data Plane via Vectorized Packet Processing (Extended Version). *Tech. Rep.* [Accessed 11 Jan. 2019].

[8] Kohler, E., Morris, R., Chen, B., Jannotti, J. and Kaashoek, M.F., 2000. The Click modular router. *ACM Transactions on Computer Systems (TOCS)*, *18*(3), pp.263-297.[Accessed 11 Jan. 2019].

[9] Pahl, C., 2015. Containerization and the paas cloud. *IEEE Cloud Computing*, *2*(3), pp.24-31. [Accessed 12 Jan. 2019].

[10] Vivek Ratan (February 8, 2017). *"Docker: A Favourite in the DevOps World"*. Open Source For U. [Accessed 12 Jan. 2019].

[11] Balalaie, A., Heydarnoori, A. and Jamshidi, P., 2015, September. Migrating to cloud-native architectures using microservices: an experience report. In *European Conference on Service-Oriented and Cloud Computing* (pp. 201-215). Springer, Cham. [Accessed 12 Jan. 2019].

[12] CN-Infra. https://github.com/ligato/cn-infra. [Accessed 12 Jan. 2019].

[13] VPP-Agent. https://github.com/ligato/vpp-agent. [Accessed 12 Jan. 2019].

[14] Protocol Buffers. https://en.wikipedia.org/wiki/Protocol_Buffers. [Accessed 12 Jan. 2019].