TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D4.31

SDK for SDN

Editor I. Trajkovska (ZHAW)

Contributors I. Trajkovska, D. Baudinot, P. Harsh, Luca Del Vecchio (ZHAW), A. Marcarini (ITALTEL), D. Christofi (PTL), Evangelos Markakis (TEIC), Kimon Karras (FINT), Luca Galluppi (HP)

Version 1.0

Date December 30th, 2015

Distribution PUBLIC (PU)

Executive Summary

This deliverable presents the current results and activities in the Task4.3 dedicated to the activity of developing software development kit - SDK for SDN. Networking in cloud datacentres currently bases on technology and protocols that were not designed for cloud environments at the first place. This has lead to unnecessary overhead and complexity in all phases of a cloud service. For instance, tunnelling protocols generate inherent cascading and encapsulation especially in multi tenant systems. The problem is further increased by the vendor specific configuration requirements and heterogeneous architectures. This complexity leads to systems that are hard to reason about, prone to errors, energy inefficient and difficult to configure and maintain. From a network application developer point of view, this is inefficient since it injects additional overhead and impedes a transparent application development. SDN reduces that complexity by not only unifying and centralizing the network configuration, but also by cutting down the protocol overhead. To address this challenge, Task 4.3 SDK for SDN is primarily focused on creating pack of libraries, which exploit the power of SDN to enable cloud native networking systems in line with the operator's technical and business requirements.

This document is organized as follows: Section 1 introduces the SDN adoption inside the datacentre deployments of Google and Facebook, including the details on their specific implementations with customized network features. Section 2 provides a discussion on the network virtualization trends as a way to demonstrate the SDN capabilities in traditional communication providers' environments such as Telco and Internet service providers (ISPs). OPNFV is described as a potentially leading community to stimulate adoption of software-defined networking (SDN) and network functions virtualisation (NFV). We then describe OpenStack Networking and OpenDaylight as main technological enables used in this task. Section 3 updates the initially defined SDK requirements, described in the T-Nova first year's deliverable D4.01, based on the lessons learnt and the current implementation of the SDK. A comprehensive State of the Art of the actual SDKs for SDN in the industry is presented and their features are compared and contrasted to the SDK developed within this task. Starting from the baseline principle for non-tunnelling multi-tenant support, some currently existing industrial implementations of tenant segregation are included. To confirm furthermore the importance of novel approaches to tenant isolation in datacentre cloud environment, we review the academic results and performance evaluations on similar technologies to ours. In Section 4, the ZHAW on premise SDN testbed is depicted as a basic testing environment for the implementations and the applications developed for the SDK. Section 5 contains the core specification of the SDK along with the architectural diagrams, offering UML representation of the currently developed SDK libraries and description for the API to be released. We focus on the internal components, their functionality and rationale, but we also relate the SDK northbound interface to the main building components of T-Nova such as the Orchestrator, the IVM and the SDN controller.

Section 5 furthermore explains the main idea behind the application driven approach, focusing on Service Function Chaining as crucial use case to further strengthen the SDN argument and validation in in cloud datacentres and generally in the entire T-

Nova eco system. At the end it includes some basic measurements and analysis to confirm the idea behind the implementation and to reason about the possible issues and challenges. Via the discussion we trace the way for the next months' advances within the scope of the SDK. Finally the main goal of Section 6 is to relate the importance of the SDK developed in T-Nova to parallel solutions currently under development (or in consideration) in domains other then a cloud data centres. Being currently hot topics in the industry, those technologies where SDN has already took off are highly relevant for the SDK for SDN developed in T-Nova.

Table of Contents

Index of Figures

INDEX OF TABLES

1. INTRODUCTION

In the current era of virtualisation where there is a focus is on maximising the utilisation of hardware resources by deploying multiple services / applications (ideally with mutually orthogonal workload characterization). Virtualisation in the cloud brings an additional dimension - self-management and programmable access to desired slices of compute/storage/network resources from a shared pool of available resources. While programmable capabilities have been strictly limited in the virtualisation layer (for example: network virtualisation typically means the creation of software isolation capabilities over existing hardware resources) the programmability of the physical network topology has lagged even further behind. Software defined networks (SDNs) provide an approach to support unified programmability of not only the virtual network elements but also the physical network elements supporting the virtualized elements in a cloud environment. In a datacentre environment where the physical links' dimensions are minimal, and the physical environment is strictly controlled, the use of SDN allows the datacentre operator to optimise link capacities by removing or minimising protocol overheads which are unnecessary in this type of controlled environment. Additionally, the focus in a datacentre primarily remains on switching rather than routing and most of the widely deployed communication protocols focus on routing creating opportunities for potential optimizations. The overheads resulting from packet fragmentation and reassembly can be decreased. In addition the implementation of a new protocol in line with the cloud datacentre architecture is also possible. To further strength the arguments already presented, in the virtualized layer, where the typical network requirements are for ensuring process isolation scope for optimization exists. Task 4.3 SDK for SDN is primarily focused on compiling a set of common libraries that would empower a datacentre application developer to easily develop and implement network optimisations in line with the operator's technical and business requirements. Big datacentre operators such as Facebook and Google have being using SDN for a number of years to fine-tune their network deployments to bring them closer to a distributed computing everything philosophy. The following subsections examine those real-world use-cases to further strengthen the SDN argument and validation of the need for deployment and proper exploitation in cloud datacentres.

1.1. Networking in Datacentres

1.1.1. SDN in Google Data Centres

Google has been employing SDN in their datacentres for many years. Their networking infrastructure has long been designed using merchant silicon from various vendors instead of relying on standard commercial of the shelf (COTS) network hardware. Designing their own hardware and software means they are freed from the usual network protocol stacks and can implement their own variations to suit their specific datacentre needs. Their network has evolved from Firehouse DC design to the current Jupiter design [1] that offers 1 Petabits/sec bisection bandwidth

in a Clos topology arrangement of switches that is centrally managed by their own network control software [2]. Their network deployment is logically a big switching fabric that is closer to their distributed software architectures than any other router-centric networking topology. Their cloud SDN networking virtualisation stack is called Andromeda shown in Figure 1-1.

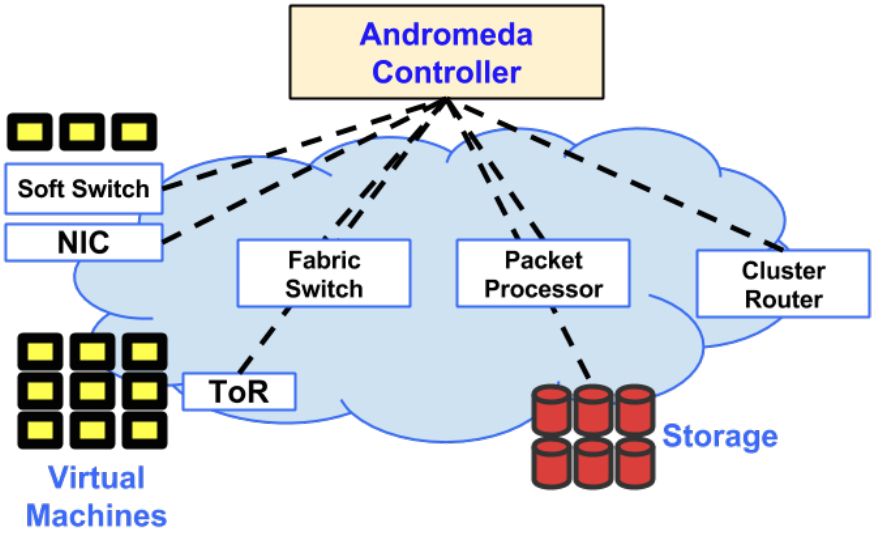


Figure 1-1: Google's SDN networking virtualization stack

Andromeda not only orchestrates the virtual networking needs of the VMs in the Google Compute Engine (GCE), but also orchestrates the top-of-the-rack (TOR) switches, network peering edges, border routers which are part of their physical network fabric. Since they have full software control on all levels of the networking fabric, from low-level hardware to high-level software, they don't have to make compromises and can achieve the full potential of the available network resources. Using Andromeda and a custom Linux port that utilises the full capabilities of their SDN fabric, they are able to improve the TCP throughput substantially (see Figure 1-2).

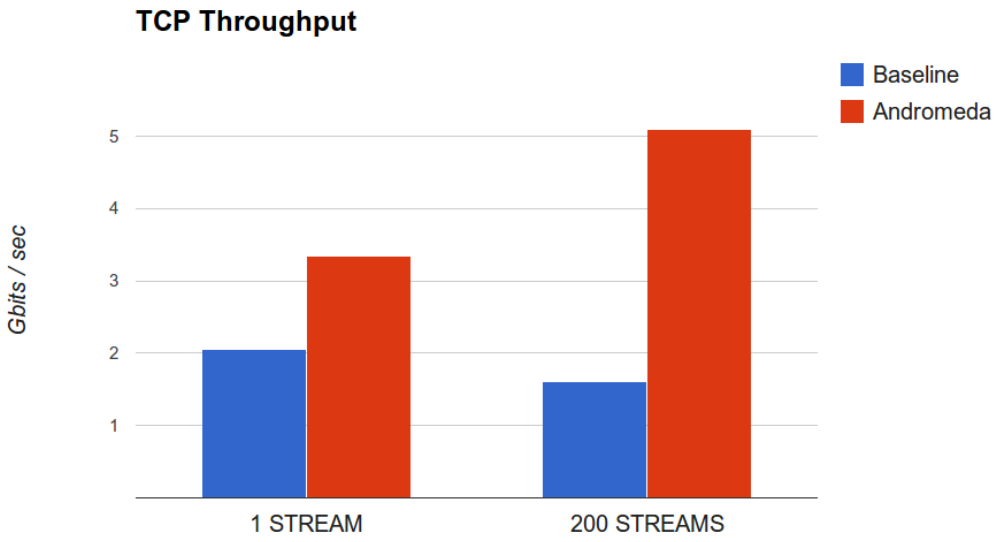


Figure 1-2: Network performance (baseline vs Andromeda)

The Google SDN use case clearly shows the capability of SDN to create a holistic as well as a synergistic effect upon the overall software architectures and the underlying networking fabric in a datacentre environment. Below we provide Facebook's analysis to further strengthen this argument.

1.1.2. SDN in Facebook Data Centres

Facebook has seen a tremendous explosion in their network traffic due to an ever-increasing user base. Their data centres across the globe now support more than 1.5 billion active users' worldwide [3], Facebook's internal traffic called machine-to-machine traffic is orders of magnitude higher than user generated traffic. While designing their network to satisfy ever increasing capacity needs for their distributed applications, Facebook's network team realised that cluster-based designs have a fundamental flaw that limits the cluster sizes to the port densities in the top-of-the rack switches. With ever increasing inter-cluster traffic demands, more ports start to be consumed which reduces the cluster sizes. So the scalable approach Facebook engineers took into account was to divide the data centre network into a set of pods and cores, where each pod consisting of 48 machines is served by 4 top-of-rack switches in (3+1) configuration as shown below in Figure 1-3 and Figure 1-4.

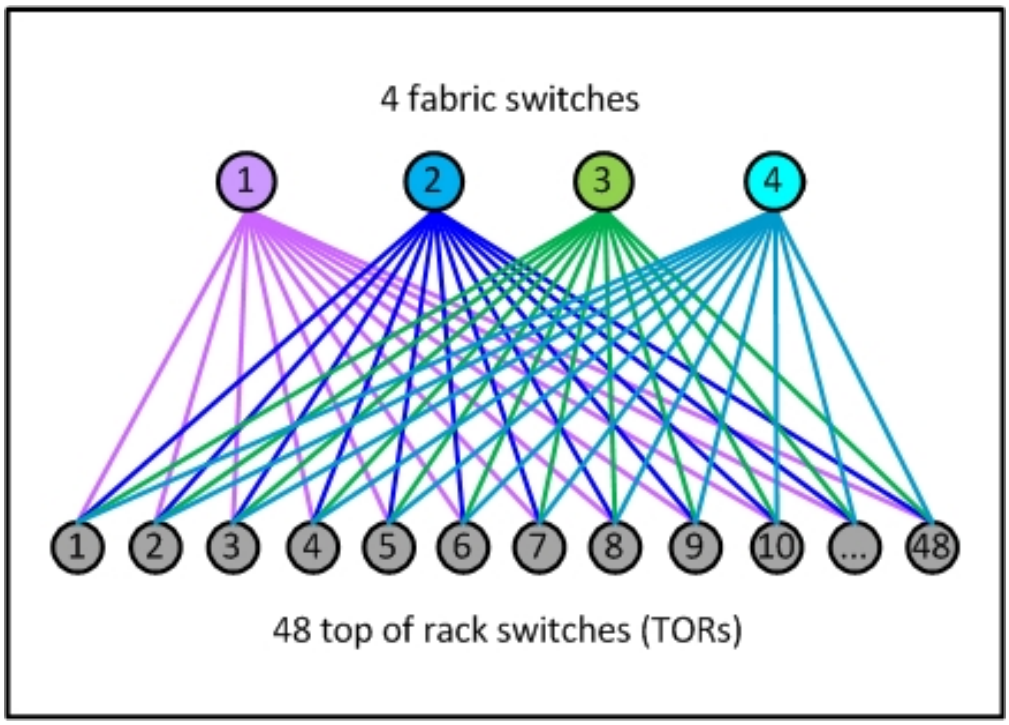


Figure 1-3: Typical Facebook pod

Their data centre fabric treats each networking hardware as a virtual cluster, wherein all their routing needs are served through BGP4 protocol. They manage all the routing rules centrally via a custom built centralised BGP controller. The routing in their datacentre is fully controlled in their custom software. Furthermore only essential protocol elements are implemented to remove any inefficiency from the protocol itself.

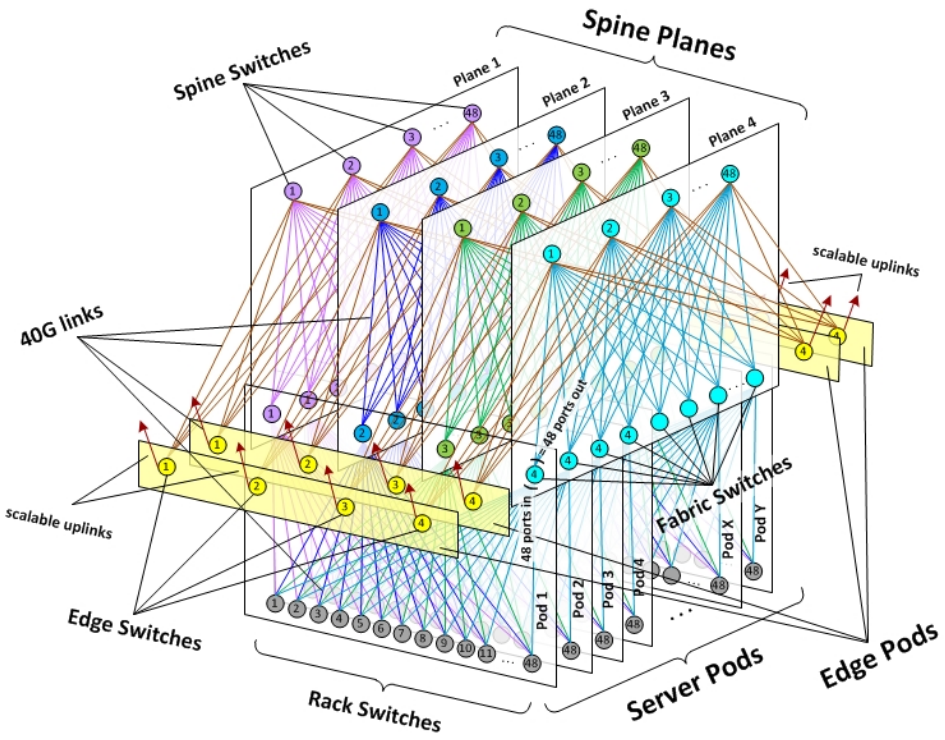


Figure 1-4: Facebook networking fabric design

Management of network hardware elements is also fully automated and is controlled by their software based management service, which automatically identifies a newly added box and the role assigned to it and configures it without any network engineer involvement. With everything controlled in software, the various elements of a virtual cluster can now be placed in any physical location in their datacentre, which makes the manageability simpler. This clearly shows the advantages of SDN approach that can support datacentre flexibility in a simplified manner.

The two SDN use cases by leading service providers show the benefits that SDN can bring in order to achieve efficiency in datacentre management along with traffic optimizations capabilities. The SDK for SDN delivered by this task will provide a set of building blocks for performing fundamental tasks, which can be used by datacentre application developers to build customised solutions to optimise their data centres as per required business and deployed application use cases.

2. TECHNOLOGICAL ENABLERS FOR SDK4SDN

2.1. Network Function Virtualization Trends

Continuing from the previous chapter, let's now look at the potential usefulness of SDN capabilities in traditional communication providers' environments such as Telco and Internet service providers (ISPs). With rising CAPEX+OPEX costs and reducing revenue margins, connectivity service providers (Telcos, ISPs, etc.) have now realised the significance of rapid development and deployment cycles. Coupled with the power of virtualisation that enables better consolidation and possible reductions in cost, the spike in interest surrounding NFV and the increase in standardisation efforts is not a surprise. With the advent of NFV, which drives migration from proprietary hardware boxes to NFs running as software processes in VMs in cloud / datacentre, the importance of SDN becomes more significant. Now virtual machines can be migrated and relocated to different locations depending on the outcome of optimisation algorithm, therefore there is a need for dynamic flow management. Furthermore, SDN northbound interfaces could provide more control to Network Application developers in the form of well-defined abstractions for faster and easier application development. This form of capability will act as a catalyst for network innovations. Optimisation algorithms would converge faster with NFV and SDN deployments as the global network state would no longer be discovered via a distributed slow-converging process, but would be maintained in the SDN controllers. Analysing the needs in typical network application use cases one could clearly identify the need for SDK for SDN in NFV centric R&D strategies. NF virtualisation requirement analysis also suggests the need for some SDK. One type of relocation could be called full virtualisation where the full functionality is virtualised and the function could be moved around on a dynamic basis in response to operational network conditions or SLA's etc. Another approach is partial virtualisation where a part of the functionality is kept in the physical hardware and the remaining functionality is separated and maintained in virtual elements within the datacentre. A prominent example of this approach is Customer Premise Equipment (CPE) virtualisation and relocation, Figure 2-1.

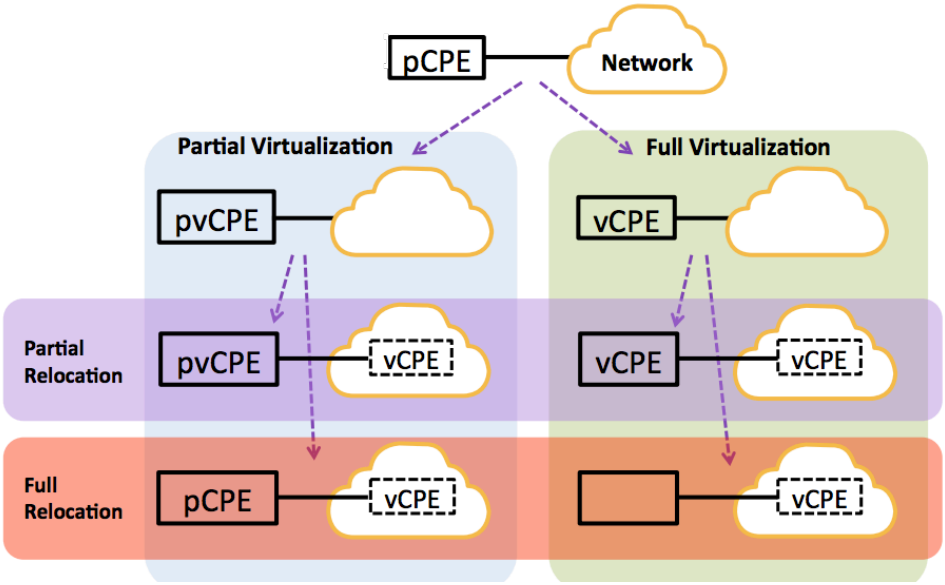


Figure 2-1: Types of function virtualization in a CPE [4]

There are a large number of NFs that could potentially be virtualised into VNFs, including:

- Switch, routers, NAT, etc.
- Firewalls, virus scanners, spam classifiers, etc.
- HSS, MME, eNodeB, NodeB, RNC, etc.
- Rating, charging, billing functions, AAA, etc.

These are just a partial list from the commonly utilised functions in a provider's setup, while potential for virtualisation of many more exists. A suitable SDK will accelerate this process by enabling network developers to design innovative network applications with appropriate interfaces to an SDN controller, and with this providing fine-grained control of the underlying network fabric. Increased SDN activity in this space is a clear indication of a concerted push in this direction. ETSI NVF-ISG, ETSI MEC ISG, Broadband Forum (BBF) and efforts in the open source communities such as OPNFV, OpenStack, Open vSwitch, DPDK, ODP are some healthy indicators for that.

2.2. Open Platform for NFV (OPNFV)

The Linux Foundation created the OPNFV initiative in 2014 right after the creation of the OpenDaylight Project in April 2013 as a leading framework to boost adoption of software-defined networking (SDN) and network functions virtualisation (NFV) [5] [6] [7] [8]. OPNFV is a carrier-grade, integrated, open source platform to accelerate the introduction of new NFV products and services, by essentially bringing together service and NFV providers, cloud and infrastructure vendors, developers' communities, and customers into a new NFV ecosystem. OPNFV was motivated by the European Telecommunications Standards Institute and ETSI NFV to achieve consistency among open standards in terms of performance and interoperability among virtualised network infrastructures. OPNFV promotes an open source network

aimed at accelerating innovation and collaboration between the participating communities based on current technological enablers.

The first milestone of OPNFV is the creation of NFV Infrastructure (NFVI) and Virtualized Infrastructure Management (VIM) using building blocks from upstream projects.

Figure 2-2 shows a diagram of the ETSI NFV architecture marking the initial focus area of the OPNFV group and the two building blocks: NFV and NFVI.

NFVI: Provides access to basic resources—compute, storage and networking—through hypervisors and SDN functions.

VIM: Manages the NVFI and provides the management capability required to deploy applications running in a virtual environment, commonly referred to as VNFs (virtual network functions).

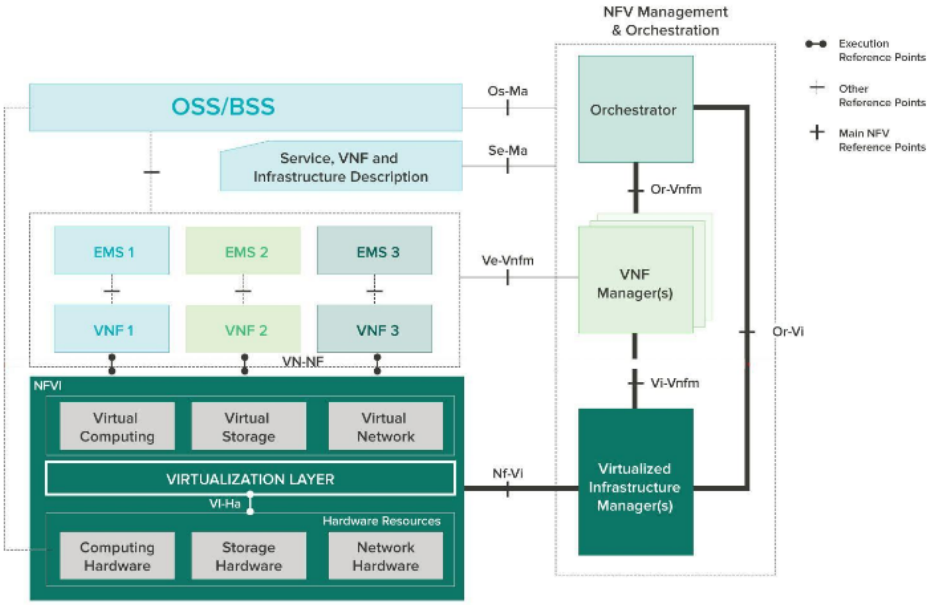


Figure 2-2: OPNFV technical overview [9]

Currently, there are 45approved projects in OPNFV categorized as: Requirements, Integration and Testing, Collaborative Development, and Documentation. Two of the most relevant projects to the activities in Task 4.3 are: (1) **OpenStack Based VNF Forwarding Graph and** (2) **Service Function Chaining**. The first one leverages the OpenStack work on VNFFG (Service Function Chain) and ONF Openflow work on service chaining, in order to achieve automatic set up of end-to-end VNF services through VNFFG so different tenants' flows can be steered through different sequences of VNFs (Service Function). The second project is focused on creating a link between two Linux Foundation projects, OpenDaylight (ODL) and OPNFV. The goal is to provide service chaining capabilities in the OPNFV platform, i.e. an ordered set of abstract service functions (e.g. NAT, load balancing, QoS and firewall) and ordering constraints that must be applied to packets and/or frames and/or flows

selected as a result of classification [10]. More details on OPNFV were described in D2.32.

2.3. OpenStack Networking

OpenStack is an open source (Apache 2.0) IaaS stack with a modular and open architecture. Specifically Neutron, the networking service, abstracts the underlying network via virtual network components and its plugin architecture. This allows very close integration with an SDN controller, such that most of the networking logic is OpenFlow enabled. The exception to this are the iptables firewall rules such as the interfaces and IP namespaces that provide L3 routing for external access of the instances. The rest of the OpenStack networking is implementation agnostic, which enables the full power of OpenFlow within an OpenStack network. More details on OpenStack Neutron and networking can be found in the following references [11].

2.4. OpenDaylight

OpenDaylight (ODL) [12] is an open source framework focused on facilitating an SDN programmability platform for network developers, end-users and customers. ODL supports a variety of networking projects, standards and protocols and has already taken a leading role in the SDN world.

Twelve founding members have actively supported OpenDaylight. It is organised in a modular way consisting of various components. ODL allows the inclusion of north or southbound projects, standards and protocols due to its extensibility. It is based on Apache Karaf – a small OSGi based runtime that provides a lightweight container onto which various components and applications can be deployed. It acts as ecosystem provider for ODL application. Using Karaf one can import different bundles in the runtime controller environment to achieve a specific functionality.

OpenDaylight is an obvious choice as an open source (EPL-1.0) SDN controller. It provides modularity via Apache Karaf. The required features for SDK4SDN include OVS, Neutron and OpenFlow integration. Especially Neutron integration has matured over the last three versions (Hydrogen, Helium and Lithium) in documentation and in stability, which is a prerequisite for a stable and reliable abstraction on top of standard OpenDaylight features.

Figure 2-3 below shows the integral components of the ODL project in the current Lithium release, together will the supported northbound applications and southbound protocols.

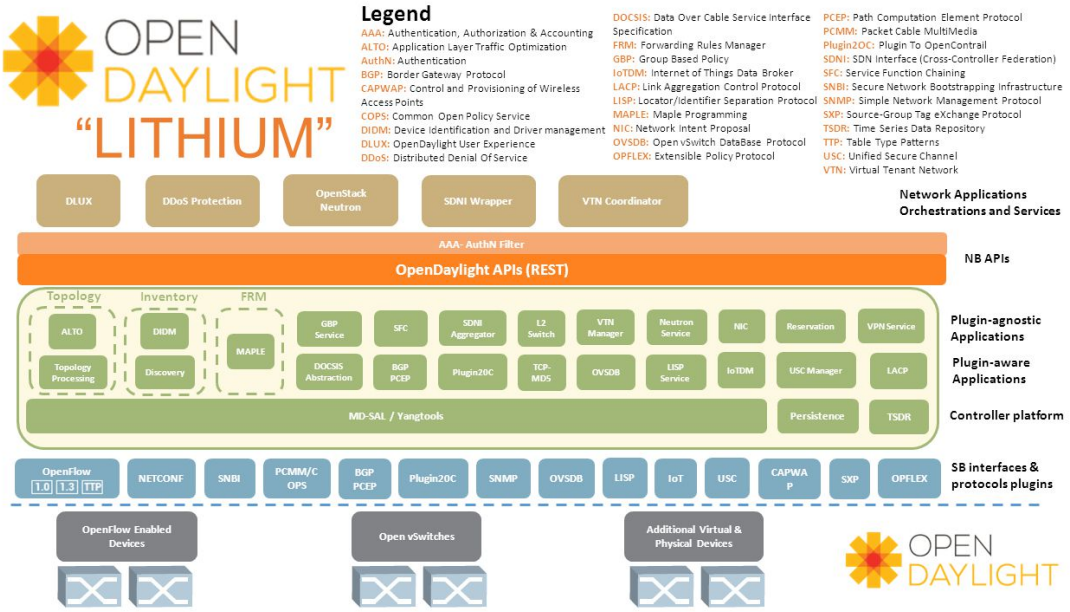


Figure 2-3: Projects and components in OpenDaylight Lithium

3. SDK FOR SDN REQUIREMENTS

The initial requirements for the SDK for SDN within T-Nova were listed in the Deliverable 4.1. This current deliverable describes the SDK progress as it was agreed and formulated after the Rome GA meeting in April 2015.

The major changes in this task concern the formerly described SDK scope "to provide high-level framework on top of existing SDN-controllers." After the amendment changes with respect to Task 4.3, the SDK changed the focus to exclusive support on OpenDaylight as unique entity to control the physical network infrastructure. OpenDaylight has emerged as de facto controller in the past couple of years with continuous engagement from the fast growing community to date. The increased number of leading and parallel ODL projects, along with the support of several SDN standards and protocols, had evolved ODL beyond only being SDN controller to a complete framework for network programmability and NFV support. Comparing the primary depicted modules of the SDK architecture to the currently deployed ODL libraries, we identified many overlaps across the scope of the ODL project and the actual T-Nova SDK. With this respect, a deeper analysis of the ODL development progress and scope has urged this task to reconsider the need for multi-controller API extensions and interfaces to be supported by the SDK. Instead, we can reference the ODL dependencies in the SDK rather than reinventing the technology and therefore omit redeployment of already existing code and tools.

We followed however the generic guidelines for design and developing SDK for SDN as described in Deliverable 4.1. Table 3-1 represents the updated requirements that are vital for the SDK in the current T-Nova vision along with the crucial requirements described by the DoW and the T-Nova consortium.

Requirement Name	Requirement Description	Justification of Requirement	Category
SDK4SDN-OpenDayLight	SDK for SDN MUST support OpenDaylight	Comes from the T-Nova consortium	Functional
SDK4SDN-Testing	SDK for SDN MUST provide testing capabilities	Comes from SDK-general	Functional
SDK4SDN-Diff-Open Flow	SDK for SDN MUST expose Open Flow differences in a safe manner	Comes from DoW	Functional
SDK4SDN-Source-CODE	SDK for SDN MUST be available open source	Comes from DoW	Functional
SDK4SDN-Libraries	SDK for SDN SHALL provide all the necessary	Comes from DoW	Functional

	dependencies and tools in order Developers can Validated its installation		
SDK4SDN-SFC-Support	SDK for SDN MUST provide capability to enable SFC	Comes from SDK-general	Functional
SDK4SDN-Java-API	SDK for SDN MUST provide a Java API	Comes from ODL	Functional
SDK4SDN-Remote-API	SDK for SDN MUST provide a remote API (REST/RPC/CQRS)	Comes from SDK-general	Functional
SDK4SDN-Documentation	SDK for SDN MUST provide sufficient documentation	Comes from SDK-general	Non-functional
SDK4SDN-Path-Connection	SDK for SDN SHALL provide end-to-end connection flow programming capabilities	Comes from SDK-general	Functional
SDK4SDN-Default-Implementation	SDK for SDN SHOULD provide working default implementations for SFC and connection based flow programming	Comes from SDK-general	Functional

Table 3-1: SDK for SDN functional requirements

3.1. Common SDK components

In networking, Software Development Kits (SDK) represents a set of software development tools that allows the creation of networking applications within the scope of certain network operative system, or networking infrastructure (like datacenter deployment). The SDK can be provided as an isolated piece of software (open source component in the SDN community) or as a proprietary component offered by the creators of specific networking technologies or products.

It is very common, that SDK includes an IDE (integrated development environment). The IDE main function is to provide a centralized programming interface and it often contains a terminal, editor, GUI (graphical user interface), debugging tool, and a compiler to create an application out of the code. Most of the SDKs contain a sample code to show the developers a way to use particular programs or libraries from the SDK. There are also SDKs that offer parts of GUIs, like buttons or icons, as well as technical documentation and tutorials. Usually, SDK is offered as free component so that the developers can relay on it for easy application creation for the company.

3.2. SDK for SDN implementations

3.2.1. SDNApp-SDK

Several companies and the community are taking the challenge to write SDK for SDN recently. One of them is OpenDaylight with their "SDNApp-SDK", Figure 3-1.

OpenDaylight describes the pain-points that occur when writing an SDN-Application and how they want to fix them with a Software Development Kit.

The main principles of the SDNApp-SDK are to provide guidelines for SDN-Application development framework. It also helps the developers to reuse and enhance existing code. With the SDNApp-SDK, OpenDaylight wants to make a common platform to validate applications that are written in OpenDaylight. The ODL community wants a framework that enables application management. The components included in the SDK version within the ODL Beryllium release are the following: data modelling, Inter-working with 3rd party tools, database services, application management, keeper services, build, deploy and sample code.

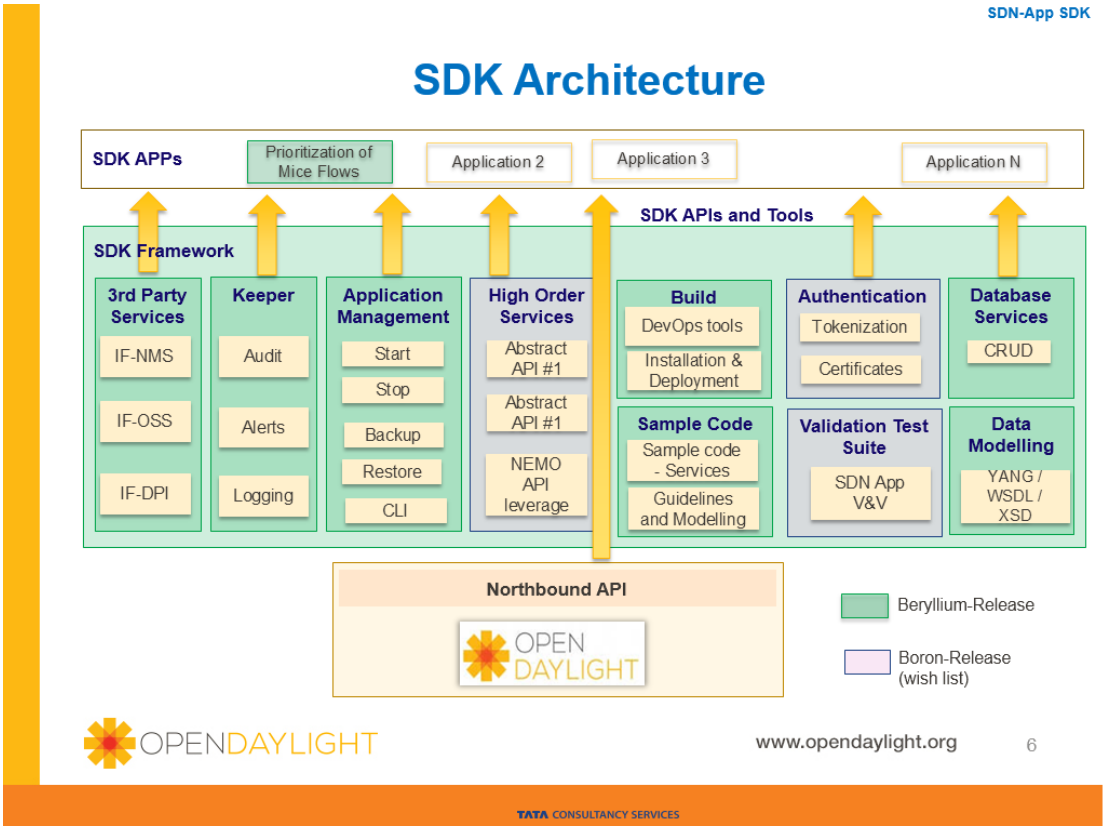


Figure 3-1: SDNApp-SDK architecture [13]

3.2.2. HP's SDK

HP also provides SDN Development Kit, Figure 3-2, with the tree main features that are, "Create", "Test" and "Validate". HP's SDK for SDN helps to simply set up a developer environment and to create applications on the top of their own SDN controller (HP VAN SDN Controller). The SDK component includes: APIs and documentation, programmers' guide, GUI Framework and Sample code.

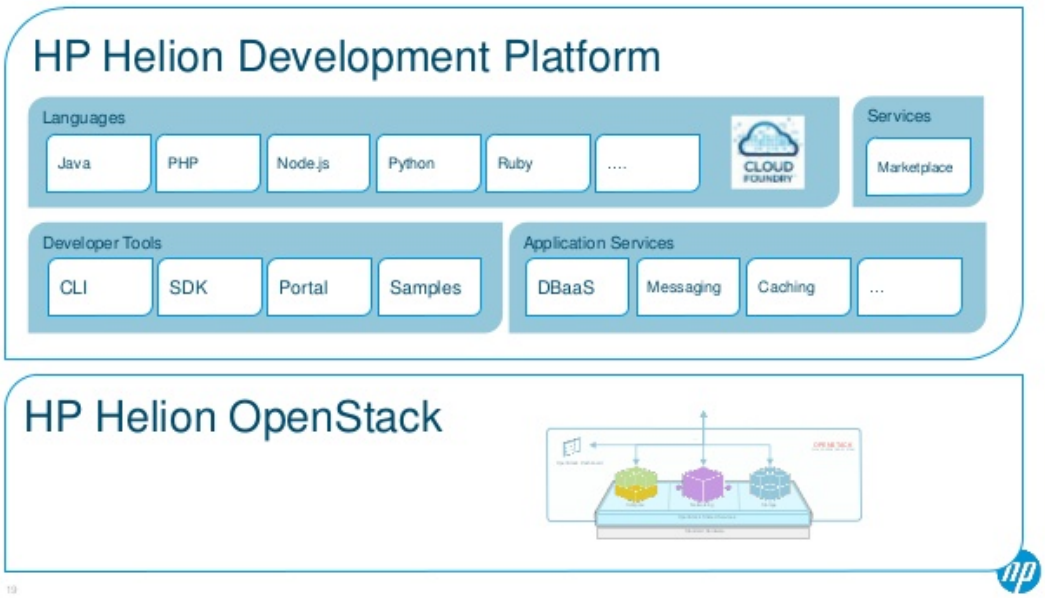


Figure 3-2: SDNApp-SDK architecture

3.2.3. Junos space SDK

Juniper Networks space SDK is an open, network-centric application development toolkit designed to enable developers to use the information embedded in the network to create unique, differentiated applications quickly, easily, and economically. It facilitates the data extraction to be later used in applications and also tends to provide a good end-user experience. Some of the network features include: real-time policy management, energy usage and tracking, custom workflows, network insight for business intelligence, correlation of user subscribed services, and policy and QoS management, as shown in Figure 3-3.

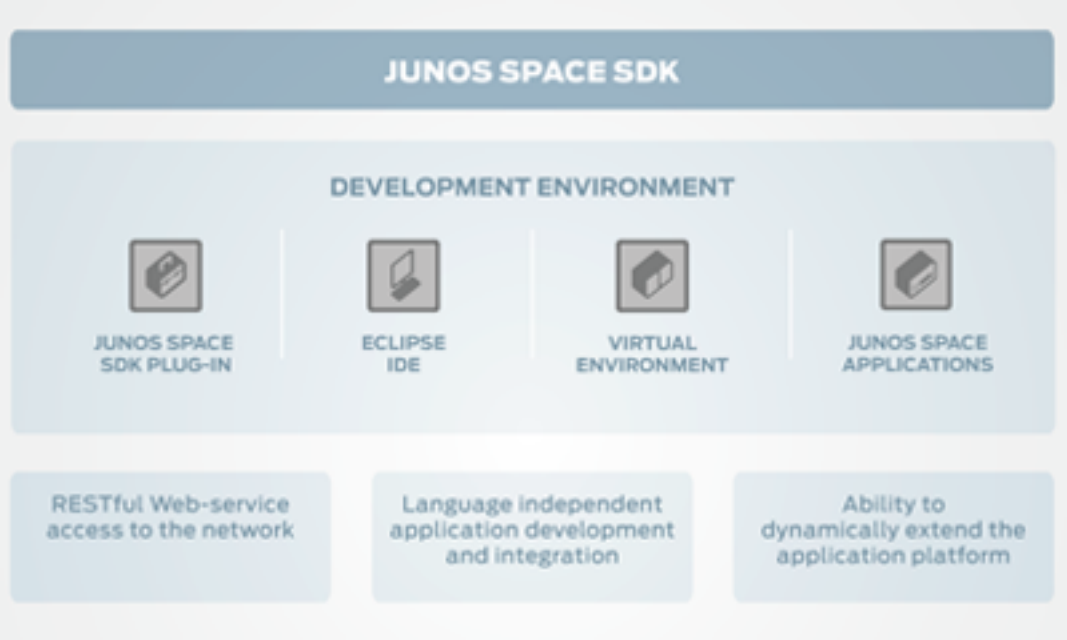


Figure 3-3: Junos Space SDK architecture [14]

3.2.4. Cisco onePK

Cisco's onePK, Figure 3-4, is a toolkit for development, as well as automation and rapid service creation. It is designed for flexibility and can integrate with PyCharm, PyDev, Eclipse, IDLE, NetBeans, and more. OnePk supports common languages like Java, C and Python. It can run on every server or directly on a network. OnePk uses APIs to serve the business needs of the costumers.

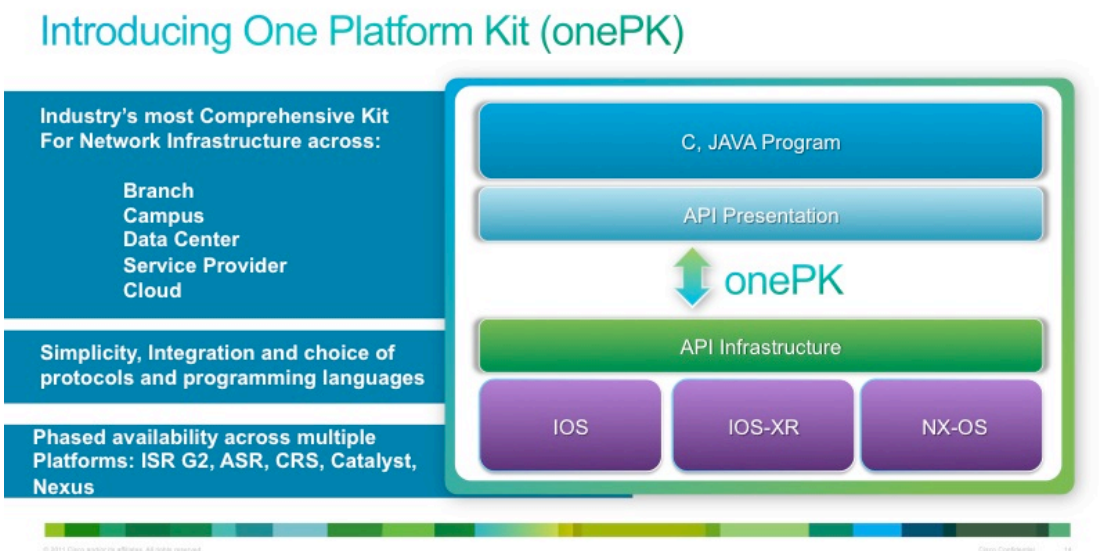


Figure 3-4: onePK architecture [15]

Cisco's onePK can be deployed in three different ways: Process Hosting, Blade Hosting and End-node Hosting. In the Figure 3-5, the three ways are displayed in detail.

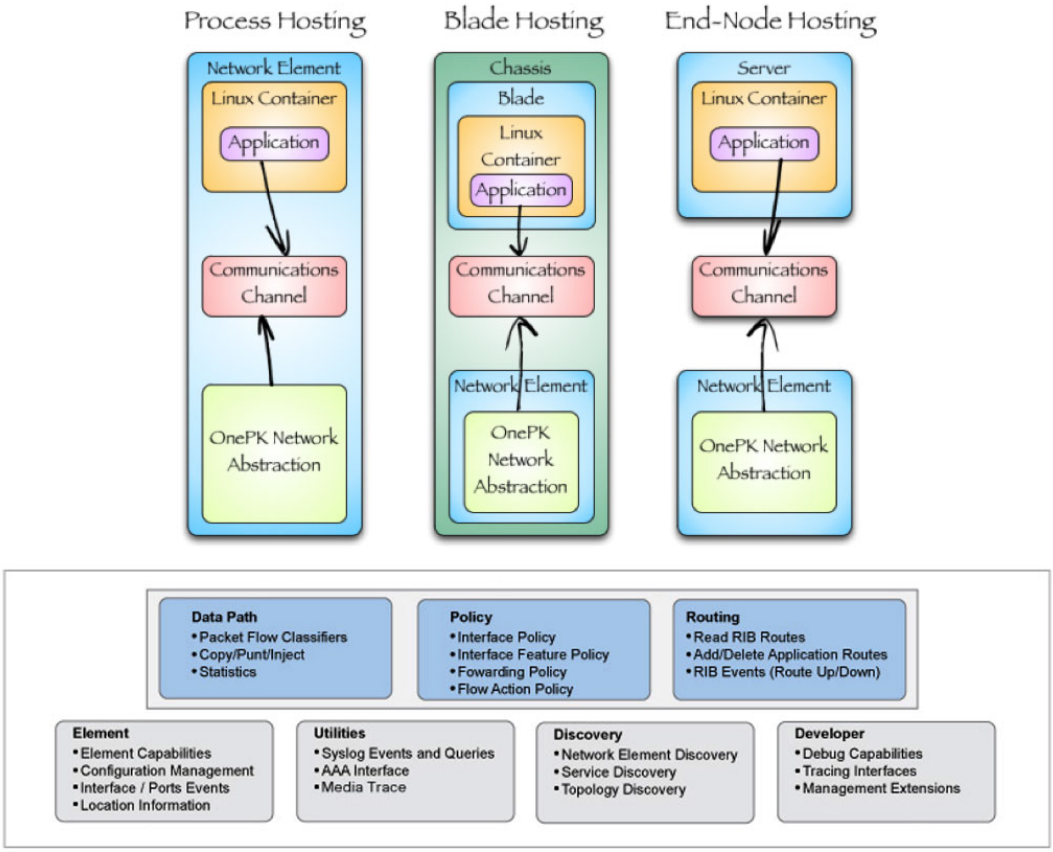


Figure 3-5: Cisco's onePK deployment options [16]

3.2.5. NetIDE

NetIDE is a European project from the FP7 framework [17], aimed to deliver an IDE integrated in Eclipse [18] as a single integrated development environment to support the development life-cycle of network controller programs and SDN environments. Figure 3-6 shows an example NetIDE network topology editor within Eclipse.

NetIDE features four objectives that are described on their website and are also listed here:

- Defines a platform agnostic representation format for network applications.
- Delivers a prototype IDE and associated tools that support the SDN development life cycle.
- Develops a prototype of a run-time environment (NetIDE Network Engine) that supports open & proprietary SDN controllers.
- Promotes the establishment of an Open SDN Model based on an Open Community of developers.

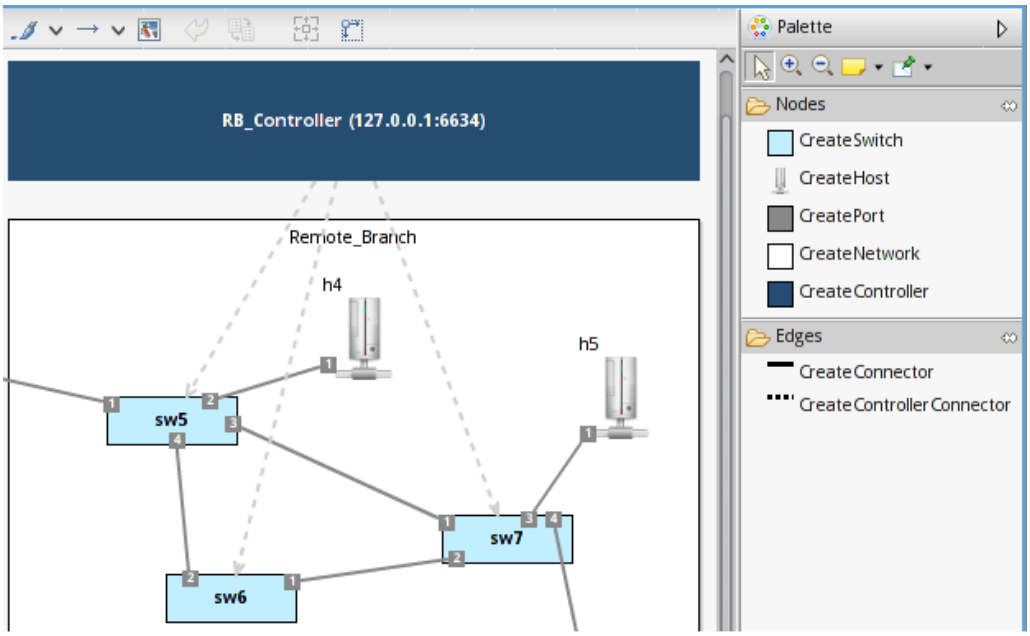


Figure 3-6: NetIDE Network Topology editor [19]

Unlike the netIDE integrated environment, the T-Nova Task 4.3 SDK for SDN is focused on datacentre network and build on the top of the OpenDaylight modular controller. NetID focuses on comprehensive network programming over multiple SDN controllers, whereas the SDK offers libraries for the network programmers who require tools for optimizing network and minimizing inefficiencies due to protocol encapsulation in Open Stack environments. Moreover the SDK developed in T-Nova aims to offer coherent communication with the main T-Nova building blocks such as the VIM and the TeNOR orchestrator via RESTful APIs with the objective to offer a seamless SDN support to the users of the T-Nova services.

3.2.6. PLUMgrid SDK

The PLUMgrid SDK [20] is a Language-based SDK to enable third-party developers and community to build distributed functions on top of IO Visor technology. Figure 3-7 depicts the key components of the PLUMgrid platform that include: PLUMgrid director, virtual domains, IO Visor, APIs, network functions and the SDK.

With the PLUMgrid SDK the software engineers can create their own APIs and network functions easily. The main components from the SDK are object models and libraries as well as compilers and domain specific languages. Using the SDK, network functions can be deployed and/or developed on their own PLUMgrid Platform at run-time and without having to reboot.

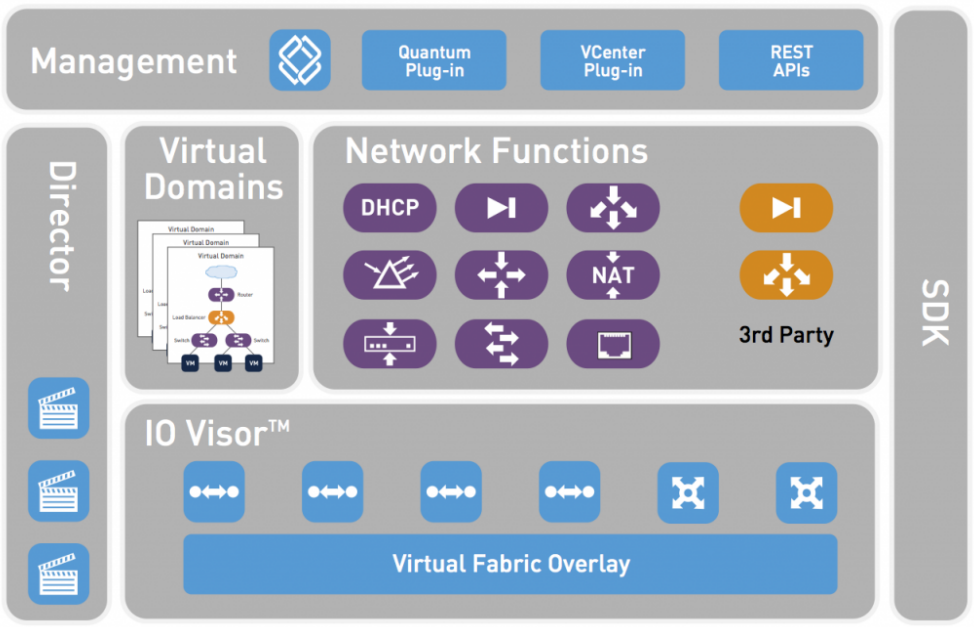


Figure 3-7: PLUMgrid SDK architecture [20]

Table 3-2 summarizes the main features of the described SDKs and the SDK being developed in T-Nova.

SDN SDKs	Supported features						
	REST API	GUI	SAMPLE CODE	DOCUMENTATION	SFC	RESILIENCE	NETWORK TENANT ISOLATION
SDNApp-SDK	x	x	x	?	?	?	x
Junos space SDK	x	x	x	?	?	x	?
Cisco onePK	x	x	x	x	?	x	?
NetIDE	x	x	x	x	?	?	?
HP SDK	?	x	x	x		x	?
T-Nova SDK for SDN	x	x	x	x	x	x	x

Table 3-2: SoTA SDKs main features comparison

3.2.7. SDK for SDN in T-Nova

The T-Nova SDK for SDN differs from the other SDKs mentioned above in the end-to-end based network discovery that is offered per tenant base. The main features are similar for all of the SDKs, however the approach to offer the network abstraction to directly enable isolation - is different. The SDK for SDN includes a network graph library as well as an Open Flow plugin LLDP discovery service, interfaces and their implementations. This is to establish an end-to-end SDN based network discovery per tenant base in Open Stack datacenters. It also contains interfaces to retrieve all the information needed for a current host in OpenStack and also creates and traverses the path between two termination points. SDK for SDN is completely SDN-based and relies exclusively on the Open Flow protocol to create abstractions of the underlying network.

One step towards data centre optimized traffic

The two-year time span from the initial DoW Task 4.3 description brings significant novelty in the SDN and ODL. The T-Nova SDK for SDN is aimed to serve as a tool rather than to be a solution itself. It bases on the emerging concepts in the community, and follows real use cases from providers and enterprises working with SDN, in order to:

- Define a set of SDN enabled libraries for creating novel networking applications inside Open Stack environment.
- Provide a support for Datacenter network control on physical L2 layer.
- Provide alternative SDN solutions to the existing networking protocols for optimized DC traffic.

For instance, tenant segregation in OpenStack hosts is an imperative for security and performance. To provide this, the current OpenStack Neutron ML2 plugin uses tunnelling and tagging techniques such as GRE and VxLAN. The drawback of applying these isolation mechanisms is an increased overhead due to a header encapsulation. Overcoming complexity in large-scale cloud environments of several hundred hosts and tenants is essential in order to enable efficient networking, improved datacentre orchestration, and optimized applications on the top of that infrastructure. The next section elaborates in details, the drawbacks and the challenges of avoiding the current tunnelling mechanisms.

3.3. Non GRE/VxLAN Isolation: Drawbacks and Challenges

In a cloud environment, one of the key aspects is to guarantee isolation between customers also in the networking resources. The adopted approach in network virtualization solutions is to build overlay networks with technologies like VXLAN or GRE on top of the underlay physical network. The overlay networks are by design

isolated customer networks, because they are only logical topologies built on top of a shared underlay physical network.

Even if the popularity and the adoption of the overlay paradigm has grown significantly during the last few years, because it can be implemented on top of a traditional IP-based underlay networks, overlay technics introduce non-negligible drawbacks on performance, in particular when data chunk size exceeds 2902 bytes due to IP fragmentations. There is a lot of academic research that highlight the performance issue as presented in Figure 3-8.

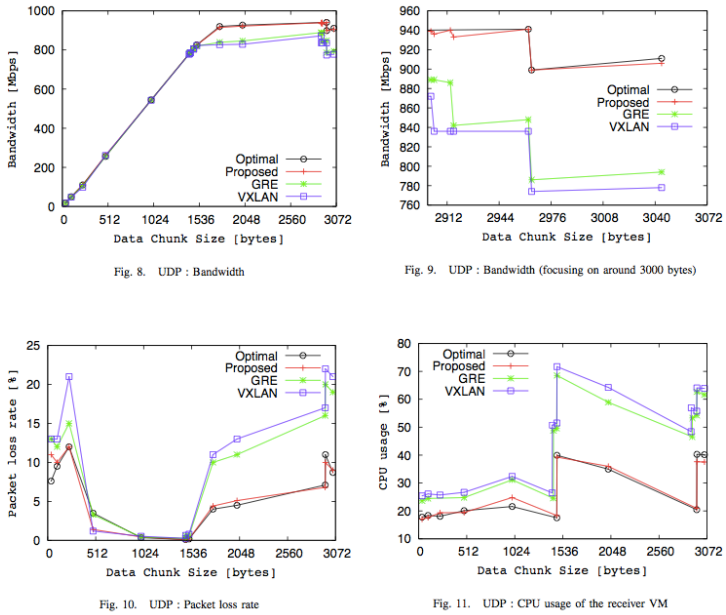


Fig. 8. UDP : Bandwidth

Fig. 9. UDP : Bandwidth (focusing on around 3000 bytes)

Fig. 10. UDP : Packet loss rate

Fig. 11. UDP : CPU usage of the receiver VM

Figure 3-8: Overlay vs Non-Overlay comparison in network virtualization

In a pure OpenFlow SDN architecture, where users can directly access and manipulate the forwarding plane of the network devices, an SDN-based approach has been adopted by academic and also commercial solutions to create isolated networks. Among the academic examples, it's worth mentioning the "Non-Tunneling Edge-Overlay Model using OpenFlow" proposed by researchers from Nagoya Institute of Technology [1]. Among the commercial solutions, we report the Extensible Network Controller (XNC – adopted by Cisco) [2] and FlowVisor (an open source Open-Flow controller) [3]. All of them will be briefly described in the following.

Non-Tunneling Edge-Overlay Model proposes to replace source /destination MAC addresses of the frames transmitted by virtual machines with physical servers addresses. Such substitution is performed by the virtual switch. The modified destination address is then restored at receiver side. The solution ensures address space isolation of each VM, and reduces the number of MAC addresses that the physical switches have to learn. Moreover, it requires less CPU usage compared to GRE/VXLAN tunneling, and facilitates VMs migration.

Cisco XNC is an Open-Flow Controller based on OpenDaylight plus other features and applications like Monitor Manager, Topology-Independent Forwarding (TIF), and

Network Slicing. Topology-Independent Forwarding provides the capability to set up your own path that will be used by data-flow (layer 3 or 4) on the network. Besides setting up the forwarding path, a TIF policy defines some properties, which describe how the traffic will be routed between source and destination. Network Slicing makes available the partition of the network based on physical or logical rules assigned to users. A slice provides an isolated network to users assigned to it. The granularity of a slice could be from a Network device to a specific Flow identified by a source and a destination IP.

FlowVisor is an open source Open Flow controller designed with multitenancy and virtualization paradigms in mind. Into the SDN framework, FlowVisor acts as a controller of controllers; it brings almost a virtualization layer between the (unique) network infrastructure and the (multiple) network controllers intelligence, offering with its proxy services each "slice" a personalized, surrogated, view of the infrastructure, and in turn hiding the infrastructure to the multiple controllers working over it. The high potential of this proxy controller approach is that it breaks usual complexity of networking scenarios, giving tools to work into its simple pieces, where complexity is often just the exception management of individually simpler tasks. Figure 3-9 describes a possible scenario of a FlowVisor implementation where different customers' VMs share a common subnet, and every IP node has a unique IP address. The components are 4 FL (Backend controller), 1 FV (FlowVisor), 1 GW, 1 DHCP server. Different tenants isolation is achieved by means of flow inhibition policies configured into general front-end controller and specific customer back-end controllers:

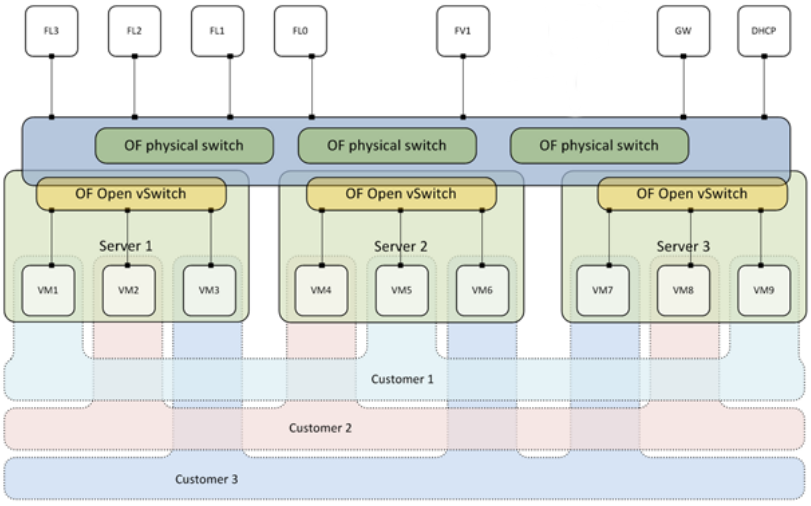


Figure 3-9: Multi-tenant scenario implementation using FlowVisor

Table 3-3 provides a comparison of the different networking approaches with the following key criteria:

- **Isolation** → identify how the solution implements the isolation between tenants

- **Scalability** → identify if the solution can be used in a large system like a cloud environment
- **Performance** → the level of performance of the solution
- **Provisioning** → identify the interface for the provisioning/decommission of new configurations
- **Programmability** → identify the ability to customize network behavior on specific conditions
- **Granularity** → identify the minimum element unit on which the operation could be performed
- **Hardware Constraint** → identify if there is some constraint on the hardware infrastructure

	VLAN	VXLAN/GRE	CISCO XNC	FlowVisor
Isolation	VLAN ID	VLAN ID	Network endpoint independent	Network endpoint independent
Scalability	Limited	Unlimited	Unlimited	Unlimited
Performance	BEST IN CLASS	GOOD Even if encapsulation/ decapsulation Require more compute operation, it can be optimized by relying on hardware offloading [i]techniques, on compliant NICs	GOOD but could be a problem if the flow table grows too much	GOOD but could be a problem if the flow table grows too much
Provisioning	CLI	CLI / REST API	REST API	REST API
Programmability	Not available	Not available	Each flow can be routed on specific path	Each flow can be routed on specific path
Granularity	L2 Segment	L2 Segment	Flow with endpoints source-destination	Flow with endpoints source-destination

			and ports	and ports
Hardware Constraint	No	No, VTEP can be implemented in SW	HW must support a pure SDN with Open Flow support	HW must support a pure SDN with Open Flow support

Table 3-3: Comparison of different networking approaches

3.4. Solutions for Tenant Isolation in Open Stack

3.4.1. Midonet

MidoNet [21] decouples IaaS cloud from network hardware, creating a software abstraction layer between end hosts and physical network. This network abstraction layer allows the cloud operator to move what have traditionally been hardware-based network appliances into a software-based multi-tenant virtual domain.

MidoNet allows users to build isolated networks in software and overlays the existing network hardware infrastructure. All component of Midonet are shown on 3-10.

Tenant isolation: provided by L3 network isolation (see Logical Switching)

Features: Fully virtualized Layer 2 to 4 Networking - MidoNet helps create switches, routers, DHCP, NAT, load balancers and firewalls among other network services.

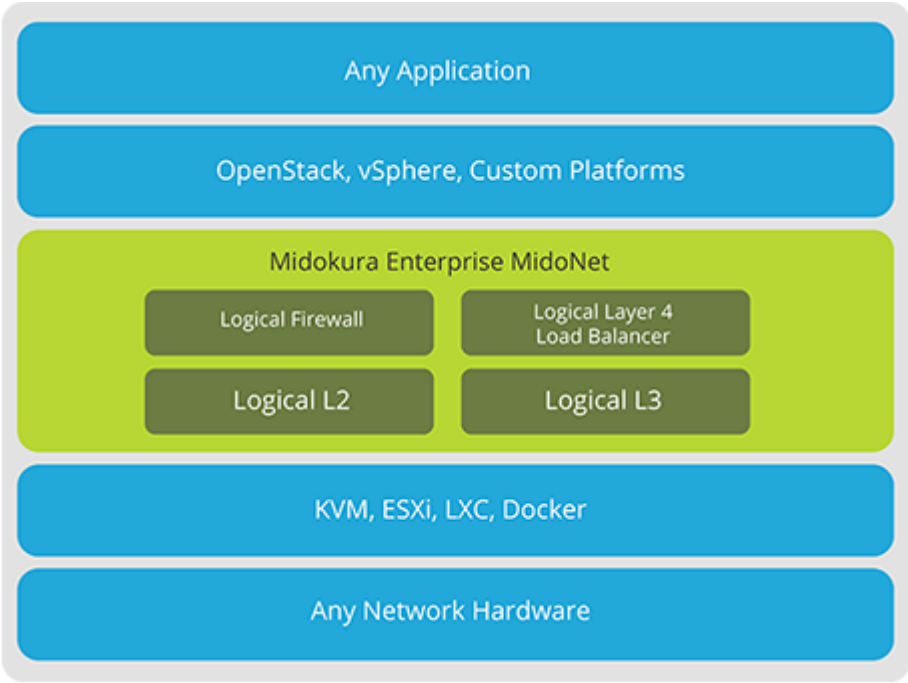


Figure 3-10: MidoNet components [21]

Logical Switching:

- Distributed virtual switching, Layer 2 (Data Link Layer) over Layer 3 (Network Layer), decoupled from the physical network without limitations of convention VLANs.
- Interconnect with VLAN/VxLAN networks (physical and virtual) via software L2 Gateway Logical Routing
- Routing between virtual networks without exiting the software container

Logical Firewall:

- Distributed Firewall that is integrated with the Linux kernel
- Enforces security policies for high packet processing performance

Logical Layer 4 (Transport Layer) Load Balancer:

- Application Load Balancing in software
- Dynamically scale up and down load balancing with compute

MidoNet uses static NAT to implement floating IP addresses in two ways:

- Bring traffic from an external network to a floating IP address for a tenant router
- Perform network address translation from the external network's public IP address to a private IP address and in the reverse direction.

3.4.2. Nuage Networks & Arista

Nuage Networks Virtual Services Platform [22] overlays all existing virtualized and non-virtualized server and network resources, Figure 3-11.

Tenant isolation: logical L3 / L2 networks provide isolation and secure multi-tenancy (see Virtual Routing & Switching).

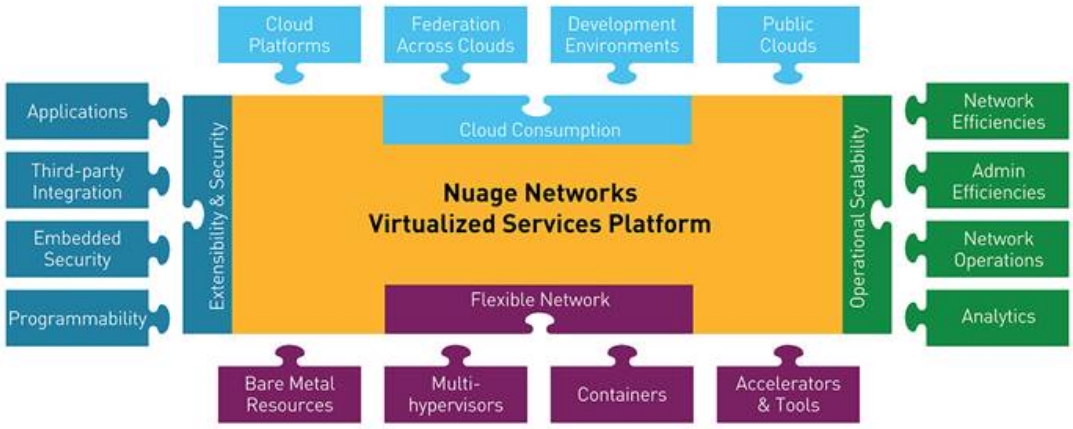


Figure 3-11: Nuage virtualized platform architecture

Cloud Consumption interface:

- Orchestrated by cloud platforms (such as CloudStack and OpenStack)
- Provisioned by customers and administrators via open interfaces (such as OpenStack Horizon)

Flexible Network interface provides:

- Control of virtualized and bare metal resources (such as network equipment from Alcatel-Lucent, Arista, HP, and others) without requiring upgrades
- Multiple server virtualization environments side-by-side, such as KVM, Docker containers and VMware

Extensibility and Security interface features enable:

- Integration with applications such as Oracle, third-party Anything-as-a-Service approaches, security appliances, and operating systems such as Red Hat and Ubuntu Linux
- Controllable network resources through policies and templates either preset by the network team or defined via an intuitive UI
- Customization

Operational Scalability interface delivers:

- Efficient, multitenant operations at cloud scale with features such as multicast and network template capabilities

Features:

Virtual Routing & Switching:

- A module that serves as a virtual endpoint for network services.

Virtualized Services Controller:

- Serves as the robust control plane of the datacenter network, maintaining a full per-tenant view of network and service topologies.

Virtualized Services Directory

- As a policy, business logic and analytics engine for the abstract definition of network services.

3.4.3. Distributed Overlay Virtual Ethernet (DOVE)

DOVE provides tunnelling and virtualization technology that allows creation of network virtualization layers for deploying, controlling, and managing multiple independent and isolated network applications over a shared physical network infrastructure [23].

Tenant isolation: logical components of the DOVE architecture (by IBM) are DOVE controllers and DOVE switches, Figure 3-12. DOVE controllers perform management functions, and one part of the control plane functions across DOVE switches. DOVE switches perform the encapsulation of layer 2 frames into UDP packets using the Virtual Extensible LAN (VxLAN) frame format, and provide virtual interfaces for

virtual machines to plug into, similarly to how physical Ethernet switches provide ports for network interface controller (NIC) connections.

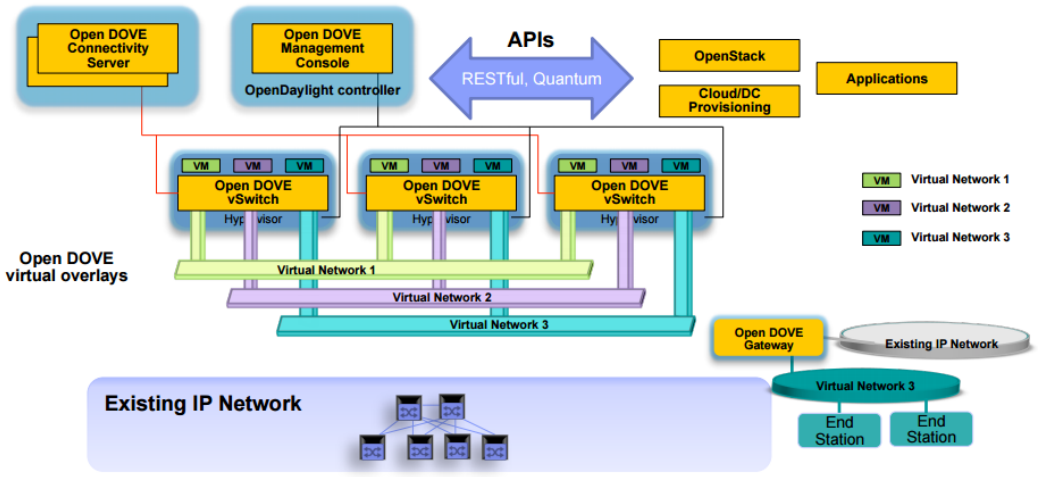


Figure 3-12: Open DOVE architecture

Features:

- No dependency on the underlying physical network and protocols
- Use of the existing IP network infrastructure
- No dependency on the IP multicast traffic

3.4.4. Calico project

Calico's pure L3 approach to data centre networking integrates seamlessly with cloud orchestration systems to enable secure IP communication between virtual machines, containers, or bare metal workloads. Calico provides a pure L3 fabric solution for interconnecting Virtual Machines or Linux Containers ("workloads"). Instead of a vSwitch, Calico employs a vRouter function in each compute node [24]. Figure 3-13 shows an overview of the addressing and connectivity inside Calico.

Tenant isolation: the vRouter leverages the existing L3 forwarding capabilities of the Linux kernel, which are configured by a local agent that programs the L3 Forwarding Information Base with details of IP addresses assigned to the workloads hosted in that compute node. The local agent also programs Access Control Lists in each compute node to enforce whatever security policy may be needed, for example to provide isolation between tenants.

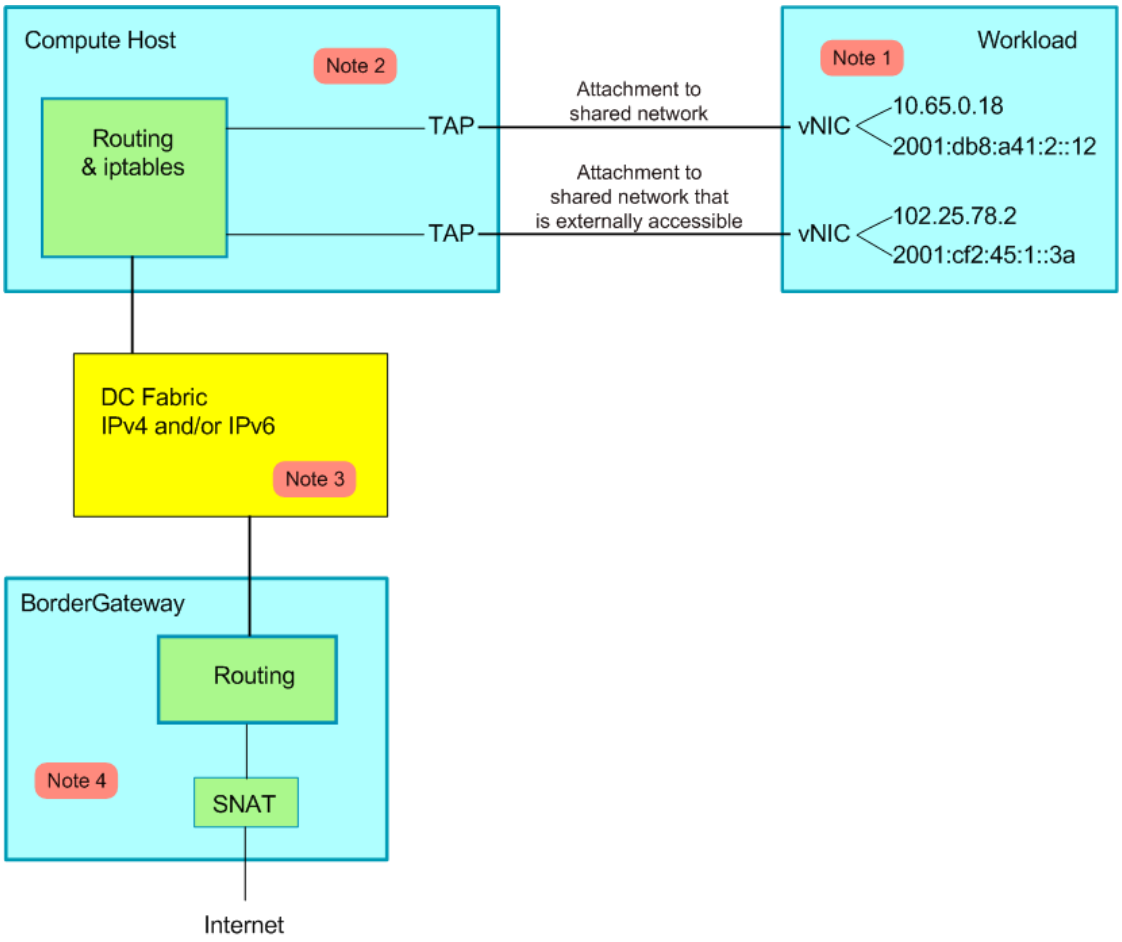


Figure 3-13: Calico addressing and connectivity overview

4. SDN TESTBED ENVIRONMENT

To be able to develop and test the networking applications for the SDK, SDN testbed was setup at the ZHAW premises. This architecture allows integrating the innovations with the fast moving open source community. The testbed was designed upon the following technological enables:

- Centralized OpenDaylight controller
- OpenStack environment
- Physical and virtual networks fully SDN enabled
- OpenDaylight dynamically loads the SDK applications during runtime

The testbed nodes are provisioned with Foreman, whereas the OpenStack environment is installed via Packstack. For more details on the provisioning process please visit Packstack and Foreman node provisioning. The testbed contains five nodes: OpenStack Compute1, Compute2, Control (Networking) all under the release of OpenStack Kilo, SDN Control - OpenDaylight Lithium, and SDN Switch running OpenvSwitch, Figure 4-1.

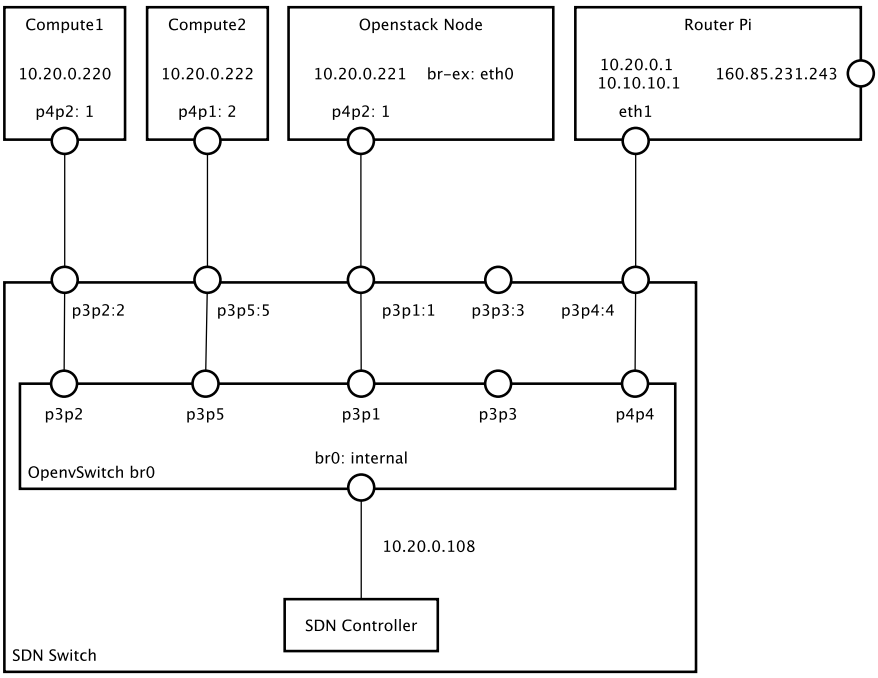


Figure 4-1: ZHAW SDN testbed architecture

4.1. Network

The Compute nodes, the OpenStack Control node and the SDN node are all connected through a network ("physical network" in our case the 10.20.0.0/24 range).

The compute nodes require 1 physical interface for the physical network, later the OVS' of the compute nodes will have a tunnel port that uses the IP attached to this interface.

The OpenStack node needs 2 physical interfaces, one is used in the same manner as the compute node interface, and the other is used by the "external bridge" br-ex, which interfaces between the "virtual network" of OpenStack and the physical network through a neutron router.

4.2. Switch / SDN Controller Node

Basic Provisioning and Configuration

The switch machine has multiple physical network interfaces with Ubuntu OS and OpenvSwitch (OVS) installed. The switch has all physical Interfaces mapped to OVS ports (p3p1...p3p5). We can control the switching of the physical network through these OVS ports. To place the switch inside the physical network 10.20.0.0/24 range, we assigned an IP to the internal port, in this case 10.20.0.108 and associated that IP to the OVS port br0. This IP connects the bridge to the underlying operating system and all OVS bridges in the environment to the SDN controller running in the same machine.

4.3. OpenDaylight node

We installed the Helium release of the OpenDaylight controller. After running it with *./bin/start*, we were able to connect to the ODL Karaf interface by running *./bin/client*. The following features have to be installed in ODL in order to be able to work with Neutron:

feature:install odl-base-all odl-aaa-authn odl-restconf odl-nsf-all odl-adsal-northbound odl-mdsal-apidocs odl-ovsdb-OpenStack odl-ovsdb-northbound odl-dlux-core

4.4. Neutron & ODL integration

For the integration of OpenDaylight with Open Stack Neutron the ODL wiki and the RDO community source was used as a reference point [25], [26].

The general idea is that when ODL is used for the network configuration, it must be the unique entity for the Open vSwitch configuration to avoid conflict with Neutron ML2 plugin. To achieve a successful integration, a clean state is required by ODL that means removing previous OpenStack Neutron configurations on the Open vSwitch within the

compute and the control nodes. This can be simply achieved by turning off Neutron server on the network controller, and Neutron's Open vswitch agents on all hosts. A successful state prompts the ODL controller connected on the newly created br-int of the vSwitch configuration on port 6633 and as OVS as manager of the compute nodes and the OpenStack node connected on port 6640:

```
root@maggie-controller ~]# ovs-vsctl show
a86e813d-b897-4caf-a0a3-38a7f000bef7
    Manager "tcp:10.20.0.108:6640"
        is_connected: true
    Bridge br-int
        Controller "tcp:10.20.0.108:6633"
            is_connected: true
```

The OVS configuration and functionalities provided by the neutron OVS agent are very well documented. Table 4-1 summarizes the OVS specific bridges and flows for each of the Open Stack nodes (Compute and Control/Neutron) in the ZHAW testbed created by ODL as a result of the SDN integration of the Open Stack ML2 plugin. [27]

Node	Bridge	Type	Description
All	All	Flow	LLDP to controller
All	All	Flow	Forwarding
All	br-int	Port	Link to br-tun
Compute	br-int	Port	Link to br-ex
Compute	br-ex	Port	Link to br-int
All	br-tun	Port	Link to br-tun
Network	br-ex	Port	Link to router
Network	br-int	Port	Link to DHCP
Compute	br-int	Port	Link to VM
Compute	br-int	Port Configuration	VLAN ID for Isolation
Compute	br-tun	Flow	Maps VLAN ID to tun. ID
All	br-tun	Flow	Broadcast
All	br-tun	Flow	Pipeline flow

Table 4-1: OVS bridges and interfaces on each Open Stack node

5. SDN4SDK ARCHITECTURE SPECIFICATION

The SDK comprises two main key concepts, one being the network data representation and the second one - the flow control on the top of the specific network abstraction.

Network data abstraction

The T-Nova SDK for SDN retrieves information from different ODL service interfaces in order to be integrated with OpenStack and fully SDN enabled. This data is combined and represented through a network graph, which can be traversed in order to find and establish connections. The data representation also makes it straightforward to reason about the network and its properties because it is a natural model for networks. Being generic enough and modular permits the SDK to accept additional modules in future that could add onto it different information in a contextual fashion.

Flow programming service

While adding use cases in our application driven design it became apparent, that OpenFlow messages are the core output of SDK for SDN. OpenFlow is very powerful but at the same time it doesn't provide context over more than one switch. The point of the abstractions in SDK for SDN is to provide that context and to allow users to program flows on top of multiple SDN switches.

5.1. Components & Interfaces

SDK for SDN is integrated as a Karaf feature into the OpenDaylight environment. It registers itself to ODL features via OSGI and depends mainly on the Neutron API, the OVSDB southbound feature and the OpenFlow plugin.

Some SDK for SDN components such as the Service Chains and the Flow Patterns are maintained by an external API generated by YANG models and stored in the ODL data store.

5.1.1. Dependencies

The SDK depends on several OpenDaylight modules. They are described in the following subsections.

5.1.1.1. OVSDB Southbound

The Network Graphs low-level components are maintained via the OVSDB Southbound plugin. It communicates with the OpenvSwitches that are connected to ODL via the management interface. OVSDB Southbound gets OVSDB notifications

and forwards them to the ODL data store where the SDK listens to node, bridge, port, and interface changes in the topology.

5.1.1.2. Open Flowplugin LLDP discovery

The topology / links between the bridge ports are discovered via OpenFlow plugin LLDP applications. This is necessary to generate complete Network Paths in the graph between connectable host endpoints. Link failure notifications that disrupt Network Paths trigger new path searches, such that the flow connections can be updated.

5.1.1.3. Neutron API

The host ports are registered at creation time on the Network Graph via the Neutron API. This allows for proactive flow programming. SDK for SDN does not push flows reactively via controller messages. It pushes flows as soon as a valid path is discovered between two or more neutron ports.

5.1.2. External Interfaces

Figure 5-1 depicts the SDK components together with the interfaces to the external components, application and the T-Nova VIM.

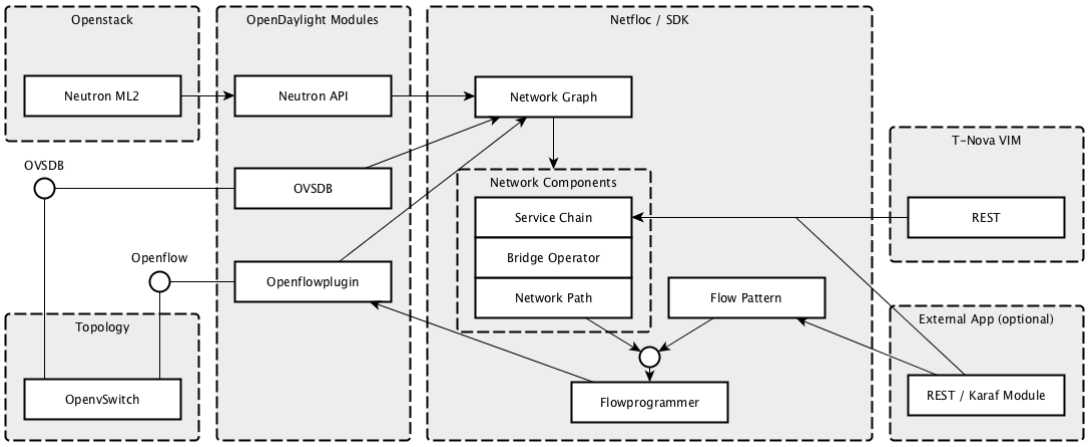


Figure 5-1: SDK for SDN architectural components and external interfaces

5.1.2.1. Flow Patterns

Flow Patterns are template functions that generate Flows. The parameters are specifically Service Chains, Network Paths or just Bridges. Flow Patterns then bind the required network components into their flow template. There are default Flow Pattern implementations for Chains, Paths and Bridges. In addition to that, one can apply REST/Java calls to those if different flows are required. The main reason for the type discrimination between chains paths and bridges is the fact that each higher sequence has a bigger scope to bind parameters. A Flow Bridge Pattern is only aware of the bridge it is applied to. Flow Path Patterns are aware of all the containing bridges (namely endpoint bridges and aggregation bridges) and their properties. Similarly to a path, a Flow Chain Pattern is aware of all the bridges along a chain and can apply any of the properties to any of the bridge flows. However, what makes a

chain semantically different from a path is that a chain contains specific service ingress and egress ports along the network flow.

5.1.2.2. Flow Bridge Pattern Parameters

Flow Bridge Patterns have the scope of a Bridge. Their parameter types are:

- Internal Ports: OVS ports with internal interfaces attached to them. They reach the host system of the OVS.
- Link Ports: OVS ports that are linked to other OVS ports.
- Host Ports (Neutron Ports): retrieved by the Neutron API

5.1.2.3. Flow Path Pattern Parameters

Flow Path Patterns have the scope of a Network Path. Their parameter types consist of a structured set of bridges:

- Endpoint Port / Neutron Port: An OVS port to which one of the respective Hosts is connected.
- Endpoint Bridge: An OVS bridge to which an Endpoint Port is attached.
- Link Port: An OVS port, which is linked to another OVS port.
- Aggregation Bridge: An OVS Bridge linked to another Aggregation Bridge or to an Endpoint Bridge.

5.1.2.4. Flow Chain Pattern Parameters

Flow Chain Patterns have the scope of a Service Chain. Their API parameter type is:

- An ordered sequence of Neutron Port references
- Odd indices of the sequence map to service ingress ports, even indices map to service egress ports

Internally the Flow Chain Pattern detects the paths between the services and uses that as an actual parameter for the template.

5.1.2.5. Service Chain

Service Chains have to be registered via the REST/Java API. A Service Chain composes of a list (in order) of Neutron ports. The Network Graph is aware where the neutron ports are, so a Flow Chain Pattern can be applied to it.

5.1.2.6. Dependencies & Linkages with T-Nova Components

On Figure 5-2, we can see the relation with the other T-Nova components as well as the interfaces the SDK would expose to third party developers. On the northbound side the SDK will expose APIs to the T-Nova IVM (or TeNOR orchestrator) in order to receive information about the VNF placement and service description. The Orchestrator has VNDF (Virtual network function descriptor) to describe the requirements and the specification of each VNF, and NSD (Network Service Descriptor) as a service manifest specification for the service required by the network

developer. This information is later parsed in a metadata file from which a heat template is generated to spawn the required VMs and deploy the VNFs in the infrastructure. From the SDK, RESTful northbound APIs will be exposed to the orchestrator (or VIM) in T-Nova in order to gather the information about the VNFs endpoints and the service description. For example in the case of SFC, the info to be included in the request is (1) the Neutron ports of the VNFs (2) the chain path as a logical subsequent connection between specific neutron ports: *[port_A, sfc_1_in, sfc_1_out, sfc_2_in, sfc_2_out, port_B]*.

One alternative to implement this is in a proactive way where the SDK queries the VIM to determine if the client has specified a request for new service (chain). Second option is to do it in a reactive manner - the information of the new service request is passed to the SDK during VNF deployment process in Open Stack. The integration is still under development and will be decided in the near future.

5.1.3. Internal Components

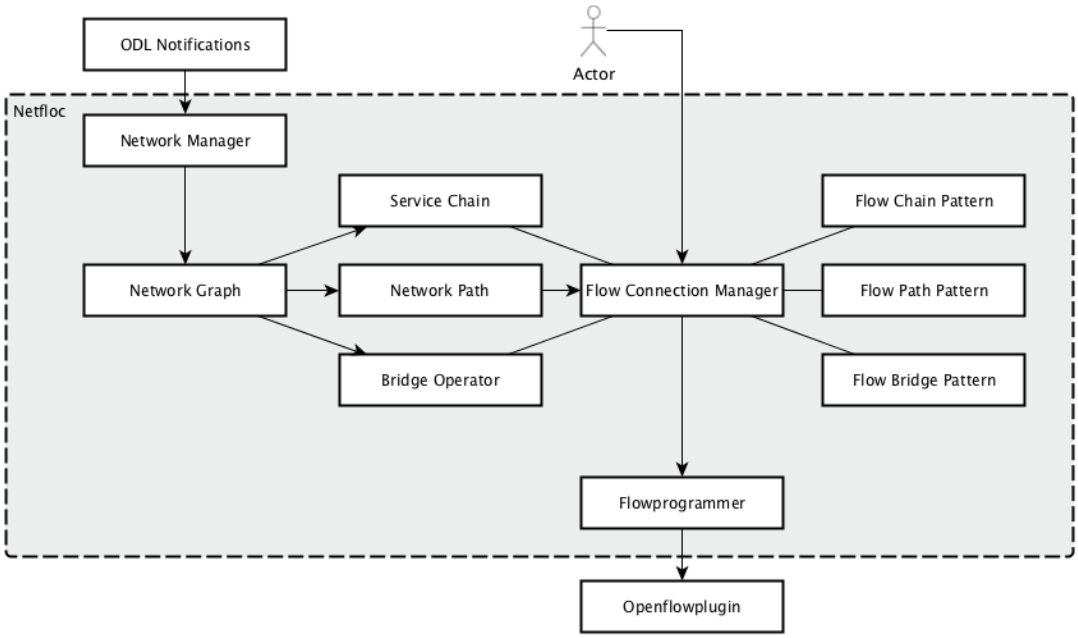


Figure 5-2: SDK for SDN internal components

At its core SDK for SDN is a flow template engine that is aware of logical network components and their relations. These components mostly get created and updated from ODL notifications that are described in the Dependencies section and held in the Network Graph. After the Network Graph discovers these components, they can be further processes by the Flow Patterns to generate OpenFlow messages.

5.1.3.1. Network Manager

The internal architecture is best described via the data flow. The Network Manager registers to the notification services of OpenDaylight mentioned in the External APIs section. It is responsible for state changes in the Network Graph.

5.1.3.2. Network Graph

The Network Graph holds the combined topology information of the OpenDaylight modules and maps them in the following fashion:

- OVSDB -> switch components (bridges, ports,...)
- Neutron API -> host networking information (IP, MAC,...)
- OpenFlow plugin -> topology / links between bridges

5.1.3.3. Flow Connection Manager

The Flow Connection Manager exposes an API to store and apply Flow Patterns and Service Chains. It maintains the relations between Patterns and Network components and pushes flows via the Flow Programmer and handles the state of the flow programming transactions.

5.1.3.4. Logical Network Components & Flow Patterns

Service Chains, Network Paths and Bridges are logical components to which the flows can be assigned/installed. They bind the parameters on Flow Patterns, which are flow template functions specific to a network component type.

5.1.3.5. Flow Programmer

The flow programmer is responsible for merging new flow entries into the ODL data store, which then get processed and executed by the OpenFlow plugin.

5.1.4. API Description

The API specification and implementation is still under development at the point of this writing. It is to be expected that it will undergo breaking changes. The API description gives the direction at which the specification is oriented, Table 5-1.

Name/Resource	**API Description Table**			
	Type	Description	Input	Output
NetworkGraph	Query	Get the graph state	/	NetworkGraph
NetworkPath	Query	Get a path between two HostPorts	Neutron Ports (src & dst)	NetworkPath
ServiceChain	Query	Get resource	ID	ServiceChain
FlowBridgePattern	Query	Get resource	ID	FlowBridgePattern
FlowPathPattern	Query	Get resource	ID	FlowPathPattern
FlowChainPattern	Query	Get resource	ID	FlowChainPattern

Apply Pattern	Command	Apply pattern to network component	Pattern ID & Bridge/Chain ID / Neutron Ports	Status
Standard Pattern	Command	Set the pattern for automatic application	Pattern ID	Status
FlowBridgePattern	Command	Create a new pattern	Pattern	Status
FlowPathPattern	Command	Create a new pattern	Pattern	Status
ServiceChain	Command	Create a new chain	List of Neutron Ports	Status

Table 5-1: SDK API description

5.2. Application Driven Design Approach

The features of the SDK are derived from real application development use cases aimed at optimizing datacenter networks, and for supporting applications networking requirements in the cloud. The SDK will support key T-Nova requirement of service function chaining (SFC) between multiple NFVs that make up a network service. Other SDK libraries & features will be derived from DC use cases geared towards minimizing tunneling overheads, maximizing throughputs, enabling data path redundancies, to name a few.

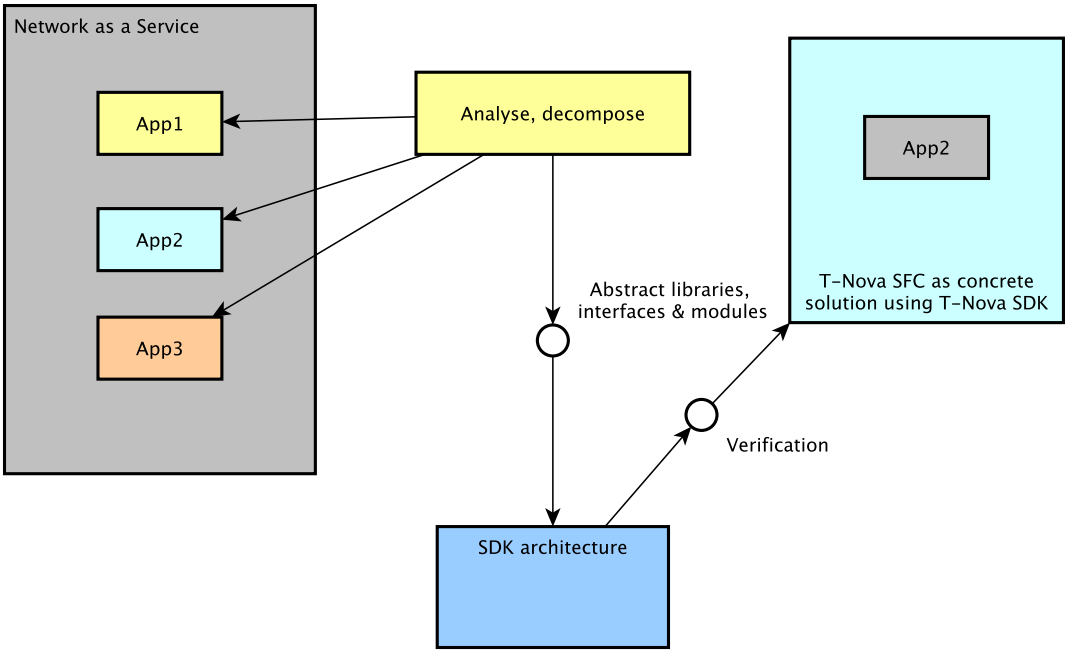


Figure 5-3: A reference development process for the SDK

The SDK design followed a top down approach, Figure 5-3: We analysed the potential issues that exist in the current datacentre networks in order to use the SDN as a tool to test and validate solutions to resolve those issues. Based on the analysis some use cases were established in order to optimize the datacentre networks and facilitate the development of networking applications. For instance, tenant segregation in OpenStack hosts is an imperative for security and performance. To provide this, the current OpenStack Neutron ML2 plugin uses tunnelling and tagging techniques such as GRE and VxLAN. The trade-off applying these isolation mechanisms is an increased throughput due to the bigger header that is created as a side effect from the encapsulation. Overcoming complexity in large-scale cloud environments of several hundred hosts and one tenant is essential in order to enable: efficient networking, improved data centre orchestration, and optimized applications on the top of that infrastructure. The SDK also should support the T-Nova requirements of enabling service function chaining (SFC) between multiple NFVs in order to create services offered by the T-Nova Marketplace. Overall the applications we have based the SDK design on the following:

- **Isolation Application:** Ensure tenant segregation using novel non-GRE/VxLAN tunneling mechanism for optimized packet header
- **Resilience Application:** Provide direct SDN control on a physical level enabling on-demand switch provision and configuration
- **Service Function Chaining:** Gather T-Nova specific SFC requirements and perform traffic classification and steering using the T-Nova deployed NVF

Based on this initial set of applications we derived and defined the required libraries to be supported in the SDK based on the common functionalities from the use case applications.

5.2.1. Isolation

The flow-programming model in SDK for SDN is connection based: Appearing Host Ports are checked against their logical connectivity via the Neutron API. If a connection is valid between two ports then a connection is established through specific forwarding flows. Essentially this is a whitelist-based approach to isolation. The Network Path (Connection) model was derived through analysing what isolation means in a fully SDN enabled network. In classic networking connections have to be filtered through tunnelling and VLANs, because the standard, reactive Ethernet switch is not completely aware of which hosts belong to which IP networks. With a centralized OpenFlow controller and an API to a datacentre networking service we have a global and complete view of the desired connections and can quasi whitelist them via OpenFlow.

5.2.2. Resilience

Normally in a network the scope of each flow is only the bridge it is applied to. But resilience is harder to achieve because dead end flows can only be detected if the scope of a flow includes the complete path of a network connection. Alternatively

forwarding flows can be forgotten (deleted) and reactively replied. Important to note here are the trade-offs between resilience i.e., automatic link failure recovery time (increases with flows deletion frequency) and the network performance (decreases with number of controller messages) [28]. In the SDK for SDN, resilience is achieved by a combination of several components such as: network path and LLDP modules. The Network Path model has the scope of a complete connection in the network. Additionally SDK for SDN depends on OpenDaylight LLDP modules from the OpenFlow plugin to discover and update the network topology. The Network Graph in the SDK discovers events for Link updates and failures on an established Network Path. If such an event occurs, the Network Graph notifies the FlowConnectionManager, which listens to path updates, that a new pattern needs to be applied and that the old one needs to be deleted.

5.2.3. Service Function Chaining

The emergence of the NFV concept and the expansion of VNF solutions have enabled service function chaining (SFC) among virtualized functions as a legitimate use case for a cloud data centre (DC). Although still in a premature state, applying SFC concepts in a fully virtualized environment requires changes and adaptations on the existing protocols in order to be able to apply the same concepts and achieve the desired behaviour on a network level. A typical burden in environments with fully virtualized functions running on virtual endpoints is the aggregated protocol encapsulation in the packets headers that is added as they traverse those endpoints.

Currently there are two key approaches to implement SFC solution in virtualized scenario: packet based and flow based. The first requires manipulation of the packets, for instance by introducing some changes in the header field (packet tagging or rewrites) [29] or simply by applying protocols that introduce one more layer of abstraction on the top of the existing header fields – designed especially for this type of service. Such dedicated protocols has been ultimately introduced by Cisco and leveraged in the Open Daylight (ODL) community to support the ODL SFC integral project. In this case an additional header called NetworkServiceHeader (NSH), is introduced in order to enforce end-to-end traffic as an overlay connection above the service chain path. The problem of such solution is that it alters the datagrams and this can potentially cause a problem in the case where the VNF that runs on some of the virtual machine (VM) hops along the chain, requires the datagrams in their original structure. One example is a virtual function such as vDPI (virtual deep packet inspection) that requires the packets in their original structure in order to enforce a correct behaviour. For further elaboration and in depth analysis of the currently existing SFC solutions among the open source community please refer to T-Nova Deliverable D4.21, Section 4.3 Traffic steering approaches in virtualized network.

In the Demos and Measurement Section we describe the initial experimentations and demo created in order to test the feasibility of the previously described Open Flow based approach. , 5-4 depicts the general idea. In this basic scenario, all the VMs are collocated on the same physical host. Alice's and Bob's VMs are final PoP used in this scenario as the ingress and egress points for the chain. We used *iperf* and *ping* to establish traffic from Bob's VM addressing Alice's as the destination address. We set up flows on the OVS *br-int* in order to steer the traffic through the VNF VM based on

matching the tap interfaces and port form each of the nodes. We previously made a port mapping based on *tcpdump* to identify the mapping between the OVS and the corresponding OpenFlow ports. Some of the flows look like the following:

ovs-ofctl add-flow priority=10,in_port=$bob_of,dl_src=$bob_mac,dl_dst=$alice_mac,actions=output:$vnf_in_of
ovs-ofctl add-flow priority=10,in_port=$vnf_out_of,dl_src=$bob_mac,dl_dst=$alice_mac,actions=output:$alice_of

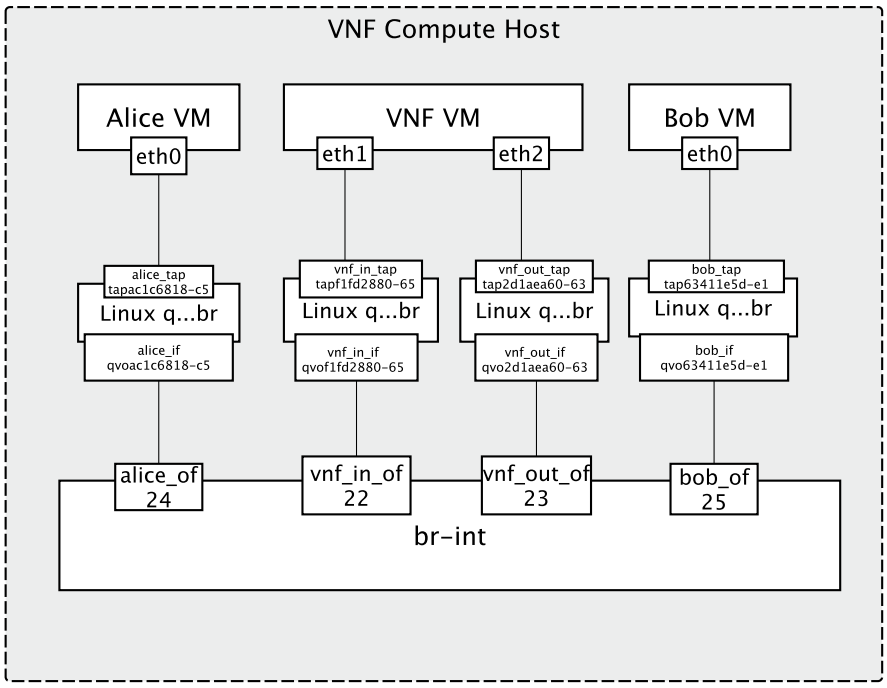


Figure 5-4: Simple SFC scenario: two endpoints and one VNF in same Open Stack node

As the image shows, the flow has the following direction:

Bob (eht0, bob_tap_tapxxx, bob_if_qvoxxx, bob_of) -> VNF (vnf_in_of, vnf_in_if_qvoxxx, vnf_in_tap_tapxxx, eth1, eth2, vnf_out_tap_tapxxx, vnf_out_if_qvoxxx, vnf_out_of) -> Alice (alice_of, alice_if_qvoxxx, alice_tap_tapxxx, eth0)

We defined two network in Neutron *sfc_test1* and *sfc_test2* for the ingress and egress raw ports as the vTC VNF requires and run the VNF with the basic functionality of specifying the packets per second (pps) using the following command:

vtc/PF_RING/userland/examples/pfbridge -a eth1 -b eth2

The VNF reads every incoming packet from interface *a* (assigned to eth1), and then sends it out through interface *b* (assigned to eth2). The VNF *pfbridge* stands as a link patch - the packets that reach the VNF will be read and forwarded the other interface. The output shows the number of packets per second traversing the VNF bridge.

In the first basic scenario we just wanted to prove a feasibility of the flow programming approach over the OVS bridges and identify possible issues and challenges that can come up from such an implementation.

The advantage of SDN is that since it is based on the Open Flow (OF) protocol, the routing can be steered over a specific networking path by programmatically applying OF based rules (flows) on the SDN controller or the virtual switch (OVS) inside the VM hosting the VNF. This is the second approach to implement SFC, based on flow programming rules, that leaves the packets untouched while applying actions on the OF ports of the switches, in order to gear the desired route of the packets in the chain. This routing logic is simplified compared to the first one, as it avoids unnecessary overheads on the top of the already existing ones (ex. in the scenario of inter tenant communication in OpenStack [30]) and the packets are left intact, completely agnostic of the existing chain. A solution based on this approach requires that the network environment is fully SDN capable in order to apply the chain rules along the full virtual network graph. The resulting routing flows need to be maintained to reflect alterations in the function chain (e.g. a VNF altering the packet header could invalid the end-to-end chain).

The objective of this first experimental scenario was to engage as much as possible from the ready-to-use T-Nova VNFs in order to test the functionality on the ZHAW testbed and in the established SFC scenario. With this in mind we wanted to pinpoint potential issues that can interfere with this approach and reiterate on alternative solutions. Clearly the idea of developing SFC support library in the SDK is mainly based on the specific T-Nova requirements that originate from the: VNFs, Orchestrator, IVM and the other components interfacing the SDK. However the optimal solution should be environment agnostic and must eventually work for any SFC regardless of the deployment scope. From the direct he vTC VNF implementation as deep packet inspector nDPI, the vTC permits changes in the header of the packet by ToS manipulation in order to make specific traffic classification. Therefore the previously described approach would not be valid for this advanced functionality of the vTC VNF.

Another potential problem we identified is that the described solution would not work fully in a scenario where two VNFs are running on the same physical host. The reason for this is the non-deterministic path (the OVS of the physical host would not recognize the hop order i.e. which of the two VNFs should the flow go to). Higher level chaining abstractions and programming languages are needed in order to allow service developers to programmatically declare the sequence the VNF traffic should follow, leaving up to the underlying runtime system the actual implementation of such rules [31] [32]. For the chain routing to be deterministic, there has to be a field that keeps track of the chain hops. Using the VLAN ID as a workaround for this purpose could be one possible approach, since the chain routing does not follow the standard Ethernet routing.

In this initial deployment, we run into a common problem in Open Stack Nova – the anti-spoofing rules. Nova uses its own or the Neutron's API for security groups in order to start an instance. Since these rules are applied by default the initial ping between Alice and Bob did not work since the packets will be dropped, as they don't carry the source MAC or IP address that was allocated to the instance. To enable this

we had to initially set the *--no-security-groups* flag (in Open Stack Kilo release). The drawback is that is works only once after the initial configuration. If the ping is resumed the rules are reapplied and therefore the iptables have to be manually manipulated to allow traffic on the 2 chains (in/out) per each of the tap interfaces. We applied a workaround for this that was proposed as a patch (Noop driver) to the nova-compute and neutron that stopped the creation of iptables rules [33].

MAC Rewrite Pattern for SFC

Based on the previous discussion about the potential SFC solutions in T-Nova environment and in order to address the potential obstacles from the discussed solution, we describe here the flow (Open Flow) pattern that uses MAC address rewriting to enable SFC forwarding.

Traffic Classification

The solution we propose hereby is to reserve two fields in the MAC address: one for the chain ID, and the other as a hop counter along the chain. The vTC is responsible for the initial MAC rewriting. After traffic is classified its destination MAC address is rewritten to the following format:

Destination MAC = <ID - Chain Identification>:<N - Number of Chain Hops>:00:00:00:00

The resulting virtual MAC address is matched along the Chain Path to forward the classified traffic to the Nth VNF in the chain:

Destination MAC = <ID>:<N>:00:00:00:00/00:00:FF:FF:FF:FF

Hop Rewrite

On each egress bridge of a specific VNF, there is a rewrite flow that matches the egress port of the VNF and the virtual destination MAC address and decreases the Number of Chain Hops by 1. This way the forwarding is always clearly determined. Forwarding to the next VNF is done via the following matching:

Destination MAC = <ID>:<N - 1>:00:00:00:00/00:00:FF:FF:FF:FF

This happens until N reaches 0.

Endpoint Rewrite

After the last VNF in a chain the packets are resubmitted to the VNF egress bridge (or sent to an according table). Before being able to reach the endpoint host the destination MAC address has to be rewritten to its origin. This is done via the following match:

Destination MAC = <ID>:00:00:00:00:00/00:00:FF:FF:FF:FF

Destination IP = <Destination IP Address of specific Host>

The controller is aware of the MAC to IP mapping via the Neutron API and can recover the original destination MAC address. The packet then gets resubmitted and matches on standard L2 forwarding again.

5.2.4. Rationale and next steps

Our original goal was to maintain the packet structure along the service chain as if it would have normally been forwarded from endpoint to endpoint. This approach was not feasible in the current setup from the following two reasons:

- The VNF deployment (physical location of each VNF) may be in such a way that we would lead to non-deterministic paths (Section 5.2.3).
- The introduction of the Traffic Classifier VNF, because it has to encode the chosen service chain into the datagrams.

This solution is our way to deal with these problems, but it is not in any way set in stone for the SDK at the current stage. Rather it shows one possible way we can implement this using the SDK while having minimal overhead. Other possible solutions could be based on tunnelling (MPLS/GRE) or other rewrite patterns, which could be implemented using the Flow Pattern abstraction of the SDK.

5.3. Implementation

5.3.1. Source Code

The SDK for SDN source code is organized mainly into three parts: (1) the Network Graph components representing the network data, (2) the service listeners and (3) the Flow Components representing the OpenFlow template engine. Also there is some organisational glue and I/O handling around it that can be ignored in this section.

5.3.1.1. OpenDaylight OSGI Service Handlers

These handlers can be categorized in three parts:

- OVSDB Southbound handlers: update the graph based on nodes, bridges, ports updates
- Neutron handlers: update the Host Ports of the graph
- OpenFlow plugin handlers: update the topology links in the graph

5.3.1.2. Graph Components

The following are the components in the Network Graph, Figure 5-5:

- INetworkOperator: graph container
- INodeOperator: represents an OVSDB Node (container for bridges)
- IBridgeOperator: represents an OVDSDB Bridge
- ILinkPort: port which links to another port
- IHostPort: neutron port & OVSDB port attached to nova instance

- IInternalPort: internal OVSDB port

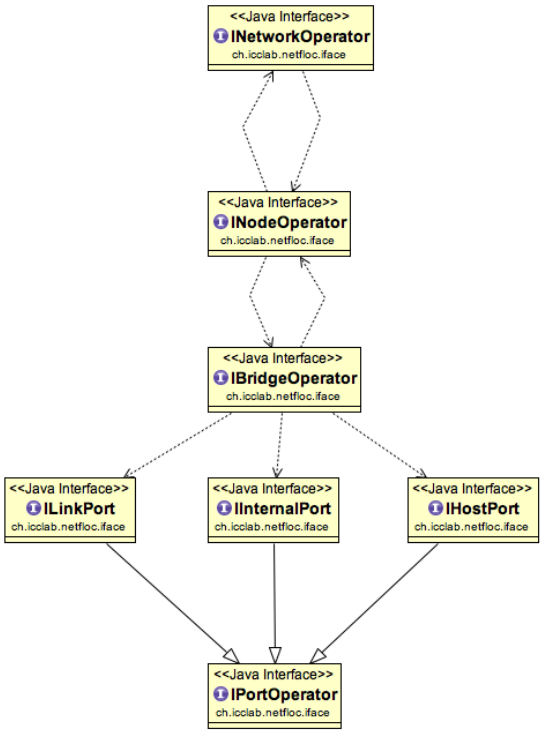
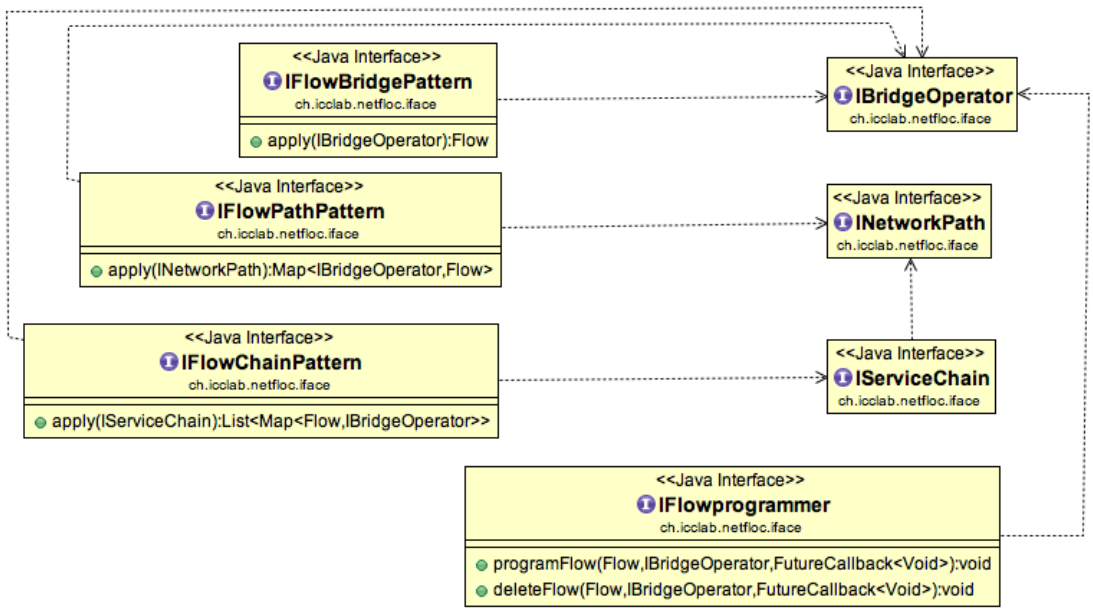


Figure 5-5: UML diagram of the Network Graph components

5.3.1.3. Flow Components

The following UML depicts the flow related components, Figure 5-6:

- IFlowBridgePattern generate Flows from and for single IBridgeOperators
- IFlowPathPattern generate Flows from and for INetworkPaths which are mainly IBridgeOperator sequences with end to end connection context
- IFlowChainPattern generate Flows from and for IServiceChain which are sequences of sequences of IBridgeOperators and chaining context



5-6: UML diagram of the flow related components

5.3.2. Deployment

1. SDN Network deployments: SDK for SDN only supports OVS based, fully SDN-enabled networks.

2. OpenStack: SDK for SDN is built and tested for OpenStack Kilo. The default FlowChainPattern for service chaining was tested by applying the Noop driver for disabling the IP tables (explained in the section above).

3. OpenStack cleanup: An OpenStack installation has to be cleaned up as described the OpenDaylight integration manual.

5. OVS setup: In addition to the OVS clean up and the compute node OVS have to be interfaced to physical NICs.

6. OpenDaylight installation: Use the standard OpenDaylight installation for OpenStack, but without installing the Karaf features. The only feature which has to be installed is "SDK for SDN", which will resolve its dependencies automatically.

5.3.3. Discussion

Metering, QoS and Load balancing

The SDK could support Open Flow meters at the same abstraction level as flow programming. This would allow applications to provide QoS or load balancing. For example a service chain could be balanced on a network level by adding redundant VNFs to the same chain hop for slower VNFs and then divide the load by bandwidth meters. The existing Network Path abstraction fit well into the QoS idea because the overview of the connection already exists in the SDK.

Regarding dynamic VNF injection, currently the workflow of updating a service chain would require deleting the one that needs to be updated and creating a new one. We are considering implementing an update command for service chains when the specification finalizes.

Service Chains are currently described as flat lists, so branching chains is not possible without creating two overlapping chains, which could lead to problems in the network state because overlapping OpenFlow flow mods are not considered unique. An Open Flow instruction is identified by its matching criteria. In the case when a network operator would overlap custom flow patterns for two different Service Chains, then changing one of them in the overlapping section would also change the other. So to enable branching functionality we need to implement a more general data structure for the Service Chain API such as a tree and otherwise disallowing / handling flow overlap so it is clear for the user that changing the state of the Service Chains only influences one chain at a time.

5.4. Demos, Tests & Measurements

This section describes the initial test case scenarios we performed inside Task 4.3 along with the process of software development. The fist demo is related to the service function chaining use case based on the flow programming approach. It uses one of the T-Nova VNFs to make traffic classification and branch the flow in two different chains. The measurements part includes a basic performance characterization of the non-tunnelling traffic between two OpenStack VMs. This is done in terms of comparison between the bandwidth and the latency in case of conventional GRE based isolation and the non-tunnelling based on Open Flow.

5.4.1. SFC demo with T-Nova VNF

We set up SFC tests with integrated T-Nova VNFs to explore possible flow patterns and technical challenges. In our test we established two distinct chains that are separated through the vTC VNF. The flow programming for this SFC example has to be tightly coupled with the output traffic of the vTC because it will manipulate packets based on their classification, which then determines the chain matching pattern via a distinctive property. The distinctive property is this case is the ToS field, so we forwarded the traffic based on the ToS field from the vTC onwards.

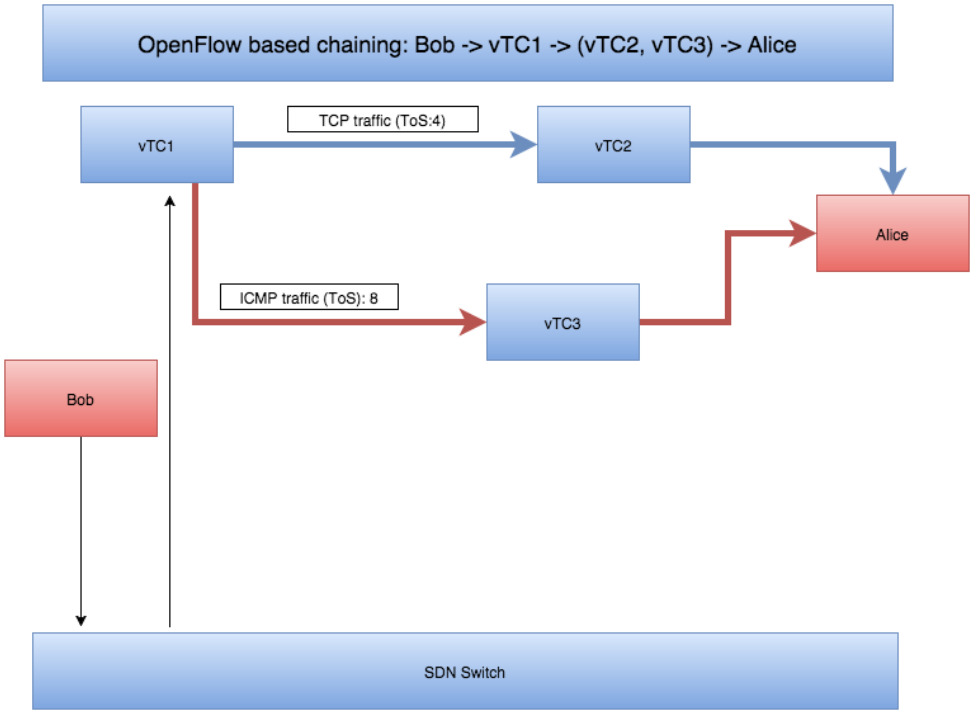


Figure 5-7: Open Flow based chaining using traffic classifier VNF

Figure 5-7 describes the setup of this test. Alice and Bob represent the communication endpoints. To reach the first VNF in the chain: vTC1, the traffic needs to be specifically forwarded to it. The first VNF represents a traffic classifier. Note that this means that SFC forwarding has to be predetermined for an endpoint. Also the traffic flow is unidirectional in this case. The traffic classifier in this test is distinguishing between UDP/ICMP and TCP traffic to determine the chain path, which is then mapped to a ToS field - 8 and 4 respectively.

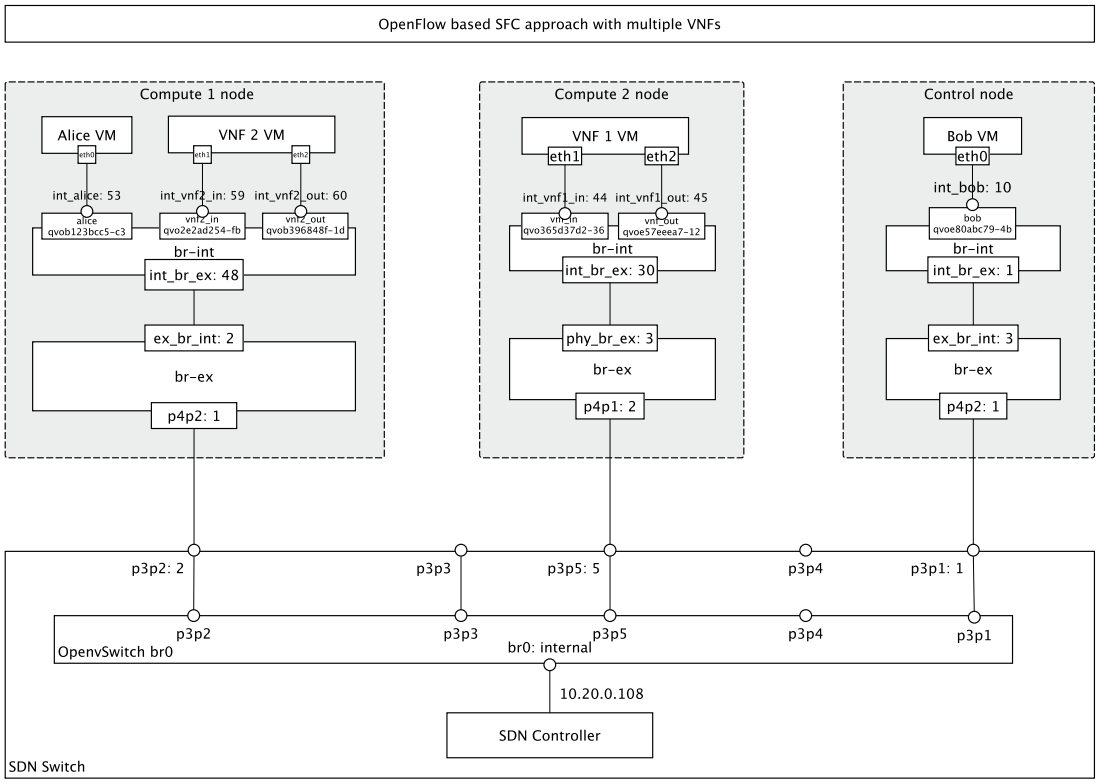


Figure 5-8: Open Flow based SFC using multiple instances of T-Nova vTC VNF

Figure 5-8 depicts the topology of the scenario along with the placement of the VNF VMs and the endpoints inside each of the OpenStack nodes on the testbed. In this scenario we used all the physical nodes of the testbed as hosts of the VNFs and as explained in the SFC section, we applied the Noop driver to disable the firewall rules in each of the nodes. As can be seen on the figure, the Linux Bridge along with the tap interfaces that previously existed inside the hosts has disappeared. This technique stopped Nova from using the hybrid VIF plugging strategy, where it places a Linux Bridge in-line between the instance's tap and br-int. Instead, it plugged the VIF (qvoxxx…) straight into br-int.

This approach has confirmed an evident issue when employing the specific VNF: If the ToS field is the only distinctive property for the forwarding decisions, then some paths cannot be deterministically established. To be able to make forwarding decisions in cases where two chain hops such that "number of hops to VNF B" + 1 < "number of hops to VNF B" while VNF A and VNF B are on the same switch, there has to be an additional information: The core of this problem lies in the fact that a bridge/switch does not have the context of a whole connection path let alone a chain. So the minimal information, which has to be encoded in the packet needs to have two components in reduced form, or in other words, a tuple of two values (a, b) where "a" describes the next hop by uniquely identifying it or by counting the remaining hops and "b" which either maps to topological switching instructions (like in source routing) or maps to a unique chain identification.

5.4.2. Initial measurement scenarios

Here we describe the initial set of network characterization measurements we performed in the case of tenant segregation based on using GRE tunnelling protocol versus the case when we program the OVS and the switch to establish VM connection via direct L2 forwarding using Open Flow.

Minimalizing the networking and protocol overhead for isolated traffic in the datacenter is one of the baseline application use cases for the SDK. We measured the performance differences between tunnelled GRE VM to VM traffic and direct L2 VM to VM traffic on our previously described OpenStack testbed. Following are the two measurement scenarios:

Scenario I: Standard OpenStack + GRE forwarding

The standard GRE network setup is described in the testbed section. In short: The integration bridge forwards the traffic to the tunnel bridge, where VLAN IDs get translated to GRE tunnel IDs (and vice versa) for inter node connections.

Scenario II: Direct L2 forwarding

The alternative is to directly attach the external bridge to a NIC on a compute node. We specifically forwarded the tested traffic though the external bridge and created a single L2 domain via Open Flow. The traffic is still isolated without using VLAN or GRE because we are specifically whitelisting the source and destination hardware addresses.

Latency

The latency was measured via the ping utility. A thousand standard sized packets where sent with a 0.5s interval. Alice was the receiver and Bob was the sender.

We encountered a significant latency difference gain for the direct L2 forwarding, Table 5-2.

Type	GRE	Direct
Min	1.561ms	0.440ms
Max	2.029ms	1.390ms
Avg	1.747ms	1.126ms
Mdev	0.090ms	0.119ms

Table 5-2: Comparative latency values for GRE and direct L2 forwarding

Throughput

The throughput was measured via the iperf utility with standard packet size and five times ten seconds of traffic, where Alice (receiving) was the server and Bob the client (sending).

However the throughput did not change significantly for both TCP and UDP measurements, Table 5-3.

Protocol	GRE	Direct
TCP	773Mbits/sec	769Mbits/sec
UDP	421Mbits/sec	420Mbits/sec

Table 5-3: Comparative throughput values for GRE and direct L2 forwarding

We realized that we didn't take enough parameters into consideration, since throughput and latency are not orthogonal values. To determine the bottleneck(s) we decided to do an additional test run with a more extensive specification.

The additional parameters, which can influence the effective throughput for the future testrune, include the following:

- Variable packet sizes
- TCP Segmentation Offloading (TSO) enabled/disabled
- Generic Segmentation Offloading (GSO) enabled/disable

6. SDN INTER-DOMAIN SOLUTIONS

6.1. SDN for Docker Containers

Currently a large amount of service and software is running on fast and distributed infrastructures that provide the necessary tools to synchronize among a numerous data centres. These apps are containerized meaning that the application is written on the container interface rather than on top of a specific operating system. This allows easier portability. Containers neither care about the underlying networking infrastructure nor have direct connection with the SDN infrastructure. However Docker uses virtual networks to connect containerized applications to the local network, and it connects containers with other containers on the same host. In this respect a number of tools are identified such as Flocker [26] or Rancher [27] where they provide the capability of using virtual overlay networks to connect containers across hosts and over larger networks (such as data centers, wide area networks and the Internet).

Furthermore, Docker acquired SocketPlane [28] as much for the talent as the technology. The latter company chief executive is Madhu Venugopal, a former senior technical leader at Cisco. SocketPlane, founded in October 2014, has focused on making it possible to network that thousands of Docker containers do interact with each other based on the computing task. The containers could host applications on a PC or data centre server. SocketPlane is developing a hybrid networking model that builds on SDN principles and applies them to native Docker environments. It is developing a programmatic platform that puts DevOps in a networking context. In this respect, SocketPlane builds [29] "VXLAN tunnels between hosts to connect Docker containers on the same virtual (logical) network with no remote/external SDN controller needed." Users will interact with a CLI wrapper for Docker that also controls how SocketPlane virtual networks are created, deleted and managed. SocketPlane uses Consul [30] as a lightweight control plane. It connects to the Consul cluster through Open vSwitch for the network connectivity. Once a Docker host is added, the agent runs as a Docker instance and connects into the cluster. The container then looks like a VM.

Additionally, multi-host SDN [34] is now available giving the ability to distributed applications to use multi-container in order to seamlessly communicate across IP networks, while being portable across any network infrastructure. This new ability is providing application portability throughout the application development lifecycle.

Finally the new trend in Docker is the use of the Plugins Mechanism not only for storage but also for networking plugins incorporating in this way a number of project and technologies such as Project Calico [35], Nuage Networks [36], Cisco [37], VMware [38], and Midokura [39]. Docker brings changes that will define a new generation of networking technologies that leverage containers across multiple machines and hosts.

6.2. SDN in Multiple Connected Datacentres

A typical scenario where the SDN controller and a respective SDK solution can find application is between different data centres. An example is OpenContrail – an open source network virtualization platform for the cloud. It is based on MPLS L3VP EVPN (for layer 3 and 2 overlays) and it coexists, as a Neutron plugin in OpenStack, with components such as: (1) vRouter that runs on top of the hypervisor in the host kernel and holds IP tables for each tenant (2) Contrail [31] agent that communicates to the SDN control node, passing BGP and Netconf control specific messages via XMPP. In the forwarding plane, it supports MPLS over GRE/UDP and VXLAN.

The Contrail solution supports function chaining between virtual networks that get connected as a result of a constraint based policy language. Policy rules look like the following:

allow any src-vn -> dst-vn svc-1, svc-2

This rule allows all traffic to flow from virtual network src-vn to virtual network dst-vn and forces the traffic through a service chain that consists of service svc-1 followed by service svc-2. In the previous example, the rule is applied when any virtual machine in virtual network src-vn sends traffic to any virtual machine in virtual network dst-vn. The system is mostly focused on traffic steering i.e injecting the traffic flows into the right virtual machines using a virtual interface. Virtual machines provide network services such as firewall, DPI, IDS, IPS, caching, etc. The system creates additional routing instances for service virtual machines in addition to the routing instances for tenant virtual machines.

Traffic is steered in the following ways:

• Route targets for route paths are modified in order to influence routing tables import/export from a routing instance to another routing instance.

• The next hops, labels, or both are modified as they are leaked from routing instance to routing instance; this is to force traffic to flow through the right sequence of routing instances and the right sequence of corresponding virtual machines.

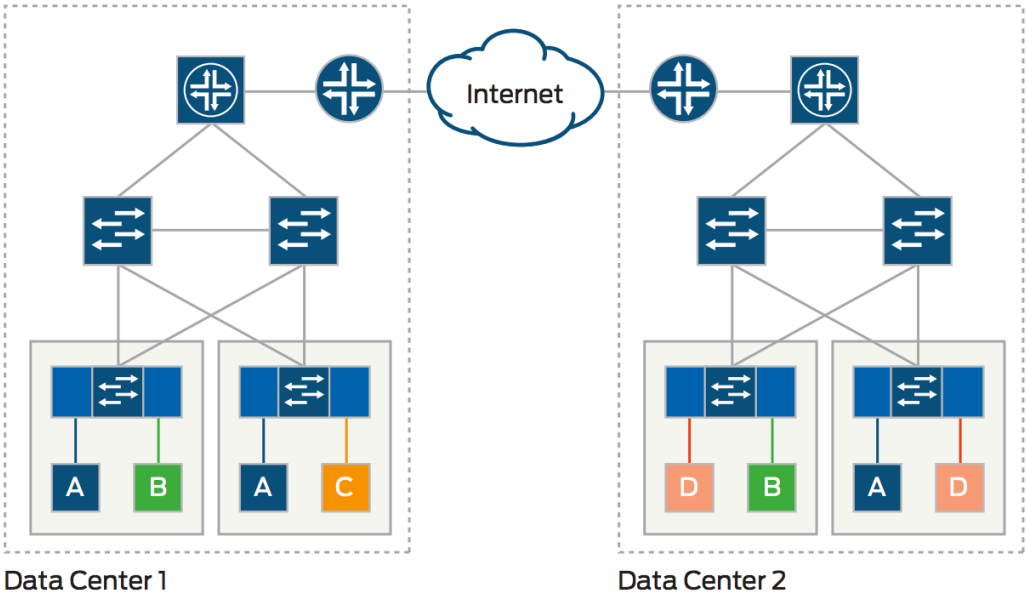


Figure 6-1: Data Centre Interconnect (DCI) using Contrail

Data Center Interconnect (DCI) is a Contrail use case, where multiple data centers are interconnected over a wide area network (WAN), Figure 6-1. Data centers can be in the following states: active/standby for disaster recovery, temporarily active/active for disaster avoidance, permanently active/active. In the active/active case, a tenant might have virtual machines in multiple data centers. DCI puts all the VMs of a given tenant spread over the data centers on the same virtual tenant network.

DCI must address the following network requirements:

- Enable storage replication
- Allow tenant networks to use overlapping IP address spaces across data centers
- Provide global load balancing (GLB)
- Allow VM migration across data centers for disaster avoidance

Multiple transport options are available for DCI interconnections, including dark fiber, SONET/SDH, DWDM, pseudowires, Layer 3 VPNs, EVPNs, etc. Unlike the data center network, bandwidth is a scarce resource in the DCI WAN, so traffic engineering (TE) is often exploited to use available resources efficiently.

6.3. SDN in LTE & Small Datacenters

The role of SDN controller gains more presence and importance within the emerging NFV world. On the other side, the VNFsare the main technological glue between Telco's and Clouds in a novel concept of "cloudifying" LTE functions as related to evolved packet core (EPC) and radio access network (RAN). There exist at a current stage network function deployments (VNFs) and support for components such as BSS, HSS, RAN, etc. [32]. In the LTE (5G) domain the SDN controller plays a fundamental role in interconnecting those virtual functions, becoming an entity to enforce the rules of traffic steering and chaining among a logical network graph, in

order to achieve certain service functionality. Applying the SFC concept inside a Telco scope is still a challenging work in progress that is going to be addressed by several ongoing European projects within the 5GPPP initiative such as SESAME [https://5g-ppp.eu/sesame/]. Adopting SDN in this mixed ecosystem of both cloudified and conventional on premise network functions deployed over bare metal, requires further analysis on the NFV concepts in such a scenario and a careful analysis of the requirements needed to trigger the desired outcome. The role of the controller in this holistic approach has been analysed in the literature and some experimental and industry implementations have already been presented [33] [34] [35] [36] [37]. However in this scenario, SDN based SFC is still an unexplored area, which will be for instance addressed in the scope of the SESAME project.

Within the open source community like OPNFV, it has recently been stated by Myung-Soon Park, Head of Emerging Technology R&D Centre from SK Telecom, that OPNFV plays a fundamental role in the process of adopting and accelerating "innovative Telco solutions, including 5G, by defining tightly integrated and standardized environment". This shows that SK Telecom is highly interested in combining OPNFV with 5G networking. SK Telecom will realize a 5G solution where OPNFV plays an important role in the 5G era. They see the Arno release to be a good candidate towards materializing the specifications.

6.4. SDN for Robotics

Robotics is shaping up to be one of the most rapidly developing areas of modern science and is poised to change the way people go about their daily lives by both taking up and fundamentally transforming existing jobs but also by creating new ones. This raises important social and ethical questions, which are outside of the scope of T-NOVA, but it also raises new challenges in the area of inter-networking of complex system that include such devices.

Given that robotics includes a very wide swath of application potentially ranging from autonomous vehicles to military drones and care-taking robots for the elderly it is only natural that this devices will play varying roles depending on the use case in which they are applies. Robotics devices will also most likely be end nodes and not routers in a network, although the latter can't be completely ruled out, as might be the case in an ad-hoc network. For the remainder of this section we will treat robotics devices as potential end nodes in a network.

One of the factors that make SDN lucrative for robotics is the flexibility it can offer in the face of rapidly changing network topologies. This will almost certainly be the case since most robotics systems are expected to be highly mobile and change environments constantly. Imagine, for instance, an autonomous vehicle that drives into the parking lot of a shopping centre. That vehicle is probably already part of a couple of separate networks (maybe the owner's home network, a general road monitoring and traffic condition update network and the vehicle manufacturer's maintenance network) but also needs to be added to the shopping centres network at that point in time (to receive for example updates on available parking spaces or shopping offers). Performing these tasks manually for a large volume of vehicles (like a busy Saturday in any shopping centre) will be impossible. SDN could help automate

this process by automatically registering the vehicle and setting up the correct network connectivity and access permissions.

This is just one example that attempts to highlight how SDN could be of great help in the area of future robotics. It goes without saying that there's a bunch host of other examples probably too numerous to list here that can similarly benefit immensely from the automated network service configuration that SDN can enable.

With this in mind, an SDK for SDN can provide valuable services in this area. One such service is facilitating debugging and testing of a system, which consists of robotics devices. Since such a device can be anything, ranging from a small drone to a big and expensive autonomous vehicle and taking into account that mobility will play a primary role in such systems, it becomes clear that the prospective developer will not have access neither to physical hardware for all devices nor to a complete test bed of the system. Thus an SDK that provides him with debugging capabilities as well as a simulation & emulation environment for testing of complex scenarios can become a useful component in such case.

6.5. SDN for FPGA Devices

FPGAs form a special class of devices that can be used within a network. This is because an FPGA is essentially a blank slate that can be moulded to whatever its user wants it to be. This means that it's very possible to create an SND-enabled design that can be configured remotely over one of the standard SDN protocols.

This is the premise behind Xilinx's SDNet **Error! Reference source not found.**, which consists of a Domain Specific Language (DSL) that is used to specify the behaviour of a system, which is then generated and programmed on the FPGA by a specialized compiler tool chain. The resulting circuit's functionality can then be modified by programming a flow table dynamically as with every router. In this regard the FPGA has been essentially made into a standard SDN-enabled network device and can be used like any other switch [40].

What makes FPGAs more flexible than a typical, off-the-shelf switch is that it provides the means to perform a two-step configuration. The first step, detailed in the previous paragraph is equivalent to a standard switch. The second step is the complete reconfiguration of the device hardware to fundamentally alter the device's functionality. For instance, the same device could in one moment be an MPLS router whose flow tables are dynamically manipulated via SDN and then it can be reprogrammed to be an SDN-enabled BGP router. An FPGA allows for this dynamically recasting of the hardware in an almost seamless transition from system to system.

Since FPGA-based network devices like the ones that can be created with SDNet are currently not compatible with existing SDN standards, SDK for SDN can bring concrete advantages in wrapping the required SDNet function calls appropriately so providing a unified interface with other SDN-enabled devices. Furthermore SDK for SDN can provide tangible benefits on the debugging and testing front by providing emulation platforms that will allow the developer to test the architectures created

without having the actual hardware present. This can significantly speed up the development phase.

6.6. Kernel-based SDK: IO VISOR project

The IO Visor Project is an open source project and gets support from a wide community of developers. The IO Visor Project enables totally new ways to create network functions. Figure 6-2 shows where the IO Visor Project stands in the Open Networking Ecosystem (ONE).

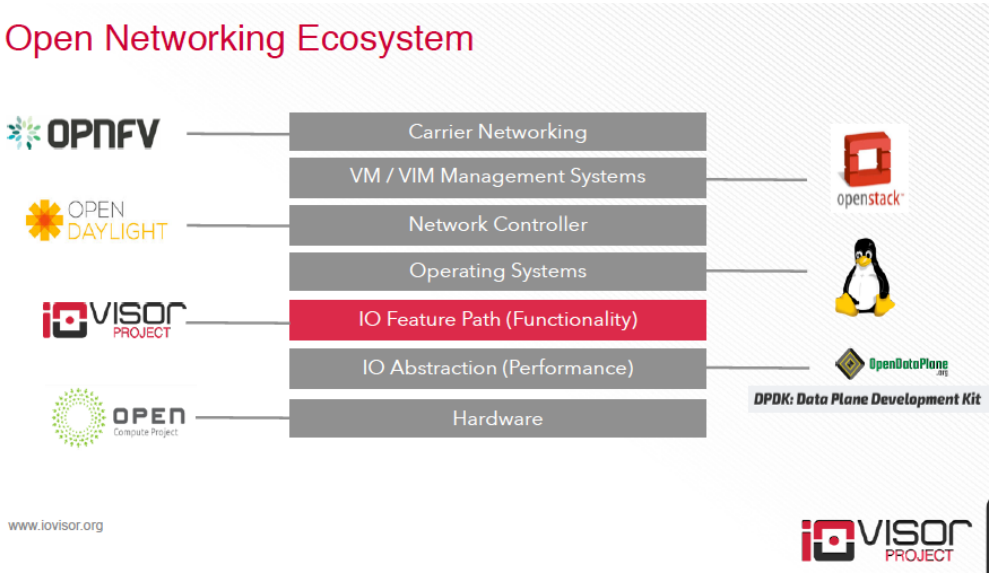


Figure 6-2: IO Visor project placement inside ONE

The main problem IO Visor wants to face is the fact that the Linux kernel isn't virtualized. This is why the kernel has to handle each request after another. That means that the kernel has to be recompiled in order to accommodate a new request. Now what IO Visor wants to achieve is that virtual machines can be added more spontaneously to the kernel space. This has the impact that developers would be able to run multiple VNFs together in the kernel with network function virtualization. Also this would allow creating a complete virtual network that is broad across multiple compute nodes. In this case data-plane processing would all be done by the kernel. IO Visor engine is the keyword, which makes mechanism like this possible. These engines, developer tools and attendant plug-ins are the things that the IO Visor project wants to achieve. If they succeed, multiple tasks could be done in kernel space. Figure 6-3 shows on the left side the networking framework, and an example workflow using the IO Visor SDK, on the right side.

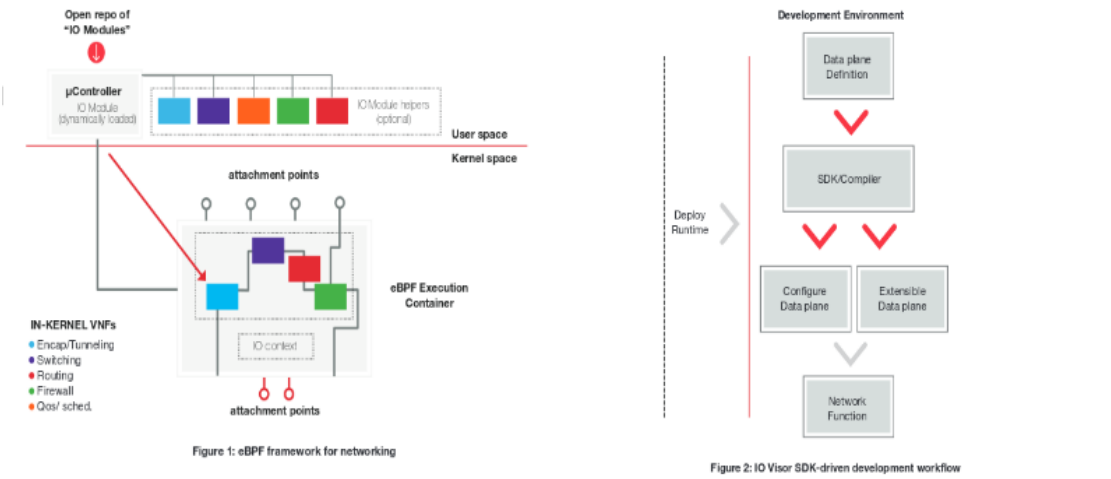


Figure 6-3: Left: The eBPF framework for networking. Right: Workflow from the IO Visor SDK-driven development

Use Cases: The IO Visor technology has been out and in use for some years now and there are some use cases, which are implemented or discussed very actively at the moment. They focus on "Network", "Security" and "Tracing".

Network: In networking, IO Visor Project enables the implementation of advanced networking functions like L3 and NAT in-kernel that are fully distributed across unlimited compute nodes and chained in-kernel to create any network topology which is moving through a virtual or a physical type of deployment.

Security: Micro-segmentation, security groups and full-fledged firewalling are the security functions that IO Visor Project enables to be implemented in-kernel and hence distributed. All this is done with providing the optimal point for traffic within the application.

Tracing: Real-time tracing as well as monitoring applications are widely offered by the IO Visor Project even with the benefit that they are directly built into the kernel.

7. CONCLUSION & NEXT STEPS

In this deliverable we summarized the work related to implementation of the SDK in T-Nova Task 4.3 "SDK for SDN". The architecture of the SDK was described along with the internal components and the interfaces to the Open Daylight, Open Stack, as well as some of the T-Nova components. The development approach was described following the use-case driven design, giving more details on traffic steering approaches using SDN to support a native Service Function Chaining as initial showcase for integration of the SDK within the T-Nova environment. Analysis of existing SDK implementations in SDN was presented and compared to the SDK developed in T-Nova. We presented extensive State of the Art for the hot technologies where SDN takes place in order to point out the high need for a component such as SDK to further support the adoption of SDN in the industry, and the academy as well. There is high focus yet dedicated on the improvement of the SDN controller libraries to enable support of the native networking protocols. This task's focus instead is beyond the work on the actual SDN tool, and rather devoted to the exploitation of the current SDN technology in order to bring a bottom up solution for the companies and the providers to implement an SDN based application in a straightforward manner, aided by the SDK.

Furthermore the task T4.3 has conducted initial test validation of the current approach and is currently dedicated on extensive work to apply complete test scenarios in order to proof and validate the achievements of the SDK compared to the standard networking implementations.

8. REFERENCES

[1] Google Fellow and Technical Lead for Networking Amin Vahdat. (2015, June) Cloud Platform. [Online]. http://googlecloudplatform.blogspot.ch/2015/06/A-Look-Inside-Googles-Data-Center-Networks.html

[2] Google. (2015) [Online]. http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p183.pdf

[3] Facebook. (2015) [Online]. http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

[4] Yaakov Stein. SDN & NFV OpenFlow and ForCES. [Online]. http://ietf.org/edu/tutorials/sdn-nfv-openflow-forces.pdf

[5] OPNFV. [Online]. https://www.opnfv.org/

[6] OPNFV. White paper. [Online]. https://www.opnfv.org/sites/opnfv/files/pages/files/opnfv_whitepaper_092914.pdf

[7] OPNFV. FAQ [Online]. https://www.opnfv.org/news-faq/faq%20-%20n144

[8] sdxcentral. What is OPNFV. [Online]. https://www.sdxcentral.com/resources/nfv/opnfv/

[9] OPNFV. Technical Overview. [Online]. https://www.opnfv.org/software/technical-overview

[10] IETF. Service Function Chaining (SFC) Architecture. [Online]. https://datatracker.ietf.org/doc/rfc7665/

[11] Redhat. [Online]. https://www.rdoproject.org/networking/networking-in-too-much-detail/

[12] OpenDaylight. Lithium. [Online]. https://www.opendaylight.org/lithium

[13] OpenDaylight. Project Proposals:SDnApp-SDK. [Online]. https://wiki.opendaylight.org/view/Project_Proposals:SDNApp-SDK

[14] Juniper. Junos Space SDK. [Online]. http://www.juniper.net/de/de/products-services/network-management/junos-space-sdk/

[15] Cisco. [Online]. http://blogs.cisco.com/wp-content/uploads/blog5.jpg

[16] Sdx central. What is Cisco onePK. [Online]. https://www.sdxcentral.com/resources/cisco/cisco-onepk/

[17] GitHub. [Online]. https://github.com/fp7-netide/IDE

]

[18] eclipse marketplace. NetIDE. [Online]. https://marketplace.eclipse.org/content/netide

[19] OpenDaylight. Project Proposals:NetIDE. [Online]. https://wiki.opendaylight.org/view/Project_Proposals:NetIDE

[20] PLUMgrid. PLUMgrid Technologies. [Online]. http://www.plumgrid.com/technology/plumgrid-platform/

[21] midokura. MidoNet Features. [Online]. http://www.midokura.com/midonet/

[22] nuagenetworks. Nuage Networks collaborates with Arista Networks for Open Networking. [Online]. http://www.nuagenetworks.net/news/nuage-networks-collaborates-with-arista-networks-for-open-networking/

[23] IBM. IBM Distributed Overlay Virtual Ethernet (DOVE) networking. [Online]. http://virtualization.info/en/news/2012/09/ibm-distributed-overlay-virtual-ethernet-dove-networking.html

[24] Calico. Calico. [Online]. http://www.projectcalico.org/

[25] OpenDaylight wiki. (2015, Dec.) OpenStack and OpenDayligh integration. [Online]. https://wiki.opendaylight.org/view/OpenStack_and_OpenDaylight

[26] RDO. (2015, Dec.) RDO networking helium-odl-juno-openstack. [Online]. https://www.rdoproject.org/networking/helium-opendaylight-juno-OpenStack/

[27] RDO. Networking in too much detail. [Online]. https://www.rdoproject.org/networking/networking-in-too-much-detail/

[28] Marcial P Fernandez, "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive," in *27th International Conference on Advanced Information Networking and Applications*, 2013.

[29] OpenDaylight. Service function Chaining in OpenDaylight using NSH protocol. [Online]. https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

[30] Openstack. Overlay encapsulation in DC network. [Online]. http://docs.openstack.org/developer/neutron/devref/openvswitch_agent.html

[31] Theophilus Benson, Nick Feamster, and Dave Levin Bilal Anwer, "Programming slick network functions. In Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15)," New York, NY, USA, 2015.

[32] Riggio et al., ""Programming Wireless Network Functions", in Proc. Of IEEE NOMS 2016," Instabul, Turkey, 2016.

[33] Github. [Online]. https://gist.github.com/djoreilly/db9c2d32a473c6643551

[34] Docker blog. (2015, Dec.) Docker multi-host networking. [Online]. https://blog.docker.com/2015/06/announcing-docker-1-7-multi-host-networking-plugins-and-orchestration-updates/

[35] Project Calico. (2015, Dec.) Project Calico libnetwork. [Online]. http://www.projectcalico.org/calico-docker-1-7-libnetwork/

[36] Nuage. (2015, Dec.) Nuage networks. [Online]. http://www.nuagenetworks.net/libnetwork-is-license-to-hyper-scale-for-docker-and-sdn/

[37] Cisco. (2015, Dec.) Cisco DC networks. [Online]. http://blogs.cisco.com/datacenter/docker-and-the-rise-of-microservices

[38] VMware. (2015, Dec.) VMware Docker network. [Online]. http://blogs.vmware.com/networkvirtualization/2015/06/vmware-docker-networking.html

[39] Midonet. (2015, Dec.) Midonet Docker networking. [Online]. http://blog.midonet.org/docker-networking-midonet/

[40] Xilinx. (2015, Dec.) Xilinx sdnet. [Online]. http://www.xilinx.com/products/design-tools/software-zone/sdnet.html

[41] OPNFV. OPNFV Arno Release. [Online]. https://wiki.opnfv.org/releases/arno

[42] OPNFV. Technical Overview. [Online]. https://www.opnfv.org/software/technical-overview

[43] OPNFV. Pharos Project: Community Test Infrastructure. [Online]. https://wiki.opnfv.org/pharos

[44] OPNFV. OpenStack Based VNF Forwarding Graph. [Online]. https://wiki.opnfv.org/requirements_projects/openstack_based_vnf_forwarding_graph

[45] OPNFV. Project: Service Function Chaining. [Online]. https://wiki.opnfv.org/service_function_chaining

[46] IETF. Service Function Chaining (SFC) Architecture. [Online]. https://datatracker.ietf.org/doc/rfc7665/

[47] PLUMgrid. PLUMgrid Technologies. [Online]. http://www.plumgrid.com/technology/plumgrid-platform/

[48] ClusterHQ. About Flocker. [Online]. https://clusterhq.com/flocker/introduction/

[49] au courant technology. Docker Virtual Networking with Socketplane.io. [Online]. http://aucouranton.com/2015/01/16/docker-virtual-networking-with-socketplane-io/

[50] Rancher. Introducing the First Platform for Building a Private Container Service. [Online]. http://rancher.com/rancher/

[51] socketplane. socketplane. [Online]. http://socketplane.io/

[52] Consul. CONSUL. [Online]. https://www.consul.io/

[53] R., A. Bradai, T. Rasheed, T. Ahmed, K. Slawomir, and J. Schulz-Zander Riggio, "Virtual Network Functions Orchestration in Wireless Networks," in *11th International Conference on Network and Service Management (CNSM)*, Barcelona, 2015.

[54] 11th International Conference on Network and Service Management (CNSM), ""SoftAir: A software defined networking architecture for 5G wireless systems" in Computer Networks, vol. 85, no.C, pp.1-18," 2015.

[55] A. Basta, W. Kellerer, M. Hoffmann, K. Hoffmann, and E.-D. Schmidt, ""A Virtual SDN-Enabled LTE EPC Architecture: A Case Study for S-/P-Gateways Functions," in Future Networks and Services (SDN4FNS)," 2013.

[56] Junaid Qadir, Basharat Ahmad, Kok-Lim Alvin Yau, Ubaid Ullah Aqsa Malik, "QoS in IEEE 802.11-based wireless networks: A contemporary review, Journal of Network and Computer Applications, Volume 55," 2015.

[57] S. Chourasia and K.M. Sivalingam, ""SDN based Evolved Packet Core architecture for efficient user mobility support," in Network Softwarization (NetSoft)," 2015.

[58] Wei Song Jun He, "Smart routing: Fine-grained stall management of video streams in mobile core networks, Computer Networks, Volume 85," Pages 51-62, 2015.

9. List of Acronyms

Acronym	Description
API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
BUM	Broadcast, Unknown unicast and Multicast
CP	Control Plane
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DC	Data Centre
DCN	Distributed Cloud Networking
DOVE	Distributed Overlay Virtual Ethernet
DPI	Deep Packet Inspection
FTP	File Transfer Protocol
FW	Firewall
GPE	Generic Protocol Extension
GRE	Generic Routing Encapsulation
GW	Gateway
HA	High Availability
HPE	Hewlett Packard Enterprise
HTTP	HyperText Transport Protocol
IP	Internet Protocol
ISP	Internet Service Provider
IVM	Infrastructure Virtualisation Management
JVM	Java Virtual Machine
L2	Layer 2
L3	Layer 3
L4	Layer 4
LB	Load Balancer
MAC	Medium Access Control

MD-SAL	Model-Driven Service Abstraction Layer
ML2	Modular Layer 2
MPLS	Multi-Protocol Label Switching
NAP	Network Access Point
NFV	Network Function Virtualisation
NFVO	NFV Orchestrator
NIC	Network Interface Card
NSH	Network Service Header
NVGRE	Network Virtualisation using Generic Routing Encapsulation
ODL	OpenDaylight
OF	Open Flow
OSGi	Open Services Gateway initiative
OVS	Open vSwitch
OVSDB	Open vSwitch Database
POP	Point Of Presence
QOS	Quality of Service
REST	Representational State Transfer
SDK	Software Development Kit
SDN	Software Defined Networking
SF	Service Function
SFC	Service Function Chaining
SFF	Service Function Forwarder
SFP	Service Function Path
TTL	Time-To-Live
UDP	User Datagram Protocol
UUID	Universal Unique Identifier
VIM	Virtual Infrastructure Manager
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNFAAS	VNF As A Service
VNFFG	Virtual Network Function Forwarding Graph
VPN	Virtual Private Network

VRS	Virtualised Routing & Switching
VSC	Virtualised Services Controller
VSD	Virtualised Services Directory
VSP	Virtualised Services Platform
VTEP	VXLAN Tunnel Endpoint
VTN	Virtual Tenant Network
VXLAN	Virtual Extensible Local Area Network
WAN	Wide Area Network
WICM	WAN Infrastructure Connection Manager