NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D4.01

Infrastructure Virtualisation and Management

Editor Michael J. McGrath (Intel)

Contributors Vincenzo Riccobene (Intel). Eleni Trouva, George Xilouris (NCSRD). Aurora Ramos (ATOS), Beppe Coffano, Luca Galluppi, Pierangelo Magli (HP). Kimon Karras (FINT), L. Zuccaro, F. Delli Priscoli, A. Pietrabissa (CRAT), G. Dimosthenous, S. Charalambides (PTL). J. Carapinha (PTIN), Piyush Harsh, Antonio Cimmino, Glorjan Çadri (ZHAW), Evangelos Markakis (TEIC), Paolo Comi (Italtel). G. Gardikis, I. Koutras (SPH)

Version 1.0

Date 19th December, 2014

Distribution PUBLIC (PU)

Executive Summary

This deliverable presents the current activities and interim results of the four active tasks in Work Package 4 of the T-NOVA project. This work package is focused on the key elements of the T-NOVA Infrastructure Virtualisation Layer (IVM). Current activities are focused on the identification of appropriate virtualisation mechanisms and enablers; implementation and characterisation of a virtualised software defined networking (SDN) control plane; implementation of an SDN software development kit; infrastructure monitoring and maintenance subsystems for the IVM.

Section 2 describes the inter task dependencies of WP4. These inter dependencies are being carefully considered and monitored to ensure that the tasks in the work package receive appropriate input to guide their activities. These inputs are important in aligning the outputs of each task in WP4 in order meet the expectations and needs of the dependent tasks.

Section 3 describes the activities of Task 4.1 which are focusing on aspects of virtual Node (vNode) resource virtualisation, workload characterisation; technology enhancements and optimisations required for supporting Virtualised Network Functions (VNFs) and Network Services (NS) deployments on vNodes. These activities have the explicit goal of determining the most appropriate composition of technology components and integration approaches to implement a functional T-NOVA IVM. The implementation of the IVM and its functional entities is required to provide a performant environment for hosting VNFs and NS.

Section 4 outlines the activities Task 4.2 which is designing and developing a virtualised SDN Control Plane to support virtual network creation and management over OpenFlow-enabled networks. The task is also investigating the most appropriate architecture for the SDN controller implementation in T-NOVA. Options being investigated are focused on centralised and distributed approaches.

The third topic of WP4 is described in Section 5. Task 4.3 is designing and implementing a software development kit (SDK) for the SDN Control Plane (SDK for SDN). The key aim of this task is to provide the SDN community with a single framework to develop SDN applications regardless of the underlying API.

Monitoring and Maintenance issues of the virtualised infrastructure are dealt with in the fourth task of WP4. Section 5 outlines the activities in the design and development of a monitoring framework that will monitor the physical and virtual resources of the IVM and make this information available to the T-NOVA Orchestration layer.

Finally Section 7 outlines the technologies that have selected for the implementation of the IVM and its functional components namely the virtualised infrastructure manager (VIM), network function virtualised infrastructure (NFVI) and transport network manager (TNM). A rational for the selection of each technology component is provided together with the alternative technologies that were investigated. A mapping of T-NOVA requirements to the technology is provided to ensure that the selected technology can support these requirements in an appropriate manner.

Table of Contents

Index of Figures

Index of Tables

1. INTRODUCTION

Work package 4 is focused on the characterisation of virtualisation mechanisms and enablers, the SDN control plane, an SDN SDK, as well as the infrastructure monitoring and maintenance subsystems for the T-NOVA system. The output of these tasks play a key role in the definition, implementation, functional testing and performance validation of the key components that will be utilised in the T-NOVA Infrastructure Virtualisation and Management (IVM) layer. This deliverable outlines the key activities and findings to date from the four active tasks in WP4.

The IVM layer is responsible for providing a performant virtualised hosting and execution environment for VNFs and NSs. The IVM is comprised of a Network Function Virtualised Infrastructure (NFVI) domain, a Virtualised Infrastructure Manager (VIM) and a Transport Network Manager (TNM) as previously described in D2.31. The IVM provides full abstraction of these resources to VNFs by using virtualisation technologies. However many virtualisation technologies find their origins in the IT domain where performance constraints can be more flexible than those required in carrier grade telecoms environments. Virtualisation has also expanded beyond its initial focus on compute resource virtualisation to encompass a variety of different technology approaches such as hardware, operating system, storage, memory and network. Collectively these approaches have enabled the complete virtualisation of infrastructure resources found it a traditional data centre.

Virtualisation is the key enabler technology that allows traditional physical network functions to be decoupled from fixed appliances and to be deployed onto industry standard servers in large Data Centres (DCs). This approach is providing key benefits to operators such as greater flexibility, faster delivery of new services, a broader ecosystem enhancing innovation in the network etc.

While virtualisation brings many benefits to Enterprise IT and more recently to the Telecoms domain it also brings many challenges particularly in achieving the same level of performance in comparison to the traditional fixed appliance approach. The composition, configuration and optimisation of the virtualised resources are critical in achieving the required levels of performance. Additionally given the origins of many virtualisation technologies such as cloud OS environments there are capability gaps that need to be addressed in order to adequately support VNF/NS type workloads. This WP is addressing some of these gaps by extending existing technologies in a manner that is compatible with their current instantiation, implementing bespoke solutions where necessary and integrating them into existing technology solutions in order to extend them. These approaches are necessary, firstly to address T-NOVA requirements and secondly to illustrate the benefits of these modifications in order to influence the communities that are developing these technologies and their roadmaps. For example a T-NOVA extension to OpenStack's functional could form the basis for an OpenStack Blueprint contribution.

Another challenge is to ensure the orchestration layer fully exploits the capabilities of the servers it manages. Currently within cloud environments the resources are highly abstracted which again causes issues for the performant deployment of VNFs and

NSs. It is important to expose the specific platform features such as unique CPU instructions and attached devices, such as acceleration cards, co-processors or Network Interface Cards (NICs) with advanced capabilities. Additionally many hardware devices, such as NICs, have additional dependencies, such as the availability of supporting software libraries - e.g. DPDK - in order for a VNF to function in an optimal manner.

The combination of hardware and software technology components plays a critical role in implementation of the IVM and its composite functional entities namely the NFVI, VIM and TNM. Individual and collective performance of these functional entities has a significant impact on the VNFs/NSs hosted within the IVM. For any given function more than one technology choice maybe available. It is therefore important to understand how the technology options will performance and specifically within the context of the T-NOVA system. These topics are addressed by Task 4.1 specifically focusing on aspects of vNode resource virtualisation, workload characterisation, technology enhancements and optimisations to support VNF/NS deployments on vNodes in a performant manner.

The lack of platform and infrastructural awareness is a major drawback since many virtual appliances have intense I/O requirements and could benefit from access to high-performance instructions, accelerators and NICs for workloads such as compression, cryptography and transcoding. Identification of the relevant platform features than can influence VNF/NS performance is also a focus within Task 4.1. Key findings from the task will be an important input into Task 3.2 in WP3 which is focused on provisioning a resource repository to the T-NOVA Orchestration layer. In addition Task 3.2 is identifying mechanisms to improve the intelligence of the VM provisioning process within the VIM. This involves exposing key platform features identified in Task 4.1 to the scheduling function within the cloud environment. This capability should help to improve VNF/NS performance through improved provisioning of appropriate resources.

In Task 4.2 the focus is on the design and development of an SDN Control Plane to support virtual network creation and management over OpenFlow-enabled networks. Starting with an in-depth analysis of the existing solutions, extensions and necessary modifications will be identified to address the requirements identified for the T-NOVA system. Specifically, the task is evaluating distributed approaches to overcome network control plane centralisation limitations and to improve the performance and reliability of the network controller. Additionally network encapsulation techniques are being considered in order to provide enhanced connectivity services in multi-tenant scenarios. In this context the key issues to be addressed are resource optimisation, QoS support and live migration of the VMs hosting the NFV applications.

The third task of WP4 is the design and implementation of a Software Development Kit (SDK) for the SDN Control Plane (SDK for SDN). The SDK for SDN task is undertaking a detailed analysis of the interactions between virtualised and physical network elements in a DC to identify potential bottlenecks and potentials for optimisations through adoption of SDN capable network elements. The task, based on the outcome of this analysis will provide libraries and code examples to alleviate some of the identified bottlenecks. A prominent aim of this task is to provide the community a single framework to develop SDN applications regardless of the underlying controller API. The northbound APIs of several popular SDN controllers will be analysed to identify the common feature-set to be unified under the initial release of the SDK, with controlled support for the disparate elements in various SDN controllers to be exposed to application developers. The SDK will also provide useful libraries to aid the development process – debugging support, test environments using Mininet or another similar frameworks. Requirements emerging from different tasks in T-NOVA will be collected, specifically from WP5 where most of the NFs are to be virtualised; the SDK design will be guided in part by these requirements in order to facilitate the NFV development process.

The final topic in WP4 relates to monitoring and maintenance issues in virtualised infrastructures. The collection and exposure of dynamic metrics reflecting the current status of the IVM is critical for supporting most of the T-NOVA use cases. Task 4.4 is designing, implementing and integrating a monitoring framework within the VIM which collects metrics from computing and network nodes (both virtual and physical), aggregates the metrics and analyses them. Generic VNF metrics are also collected and processed. As an output of the metrics processing workflow, selected measurements and alarms/events are communicated to the Orchestrator via an API to be also defined in this task, in order to facilitate service mapping and management procedures.

2. WP4 Inter Task Dependencies

The outputs of the tasks within WP4 have a number of key dependencies with other tasks in WP4 and with tasks in other work packages, including WP3 and WP5 as outlined in Tables 2.1 to 2.4. Therefore on-going close cooperation and coordination between the dependent tasks will be required to ensure that the outputs are appropriate and meet the expectations of the dependent task. The following tables provide a brief description of the task dependencies.

2.1. Task 4.1 Resource Virtualisation Task Dependencies

Dependent Task	Dependency
Task 3.1 – Orchestrator Interfaces	Task 4.1 will provide an input on the metadata that must be sent by the Orchestrator in order to support platform aware VNF scheduling in OpenStack Nova
Task 3.2 - Infrastructure Repository	Task 4.1 will identify appropriate platform features that should be collected and stored in the Infrastructure Repository. These features are expected to be useful during the OpenStack Nova scheduling and filtering processes in order to improve VNF placement decisions.
Task 3.3 - Resource Mapping	Task 4.1 will identify the workload, platform and infrastructure features that have a significant impact on VNF performance. The features will act as inputs into the design and development of the resource mapping algorithm which maps VNF to the most appropriate platform locations.
Task 4.5 - Infrastructure Integration and Deployment	Task 4.1 will help to define the technology components including both software and hardware and their most appropriate configuration required to implement a performant IVM in Task 4.5.
Task 5.4 - Performance Assessment and Optimisation	Task 4.1 will provide some best practices relating to the design and the configuration of virtualisation infrastructure which will be taken as an input by Task 5.4 which is focused performance assessment and optimisation of T-NOVA's VNFs.

Table 2.1 Outline of Task 4.1 inter-task dependencies

2.2. Task 4.2 SDN Control Plane Task Dependencies

Dependent Task	Dependency
Task 4.1 - Resource Virtualisation	Task 4.1 is investigating resource allocation for various processing workloads in order to share the same resources efficiently. SDN control plane plays a

	significant role when the processing workloads requiring network resources. Therefore, the SDN control plane will allow sharing of the same network resources by creating and maintaining isolated vNets.
Task 3.1 - Orchestrator Interfaces	One of the objectives of Task 3.1 is the implementation of the southbound interfaces, which will communicate with the underlying layers; the SDN control plane and the cloud controllers. Thus, Task 4.2 is directly related with Task 3.1, which will provide the communication between the SDN controller and the Orchestrator.
Task 3.2 (Infrastructure Repository)	The discovery engine of Task 3.2 will interact with the SDN controller in order to store network topology information.
Task 4.5 Infrastructure Integration and Deployment	Task 4.5 will handle the deployment and integration activities for T-NOVA's infrastructure. Part of this work includes the validation of the SDN Control plane that will be defined and developed in Task 4.2. Thus, the work that will be carried out in Task 4.2 will be one of the initial starting points in Task 4.5.

Table 2.2 Outline of Task 4.2 inter-task dependencies

2.3. Task 4.3 SDK for SDN Task Dependencies

Dependent Task	Dependency
Task 2.4 – Specification of IVM	Task 4.3 will enable the user to fulfil the requirements defined by Task 2.4 in terms of setting up, managing and monitoring networks.
Task 4.2 - SDN Control Plane	Task 4.2 is focused on providing an abstraction layer to SDN application developers building network applications allowing them to avoid having a detailed understanding of the underlying controller. Task 4.3 will support the SDN Controller northbound interface, by building an abstraction layer and exposing it to SDN applications developers.
Task 3.4 - Service Provision, Management and Monitoring	Task 3.4 will use a generic northbound interface provided by Task 4.3, allowing the implementation of the Orchestrator to be agnostic of the underlying SDN controller. The T-NOVA framework is not strictly dependent on the specific SDN controller, as long as the SDK toolset provides translation of the generic-to-specific-new-controller northbound interface.
Task 5.3 - Development of VNFs	The VNFs developed in Task 5.3 may require significant interaction with the underlying SDN network infrastructure. The VNF developers will use the SDK to simplify the interaction with the actual SDN controller.

Table 2.3 Outline of Task 4.3 inter-task dependencies

2.4. Task 4.4 Monitoring and Maintenance Task Dependencies

Dependent Task	Dependency
Task 2.4 - Specification of IVM	IVM specifications and requirements drive Task 4.4 design and implementation decisions.
Task 3.1 - Orchestrator. Interfaces	The approaches taken in Task 4.4 affect the interface to the Orchestrator for communication of monitoring metrics.
Task 3.3 - Service Mapping	Service mapping strongly depends on IVM metrics.
Task 3.4 - Service Provision, Management and Monitoring	IVM metrics are essential for proper service monitoring.
Task 4.1 - Resource Virtualisation	The technical approach of the monitoring framework strongly depends on the technical specifications of the NFVI substrate.
Task 4.2 - SDN Control Plane	The technical approach of the monitoring framework strongly depends on the technical specifications of the SDN Control Plane.
Task 4.5 - Infra. Integration and Deployment	The IVM monitoring framework is one of the components to be integrated by Task 4.5.
Task 5.3 - Development of VNFs	The implementation of NFs will affect how NF resources will be monitored.
Task 6.3 - User Dashboard	It is assumed that some IVM metrics will be presented on the dashboard.
Task 6.4 - SLAs and Billing	SLA monitoring procedures strongly depend on IVM metrics.

Table 2.4 Outline of Task 4.4 inter-task dependencies

3. Resource Virtualisation

This section relates to the activities of Task 4.1, which is focused on the identification, characterisation and optimisation of the hardware and software components that will be used in the implementation of the T-NOVA IVM. Task 4.1 is also examining the inter-relationship of VNFs and their host virtualised environments. The task will also look at the challenges than can exist around co-competing optimisation criteria e.g. performance vs resource consumption, cost vs reliability, infrastructure homogeneity vs heterogeneity etc. The key outputs of this task will be a set of best practices and insights regarding the appropriate configuration of the infrastructural components and the technologies to be used in the implementation of the T-NOVA IVM.

The VNFs and the Networks Services that are composed from them have varying compute, storage and network requirements that are context specific. It is therefore important from an IVM point of view to develop an understanding of how VNF type workloads interact and consume resources in their host environments and how these interactions vary on a temporal basis. While Task 4.4 is looking at the collection and exposure of dynamic IVM system metrics to the Orchestration layer, Task 4.1 will specifically work on identifying both dynamic and static metrics that are correlated with VNF performance and its host environment. These metrics should enable insights into the specific composition of resources and their configuration.

The ESTI NFV Group Specification provide some guidance on the types of metrics that should monitored and will inform the initial set of metrics to be capture and analysed [1]. While metrics can play an important role in the characterisation of VNF workloads and their environments the volume of potential metrics can be overwhelming and can dilute their value particularly in an operational context. Additionally, the identification of the key static metrics is a key input into Task 3.2 which is focused on the implementation of a resource repository for the T-NOVA Orchestrator.

While current cloud environments do track some limited static metrics in terms of platform characteristics - e.g. CPU speed - these are very limited in scope. They are currently not considered sufficient for the intelligent placement of VNFs onto virtualised infrastructures. For example a VNF which has a dependency on DPDK (Data Plane Development Kit [2]) libraries for accelerating packet processing performance cannot be deployed onto the appropriate compute node without using enhanced compute scheduling mechanisms. An initial step in this direction is to identify non-generic platform features both hardware e.g. AES-NI, TXT, SR-IOV capable NICs and software e.g. DPDK libraries, which need to be exposed to allow the Orchestrator to make better workload placement decisions. Additionally, mechanisms, which can be utilised to expose these, enhanced platform features to the scheduling and filtering mechanisms in cloud compute environments, will also be explored.

As a technology, NFV encompasses a wide variety of network functions which have a diversity of resource requirements. It is important therefore to develop an understanding of the workload types and their affinity for certain platform features and technologies. While it is not possible to identify all the affinities for all VNFs within the scope of Task 4.1, the development of a robust methodology is possible.

To support workload characterisation activities in the task, a flexible test-bed platform is being developed which will be composed of the technologies that are relevant to the IVM and will enable to rapid evaluation of technologies that may emerge over the lifetime of the T-NOVA project. In the design and implementation of the test-bed industry initiatives such as ONP (Open Network Platform) and OPNFV (Open Platform for NFV) are being monitored closely and their outputs are being utilised were appropriate.

The Task 4.1 test-bed will also make use of instrumentation to capture a full set of metrics from system counters that will be reduced to a set that are most highly correlated with workload or system performance. In this way, the test-bed will enable the benchmarking of potential technologies and workloads. Key considerations such as VM start-up time, network latency etc. will be investigated.

Finally the Task 4.1 will develop a set of Best Known Methods (BKMs) for virtualised environment implementation for the performant deployment and management of VNFs. It is expected that these BKMs will be used by other tasks in the setup and configuration of test infrastructures for their task activities. It is also expected that output of Task 4.1 will be used in WP7 for guiding the pilot integration and field trials.

3.1. Candidate Technology Selection and Rational

A key activity for Task 4.1 is the identification of the key software and hardware components that will form the IVM platform and specifically each functional entity within the IVM. While more than one technology may exist for a specific role within the IVM (e.g. SDN Controller) initial selections are made around the most appropriate match to the requirements identified in D2.31 and the level of community or commercial support for the technology. The section presents the initial set of platform, hardware and software technologies that will be used in the implementation of a tested to evaluate the technologies, determine the most appropriate configuration and to develop optimisations for VNF deployments within a cloud environment.

3.1.1. Platforms

Open source software and open standards are playing a key role in networking, communications, and cloud infrastructure by supporting the transition from fixed-function, complex network equipment based on proprietary architectures, to solutions based on lower cost and open technologies.

Two of the leading approaches designed to address the needs of telecommunication industry are NFV and SDN. These approaches enable demand-driven scalable service provision across pooled elastic infrastructures and have been discussed in detail in previous T-NOVA deliverables.

In order to accelerate the migration towards SDN and NFV, technology companies have been developing platform based solutions that combine interoperable hardware and open source software ingredients based on standards that enable Telco’s evaluate and deploy NFV and SDN solutions into their networks. A key effort in this direction is Intel's ONP, which provides an application-ready solution supported by the open software community, commercial software, system integration alliances, and industry standards bodies. ONP accelerates and simplifies the deployment of those technologies, extending the capability to test, deploy and scale new generation services. It involves two reference design specifications, respectively related to switches [3] and servers [4].

Of particular interest to T-NOVA is, the *ONP Server Reference Design* based on a set of open source software components and software/hardware configurations integrated together on standard servers to deliver a working platform and an infrastructural framework for efficiently virtualising Network Functions (NFs).

In ONP, the Fedora 20 (64-bit) Linux distribution is the based Operating System (OS). Integration of *QEMU*[1]*-KVM* Virtual Machine Monitor (VMM) is provided to support the execution of VMs on the physical infrastructure coupled with libvirt as the hypervisor manager, OpenStack as the Cloud Controller to control the VM lifecycle (instantiation, resource allocation, termination, and so forth) and OpenDaylight as the Network Controller, to control the traffic paths between Virtual Network Function Components (VNFCs). In terms of virtual switching technology, ONP proposes Open vSwitch and Intel's DPDK vSwitch as the open standard solution.

In the latest release, OpenDaylight is not integrated with OpenStack but only with the virtual switching technology. Analysing the T-NOVA IVM requirements, the integration between Neutron and OpenDaylight is necessary in order to provide the deployment of virtual network across the physical infrastructure. For this reason, the ONP requires extension with the integration of the ML2 Plugin [6] from a T-NOVA perspective, integrating the OpenStack Neutron network component with the Control Plane controller (i.e. OpenDaylight). Figure 3.1 provides a mapping of the main software and hardware components in ONP with respect to the functional entities of the T-NOVA IVM.

OPNFV
AN OPEN PLATFORM TO ACCELERATE NFV
A Linux Foundation Collaborative Project

In September 2014 the Linux Foundation announced the Open Platform for NFV Project (OPNFV) [7] which is focused on developing carrier-grade, integrated, open source reference platform. The initial scope of OPNFV will be on building NFV Infrastructure (NFVI), Virtualised Infrastructure Management (VIM), and including application programmable interfaces (APIs) to other NFV elements, which together form the basic infrastructure required for VNF and Management and Network Orchestration (MANO) components. The platform is expected to be based on existing open source projects including OpenDaylight, OpenStack, Open vSwitch and the Linux kernel among others [7]. Task 4.1 will monitor the outputs of this project closely and will integrate outputs as appropriate.

[1] QEMU is an open source Virtual Machine emulator [5] QEMU. (2014). *Open Source Processor Emulator*. Available: http://wiki.qemu.org/Main_Page.

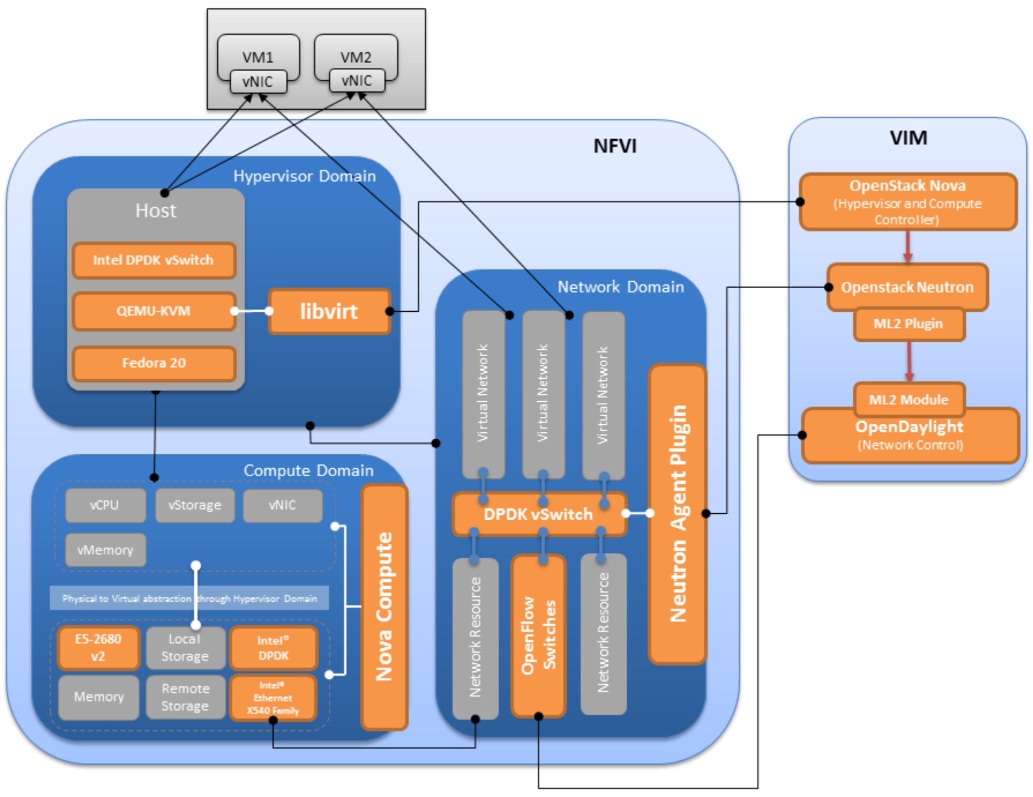


Figure 3.1 Mapping of ONP components to T-NOVA IVM.

3.1.2. Hardware

This section describes the key characteristics and capabilities of the hardware components considered in the design and deployment of the Task 4.1 testbed. In addition to identifying the hardware components, Task 4.1 will also evaluate their performance, as well as interrogating various configuration options to identify the appropriate set of hardware resources and configurations for use in Task 4.5 (Infrastructure Integration and Deployment).

3.1.2.1. SDN Switches – Features and Capabilities

The most important feature introduced by the SDN approach is the separation of the Control Plane (CP) and Data plane. SDN switches (both physical and virtual) are based on the replacement of the local CP with a programmatic interface supporting standard flow control protocols (see Figure 3.2). This supports the migration of the network device CP to a centralised controller that has holistic view of the overall network and can dynamically respond to changes: the controller makes decisions about the CP and automatically configures the switches accordingly through their northbound interface.

The most common protocol used to implement this interface is OpenFlow [8]. It is an open standard that defines how the controller interacts with the Data Plane and makes adjustments to the network, adapting to changing requirements or conditions. When a switch receives an unknown packet it sends a *packet-in* event to the controller, requesting instructions on what actions to apply in order to process the packet appropriately; the controller installs on the switch one or more entries in the flow tables that match the specific packet providing a set of actions to be executed during the forwarding phase. Thereafter, each time the switch will receive a new packet that matches the new flow entries; it will execute the specified actions without recall to the controller.

While OpenFlow is regarded as de-facto standard protocol for SDN, new vendor led open source SDN protocols are also emerging, such as OpFlex from Cisco [9]. OpFlex takes a different approach to switch configuration. Instead of sending specific configuration instructions to downstream networking equipment, it sends down an application policy or the application's network requirements, allowing the devices to self-configure accordingly. OpFlex comprises of both the protocol and the set of standards used to communicate the policies.

It is important for the T-NOVA project to continuously monitor developments in both the OpenFlow protocol and other protocol initiatives to ensure that the T-NOVA solution evolves appropriately overtime.

SDN switches are commonly focused on Top-of-Rack (ToR) access switching providing 1Gbe (or faster) connectivity to servers with high-speed uplinks to the next level of aggregation switching. Switches with SDN support are available from a variety of vendors, including Extreme Networks, Cisco, NEC, etc. The first generation of SDN switches such as the PICA8 supported 1 Gbps line rate speed. Most SDN switches commercially available at present support at least 10 Gbps line rates. Many 10 Gbps switches offer from 2 to 4 40Gbe uplink ports to provide switch fabric connectivity. Switches which provide various fixed configurations of 10, 40 and 56 Gbe QSFP/SPF+ ports are also available from vendors such as Mellanox, Arista, Brocade, etc. However they can be relatively expensive for large-scale deployment.

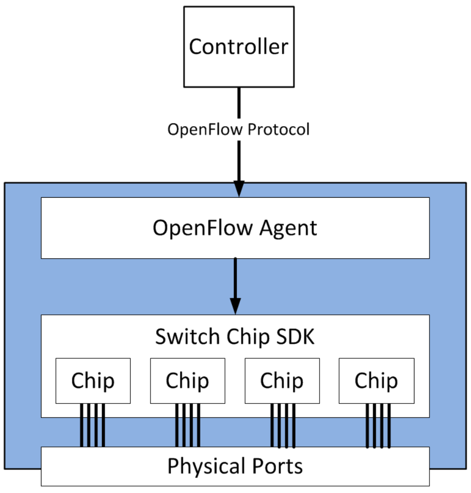


Figure 3.2 High level architecture of an SDN switch

SDN switches typically support Layer 2 and Layer 3 forwarding in 48 or 64 port configurations with IPv4/IPv6 support. The physical switch connections are mainly SFP+ with either optical cabling or direct attached copper cabling. Some vendors also offer switches with 10Gbase-T connections over Category 6/7 cabling preferable for lab base test beds due to the lower cost of connections.

An important feature which can affect performance of an SDN switch is the memory used to store forwarding tables. OpenFlow 1.0 switches typically used the Ternary Content-Addressable Memory (TCAM), which is a specialised type of high-speed memory that searches its entire contents in a single clock cycle. TCAM supports efficient flow instantiation updates from an SDN controller and its lookups match explicit 1s and 0s but also have a "don't care" bit. In OpenFlow this is referred to as a wildcard bit often depicted in diagrams with a '*'. However, the use of TCAM in an SDN switch is not without issues. TCAM is power hungry, expensive and can have a large silicon footprint, and is often the most expensive component on the switch [10]. Some vendors now use a blend of BCAM memory, SRAM, NPUs and software algorithms to perform ternary lookups instead of utilising expensive TCAM.

From a T-NOVA system perspective there is a number of considerations regarding the choice of switch for the IVM layer that comes from the analysis of the requirements, outlined in D2.31. First, the switch should support the OpenFlow protocol, due its open source nature and its broad industry adoption, OpenDaylight support and its de facto industry standard status. At a minimum the switch needs to support OpenFlow version 1.0. Ideally, the switch should feature OF version 1.3 which is currently supported the Helium release of OpenDaylight. Additional support for open source protocols such as VxLAN is desirable to support alternative research configurations/investigations if necessary. Use of TCAM memory would be desirable to maximise flow table lookup performance. Finally, the switch vendor should have a robust and timely roadmap for SDN support to ensure that the switch can be upgraded to the latest technology developments over the lifetime of the T-NOVA project.

3.1.2.2. Network Interface Cards (NICs)

The cost of 10GB Ethernet has fallen significantly over the last few years and it is becoming more common place in many datacentres. Typically a 10GB NIC costs in the range of $500 - $1000 depending on its feature set. 10-gigabit Ethernet (10GE, 10GbE, or 10 GigE) was first defined by the IEEE 802.3ae-2002 standard. The standard only defines full duplex point-to-point links which are generally connected by network switches. Connections between ports can be provided by either copper or fibre cabling. The standard supports a number of different physical layer (PHY) standards including XFP, XENPAK, QSFP, enhanced small form-factor pluggable transceiver, (SFP+) and 8P8C (RJ45). SFP+ has become the most popular socket on 10GbE systems, however with additional cost considerations over copper. Due to the high bandwidth requirements, higher-grade copper cables are required in comparison the 10/100/1000 MB standards. Category 6a or Class F/Category 7 cable are necessary for links up to 100m in length. 10GbE network interface cards are available from several manufacturers with a variety of options as outlined in Table 3.1.

Feature	Options
Physical Connectivity	XFP, XENPAK, QSFP, (SFP+) and 8P8C (RJ45).
Power Management	Thermal Design Power (TDP)
Technology Support	IWARP/RDMA, Power Management, SR-IOV, VMDq, On-chip QoS, DPDK support, Unified Networking (LAN/SAN (e.g. iSCSI) traffic support on the same network fabric etc. Packet Filtering VLAN Support

Table 3.1 Typical NIC Characteristics

Another key consideration is backwards compatibility. Some NICs will only support 10GB connections while others provide backward compatibility with existing 1000Base-T networks such as Intel's x540 Converged Ethernet NICs.

Most 10GB NICs support SR-IOV, which is a PCI SIG standard that allows a single PCIe device to be subdivided into multiple virtual instances. These virtual instances, known as virtual functions, can be assigned to separate VMs and appear to the VM as its own individual NIC, without the need for packet traffic to traverse the hypervisor layer. Theoretically up to 256 VFs can supported, however currently the practical limit of 64 VFs appears to be the upper limit for most devices. From a T-NOVA system perspective, the use of SR-IOV capable NIC introduces some important considerations. Many SR-IOV NICs include basic L2 hardware switching on the NIC. As a result, VM-to-VM traffic can be extremely fast (up to 40 Gbps). However, when traffic patterns change from VM-to-VM communications on the same SR-IOV NIC to VM-to-VM on different cards, performance can change significantly. Depending on the path and the involved NICs (i.e. SR-IOV to SR-IOV vs SR-IOV to non SR-IOV NICs) performance will drop back to the physical layer speed or lower.

3.1.2.3. Compute Platform

The compute platform is a critical component in supporting the functional deployment of VNFs. Initially, VNFs were deployed onto standard X86 servers that were designed for Enterprise Cloud environments. While supporting virtualisation technologies, standard servers lacked features to support VNFs that had high packet processing requirements. Compute platforms that are targeted specifically at large-scale communications infrastructure systems have started to emerge. Intel now offers a platform targeted at NFV solutions which is based around its Xeon E5 v2/v3 product family and 89xx communication chipsets. This platform provides hardware-based acceleration and the general purpose processing needed for Telco workloads. The platform provides a number of key technologies to improve the performance of NFV workload types, including QuickAssist, XL710 40GbE Ethernet Controllers and PCI Express.

The chipset series provides hardware-based cryptographic and compression acceleration capabilities for a wide range of communications infrastructure applications, such as high-end security appliances, enterprise routers, and wireless infrastructure. The Intel Data Plane Development Kit (DPDK) complements the platform by improving packet processing speeds to handle increasing network traffic data rates and associated infrastructure control/signalling requirements. The chipset series also provides hardware offload assistance up to 20 Gbps for virtual private networks (VPNs) and helps storage and network optimisation applications better handle compression and decompression tasks.

A range of processor options allows developers to create a family of products based on one design. The specific number of cores required will typically be VNF dependent, relating to the number of threads that have to be supported (10 cores – 20 threads, 8 cores – 16 threads, 6 cores - 12 threads). For multi socket systems QuickPath Interconnects (QPI) provide low latency connections between the processors (as shown in Figure 3.2) which is important for bandwidth intensive applications.

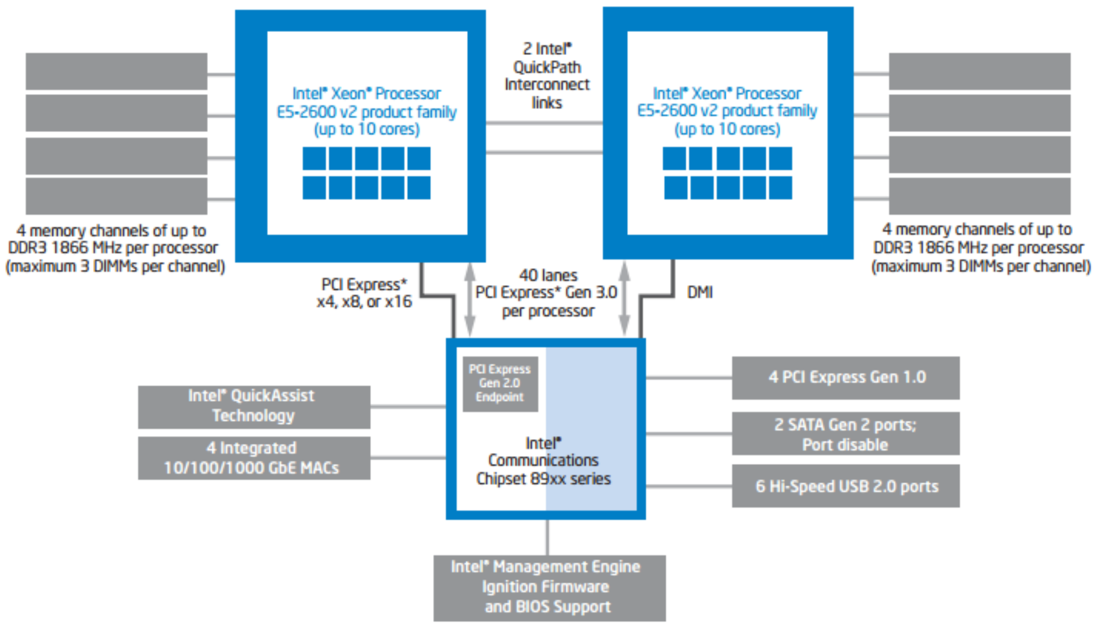

Intel® Xeon® Processor E5-2600 v2 product family (up to 10 cores)
2 Intel® QuickPath Interconnect links
Intel® Xeon® Processor E5-2600 v2 product family (up to 10 cores)
4 memory channels of up to DDR3 1866 MHz per processor (maximum 3 DIMMs per channel)
4 memory channels of up to DDR3 1866 MHz per processor (maximum 3 DIMMs per channel)
PCI Express* x4, x8, or x16
40 lanes PCI Express* Gen 3.0 per processor
DMI
PCI Express Gen 2.0 Endpoint
Intel® Communications Chipset 89xx series
Intel® QuickAssist Technology
4 Integrated 10/100/1000 GbE MACs
4 PCI Express Gen 1.0
2 SATA Gen 2 ports; Port disable
6 Hi-Speed USB 2.0 ports
Intel® Management Engine Ignition Firmware and BIOS Support

Figure 3.3 Dual-socket -socket configurations of Intel E5-2600 v2 Xeon processor with 89xx communications chipset

3.1.3. Software

In this section the most important software components relevant to the investigations of Task 4.1 are discussed.

3.1.3.1. Open vSwitch and DPDK vSwitch

Virtual Switch (vSwitch) technology is a key component in realising NFV. Various vSwitch technologies including Open vSwitch and DPDK vSwitch have previously been described in D2.31. The section provides additional information on the candidate technologies in order to further explore the different configuration options and implementation features.

Open vSwitch (OvS) [11] is a production quality, multilayer virtual switch licensed under an Apache 2.0 open source license and it represents the de facto standard technology in terms of vSwitches. Intel DPDK vSwitch is a branch of OvS, which couples the original software switching technology with DPDK in order to improve the performance of OvS, while maintaining its core functionality. The OvS source code has been modified to enable fast packet switching and improves small-packet performance.

Along with the DPDK vSwitch source code, a specific version of QEMU is also provided: to enable efficient inter-VM communications by interfacing with the accelerated vSwitch at the hypervisor layer.

DPDK vSwitch currently provides two communication methods between the Virtual Machine (VM) and the host: *Userspace vHost* and *IVSHMEM* (Inter Virtual machine SHared MEMory).

The Userspace vHost mechanism provides a virtio Poll Mode Driver (PMD) as a software solution for fast guest-VM-to–guest-VM communications and guest-VM-to-host communications. vHost is a kernel module which works as the backend of virtio (a para-virtualisation driver framework) to accelerate the traffic from the guest to the host. The DPDK kernel NIC interface provides the ability to attach vHost traffic to userspace DPDK applications. Together with the DPDK PMD virtio, it significantly improves the throughput between guest and host. Further details regarding these mechanisms are available in the DPDK Programmer’s guide [12].

With the IVSHMEM mechanism, shared memory between the VM and the host is utilised in order to improve the performance of information exchange. As a result of the shared memory, zero copies between the guest and switch are required. This option can be particularly useful for trusted applications that require very fast small packet throughput. However, using this option means the Linux Kernel network stack is completely isolated from the packet processing. Both the mechanisms and configuration options are currently being investigated in Task 4.1.

In order to select the appropriate virtual switching technology, from a technical perspective, a set of tests was been performed to measure the relative throughput, comparing the performance levels of Open vSwitch and DPDK vSwitch. The results of this comparison are presented in Section 3.3. DPDK vSwitch delivers superior performance, based on the current versions of these technologies. However Intel has recently announced that they are ceasing investment in DPDK vSwitch and will instead focusing their efforts on OvS and advancing hardware acceleration. Intel’s new mainstream OvS code called “DPDK-netdev” is already present in OvS version 2.3; however it is currently only available as an experimental feature and not all the DPDK mechanisms are fully supported. It is expected the next release of OvS (version 2.4) early next year will feature fully DPDK support. It is therefore expected that in T-NOVA will adopted future releases of OvS for the IVM given recent developments.

3.1.3.2. OpenStack

The Cloud Controller candidate solution selected for T-NOVA is OpenStack (see Section 7), which provides the software components for building and managing cloud computing platforms for public and private clouds. Currently the Icehouse version of OpenStack is the reference version for the project.

The role of the OpenStack platform in T-NOVA is twofold. First, it supports the deployment and lifecycle management (in cooperation with the Orchestration layer) of the VNFs deployed on VMs within the cloud infrastructure. It also provides a common virtualisation layer across different platforms making the VNFs independent of the actual underlying physical infrastructure.

It is worth noting that OpenStack was originally designed to address enterprise cloud environment needs, managing the Compute and Hypervisor domains. From a T-NOVA perspective, the Infrastructure Network domain is as important as the Compute and the Hypervisor domains. Task 4.1 activities have been focusing on identifying current gaps in OpenStack which need to be addressed in order to deliver a cloud environment suitable for NFV/SDN. The gaps identified to date relate to the exposure of granular platform features and characteristics to the Orchestrator. This is necessary in order for the orchestrator to make the most appropriate allocation decision according the best of match VNF service characteristics, constraints and available infrastructural resources.

Technologies and features that are crucial for Telco workloads in terms of performance include DPDK, co-processors (GPUs or FPGAs), SR-IOV capable NICs, Non-Uniform Memory Access (NUMA) awareness, and so forth. Task 4.1 is focusing on identifying which features and mechanisms are required to deliver increased platform awareness within OpenStack. Task 3.2 is utilising this work to implement an actual working solution within the context of OpenStack and to expose the infrastructural landscape with increased fidelity to the T-NOVA Orchestration layer. While platform awareness is important, it serves no useful purpose unless the information can be utilised in an effective manner.

OpenStack’s scheduling mechanism (called Nova Scheduler) uses a filter-based approach in the form of a Filter Chain to make decisions regarding the dispatching of compute (and volume) requests. This mechanism needs to utilise the additional platform information effectively in the scheduling process. The Filter Chain can be composed by built-in filters (already provided by OpenStack) or custom filters (extending the standard catalogue of filters). At the end of the filtering process, a list of acceptable hosts is provided and eventually subjected to a weighting process to choose a node where to deploy a VM.

3.1.3.3. SDN Controller Selection

During the selection process of the SDN controller by the T-NOVA consortium the level of community activity and support were considered. With this criteria in mind the initial set of SDN controller options for T-NOVA were identified as: Ryu, Trema, OpenDaylight, OpenIRIS, MUL and OpenContrail. ONOS from ON.Lab is currently under development and potentially may offer interesting capabilities such as a promised 100-millisecond recovery and the ability to process 1 million requests per second. ONOS is designed to support a number of use cases such as SDN control of multilayer networks. The potential value is the ability for service providers to operate their complete networks assets i.e. both packet and optical in an integrated manner. Other potential advantages of ONOS include the ability to reduce overprovisioning; and SDN-based WAN control (use of MPLS as the data plane, with an SDN Control Plane) [13]. However; ONOS was not included as an option as at the time of writing this deliverable its first release named Avocet was only made available on December 5th, 2014.

Another consideration in the selection of the SDN controller was the consortium's hands-on experience of the various controller options. When this criterion was applied, Ryu and OpenDaylight were the remaining options.

Analysis of the feature sets and roadmaps for the two remaining candidates resulted in the selection of OpenDaylight. OpenDaylight is a highly scalable open source controller platform written in Java. It is designed to be a modular SDN platform which differentiates it from many of the controllers reviewed in D2.31. It comes with support for an abstraction layer above the southbound interface, a Graphical User Interface (GUI), northbound interface abstractions, network discovery, pluggable southbound interfaces, L2 and L3 learning, path provisioning and a flexible northbound interface using Representation State Transfer APIs (REST APIs). Statistics collection mechanisms are offered within an OpenFlow plugin which are accessible through REST APIs. Improved customisation of the platform deployment is now supported in the latest release (Helium) through the use the Apache Karaf [14] container. This gives the user significant flexibility in defining the complexity of their deployment, the required feature set and footprint of the controller.

The available modules can be utilised for performing various tasks such as data gathering, network devices identification and management, etc. based on field-proven and popular technologies, such as Java, OSGi, REST, etc.

3.1.3.4. ML2 Plugin

The OpenStack Modular Layer 2 (ML2) plugin is a framework that allows OpenStack Neutron to simultaneously utilise a variety of Layer 2 networking technologies found in data centres. It currently works with the existing Open vSwitch, Linux Bridge, and Hyperv L2 agents. The ML2 framework was designed with a view to greatly simplify adding new L2 networking technologies. OpenDaylight leverages the integration the ML2 plugin provides for Neutron via a specific driver to enable communication between Neutron and OpenDaylight. On the SDN controller side, OpenDaylight has northbound APIs to interact with Neutron and to use OVSDB for southbound configuration of vSwitches on compute nodes. OpenDaylight can therefore manage network connectivity and setup GRE or VXLAN tunnels for compute nodes.

3.2. Proposed Architecture of Virtualisation Testbed

One of the main activities of the Task 4.1 is the implementation of a testbed platform to support workload and technology characterisation activities within the task. The initial architecture implemented for the experimental work outlined in the next section is shown in Figure 3.4. The development and deployment of this architecture is an ongoing activity within the task. It is expected that the architecture will evolve over the course of the task towards a more data centre oriented configuration.

The testbed is currently composed of three nodes: one controller and two compute nodes. The Controller acts as VIM (Virtual Infrastructure Manager - see Figure 3.1), and hosts the Cloud Controller (OpenStack Nova and Neutron) along with the Network Controller (OpenDaylight), integrated via the Neutron ML2 plugin.

The compute nodes include Nova Compute, which communicates with the controller through the management network. Virtualisation of the compute resources is based on the use of a KVM hypervisor and a libvirt hypervisor controller. Open and DPDK vSwitches deliver VM connectivity through the Data Network.

From a hardware perspective, all the hosts include an X540-T2 NIC, which has dual Ethernet 10GB ports, supporting SR-IOV and DPDK technologies: one port is connected to the Management Network and the other is connected to the Data Network. The controller and one of the compute nodes are based on Intel i7 4770, 3,40Ghz CPUs with 32 GB of RAM, the other compute node is a dual socket server with XEON E5 2680 v2, 2.8GHz CPUs and 96GB of RAM. This XEON E5 computing architecture provides 10 cores per processor (20 cores in total), a set of platform features of interest to T-NOVA (e.g. VT-x, VT-d, Extended page tables (EPT), TSX-NI, Trusted Execution Technology (TXT)) and 8GT/s Quick Path Interconnects for fast inter socket communications.

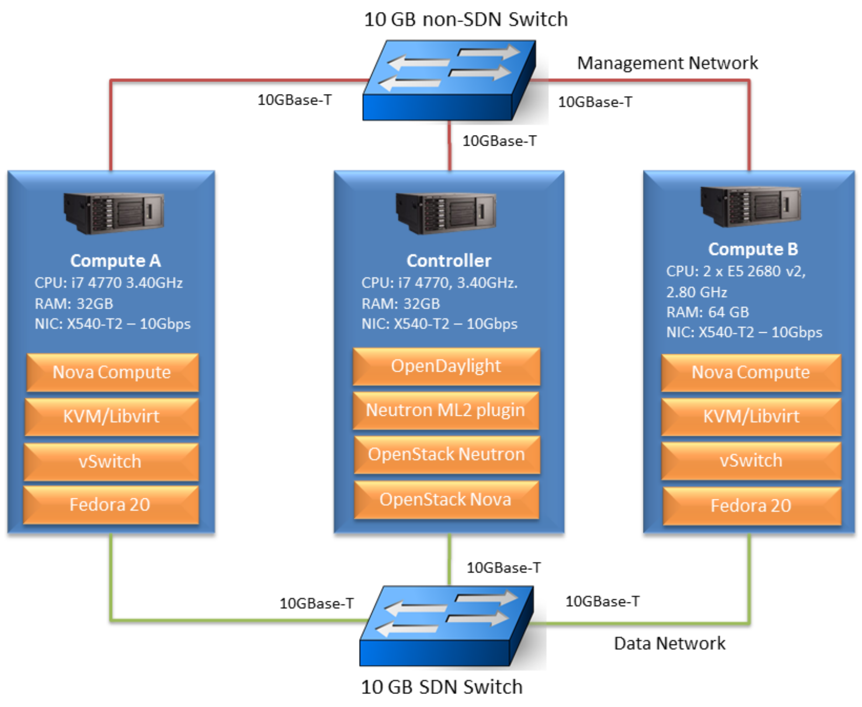


Figure 3.4 Proposed Architecture for the Virtualisation Testbed

3.3. Characterisation and Optimisation Experimental Protocols

As previously outlined, Task 4.1 is focused on technology, platform and workload characterisation activities. In order to apply a structured approach to these activities an initial experimental protocol has been defined.

In the migration roadmap from application-specific hardware to the generic hardware/software environment as envisioned by T-NOVA, a definition of best practices is required in order to help system owners (or administrators) to provide a suitable infrastructure, as well as Orchestrators to schedule VNF deployments in a manner, which achieves SLA fulfilment and optimal resource usage.

Based on the analysis of the IVM requirements in D2.31 an initial set of experiments have been performing. These experiments are focused on identifying metrics of potential interest for the quantitative evaluation of VIM performance in an effort to developing proposals with respect to potential optimisations. Table 3.2 lists the metrics identified to date. The list is not exhaustive and will be updated as appropriate in Deliverable 4.1.

It is important to identify which are the most influential parameters and to capture empirical results that indicate whether and to what extent they affect system performance. Table 3.2 specifies a concise list of parameters that could possibly affect the performance of VNFs and, consequently, Network Services.

Metrics	Description	Possible Influencing Parameters
Network Throughput	The throughput provided by the virtual network environment. It is measured as the difference from the number of sent and received packets.	• Network technologies configuration ○ DPDK ○ SR-IOV and VT-d • NUMA CPU Pinning • Core Pinning • Hugepages Size
Network Latency	This time interval starts when a packet leaves the source and ends when the packet reaches its destination.	• Network technologies configuration ○ DPDK ○ SR-IOV and VT-d • NUMA CPU Pinning • Core Pinning • Hugepage Size
VM Deployment Time	This time interval starts when the Orchestrator sends the *VM creation* command and ends when the VM is available over the Network.	• Image Size • Number of vNICs • Type of vNICs • Storage technologies and configuration
Multiple VMs Deployment Time	This time interval starts when the Orchestrator sends the *VM creation* command for the first VM and ends when the last VM is	• Average image size • Number of vNICs • Type of vNICs • Number of VM deployment

	available over the Network.	requests • Storage technologies and configuration
Live Migration Down Time	This time interval starts when the VM is no more reachable through the network at the source node and ends when the VM is reachable again at the destination node.	• Network technologies configuration • Storage technologies and configuration
Live Migration Overall Time	This time interval starts when a VM receives the command to migrate and ends when the VM is reachable over the network after the migration.	• Network technologies configuration • Storage technologies and configuration

Table 3.2 Selected performance metrics for the IVM.

The design of experimental protocols is an on-going activity for Task 4.1. Some initial experiments have been completed and are described in the next section.

3.3.1. Initial Experimental Protocols

The experiments performed so far have been focused mainly on the network throughput metric. The Internet Engineering Task Force (IETF) developed RFC2544 [15] which outlines a benchmarking methodology for network Interconnect Devices. The methodology defines performance metrics such as latency, frame loss percentage, and maximum data throughput.

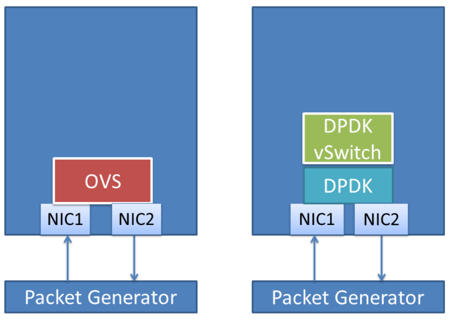
Using the RFC as a basis, throughput was measured in millions of frames per second where the frame size refers to Ethernet frames ranging from smallest frames of 64 bytes to largest frames of 1518 bytes. For 64-byte frames, a line rate of 10 Gbps translates to 14.88 million packets per second for unidirectional traffic.

The Device under Test (DUT) has 2 NICs, both connected to a packet generator: one NIC receives the packets, whereas the other NIC is used to send back the traffic to the packet generator that measures the throughput. The packet generator selected for the experiments was DPDK Pktgen [16] which is an open source version of the Linux Foundation Pktgen based on Intel's DPDK library. It was selected due to its free availability and its ability to send packets at 10Gbps line rate speeds. It is possible to physically assign one or more CPU cores directly to the sending and receiving processes over the NICs. In the current configuration, one core was assigned to the processor that generates the packets and one core is assigned to each transmission queue for transmitting packets onto the network. A new feature introduced in the latest release is the capability to run more than one instance on the same host which can be exploited in the creation of different packet flows.

To maximise the efficiency of the packet generator, a Command Line Interface (CLI) is available to set and start the transmission of the network traffic. Moreover, it is possible to setup scripts using the LUA programming language [17] to automate the packet generation process, defining traffic profiles and the behaviour of the packet generator. Exploiting this feature for the purpose of this experiment, a LUA script has been implemented, following the RFC 2544 recommendations, with different packets sizes, automating the test for the various configurations under test.

3.3.1.1. Open vSwitch vs. DPDK vSwitch Throughput

Testing has been focused initially on comparing the throughput of both the Open vSwitch and the DPDK vSwitch technologies in two different scenarios, respectively shown in Figures 3.5 (a) and (b).



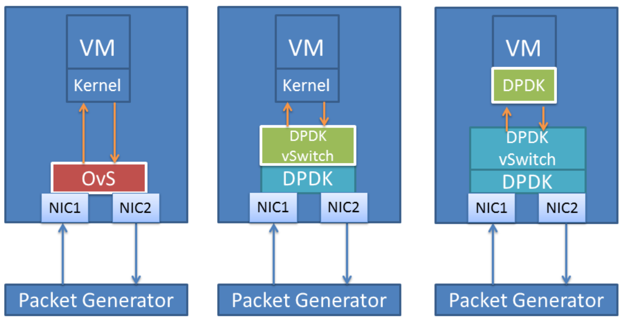


Figure 3.5 (a) First testing scenario **Figure 3.6 (b) Second testing scenario**

In the first scenario a physical-port-to-physical-port communication was implemented by the vSwitch, which basically forwards the traffic received through NIC1 onto NIC 2. The results obtained are shown in Figure 3.6.

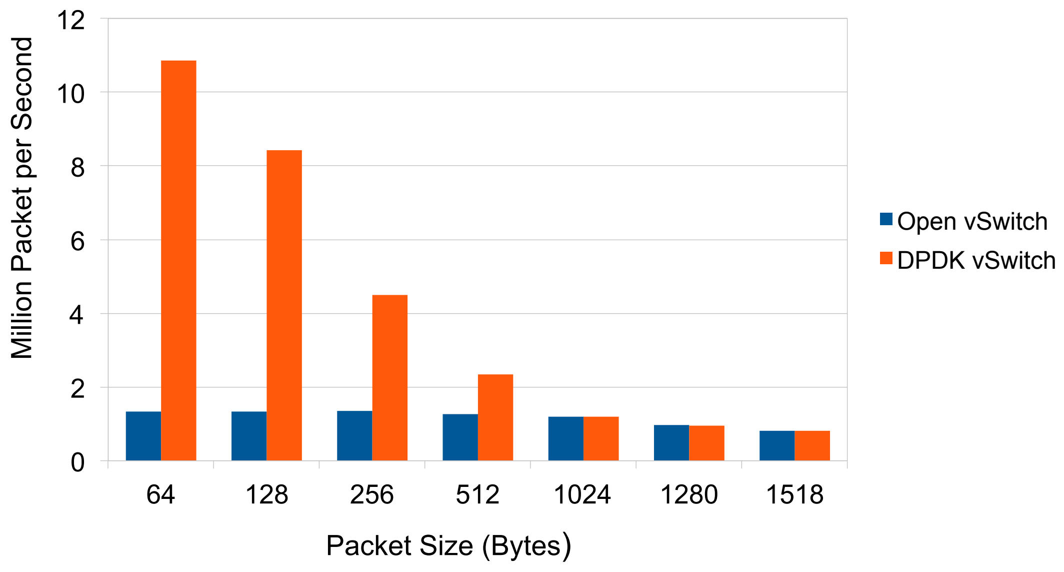


Figure 3.7 Throughput comparison for the first testing scenario

The results clearly show that DPDK vSwitch provides significantly better packet switching performance with respect Open vSwitch. In order to have a complete comparison, the results of the second scenario (as shown in Figure 3.5(b)) are shown in Figure 3.7.

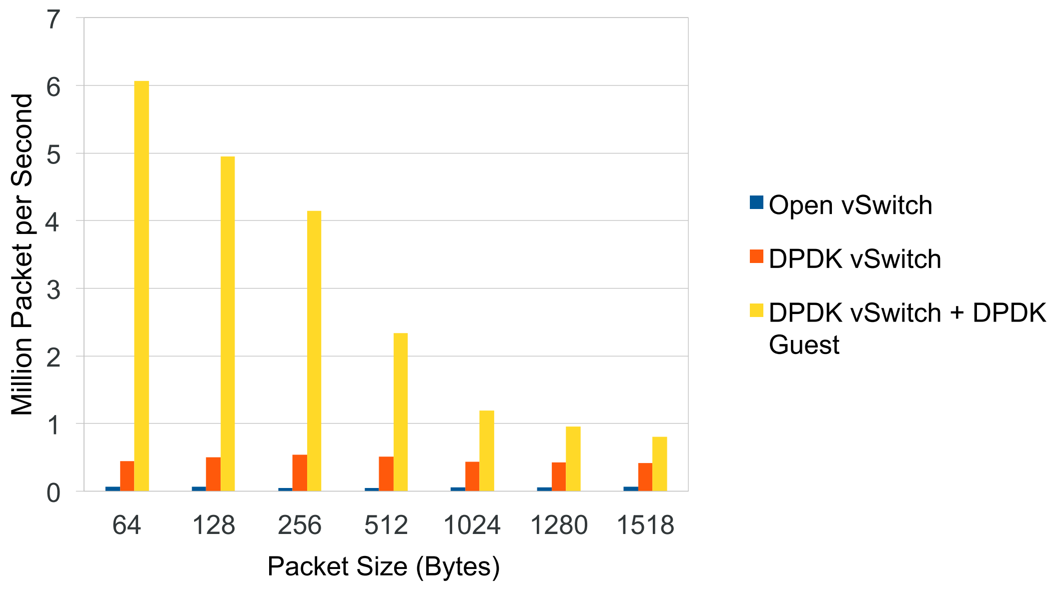


Figure 3.8 Throughput comparison for second test scenario

From these results it is clear that the major bottleneck at the infrastructure layer is related to the communication between the VM and the physical host. This is the main reason why developers are encouraged to use the DPDK library within the VM in order to achieve maximum throughput. Using DPDK in virtual switching technology (see Figure 3.7) provided superior performance in comparison the current mainstream version of OVS. As previously discussed, these results will most likely change, due to Intel's renewed focus on integrating DPDK capabilities into the upcoming OvS 2.4 mainstream code release.

The remaining experimental results described in this section referred to the second configuration of scenario two (using DPDK vSwitch and the Linux kernel packet processing within the VM), since at the VIM level, there is no knowledge respect the specific VNF a VM is hosting.

3.3.1.2. Processor Pinning influence on Throughput

Processor pinning (or core pinning) enables the binding of a process to specific cores within the CPUs in a manner that the process runs only on the specified core(s).

A number of tests have been performed to identify the impact of processor pinning on vSwitch and VM performance.

With DPDK vSwitch, it is mandatory to allocate cores that are dedicated to switching operations. The minimum number of cores which can be assigned to DPDK vSwitch is four. However, in order to tune the performance of the vSwitch, the impact of increasing the number of allocated cores was investigated.

In Figure 3.8 throughput versus the number of allocated cores is shown. The results indicated that the optimal assign of cores to vSwitch is four. In fact, increasing the number of cores beyond four did not provide any measurable increase performance in most cases. The data clearly indicates that the allocation of additional cores beyond the default configuration of four can be considered as a waste of physical resources that could otherwise be allocated to VMs.

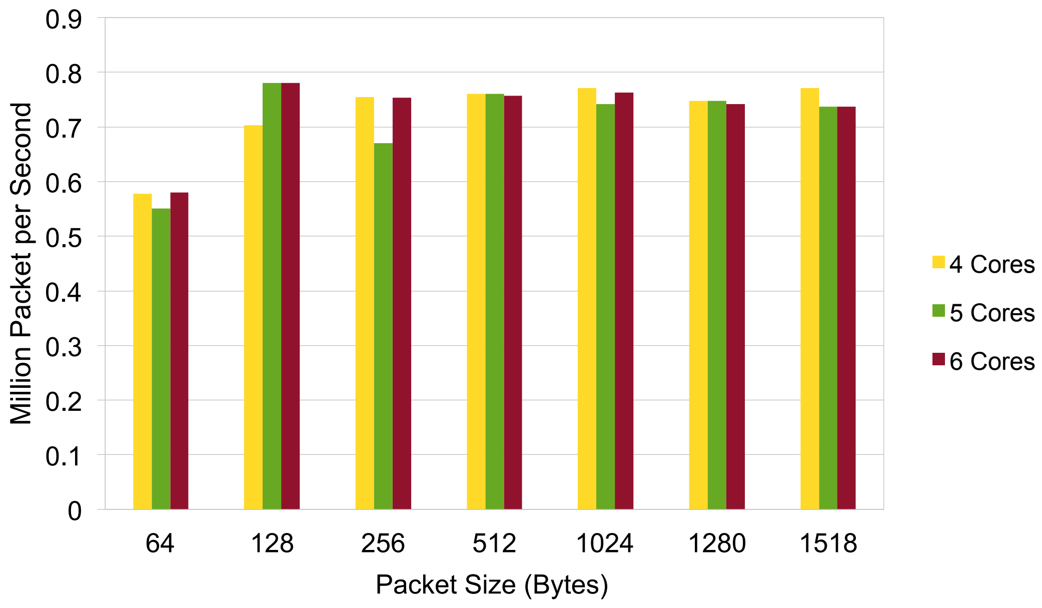


Figure 3.9 Throughput varying the core pinning configuration for DPDK vSwitch.

Similar experiments have been conducted for a VM, in order to analyse the extent in which a VNF can be influenced by processor pinning. Results are shown in Figure 3.9. Four cores were statically assigned to the vSwitch for this experiment.

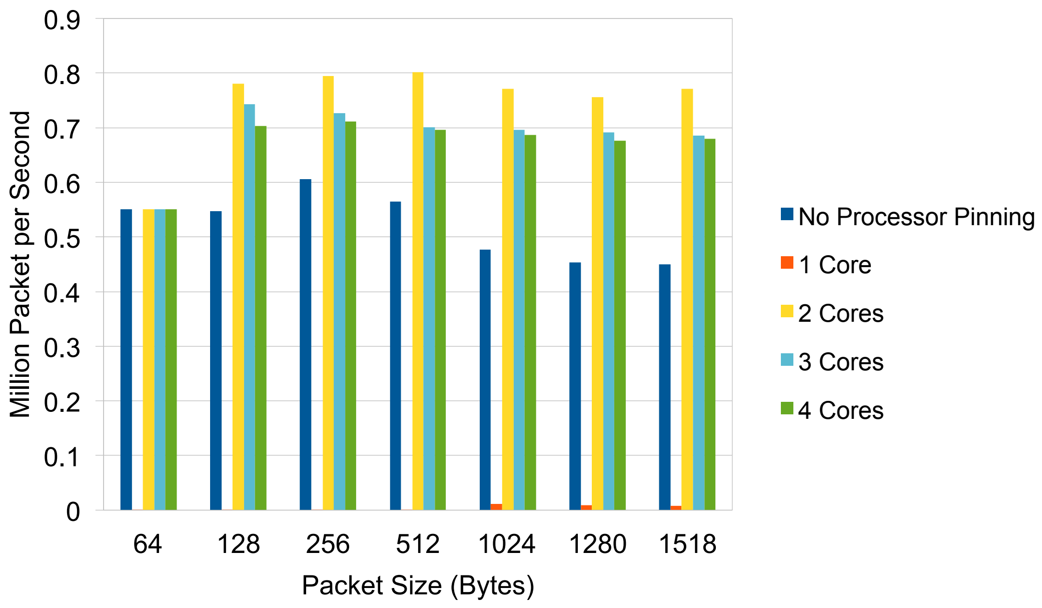


Figure 3.10 Throughput varying the core pinning configuration for the VM.

The results show that the usage of processor pinning can help to achieve improved performance, if properly configured. In this experimental configuration a VNFC, using just one core for processing incoming network traffic was found to be insufficient to manage the processing overhead required by the VM. Assigning two cores (yellow bars) it is possible to achieve improved performance in comparison to a non-pinned configuration (blue bars). These experiments to date are based on a single VM deployment onto a server with two processors with 10 cores each with the assigned cores on CPU1.

3.3.1.3. Influence of NUMA Awareness on Throughput

Non-Uniform Memory Access (NUMA) is a computer memory access design used in multiprocessing, where the memory access time depends on the memory location relative to the processor. In multiprocessor systems the processors can be grouped together with their own memory and possibly their own I/O channels. Each group of CPUs and memory is called a NUMA node. Each CPU can also access memory associated with another NUMA node in a coherent way however this is slower and less efficient in comparison to accessing local memory. The allocation of CPUs and hardware resources to a NUMA node is a hardware vendor specific implementation.

The test system utilised for the investigated NUMA configurations had two NUMA nodes, containing one CPU per each. The NIC used for the experiment (Intel Ethernet Controller X540-T2) was installed on a PCI slot belonging to the first NUMA node.

During the experiments, the throughput has been measured for two different configurations: in the first one the VM is pinned on two cores belonging to the first NUMA node (CPU1), whereas in the second the VM is pinned on two cores belonging to the second NUMA node (CPU2). The results obtained are shown in Figure 3.10.

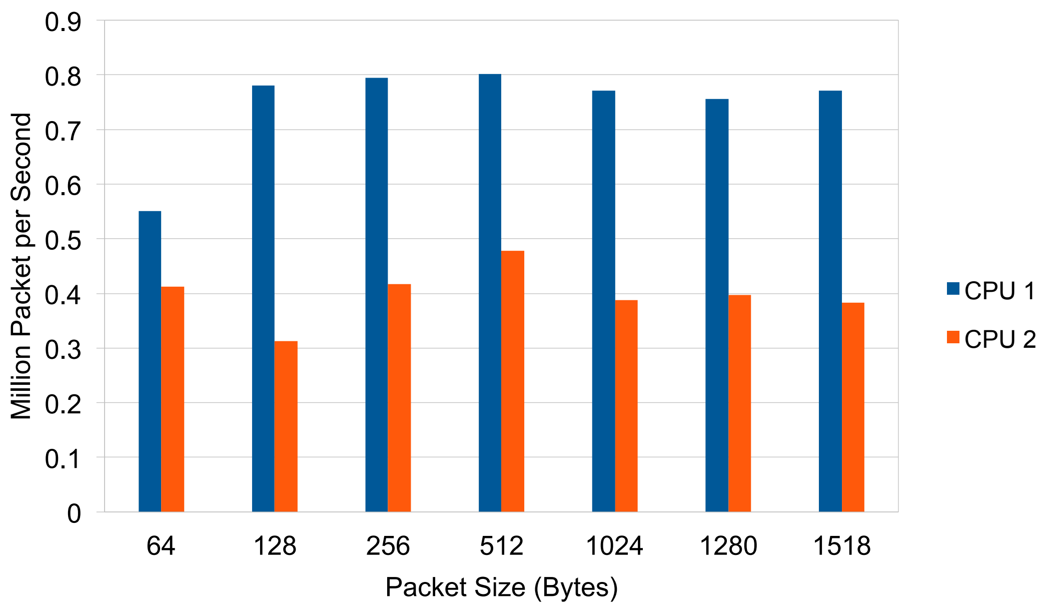


Figure 3.11 The effect of core pinning configuration on throughput

The performance obtained in the second scenario was approximately 50% less than in scenario one, indicating that NUMA awareness can have a significant influence on system performance and should be considered appropriately at the Orchestration layer and within the VIM functional entity of the IVM.

3.3.1.4. Hugepages Size Throughput

When executing instructions in an x86 architecture both the CPU and OS mark the RAM as being used by a process. For efficiency, the CPU usually allocates RAM in 4K blocks (default value for Linux) named pages. Since these pages can be swapped to the disk, the memory addresses are virtual and the operating system has to keep track of which page belongs to which process and where they are stored on the disk. As the number of pages increases, more time is taken to find where the memory has been mapped too. Newer CPU architectures and operating systems support bigger pages (so less time spent on look-ups as is the number of pages required). This feature is called Hugepages. Since the usage of DPDK vSwitch requires the Hugepages, some tests have been performed by changing the size of the pages from 2Mbytes to 1Gbyte. The results obtained are shown in Figure 3.11.

Initial results obtained indicated there is no significant advantage in terms of throughput by increasing the Hugepage size. Additional tests will be carried in order to determine if the size of the Hugepage has an influence on other systems metrics such as latency.

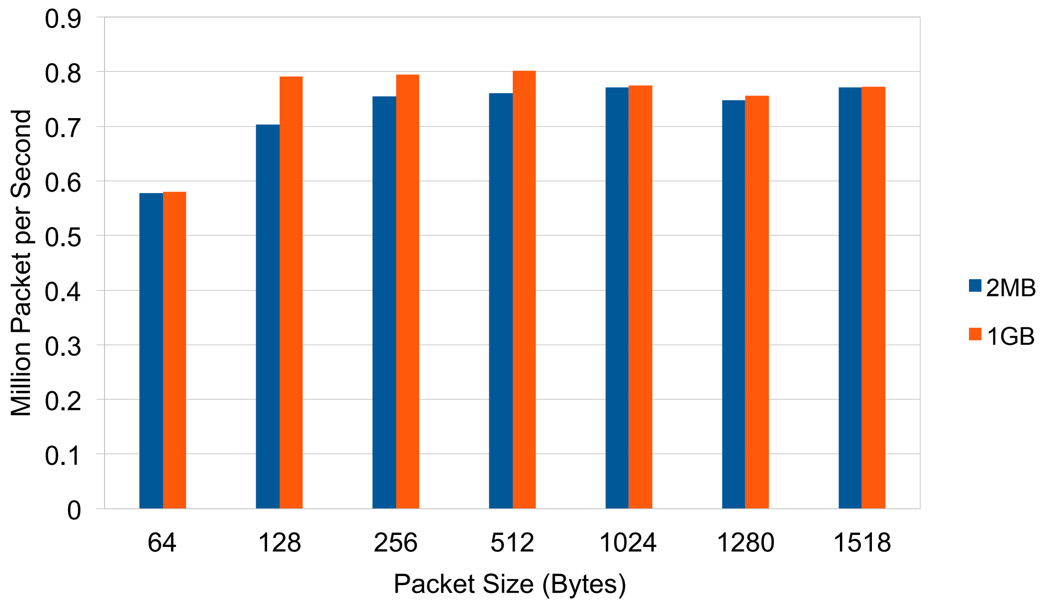


Figure 3.12 Packet throughput performance with huge page sizes of 2MB and 1GB

3.3.2. Planned Experimental Protocols

Starting with the parameters listed in Table 3.2, a set of scenarios are currently in development and will be used to plan a set of experimental protocols to determine if the parameters identified have a quantifiable impact on performance. For example, one representative scenario involves the use of DPDK vSwitch to process the traffic between VMs at a software level as well as the exploitation of hardware features, such as VT-d and SR-IOV technologies, to reduce the overhead due to the VMM. Live migration is another scenario of interest, where the bandwidth dedicated to the information exchange between the source and the destination of a live migration could affect both migration delay and down time. The test-bed defined in Section 3.2

will be used to execute the protocols based on different system configurations, network technologies etc.

Networking experiments may also involve the creation of simple service chains, each of them intended as a chain of VMs traversed by the same traffic flow. Since the analysis of the VNF application per se is out of the scope of the VIM (in T-NOVA this is delegated to the VNF developer) the goal here is to analyse the impact of the underlying technologies and, for this purpose, the simplest known VNF can be used: the L2/L3 Linux kernel stack.

Moreover, exploration of the different storage technologies available that can be used within the T-NOVA IVM will be necessary, with a focus on configuration parameters. An initial high level classification of different approaches/technologies with respect to specific scenarios has been performed. Due to the high level of potential customisation, not all the options will be considered in the experimental plan. The focus is explicitly on those that are considered to most applicable to VNFs.

The following scenarios and options have been identified:

Scenario 1. Boot volumes of VMs can be located on:

a. Local disks of the Nova Compute Node
b. Shared disks (of Nova Compute Nodes) residing on a SAN or an IP based storage array; in this case a clustered file system is necessary in order, for example, to control SCSI reservations when multiple Compute Nodes access the shared volume
c. Disks presented by a Cinder Block Storage Node, disks that, in turn, can be:
    i. Local disks of the Cinder Node
    ii. Shared disks (of the Cinder Node) residing on a SAN based or IP based storage array; Cinder Volume Agent would be moved onto the Controller Node and will manage the SAN or IP based storage array (in case of where disks local to the Cinder Node are used, the Cinder Volume Agent would run on the Cinder Node itself).

Scenario 2. Additional (non-boot) volumes of VMs can be located on:

a. Local disks of a Cinder Storage Node
b. Shared disks of a Cinder Node residing on a SAN or IP based storage array; as highlighted above it would be the Controller Node which manages the SAN or IP based storage array

Scenario 3. Volumes containing the VMs master images should be located on the Object Storage: local disks of a Swift Object Storage Node; Swift has been designed to be a multi-master replicated Object Storage, so creation or modification of an object on a Swift Node is immediately replicated to the other Swift Nodes.

As highlighted by the list above, more than one configuration can be used for each scenario and technology (with the exception of the Swift Object Storage). Any configuration listed above could influence in diverse way behaviour and performance of the cloud environment, impacting on some of the metrics listed at the beginning of this section.

3.4. Conclusions and Future Work

Task 4.1 is focused on the identification, characterisation and optimisation of the hardware and software components that will be used in the implementation of the T-NOVA IVM. The initial selection of candidate technologies has been completed and presented in Section 7. These candidate technologies have been utilised in the design and implementation of an IVM testbed. Initial technology characterisation experiments have been conducted, which can be used to improve packet processing performance. An extensive experimental protocol which will focus on workload and technology characterisation is being developed. The output of this work will help to identify dynamic and static metrics that are most highly correlated with workload or system performance. Task 4.1 is also developing a set of Best Known Methods (BKMs) for virtualised environment implementation for the performant deployment and management of VNFs. These methods will be reported in Deliverable 4.1 and will be used by a number of other tasks in the development of subsystem components for the T-NOVA system.

4. SDN Control Plane

The SDN Control Plane plays a key role in the T-NOVA system with responsibility in the southbound direction for the configuration, management and monitoring of the SDN-compatible network entities. Northbound it is responsible for delivering enhanced network connectivity services to the Orchestrator and management systems.

SDN has the potential to deliver benefits to NFV applications with a scalable, elastic and on-demand network infrastructure, leveraging the programmability of southbound network elements. However such elements, both physical and virtualised, need to be properly configured to address the applications' requirements. This challenging task is the main objective of the SDN Control Plane.

In this regard, Task 4.2 proposes to design and develop an enhanced SDN controller for network services provisioning to support NFV applications. The activities within the task have been split into following focus areas:

- **Programmatic Network Control**: Dynamic and intelligent control of network resources, thus enabling responsiveness to variable conditions, such as user behaviour dynamics, application lifecycle, network performance, monitoring events (e.g. congestion, network malfunction), as well as flexible establishment of service function chaining.
- **Network Virtualisation**: Deals with the deployment of virtual networks (vNets) supporting QoS and overlay encapsulation, through analysis of frameworks (i.e. Open vSwitch), protocols (i.e. OpenFlow) and tunnelling solutions (i.e. NVGRE, VxLAN). The key output of this activity is to provide an open, flexible and extensible interface for the instantiation, configuration and monitoring of isolated virtual networks.
- **Control Plane Virtualisation**: Refers to the virtualisation of the network controller to ensure reliability and high availability in large-scale scenarios. For these purposes, cloud computing capabilities combined with distributed clustered approaches are being investigated in order to ensure elasticity, auto-scaling and load balancing of the SDN control plane.

In Task 4.2, work has initially focused on determining the SDN platform that most appropriately addresses the T-NOVA requirements. This selection was carried out after a thorough analysis of the SDN controller implementations currently available, the features they offer, the mechanisms they support for enhanced network services (i.e. slicing, chaining, QoS), the way they approach the distribution of the control workload. The information presented in this section reports the progress on these activities.

4.1. Key Requirements

The first step involved the identification of key requirements affecting the network controller procedures and mechanisms. Table 4.1 provides a summary of the high-level requirements identified in T-NOVA concerning the SDN Control Plane. A full list of requirements have been collected and documented in D2.31.

Requirement	Description
Network connectivity and isolation	Applications and services must be connected to isolated networks, ensuring that the processing of packets on each network is independent of all the others.
Resource Monitoring	The provision of monitoring information should make management and orchestration entities aware of the status and performance of the network infrastructure
QoS support	Applications and services may have specific performance needs, requiring mechanisms for QoS provisioning over the network infrastructure.
Performance	In large-scale scenarios where many nodes need to be controlled, the control plane may suffer slower performance in terms of processed requests per second/average response time. Therefore, mechanisms to limit this issue should be provided.
Scalability	The control plane should adapt to a variety of applications and scale according to their network load. This means that in some cases a distributed control plane may be required; therefore the T-NOVA control plane must be able to accommodate this requirement.
Robustness/Fault tolerance	The controller itself might fail and therefore leave the network inoperable. Through redundancy mechanisms, it must be guaranteed that the controller does not represent a single point of failure.
Service chaining support	The network controller must be able to dynamically enforce and modify the chaining of network service functions.
Inter-datacentre connectivity	The solution adopted for the control plane should be able to support inter-datacenter (inter-DC) connectivity, as in many practical cases this will be required due to the physical dispersion of resources.

Table 4.1 Requirements mapping for Task 4.2

4.2. Generic Architecture of the SDN Control Plane

Within Task 4.2, a key activity was the design of the preliminary architecture for the network controller. The components, modules and interfaces of the T-NOVA SDN Control Plane were identified.

Figure 4.1 shows the SDN control plane functional architecture; it has been defined starting from an idealised SDN framework model which has been selected as an initial reference point [18]. It has then been extended and properly adapted to fulfil the requirements previously described.

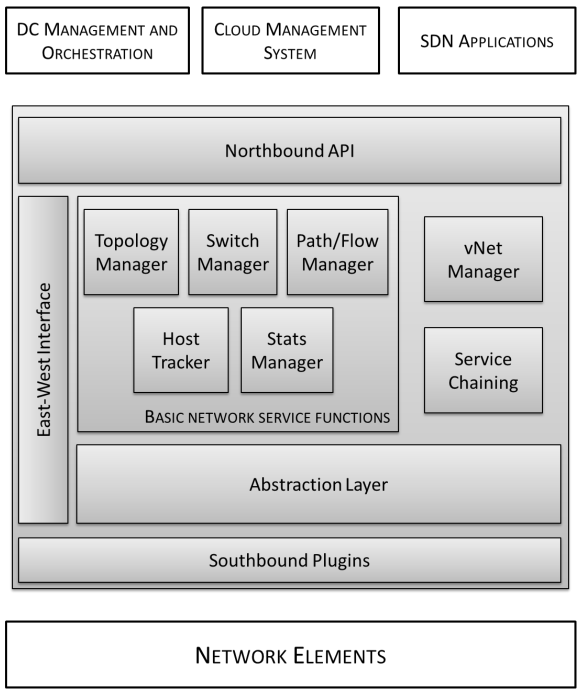


Figure 4.1 T-NOVA SDN Control Plane Architecture

4.2.1. Functional components

Table 4.2 outlines the main functional components that have been identified with a brief description of their role within the network controller.

Component	Functionalities
Topology Manager	The Topology Manager learns and manages topology information specific to devices and their reachability. Information gathered about the networks' elements is essential to discovering the topology.
Network Element Manager	The Network Element Manager stores, manages and provides the details (e.g. switch id, SW version, capabilities, etc.) of the network nodes as they are discovered.
Path/Flow Manager	This module provides the flow programming services including forwarding rule installation and removal of data paths configurations. Typically used when high-level policies specified by the northbound are translated into flows by a service module (Service Chaining, Slice Manager) that in turn communicates with this module to proactively push the flows down to network elements. Path reconfiguration (after network failures or VM migration) and QoS support are in charge of this module.

Component	Functionalities
Host Tracker	The host tracker module learns, statically or dynamically about VM hosts in the network. It stores and provides host information, such Host's IP address, MAC address, switch ID, port, and VLAN. Moreover it periodically refreshes the hosts' data to track the element location (switch, port, MAC, or VLAN), and notifies the listening applications of host related events.
Stats Manager	This module stores and provides network statistics data with different levels of data granularity (flow, port and table statistics).
vNet Manager	This functional module allows the creation of multiple, isolated, virtual tenant networks on top of a single physical network, in order to enable complete separation between the logical and physical plane, hiding the complexity of the underlying network and optimising network resources usage.
Service Chaining	This functional module supports applying service chains as ordered graph of network services (e.g. firewalls, load balancers) by configuring accordingly traffic steering.

Table 4.2 SDN Control Plane Functional Components

4.2.1.1. Distributed Control Plane

In order to address performance, scalability and fault-tolerance requirements, a distributed approach for the deployment of the SDN controller platform is under investigation.

In this regard, following the concept of a "distributed, but logically centralised" controller [19] [20], the SDN control plane in T-NOVA proposes the virtualisation of the network controller through multiple instances organised in clusters, while keeping the benefits of centralised network control. The core concept is to exploit cloud-computing capabilities to virtualise each instance of a controller on dedicated virtual machines, allowing dynamic deployment and enabling the distribution of the network control workload across the cluster.

To support deployments in a distributed scenario, new functional components are required. These additional components, shown in Figure 4.2, have the goal of extending the architecture defined before, where each CP block corresponds to a single instance of the control plane.

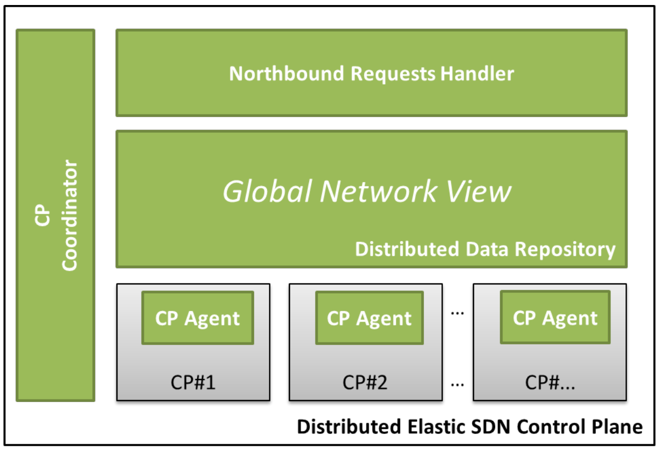

CP Coordinator
Northbound Requests Handler
Global Network View
Distributed Data Repository
CP Agent
CP#1
CP Agent
CP#2
...
CP Agent
CP#...
Distributed Elastic SDN Control Plane

Figure 4.2 Distributed SDN Control Plane Architecture

In Table 4.3 a brief description of all the Distributed SDN Control Plane components is provided.

Component	Functionalities
Distributed Data Repository	This component is responsible for consistently maintaining a global view of the network across the control plane instances belonging to the cluster. The information collected is useful maintaining a global view of the topology and the state of the network, including switch, port, link, and host status. Northbound applications/internal CP components can take advantage of the global network view in making forwarding and policy decisions, which are in turn stored into the network view. Mechanisms for the distribution of the network state among the CP instances require evaluation and analysis that will be explored later in the project.
Northbound Request Handler	Mainly responsible for distributing northbound requests among the available controller instances. It is essential to make the network control plane accessible by the northbound API as a unique single instance.
CP Coordinator	The CP Coordinator supervises and coordinates the operation in the cluster. Specifically it has to: • Properly instruct the Northbound Requests Handler in spreading the northbound requests. • Dynamically configure the controller-to-switch connections by assigning each switch to one or more controller instances. • Decide whether to add or remove a controller instance to the cluster depending on the network needs • Monitor the status of the cluster The role of coordinator may be carried out by one of the CP instances available in the cluster, by means of a procedure of leader election.

Component	Functionalities
CP Agent	The CP Agent is in charge of collecting information about the current resource utilisation (CPU load, memory usage, control messages arrival rate, etc.) at each CP instance and enforcing the switch-to-controller instance connection rules as established by the Coordinator. These rules are used by each switch to identify the controller instance/s to which the southbound requests must be forwarded.

Table 4.3 Distributed SDN Control Plane Components

4.2.2. Interfaces

The following is a description of main functionalities supported at the control plane interfaces divided into Northbound, Southbound and West-Eastbound. The detailed specification of these interfaces is currently ongoing; therefore the following is a provisional description of the roles and high-level functions.

- **Northbound Interface:**
  This interface identifies the application programming interface (API), often RESTful, serving the higher systems such as orchestrators or cloud managers or applications to gather network intelligence, run algorithms and manage network resources.
- **Southbound Interface:**
  This interface supports the exchange of control information between the physical and virtual switches and the SDN controller platform. The southbound interface is capable of supporting multiple protocols, proprietary and standards based, can be used for flow programmability (e.g. OpenFlow 1.0/1.3, BGP-LS, etc.) and device configuration (e.g. OVSDB, SNMP, NETCONF) of the data plane entities.
- **Westbound Interface:**
  This interface is required to address the scalability and high-availability requirements imposed by DC environments. It supports the control of large-scale DCs by enabling the interconnection of multiple SDN controllers, organised in clusters.
- **Eastbound interface**
  This interface is used for the communication with control planes of non-SDN domains (i.e. MPLS) and it is also responsible for managing inter-DC connectivity.

4.3. Candidate Solutions

Given the rising number of SDN solutions available an important task concerns the selection of appropriate technologies for the development of the functional components described previously. Therefore, a detailed survey among existing solutions was carried out with the goal of selecting the best technologies as the initial starting point for the implementation of the T-NOVA SDN Control Plane. Specifically, the following topics were addressed:

- SDN controllers
- Distributed controller approaches
- Network tunnelling protocols
- Network virtualisation frameworks

Sections below present the complete list of candidate technologies and the rationale behind the selection.

4.3.1. SDN Controller

4.3.1.1. OpenDaylight

Table 4.4 presents a list of the currently available SDN controllers.

Name	Descriptions
NOX	Open-source controller developed by Nicira Networks, implemented in C++ and Python. It offers support for the OpenFlow v1.0 protocol. NOX is not being actively developed at this time.
POX	Open-source controller developed by Nicira Networks, implemented in Python. It offers support for the OpenFlow v1.0 protocol. POX is not being actively developed at this time.
Maestro	Open-source controller developed by Rice University, implemented in Java. It offers support for the OpenFlow v1.0 protocol. Maestro is not actively developed at this time.
Beacon	Open-source controller developed by Stanford University, implemented in Java. It offers support for the OpenFlow v1.0 protocol. Beacon is not being actively developed at this time.
Floodlight	Open-source controller developed by Big Switch Networks, implemented in Java. It offers support for the OpenFlow v1.0 protocol and a Quantum plug-in for OpenStack support. Floodlight is not being actively developed at this time.
ONOS	Open-source SDN controller platform developed by ON.LAB. ONOS is being actively developed. The first public release was made available on the 5th of December.
Ryu	Open-source controller developed by NTT, implemented in Python. It offers support for the OpenFlow v1.0, OpenFlow v1.2, OpenFlow v1.3 and OpenFlow v1.4 protocols, as well as OpenStack support. Ryu is being actively developed at this time.
Nodeflow	Open-source controller developed by CISCO, implemented in Javascript. Nodeflow is not being actively developed at this time.
Trema	Open-source controller developed by NEC, implemented in C and Ruby. It offers support for the OpenFlow v1.0, OpenFlow v1.2 and OpenFlow v1.3.X protocol and a Quantum plug-in for OpenStack support. Trema is being actively developed at this time

Name	Descriptions
OpenIRIS	Open-source controller developed by ETRI, implemented in Java. It offers support for the OpenFlow v1.0.1 to v1.3.2. OpenIRIS is being actively developed at this time.
MUL	Open-source controller developed by Kulcloud, implemented in C. It offers support for the OpenFlow v1.0, OpenFlow v1.3 and OpenFlow v1.4. MUL is being actively developed at this time.
Jaxon	Open-source controller based on NOX and implemented in Java. It is not being actively developed at this time.
OpenContrail	Open-source SDN platform developed by Juniper Networks. The OpenContrail Controller, which is part of the platform, is implemented in Python, while the projects comprising OpenContrail are implemented in various programming languages (Python, C++ and JavaScript). It offers OpenStack support but the current version lacks of OpenFlow support. OpenContrail is being actively developed at this time.

Table 4.4 Alterative SDN Controller Considered

The first criterion to be applied in the SDN controller selection process was to consider only those still under active development. The next step was to take into account the consortium's hands-on experience with the various controller options. Ryu and OpenDaylight were the solutions that emerged based on this criterion.

The final choice between these two candidates was the OpenDaylight platform due to the high level of community support, strong roadmap, growing maturity and its numerous features, as outlined in Sections 3.1.3.3 and 7.0.

4.3.2. Distributed Control Plane

4.3.2.1. ODL Clustering Service

OpenDaylight supports a cluster based High Availability (HA) model where several instances of ODL controller act as a single logical controller. The global state of the network is maintained through a distributed data store.

The Clustering Service Provider module is responsible for providing the clustering services to all the functional components of the controller as well as to applications on the northbound side of the controller. From the northbound side the cluster is accessible via a REST API; a request can land on any controller in the cluster. On the southbound side (specifically OpenFlow), switches need to be explicitly connected to the controllers in the cluster via their IP address.

The Connection Manager is the module responsible for managing connection information between the ODL instances and the OF switches. For the time being, the connection schemes supported are: *SINGLE_CONTROLLER* (all the switches connected to only one controller) and *ANY_CONTROLLER_ONE_MASTER* (any switch connected to any controller, with only one master). Other schemes (i.e. *ROUND_ROBIN* and *LOAD_BALANCED*) have been defined but have not yet been implemented.

In this regard, the T-NOVA SDN Control Plane proposes to extend the Clustering Service offered by ODL with an algorithm implementing the *LOAD_BALANCED* connection scheme. This form of algorithm will be in charge of determining when to add/remove controllers to/from the cluster and to dynamically balance the switch-to-controller connections according to the current controllers' load estimation. Specifically it has to assign each switch to a subset of controllers, of which one will be the master, in order to reduce the control traffic load while maintaining resiliency. The dynamic switch assignment will be achieved through a migration protocol to minimise packet loss or duplication.

Name	Description
Pratyaastha	It distributes the SDN control plane application state (consisting of network flows related to a VNFs or vNets) as a variant of a multi-dimensional bin-packing problem. The goal is to reduce the operating cost of controllers and reduce flow setup latencies, in comparison to other approaches that use static assignments of SDN switches to controllers and make use of distributed data stores for state sharing. Application states and controller load may change dynamically, and if needed states and switches may need to be reassigned as well as new controllers added or removed as per the load. When switch reassignment is required, Pratyaastha proposes to use the switch migration protocol of ElastiCon.
ElastiCon	ElastiCon is a distributed controller architecture in which the controller pool is dynamically grown or shrunk depending on the traffic conditions. The SDN switch mapping to a controller is not static and an OpenFlow compliant switch handover protocol has been created to support dynamic reallocation of a switch to a different controller. The switch migration algorithm ensures the aliveness and safety of operations. ElastiCon exploits the features introduced in OpenFlow 1.3 where a controller can be configured as master, equal, or slave. The controller can register its role with the switch it is managing. A switch can connect to multiple controllers, but only one of them acts as the master controller. The ElastiCon distributed controller design consists of clusters of autonomous controller nodes that coordinate among themselves to provide consistent control logic. ElastiCon also uses a distributed data store that helps the cluster of controllers to coordinate itself and thus to show a behaviour similar to a logically centralised controller.
ONOS	ONOS is an open source distributed network operating system initially developed by researchers at Stanford University. The core architecture is implemented by using several other open source components including Zookeeper (for distributed coordination), Cassandra (in memory DHT), Titan (graph DB), to name a few. The project uses Floodlight as the SDN controller that receives configurations from control applications running on top of ONOS and sets up the necessary flows between forwarding devices using OpenFlow. The architecture allows for multiple SDN controllers to

	manage parts of network components exclusively (sharding) and algorithms to minimise message flows between network silos. High availability with failover is enabled through a distributed registry (Zookeeper). The network topology is learned by monitoring OpenFlow events (such as PKTIN) and eventually achieves consistency. ONOS reduces the complexity of the network by segregating network topology maintenance and the path computation process. At the time of writing - dynamic clustering in ONOS was not recommended and the static clustering mechanism was suggested

Table 4.5 Alternative Distributed Control Plane Technologies

4.3.3. Network Virtualisation

Network virtualisation deals with the decoupling of the hardware elements that form a physical network, from the logical networks operating over it to enable application or tenant isolation. Network virtualisation is accomplished by means of frameworks and technologies providing network abstraction over the physical environment. In the following sections, existing solutions designed for this scope are presented.

4.3.3.1. OpenDaylight VTN

OpenDaylight Virtual Tenant Network (ODL VTN) is a framework that provides multi-tenant virtual network on an SDN controller. As such it implements a logical abstraction plane that enables the complete separation of the logical plane from physical plane. Multiple logical networks can be applied on the same physical infrastructure, while at the same time remaining completely segregated from each other. Networks for applications and end user needs can be deployed without knowledge of the underlying physical network topology. ODL VTN networks are constructed by mean of the following objects: virtual bridge (vBridge), virtual router (vRouter), virtual interface (vInterface) and virtual link (vLink). Figure 4.3 shows a simple logical network.

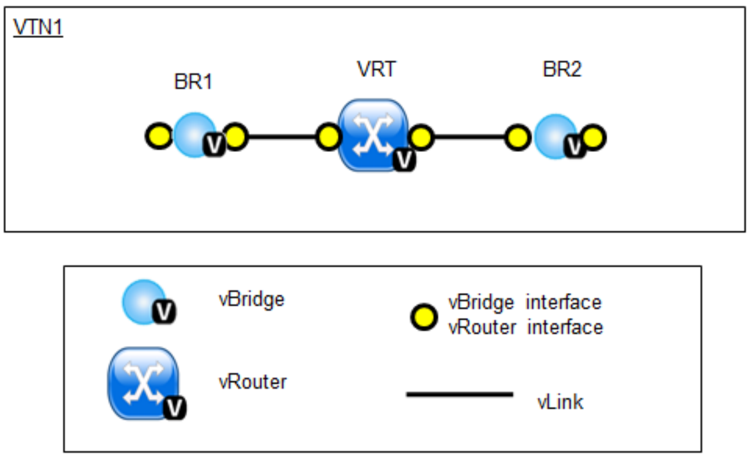


Figure 4.3 Simple VTN network

In a more complicated real world scenario, the following objects are also provided: virtual tunnel (vTunnel), virtual Tunnel End Point (vTEP), virtual connectivity between controlled networks (vBypass). Figure 4.4 shows this more complex logical network.

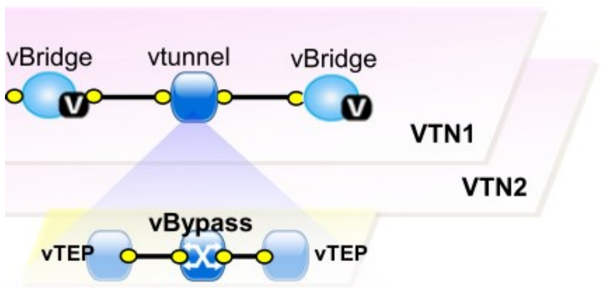


Figure 4.4 VTN tunnels

Another function in ODL VTN is the Flow Filter that can be comparable in functionality Access Control Lists (ACLs). In other words communication can be allowed, prohibited or redirected based upon particular conditions that a packet can meet.

With ODL VTN virtual networks can also be configured across multiple SDN controllers. This feature applies usually in a multi-DC environment. In this case it is still possible to apply a single Flow Filter policy to VTN virtual objects distributed over different DCs.

ODL VTN main components are the VTN Coordinator and VTN Manager that, as shown in Figure 4.5, communicates via Web APIs, implemented by REST protocol. ODL VTN uses three different methods of mapping frames to virtual networks: port, VLAN and MAC mapping.

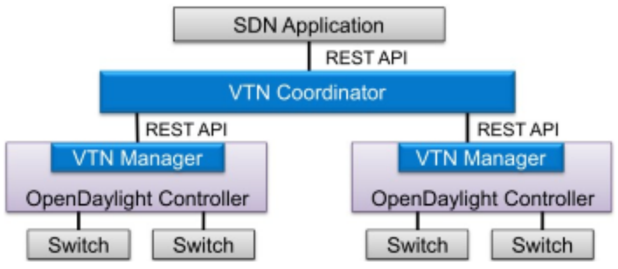


Figure 4.5 VTN high-level architecture

4.3.3.2. OpenDOVE

Open DOVE is an overlay network virtualisation platform for the DC. It runs over any IP network and provides logically isolated multi-tenant networks with layer2 or layer3 connectivity. The OpenDove components are:

- oDMC (OpenDove Management Controller): acts as the control/access point into the OpenDove management plane
- oDCS (OpenDove Connectivity Server): acts as the control point for the OpenDove control plane

- oDGW (OpenDove Gateway): provides gateway functionality between overlay and the outside world, thus enabling OpenDove to provide connectivity to external networks
- OpenDove OVS Agent: provides the interface between OVS and OpenDove control plane.

The relationship between these components is shown in Figure 4.6

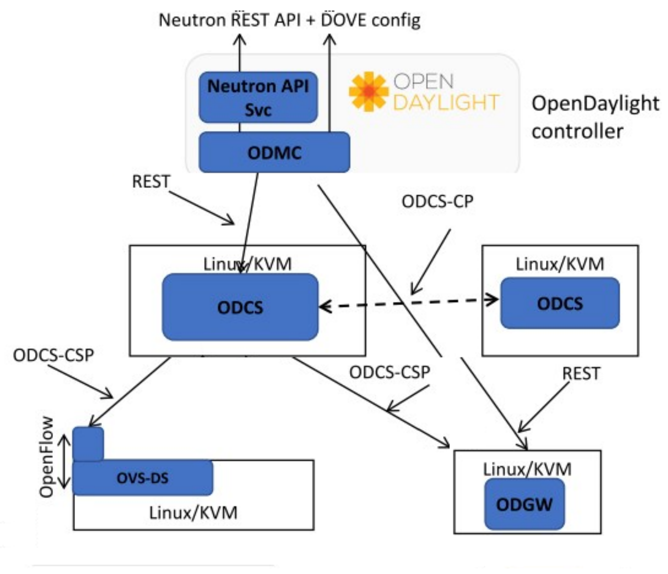


Figure 4.6 OpenDove components

The main OpenDove features are: control plane implementation with address, policy and mobility management; management interfaces for programmatic configuration, including OpenStack enablement; open data plane implementation for Linux/KVM and VxLAN encapsulation; software gateway for connecting to non-virtualised network and external hosts. OpenDove virtual switches are implemented on Open vSwitch. It leverages OVS native encapsulation/tunnelling support (VxLAN frame format). OpenDOVE has not been submitted to the incubated projects list for the Helium version of ODL.

4.4. Implementation Choices

A mapping of the T-NOVA SDN controller components and the OpenDaylight modules is being carried out. Table 4.7 summarises the initial analysis aimed at identifying the extensions required to be implemented in ODL to address the gaps in the SDN control plane functionalities requested by the T-NOVA system.

Functionality in T-NOVA	Functional Module in OpenDaylight	Extensions to be implemented
Path/Flow Manager	Forwarding Rule Manager / Flow Programmer	QoS Support Path reconfiguration
vNet Manager	VTN Coordinator VTN Manager	Support for live migration of VM

Functionality in T-NOVA	Functional Module in OpenDaylight	Extensions to be implemented
Traffic Steering	VTN Coordinator VTN Manger	Support for service chaining through the enforcement of traffic steering policies.
High Availability/Clustering	Clustering Service	Support for dynamic deployment of control plane instances.
Northbound request handler	None	Solutions to be investigated: • Anycast virtual IP • Round-robin DNS • HTTP load-balancer
Southbound traffic balancing	Connection Manager	Provide a load balanced connection scheme across the clustered controllers by migrating switches from overloaded controllers to lightly-loaded ones.

Table 4.6 Mapping between T-NOVA and OpenDaylight Components

4.5. Conclusions

OpenDaylight was selected from the available open source SDN controllers as the reference framework for the SDN control plane software implementation in T-NOVA. In addition, the Virtual Tenant Network service has been selected as the first option to support the Network Virtualisation functionalities. Finally, the clustering service included in OpenDaylight has been evaluated as an effective solution for the distributed control plane.

5. SDK FOR SDN

Modern computing is rapidly becoming dominated by resources being provisioned on demand, for the duration of the task being processed, and is increasingly being driven by the credit-card mind-set (instant-gratification), far away from the resource-requisition model that was satisfied through a centralised IT. The compute (CPUs), storage and networking needs of the consumers are being increasingly met by cloud computing providers - both internal and external to an organisation. In order to provide on-demand self-provisioning, the underlying network fabric has to be increasingly flexible. SDN is being seen as a likely paradigm that can support consumers' desires for a programmable network (management and configuration). Software defined networks can also be seen as a means to reduce the complexity and costs of the legacy networking setups, enabling innovation through rapid, uncomplicated deployment of protocol optimisations, and also possibly improving the throughput in a virtualised cloud environment by rendering the layers of encapsulation of various protocols (for tenant segregation) unnecessary.

Since the advent of OpenFlow [21] and with the maturity of several software implementations of basic network functions – open vSwitch [11] firewalls [22] etc., there is an increasing diversity of SDN controllers available today.

SDN controllers today not only allow management of flows (typically through OpenFlow protocol support in the physical/virtual switches), but also unified access to popular packages (open vSwitch), and other network protocols (ICMP), and in some cases to popular cloud platform networking management modules, for example OpenStack Neutron. Additionally, the telecommunication industry is gearing up for NFV framework adoption, and after the NFV recommendation from the ETSI MANO [23] working group, such activities have been gathering pace.

With the increasing adoption of SDN in big data centres and the push from telecommunication vendors worldwide towards NFV – the developer community is seeing increased activity in the SDN application development space. There is also an increasing diversity in SDN controllers developed by numerous consortia and individual companies. In order to sustain the quality development of SDN applications a uniform way is required for programmers to access the various SDN capabilities exposed by numerous controllers. Moreover a convenient compilation of useful networking libraries will aid the SDN application development process tremendously.

The SDK for SDN task in the T-NOVA project aims to alleviate some of the pain points of SDN application developers, and DC implementers by providing a toolkit that will abstract out the differences of the northbound APIs (nAPIs) of popular SDN controllers. The SDK for SDN task is also evaluating the fine-grained interactions between the virtualised and physical aspects of DC networking in order to better understand the bottlenecks in the data and control path, so as to include the necessary tools to aid DC planners.

Typically, a software development kit can comprise of the following:

- Language specific API wrappers
- Debugging capabilities typically built into the integrated development environment
- Testing and runtime environments (example: Mininet [24] for SDN applications)
- Sample codes showing how to use the SDK
- Supporting technical notes clarifying and supplementing the original feature usage guide

Some SDKs can provide additional functionalities such as a widget library, GUI builder, etc. for visual composition of features in the target application. There is also a need to compile a priority list of features needed by other tasks in this project, along with generally desired features to have in such a SDK. Understanding the interactions between virtual and physical network elements in a DC is critical in order to develop an understanding of the feature list to be supported by this task. In the following sections the requirements on the SDK for SDN are identified, along with an initial discussion on a possible architecture for the target SDK for SDN. The links and dependencies on other T-NOVA tasks and work-packages are described.

5.1. State of the Art

The following sections provides an brief review of the current state of the art that has relevance to the activities of Task 4.3

5.1.1. NetIDE Project

In order to identify possible overlap areas between NetIDE [25] and T-NOVA (WP4) projects, a brief overview of both the project scopes and activities is required together with an analysis of the possible synergies and differences between the two projects. Although project external goals would be the provisioning of development environment, to be utilised by SDN programmers, this analysis provides a comparison that assesses and identifies the key differences.

5.1.1.1. Overview

NetIDE is an FP7 project which started, together with T-NOVA, in January of 2014 with a duration of 36 months. The project is focused on delivering a single IDE to support the whole development lifecycle of portable network controller programs in a vendor-independent fashion.

The project objectives encompass:

- An abstraction layer for developing SDN platform independent network programs
- An IDE and associated tools that work on objects in the SDN abstraction layer
- A framework that interfaces the SDN abstraction layer with real and simulated/emulated network appliances
- Proof-of-concept implementations of network applications and services

The project is defining a framework for network applications developers who need to re-code their solutions every time they encounter a network infrastructure based on a different controller. Since the developers would have to develop different solutions for different high-level control plane network programming languages (Frenetic [26] Procera [27] etc.), this may result in network programs that are neither reusable nor shareable. The advantages of OpenFlow (uniform interface between the controller and the network infrastructures) are therefore not fully utilised. NetIDE proposes to deliver a single Integrated Development Environment to support the whole development lifecycle of network controller programs in a vendor-independent fashion.

The proposed NetIDE abstractions will allow the development of SDN solutions independent from the actual SDN controller used in each case. In particular NetIDE will:

- Define an abstraction layer for developing solution-independent SDN programs;
- Design and implement an IDE and associated tools that work on objects on the SDN abstraction layer;
- Provide a framework that interfaces the SDN abstraction layer with real and simulated/emulated network. In particular the development environment is controller-agnostic.

The overall architecture of the NetIDE project is shown in Figure 5.1, which separates into three respective blocks: the developer toolkit, the network app engine (mediator) and the network elements / emulators.

The developer toolkit represents the programming environment based on the NetIDE concept of Interchange Representation Format (IRF) - a central language element (lingua franca) [28] that covers orthogonal aspects of deployment models of different SDN approaches. It consists in a set of integrated tools, in an Eclipse-like environment, that allows software developers to code, configure, and deploy network Apps.

IRF provides a common representation of the network that is later processed by NetIDE tools (e.g. debugger, compiler, etc.). Network programs described through different languages can be transformed to IRF (and vice versa). Network programs described through IRF can be executed on top of different controllers thanks to a set of specific driver.

A library that translates the various constructs in network applications into representation to be used by the mediator layer below supports the IRF: NetIDE Network App Engine. This is a runtime environment that hosts Network Apps and acts as virtual controller of the network, leveraging existing network controllers.

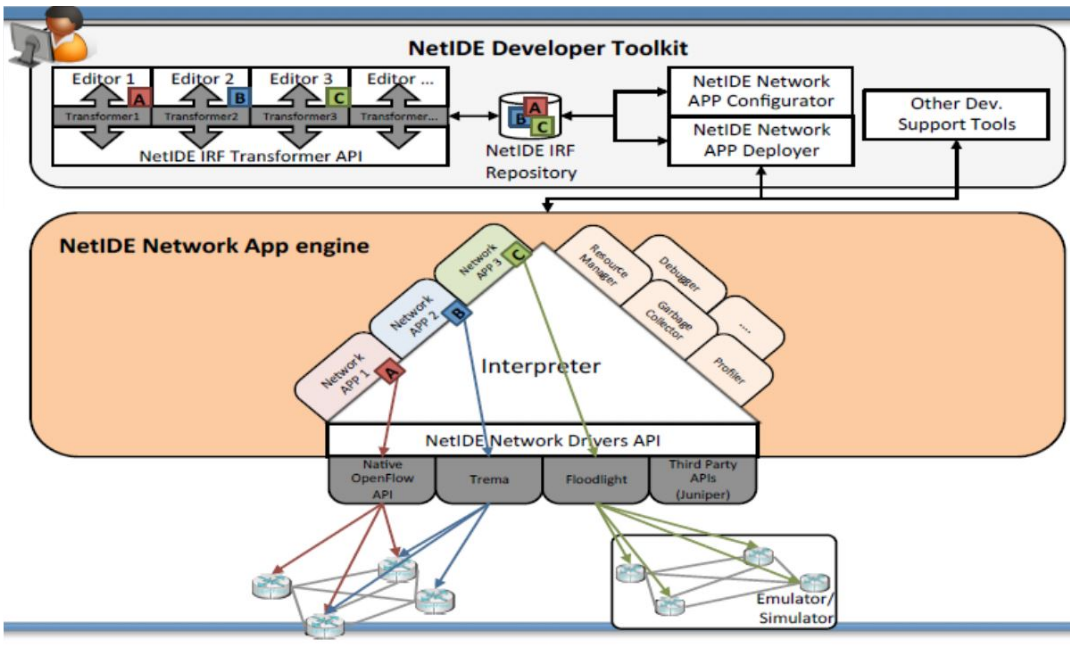


Figure 5.1 NetIDE Architecture

For achieving interoperability between controllers, NetIDE utilises pyretic [29] and its backend (Figure 5. 2).

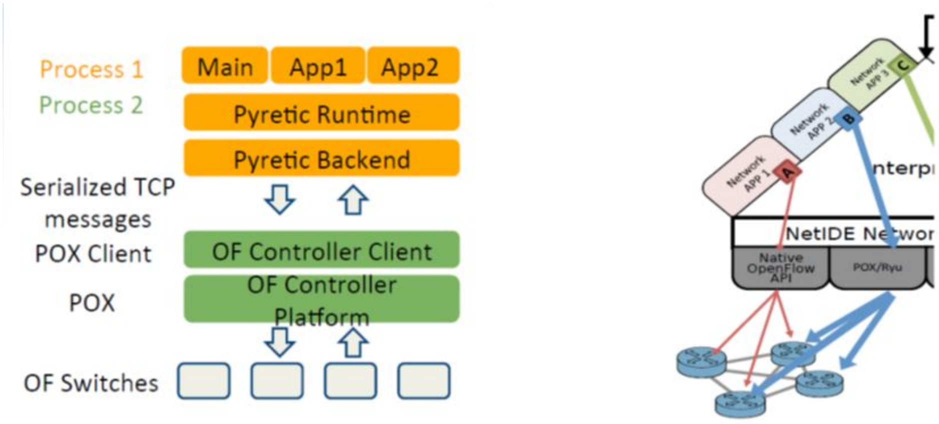


Figure 5.2 Pyretic

Subsequently, the client is rewritten for the Ryu controller. In this way the portability of applications by POX Ryu and vice versa is enabled. Furthermore an alpha version for OpenDaylight (ODL) has already been developed.

5.1.1.2. Differences between NetIDE and T-NOVA SDK

NetIDE is a project with a large allocated budget and therefore its objectives are broader compared to those of Task 4.3. In any case, both share the objective of the creation of a programming framework for SDN. While Task 4.3 is more focused on achieving unification of SDN controller nAPIs, NetIDE in comparison is oriented towards architectures that may involve LAN, Core and Metro networks and in principal any type of configuration or environment with programmable switches or NEs. The focus of SDK4SDN is restricted to particular topologies and sectors: DC SDN and SDN for orchestration of NFV. Therefore Task 4.3 represents both a subset of NetIDE's scope (i.e. DCs) but also an extension of NetIDE objectives because it is dealing with a particular architecture design of NFV (Orchestration). NetIDE does not address de facto cloud architectures or specific platforms like OpenStack.

An analysis of differences is presented in Figure 5.3. In the left side brackets, some annotations regarding NetIDE objectives are shown, and thoughts about targeting NFV and utilisation of cloud infrastructures are analysed.

In the right side brackets, T-NOVA and SK4SDN objectives are shown and annotated next to the different layers of the NetIDE architecture. One important requirement of T-NOVA is that the SDK will be used in the NFV VIM / orchestration phase thus at the quasi runtime phase, compared to the NetIDE programming environment, which is positioned for use during the design phase of SDN applications.

Furthermore, Task 4.3 will also focus on the interplay between physical and virtual network elements in a DC with a focus on providing tools to assess the bottlenecks in the data plane and libraries for programmers to support them in alleviating those bottlenecks.

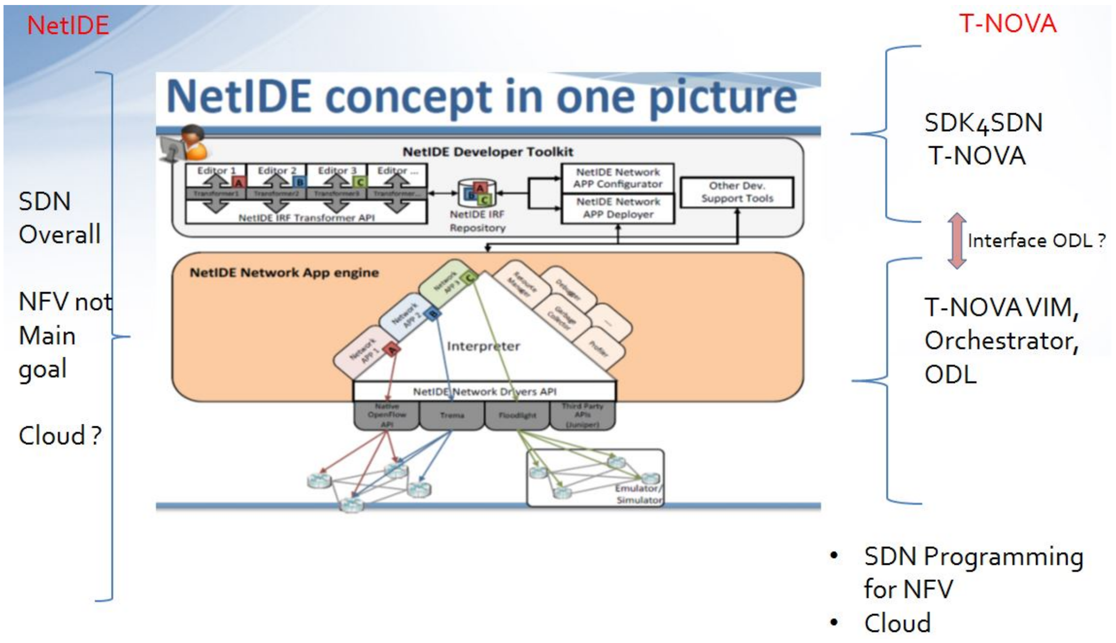


Figure 5.3 Main Differences T-NOVA versus NetIDE

In particular NFV orchestration involving SDN will deal with problems such as:

- Network Virtualisation and Cloud => Layer encapsulation, tunnelling, preconfigured virtual network flows between the VMs implementing the NFVs and the VMs and storage servers.
- Defining a library to help programmability of configurations or pre-configuration of SDN elements, both physical and virtual, for specific blocks of network functions

The SDK may address, for example, inclusion of existing OpenStack Neutron nAPIs features for SDN based elements. Table 5.1 shows comparison of the characteristics of Task 4.3 with the NetIDE project.

Project	Network segment	NFV	Cloud Orchestration	ODL	Phase	RYU	Base
NetIDE	all			yes	design	yes	Pyretic
T-NOVA SDK	DC	yes	yes	yes	Run time	yes	Jcloud java

Table 5.1 NetIDE versus T-NOVA

5.1.2. Popular SDN Controllers and API Comparison

Currently there is a variety of SDN controllers available in the market. Since analysing all of them would require a significant effort, an internal analysis of the popular SDN controllers within the project has been conducted which narrowed the controller choices down to two candidates – OpenDaylight and Ryu. In the first iteration of the SDK release, the task will aim at supporting both. A brief description of the controllers and a preliminary analysis of their APIs is provided in the following sections.

5.1.2.1. Ryu

Ryu is an open source, component-based software defined networking framework, which is written in python. Developers can easily create network management and control applications by using software components with well-defined APIs provided by Ryu.

Ryu supports various protocols such as OpenFlow 1.0, 1.2, 1.3, 1.4, NETCONF [30] and OFconfig [31]. At the Northbound APIs layer, Ryu has an OpenStack Neutron plug-in that supports VLAN and GRE configurations. Ryu also supports a REST interface for its OpenFlow operations.

Figure 5.4 shows the Ryu Framework Architecture. There are numerous "Ryu built-in apps", including firewalls, topology discover, tenant isolation, etc.

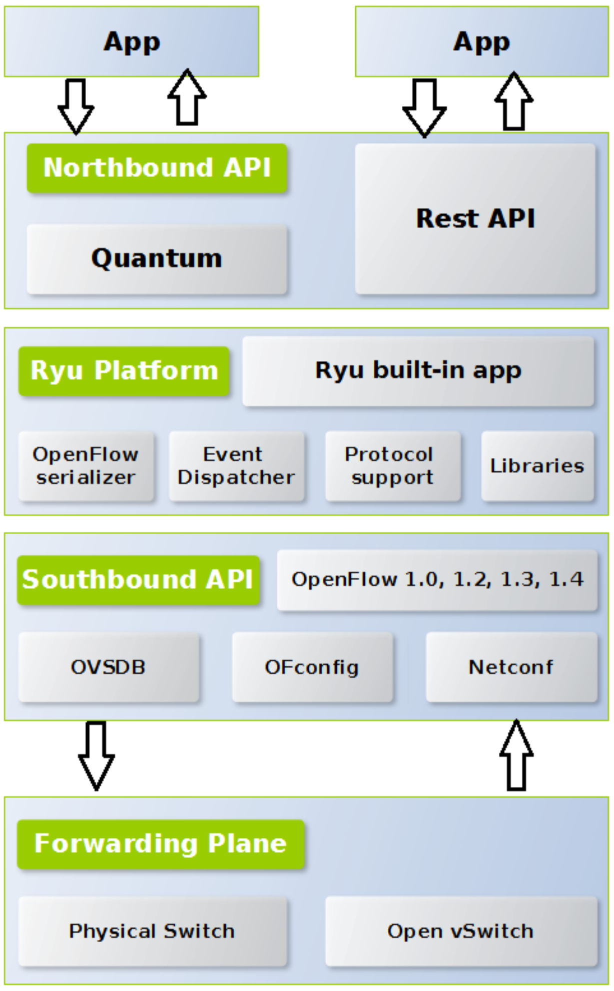


Figure 5.4 Ryu Architecture

5.1.2.2. OpenDaylight

OpenDaylight is a modular and pluggable open source framework written in Java. The controller is contained within a Java Virtual Machine (JVM) and can be deployed in any operating system platform that supports Java.

OpenDaylight supports the Open Service Gateway Initiative (OSGi) [32] framework and the REST interface for the nAPIs. For applications that are running in the same address space as the controller, the OSGi framework is used; whereas for applications that are not running in the same address space as the controller a REST API is utilised.

To perform required network tasks like host tracker or switch manager, the controller platform contains a collection of dynamically pluggable modules. The southbound interface supports multiple protocols such as OpenFlow 1.0, 1.3, Border Gateway Protocol (BGP), NETCONF, etc. Figure 5.4 shows the OpenDaylight Framework Architecture.

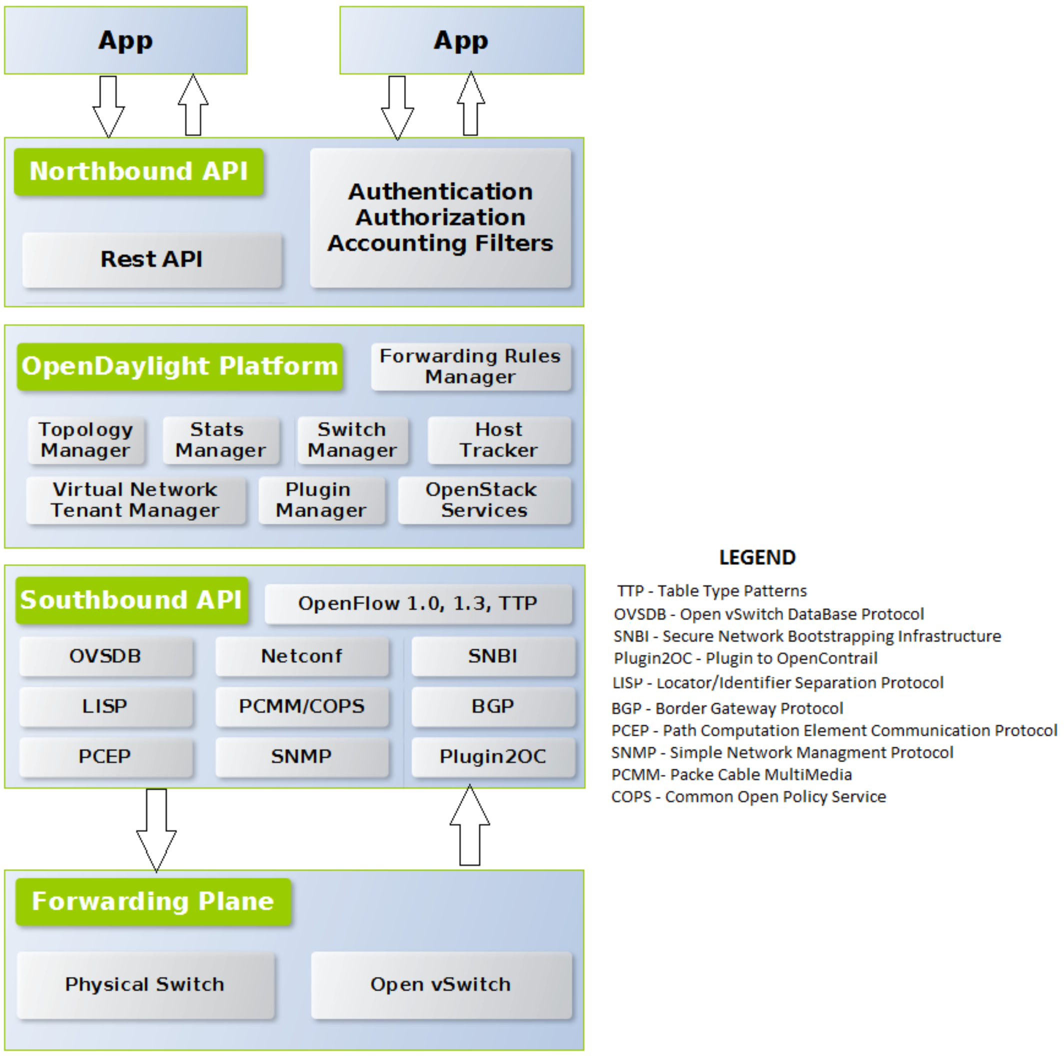

App
App
Northbound API
Rest API
Authentication Authorization Accounting Filters
OpenDaylight Platform
Forwarding Rules Manager
Topology Manager
Stats Manager
Switch Manager
Host Tracker
Virtual Network Tenant Manager
Plugin Manager
OpenStack Services
Southbound API
OpenFlow 1.0, 1.3, TTP
OVSDB
Netconf
SNBI
LISP
PCMM/COPS
BGP
PCEP
SNMP
Plugin2OC
Forwarding Plane
Physical Switch
Open vSwitch
LEGEND
TTP - Table Type Patterns
OVSDB - Open vSwitch DataBase Protocol
SNBI - Secure Network Bootstrapping Infrastructure
Plugin2OC - Plugin to OpenContrail
LISP - Locator/Identifier Separation Protocol
BGP - Border Gateway Protocol
PCEP - Path Computation Element Communication Protocol
SNMP - Simple Network Managment Protocol
PCMM- Packe Cable MultiMedia
COPS - Common Open Policy Service

Figure 5.5 OpenDaylight Architecture

5.1.2.3. Comparison of Ryu and OpenDaylight N-APIs

Tables 5.2 and 5.3 outline some of the standard modules and extensions of the Ryu and OpenDaylight platforms where similar classes of REST methods can be found. For example: in the OpenStack support comparison matrix (Table 5.2), in Network Configuration operations, the **GET /v1.0/networks** method provided by rest.py API returns the list of networks. In OpenDaylight, the respective functionality can be traced in the "Neutron Networks" extension and the actual REST call is: **GET /controller/nb/v2/neutron/networks**.

Operations	OpenStack	
	Ryu Module File	OpenDaylight Extension Name
Router Configuration		Neutron Routers Northbound
Firewall		Neutron Firewall Northbound

Configuration		Neutron Firewall Policy Northbound Neutron Firewall Rules Northbound
Network Configuration	rest.py	Neutron Networks Northbound
Switch Configuration	rest_conf_switch.py	
Port Configuration	rest.py	Neutron Ports Northbound
Flow Programming		
Statistics		
Topology		

Table 5.2 Comparing the Ryu and OpenDaylight support for OpenStack (Modules)

Operations	**Generic Network**	
	Ryu Module File	OpenDaylight Extension Name
Router Configuration	rest_router.py	Static Routing Northbound
Firewall Configuration	rest_firewall.py	
Network Configuration		
Switch Configuration		Switch Northbound
Port Configuration		
Flow Programming	ofctl_rest.py	Flow Programmer Northbound
Statistics	ofctl_rest.py	Statistics Northbound
Topology	rest_topology.py	Topology Northbound JAXRS

Table 5.3 Ryu and OpenDaylight Controllers' support towards generic networks (Modules)

The subset of REST calls available in the two SDN controllers that support OpenStack Neutron, and generic network elements are presented in Tables 5.4 and 5.5.

Operations	**OpenStack**	
	Ryu REST API	OpenDaylight REST API
Return a list of networks	GET /v1.0/networks	GET /controller/nb/v2/neutron/networks

Create a new network	POST /v1.0/networks/(network-id)	POST /controller/nb/v2/neutron/networks
Update a network	PUT /v1.0/networks/(network-id)	PUT /controller/nb/v2/neutron/networks/(netUUID)
Delete a network	DELETE /v1.0/networks/(network-id)	DELETE /controller/nb/v2/neutron/networks/(netUUID)
List all ports	GET /v1.0/networks/(network-id)/	GET /controller/nb/v2/neutron/ports
Create new port	POST /v1.0/networks/(network-id)/(dpid)_(port-id)	POST /controller/nb/v2/neutron/ports
Update port	PUT /v1.0/networks/(network-id)/(dpid)_(port-id)	PUT /controller/nb/v2/neutron/ports/(portUUID)
Delete port	DELETE /v1.0/networks/(network-id)/(dpid)_(port-id)	DELETE /controller/nb/v2/neutron/ports/(portUUID)

Table 5.4 REST APIs Comparison for OpenStack Neutron Support

Operations	**Generic Networks**	
	Ryu REST API	OpenDaylight REST API
Get the route data	GET /router/(switch_id)/(vlan_id)	GET /controller/nb/v2/staticroute/(containerName)/route/(route)
Add a new route	POST /router/(switch_id)/(vlan_id)	PUT /controller/nb/v2/staticroute/(containerName)/route/(route)
Delete a route	DELETE /router/(switch_id)/(vlan_id)	DELETE /controller/nb/v2/staticroute/(containerName)/route/(route)
Add a flow configuration	POST /stats/flowentry/add	PUT /controller/nb/v2/flowprogrammer/(containerName)/node/(nodeType)/(nodeId)/staticFlow/(name)
Delete a flow configuration	POST /stats/flowe	DELETE /controller/nb/v2/flowprogrammer/(containerName)/node/(n

	ntry/delete	odeType)/(nodeId)/staticFlow/(name)
Modify a flow configuration	POST /stats/flowentry/modify	PUT /controller/nb/v2/flowprogrammer/(containerName)/node/(nodeType)/(nodeId)/staticFlow/(name)
Get flow statistics for a node	(switch) GET /stats/flow/(dpid)	GET /controller/nb/v2/statistics/(containerName)/flow/node/(nodeType)/(nodeId)
Get all the links configuration	GET /v1.0/topology/links	GET /controller/nb/v2/topology/(containerName)/userLinks

Table 5.5 Ryu and OpenDaylight REST support for generic networks

As evident from the tables there are many points where the REST APIs provided by the northbound layers of these two SDN controllers diverge considerably, or in some cases are missing. This further strengthens the need for a unified layer to facilitate software development for SDN applications in an environment agnostic manner.

5.1.3. Popular Cloud Libraries

The last number of years has seen the proliferation in cloud platforms, both from commercial vendors and open source communities. Notable platforms are OpenStack, Apache CloudStack, OpenNebula, Eucalyptus, etc. Each platform has their own programmable interfaces, and there have been significant efforts by the community to unify these interfaces to simplify application development over them. Today in the SDN space we see similar needs, therefore in order to understand how community solutions for cloud interface differences were devised, and to gain design insights for the SDK for SDN solution, two most popular community cloud libraries were analysed.

5.1.3.1. Jclouds

Apache jclouds is an open source library that provides an abstraction with Java or Clojure API to control and manage various cloud software platforms such as Amazon, Openstack or VMware vCloud [2]. Jclouds is extremely flexible and allows developers to invoke not only portable operations (i.e. operations that are exposed by all supported cloud platforms), which are controlled by services like ComputeService and BlobStorage, but also platform-specific operations (i.e. operations that are specific to a single platform, or a subset of the supported platforms). Jcloud's main concepts are:

- Views: Views are portable abstractions to allow writing code that uses generic cloud services. The concept is similar to JDBC for a DB. The actual views available are: ComputeService, BlobStorage and LoadBalancer.

[2] A of supported providers can be found here https://jclouds.apache.org/reference/providers/

- APIs: APIs represents the specific call made in order to perform some action on the cloud. Most APIs are HTTP based (SOAP or REST), but in some cases this may be different, depending on the specific cloud.
- Providers: A provider is a specific cloud platform API plus some specific values like the endpoint URL. Each provider in jclouds implements one or more Views.
- Contexts: A context represents a specific connection to a particular provider. The concept is like a database connection to a specific DB.

In order to create their own applications with jclouds, developers instantiate a context and connect to a specific Provider. As soon as the connection is established, developers can use Views to invoke portable operations exposed by the provider, or APIs for provider-specific operations.

5.1.3.2. libcloud

Apache libcloud is a Python library for interacting with many popular cloud service providers through a unified API. Similar to jClouds, it was created in order to allow developers to build applications that can work seamlessly with all supported cloud providers. libcloud can manage the following categories of resources:

- Compute – Cloud Servers and Block Storage services, e.g. Amazon EC2, OpenStack Nova/Cinder
- Object Storage – e.g. Amazon S3, OpenStack Swift
- Load Balancer – e.g. Amazon Elastic Load Balancer, Rackspace Load Balancer
- DNS – e.g. Amazon Route 53, Google DNS

For a full list of supported providers, please refer to the project official website [33]. Similar to jClouds, Libcloud supports both cross-platform and platform-specific operations: the former are exposed through a common, unified API layer, while the latter are exposed through API extensions. Furthermore, Libcloud supports a driver-based architecture, which allows integrating new cloud platforms that are not natively supported, by developing and registering a 3rd party driver.

5.2. Requirements Gathering

An SDK must take into account various requirements that affect design and implementation issues. Three main categories can be identified here:

- The first is the software design phase where a number of design tricks can help in the implementation phase.
    - The use of abstract classes allow you to change how an object works without breaking code
    - The limited use of inherited classes only exposes absolutely necessary functions.

- The use of small interfaces will limit the chance of breaking existing code while the reduction of big interfaces benefits the SDK by not limiting the SDK to the predefined design
- The second is end user design issues where again a number of design tricks can help in the implementation phase.
    - The use of a good definition in describing the terms used inside the SDK provides a good starting point.
    - The use of namespaces based on how often a set of objects will be used.
    - The use of intelligent code completion providing in this way can be an instant help in the programmer.
- The third is Language and Terms Design where the main concept is not to use terms that people need to look up.
    - Language and terms really help determine what objects and functionality need to be defined as code.
    - The language used must be self-evident to other developers.

Apart from the generic guidelines above, which are vital for the successful design and implementation of any SDK, the SDK for SDN in T-NOVA has some specific requirements that have been captured in the Table 5.6.

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-OpenFlow	SDK for SDN MUST expose OpenFlow differences in a safe manner	Comes from DoW	Functional
SDK4SDN-Source-CODE	SDK for SDN MUST be available as open source software	Comes from DoW	Functional
SDK4SDN-Contrlollers	SDK for SDN SHALL support a variety of SDN Controllers	Comes from DoW	Functional
SDK4SDN-Libraries	SDK for SDN SHALL provide all the necessary dependencies and tools in order that developers can validated their installation	Comes from DoW	Functional
SDK4SDN-Languages	SDK for SDN-MUST support multiple languages	Comes from DoW	Functional

SDK4SDN-OpenFlow Versions	SDK for SDN support multiple OpenFlow versions	Comes from DoW	Functional

Table 5.6 SDK for SDN Functional Requirements

Considering these requirements and leveraging the approaches of the jCloud and libcloud libraries for cloud platforms, first iteration of SDK for SDN platform architecture is outlined in Section 5.3.

5.3. Initial High Level Architecture

Figure 5.6 shows the initial high-level architecture of the SDK for SDN tool to be developed in this task. The design phase of this task has not yet concluded (at the time of writing of this deliverable), and therefore it is likely that the architecture may undergo additional changes as the task progresses.

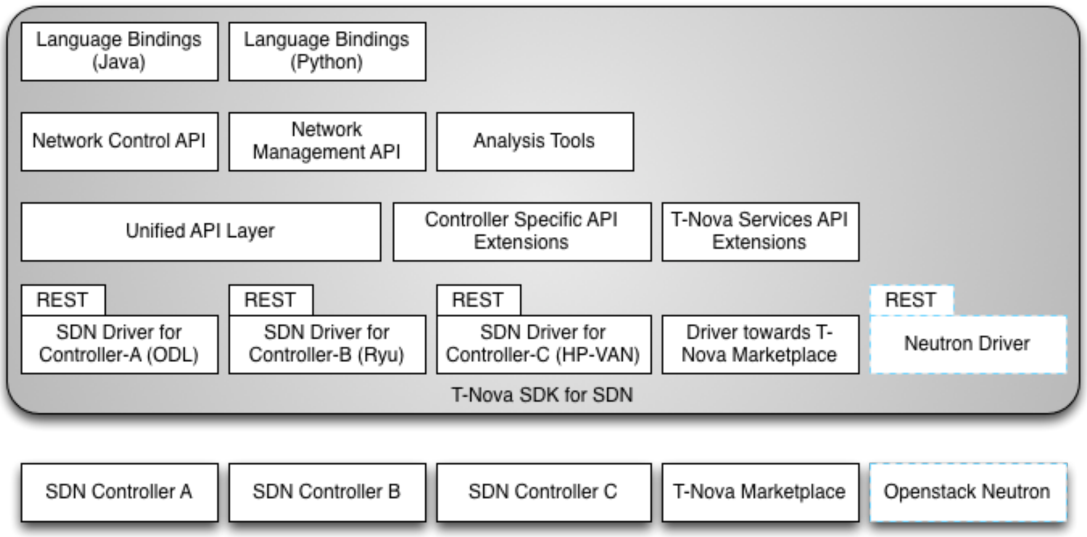


Figure 5.6 SDK for SDN Initial Architecture

This initial iteration only shows the architectural elements for the unification of the different APIs exposed by the various popular SDN controllers. The elements shown outside of the boxed area are not part of the SDK but are external elements over which the SDK will be developed and deployed. At the bottom layer of the architecture are the various drivers for the target SDN controllers who's APIs are to be supported by the SDK, apart from any T-NOVA specific integration features. The SDN controllers APIs are analysed and common feature sets are offered out from the Unified API Layer component. Controller specific features, which cannot be made available by the Unified API Layer, can be exposed to the developers by individual Controller Specific API extensions.

APIs developed by the various T-NOVA services for 3rd party integrators, can be supported as part of the SDK as well. An example case using the T-NOVA Marketplace component is indicated in the architecture. These will be offered through individual extension modules in the SDK in phases. Furthermore, providing support for OpenStack Neutron is being considered, and is therefore shown as a dashed-box in the architecture.

The APIs and features to be supported by the SDK can be generally categorised into network control and network management APIs. In addition - the SDK will provide additional tools to facilitate application debugging, network packet analysis to understand interactions between the virtual and physical network elements in a DC, and additional tools and code examples to help an application developer optimise the DC networking fabric.

The highest layer in the architecture diagram is the popular programming language bindings, which are the Java/Python API wrappings. Other language supports can be incorporated at a later date depending on the community demand for such bindings. The architecture will evolve in coming months to include mechanisms to include testing and runtime environments using Mininet [24] or similar external platform. Discussions on a clean-slate network research approach by including appropriate libraries (frame detection, error correction codes, etc.) is still on-going, and the overall architecture document will be updated as this task progresses.

5.4. Conclusions and Next Steps

The SDK for SDN task will not only provide common development libraries (with support for multiple SDN controllers) to the SDN application developers, it will also undertake a detailed analysis of interactions among virtual and physical network elements in a typical cloud DC. The aim of the study will be to identify bottlenecks in the network stack, and to develop specific libraries to aid developers in the minimisation or removal of bottlenecks. In this section, an initial draft architecture of the SDK has been presented which over the course of next couple of months will be further refined based on the close interactions planned with other tasks in T-NOVA. The two initial SDN controllers that will be targeted by this task will be OpenDaylight and Ryu - the final two controllers filtered by the T-NOVA consortium. Other controllers' support can be included over the course of time if required.

API language bindings for Java and Python will be provided to aid the development process. The goals for the next few months in this task are to finalise the API analysis of the northbound APIs exposed by OpenDaylight and Ryu, and to refine the architecture further with inputs from other tasks in T-NOVA. Actual code development will commence early next year together with the cloud DC analysis.

6. Monitoring and Maintenance

This section outlines the work currently being carried out in Task 4.4 (Monitoring and Maintenance). Task 4.4 focuses on the implementation and integration of a monitoring framework, which is able to extract and process monitoring information from both physical and virtual nodes at the IVM level. In other words, the scope of the monitoring framework being developed in Task 4.4 covers the two lower layers of the T-NOVA architecture, namely the NFVI and VIM.

Metrics are collected at the NFVI layer, processed at the VIM and forwarded to the upper layers (Orchestrator and Marketplace). Task 4.4 focuses specifically on the collection of dynamic metrics, i.e. metrics which change frequently in relation to resource usage. Static information reflecting the status and capabilities of infrastructure, e.g. number of installed compute nodes, processing resources per node etc. are assumed to be handled by Task 3.2 (Infrastructure Repository).

6.1. Requirements Overview and Consolidation

Deliverable D2.31 has defined and identified architectural concepts and requirements for the IVM (NFVI and VIM) layer. The technical requirements for the IVM monitoring framework can be directly derived/inherited by the specific IVM requirements. Table 6.1 outlines the IVM requirements that directly affect the monitoring framework and their required specialisations within the monitoring framework.

IVM Req.ID	IVM Requirement Name	Requirement specialisation for the Monitoring Framework
VIM.1	Ability to handle heterogeneous physical resources	The MF must provide a vendor agnostic mechanism for physical resource monitoring.
VIM.3	API Exposure	The MF must provide an interface to the Orchestrator for the communication of monitoring metrics.
VIM.7	Translation of references between logical and physical resource identifiers	The MF must re-use resource identifiers when linking metrics to resources.
VIM.9	Control and Monitoring	The MF must monitor in real time the physical network infrastructure as well as the vNets instantiated on top of it.
VIM.24	Virtualised Infrastructure Metrics	The MF must collect utilisation metrics from the virtualised resources in the NFVI.
C.7	Compute Domain Metrics	The MF must collect compute domain metrics.
H.1	Compute Domain Metrics	The MF must collect metrics from the Hypervisor.

H.12	Alarm/Error Publishing	The MF must receive and process any alarms sent by the Hypervisor.
N.6	Usage monitoring	The MF must collect metrics from physical and virtual networking devices.
N.9	OpenFlow	The MF must leverage OpenFlow monitoring capabilities.

Table 6.1 IVM requirements which affect the monitoring framework

By consolidating the aforementioned requirements, it becomes clear that the basic functionalities required the IVM monitoring framework are as follows:

- Collection of IT and networking metrics from virtual and physical devices of the NFVI. It should be noted that at the IVM level, metrics correspond only to physical and virtual nodes and are not associated to services since the VIM does not have knowledge of end-to-end Network Services. Metrics are mapped to Network Services at the Orchestrator level;
- Processing and generation of events and alarms;
- Communication of monitoring information and events/alarms to the Orchestrator in a scalable manner;

6.2. Challenges and Innovations

With regard to the basic functionalities identified in the previous section, metrics collection (Functionality 1) can be achieved by re-using a number of the pre-existing monitoring mechanisms for virtualised infrastructures, as surveyed in the following section. Apart from selecting and properly integrating the appropriate technologies and possibly selecting the appropriate set of metrics, limited progress beyond the state-of-the-art should be expected in this field.

On the other hand, the actual challenges and envisaged innovation of the monitoring framework are seen to be associated with Functionalities 2 and 3. Specifically, the following challenges have been identified:

- *Events and alarms generation:* Moving beyond the typical approach, which is found in most monitoring systems and is based on static thresholds (i.e. generate an alarm when a metric has crossed a pre-defined threshold) the aim is to study and adopt dynamic methods for fault detection. Such methods should be based on statistical methods and self-learning approaches, identifying outliers in system behaviour and triggering alarms reactively or even proactively (e.g. before the actual fault has occurred). This anomaly detection procedure, in the context of T-NOVA, can clearly benefit from the fact that the monitored services are composed of VNFs rather than generic VMs. As virtual appliances dedicated to traffic processing, VNFs are expected to expose some common characteristics (e.g. the CPU load is expected to proportionally rise, not necessarily linearly, with the increase of processed traffic). A significant deviation from this correlation could, for example, indicate a potential malfunction.

- *Communication with the Orchestrator*: With this functionality, scalability is the key requirement that needs to be fulfilled. In an operational environment, the Orchestrator is expected to manage tens or hundreds of NFVI-PoPs (or even thousands, if micro-data centres distributed in the access network are envisaged). It is impossible for the Orchestrator to handle the full set of metrics from all the physical and virtual nodes under its control. The challenge is to optimise communication of monitoring information to the Orchestrator so that only necessary information is transmitted. This optimisation does not only imply fine-tuning of the polling frequency, careful definition of a minimal set of metrics or the proper design of the communication protocol, but also requires an intelligent aggregation procedure at the VIM level. This procedure should achieve the grouping/aggregation of various metrics from various parts of the infrastructure as well as alarms, and the dynamic identification of information that is of actual value to the Orchestrator.

To achieve the aforementioned innovations, Task 4.4 work plan involves in its initial stage the establishment of a baseline framework which fulfils the basic functionalities by collecting and communicating metrics and, as a second step, the study, design and incorporation of innovative techniques for anomaly detection and metrics aggregation.

6.3. Monitoring Frameworks for Virtualised Infrastructures Survey

This section presents a brief overview of existing frameworks for monitoring virtualised IT infrastructures as well as SDN-enabled networks, and discusses the technologies which could be partially re-used in T-NOVA.

6.3.1. IT/Cloud monitoring

6.3.1.1. OpenStack Telemetry

OpenStack's Telemetry module, formerly called Ceilometer, reliably collects measurements with respect to the utilisation of physical and virtual resources that comprise deployed clouds. Telemetry persists data for subsequent retrieval and analysis and triggers actions when defined criteria are met. It efficiently collects the metering data of guest machines (VMs) and the hosts (Nova), the network, the Operating System images (Glance), the disk volumes (Cinder), the identities (Keystone), the object storage (Swift), the orchestration (Heat), the energy consumption (Kwapi) and also user-defined meters.

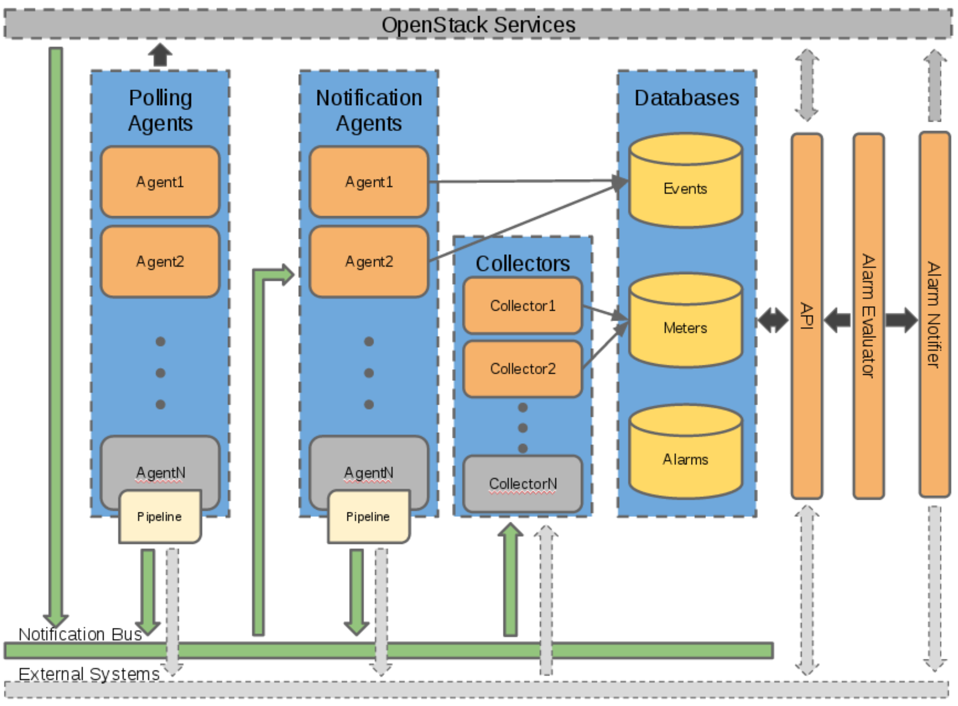


Figure 6.1 Overview of Openstack Telemetry architecture

Figure 6.1 depicts an overall logical architecture of the Telemetry module. Each of the telemetry services are designed to scale horizontally. Additional workers and nodes can be added depending on the expected load. The system consists of the following basic components:

- *Polling agents*; these are:
    - Compute agents (ceilometer-agent-compute): they run on each compute node and poll for resource utilisation statistics;
    - Central agents (ceilometer-agent-central): run on one or more central management servers to poll for resource utilisation statistics for resources not tied to instances or compute nodes;
- *Notification agents*; these run on one or more central management servers to monitor the message queues (for notifications and for metering data coming from the agent);
- *Collectors* (ceilometer-collector): designed to gather and record event and metering data created by notification and polling agents.
- *Databases*, containing Events, Meters and Alarms; these are capable of handling concurrent writes (from one or more collector instances) and reads (from the API module);
- An *Alarm Evaluator and Notifier* (ceilometer-alarm-notifier): Runs on one or more central management servers to allow configuration of alarms based on threshold evaluation for a collection of samples.
- An *API* module (ceilometer-api): Runs on one or more central management servers to provide access to the data from the data store.

Telemetry offers three independent ways to collect metering data, allowing easy integration of any OpenStack-related project which needs to be monitored:

- Via listening to events generated on the notification bus and transforming them into Ceilometer samples. This is the preferred method of data collection, since it is the most simple and straightforward. It requires, however, that the monitored entity uses the bus to publish events, which may not be the case for all OpenStack-related projects.
- Via pushing information to Telemetry, which requires adding a component to each of the nodes that need monitoring, thus, making data collection more complex. It is the recommended solution for modules which do not use the message bus.
- Via polling information by Telemetry, which polls the APIs of the components being monitored at regular intervals to collect information. The data are stored usually in a database and are available through the Ceilometer REST API. This method is least preferred due to the inherent difficulty in making such a component resilient.

Each meter measures a particular aspect of resource usage or on-going performance. All meters have a string name, a unit of measurement, and a type indicating whether values are monotonically increasing (cumulative), interpreted as a change from the previous value (delta), or a standalone value relating only to the current duration (gauge). Samples are individual data points associated with a particular meter and have a timestamp and a value. The aggregation of a set of samples for a specified duration (start-end time) is called a statistic. Each statistic also has a period associated with it, which is a repeating interval of time that the samples are grouped for aggregation. Currently there are five aggregation functions implemented: *count*, *max*, *min*, *avg* and *sum*.

These metering data can go through pipelines, composed by chains of transformers that change the data before sending them to the collector via a publisher. A transformer can be, for example, a unit conversion, a rate of change calculation and an accumulator of metering data. Telemetry is able to publish the metering data multiple times to multiple destinations, possibly using a different transport method (RPC, UDP, files) and frequency of publication. The pipelines can be configured via a YAML file.

As already mentioned, another feature of Telemetry is alarming. An alarm is a set of rules defining a monitor of a statistic that will trigger when a threshold condition is breached. An alarm can be set on a single meter, or on a combination of meters and can have three states, *alarm* (the threshold condition is breached), *ok* (the threshold condition is not met) and *insufficient data* (not enough data has been gathered to determine if the alarm should fire or not). The transition to these states can have an associated action, which is either writing to a log file or an http post to a URL. The concept of a meta-alarm is also supported; meta-alarms aggregate over the current state of a set of other basic alarms combined via a logical operator (AND/OR). For example, a meta-alarm could be triggered when three basic alarms are active at the same time.

6.3.1.2. Other Cloud/Data Centre Monitoring Tools

In this section a brief description of the most popular tools used for monitoring cloud and data centre architectures is given. Prior to the introduction of the OpenStack

Telemetry module, several projects and related tools were used for monitoring and metering an OpenStack-based cloud. The presented tools can still be used, especially if these tools are monitoring others parts of the infrastructure as well.

Zabbix

Zabbix [34] is an open source, general-purpose, enterprise-class network and application monitoring tool that can be customised for use with OpenStack. It can be used to automatically collect and parse data from monitored cloud resources. It also provides distributed monitoring with centralised Web administration, a high level of performance and capacity, JMX monitoring, SLAs and ITIL KPI metrics on reporting, as well as agent-less monitoring. An OpenStack Telemetry plugin for Zabbix is already available.

Using Zabbix the administrator can monitor servers, network devices and applications, gathering statistics and performance data. Monitoring performance indicators such as CPU, memory, network, disk space and processes can be supported through an agent, which is available as a native process for Linux, UNIX and Windows platforms. With OpenStack infrastructure Zabbix can monitor:

- Core OpenStack services: Nova, Keystone, Neutron, Ceilometer (OpenStack Telemetry), Horizon, Cinder, Glance, Swift Object Storage, and OVS (Open vSwitch)
- Core infrastructure components: MySQL, RabbitMQ, HAProxy, memchached, and libvirtd.
- Operating system statistics: Disk I/O, CPU load, free RAM, etc.

Zabbix is not limited to OpenStack cloud infrastructures: it can be used to monitor VMware vCenter and vSphere installations for various VMware hypervisor and virtual machine properties and statistics.

Nagios

Nagios is an open source tool that provides monitoring and reporting for network services and host resources [35]. The entire suite is based on the open-source Nagios Core which provides monitoring of all IT infrastructure components - including applications, services, operating systems, network protocols, system metrics, and network infrastructure. Nagios does not come as a one-size-fits-all monitoring system with thousands of monitoring agents and monitoring functions; it is rather a small, lightweight system reduced to the bare essential of monitoring. It is also very flexible since it makes use of plugins in order to setup its monitoring environment.

Nagios Fusion enables administrators to gain insight into the health of the organisation's entire network through a centralised view of their monitoring infrastructure. In addition, they can automate the response to various incidents through the use of the Nagios Incident Manager and Reactor. The Network Analyser, which is part of the suite, provides an extensive view of all network traffic sources and potential security threats allowing administrators to quickly gather high-level information regarding the status and utilisation of the network as well as detailed data for complete and thorough network analysis. All monitoring information is stored in the Log Server that provides monitoring of all mission-critical infrastructure components – including applications, services, operating systems, network protocols, systems metrics, and network infrastructure.

Nagios and Telemetry are quite complementary products which can be used in an integrated solution. Enovance has developed plugins for the Nagios monitoring environment to ensure that OpenStack components are still functioning, while the ICCLab, which operates within the ZHAW's Institute of Applied Information Technology, has developed a Nagios plugin which can be used to capture metrics through the Telemetry API, thus allowing Nagios to monitor VMs inside OpenStack. Finally, the Telemetry plugin can be used to define thresholds and triggers in the Nagios alerting system.

Shinken

Shinken is an open source system and network monitoring application [36]. It is fully compatible with Nagios plugins. It started as a proof of concept for a new Nagios architecture, but since the proposal was turned down by the Nagios authors, Shinken became an independent tool. It is not a fork of Nagios; it is a total rewrite in Python. It watches hosts and services, gathers performance data and alerts users when error conditions occur and again when the conditions clear. Shinken's architecture is focused on offering easier load balancing and high availability capabilities. The main differences and advantages toward Nagios are:

- A more efficient distributed monitoring and high availability architecture
- Graphite integration in the Web UI
- Improved performance, mostly due to the use of a distributed database (MongoDB)

Icinga

Icinga is an open-source network and system monitoring application which originated from a Nagios fork [37]. It maintains configuration and plug-in compatibility with the latter. Its new features are as follows:

- A modern Web 2.0 style user interface;
- An interface for mobile devices;
- Additional database connectors (for MySQL, Oracle, and PostgreSQL);
- RESTful API.

Currently there are two flavours of Icinga that are maintained by two different development branches: Icinga 1 (the original Nagios fork) and Icinga 2 (where the core framework is being replacement by a full rewrite).

Zenoss

Zenoss is an open source monitoring platform released under a GPLv2 license [38]. It provides an easy-to-use Web UI to monitor performance, events, configuration, and inventory. Zenoss is one of the best options for unified monitoring as it is cloud-agnostic and open source. Zenoss provides powerful plug-ins named Zenpacks, which support monitoring on hypervisors (ESX, KVM, Xen and HyperV), private cloud platforms (CloudStack, OpenStack and vCloud/vSphere), and public cloud (AWS). In OpenStack Zenoss integrates with Nova, Keystone and OpenStack Telemetry.

Ganglia

Ganglia is a scalable distributed system monitor tool for high-performance computing systems such as clusters and grids [39]. Its structure is based on a hierarchical design using a tree of point-to-point connections among cluster nodes. Ganglia is based on an XML data representation, XDR for compact and RRDtool for data storage and virtualisation. The Ganglia system contains:

1. Two unique daemons, gmond and gmetad
2. A PHP-based web front-end
3. Other small programs

gmond runs on each node to monitor changes in the host state, to announce applicable changes, to listen to the state of all Ganglia nodes via a unicast or multicast channel based on installation, and to respond to requests. gmetad (Ganglia Meta Daemon) polls at regular intervals a collection of data sources, parses the XML and saves all metrics to round-robin databases. Aggregated XML can then be exported.

Ganglia’s web frontend is written in PHP. It uses graphs generated by gmetad and provides collected information like CPU utilisation for the past day, week, month, or year. Ganglia has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes. However, further work is required in order for it to become more cloud-agnostic.

StackTach

StackTach is a debugging and monitoring utility for OpenStack that can work with multiple Data Centres, including multi-cell deployment [40]. It was initially created as a browser-based debugging tool for OpenStack Nova. In the interim StackTach has evolved into a tool that can perform debugging, monitoring and auditing. StackTach is quickly moving into Metrics, SLA and Monitoring territory with version 2 and the inclusion of Stacky, the command line interface to StackTach. StackTach contains a worker process that reads notifications from the OpenStack’s RabbitMQ queues and stores them in a database. From there, StackTach reviews the stream of notifications to glean usage information and assemble it in an easy-to-query fashion. Users can inquire about instances, requests, servers, etc. using the browser interface or the Stacky command line tool. Rackspace is working on StackTach integration with Telemetry.

Healthmon

Healthmon by HP is focused on delivering a unique point of contact for all OpenStack Cloud Resources and Infrastructure monitoring requirements, covering Inventory, Utilisation and Alert [41]. Some key characteristics of Healthmon are:

- Monitoring Service for Cloud Resources and Infrastructure with a pluggable framework;
- Resource Model and Persistence;
- Cloud Resource Life Cycle Event/Alarm Collection (compute host, instances, bare metal, network, etc.);
- Cloud Service Event Collection Framework (Nova, Keystone, etc.);
- Integration with Telemetry;

- RESTful access to the stored Cloud Resource Inventory, utilisation / usage data;
- Data Provider;
- Proxy Driver for the underlying hypervisor (KVM, ESX and Hyper-V are supported).

Healthmon integrates with the Horizon interface, adding a dedicated Healthmon tab to it.

SeaLion

SeaLion is a cloud-based system monitoring tool for Linux servers. It installs an agent on the system, which can run as an unprivileged user [42]. The agent collects data at regular intervals across servers and this data will be available on your workspace. Sealion provides a high-level view (graphical overview) of Linux server activity. The monitoring data are transmitted over SSL to the SeaLion servers. The service provides graphs, charts and access to the raw gathered data.

MonALISA

MONitoring Agents using a Large Integrated Services Architecture (MonaLISA) is based on Dynamic Distributed Service Architecture and is able to provide complete monitoring, control and global optimisation services for complex systems [43]. The MonALISA system is designed as a collection of autonomous multi-threaded, self-describing agent-based subsystems which are registered as dynamic services, and are able to collaborate and cooperate in performing a wide range of information gathering and processing tasks.

The agents can analyse and process the information in a distributed way, in order to provide optimisation decisions in large-scale distributed applications. The scalability of the system derives from the use of a multithreaded execution engine, that hosts a variety of loosely coupled self-describing dynamic services or agents, and the ability of each service to register itself and to be discovered and used by any other services, or clients that require such information. The system is designed to easily integrate existing monitoring tools and procedures and to provide this information in a dynamic, customised, self-describing way to any other services or clients.

By using MonALISA the administrator is able to monitor all aspects of complex systems, including:

- System information for computer nodes and clusters;
- Network information (traffic, flows, connectivity, topology) for WAN and LAN;
- Monitoring the performance of applications, jobs or services; and
- End-user systems and end-to-end performance measurements.

collectd, StatsD and Graphite

Cloud instances may also be monitored by using a collection of separate open source tools. collectd is a daemon which collects system performance statistics periodically and provides mechanisms to store the values in a variety of ways [44]. collectd gathers statistics about the system it is running on and stores this information. These statistics can then be used to find current performance bottlenecks (i.e. performance analysis) and predict future system load (i.e., capacity planning). collectd is written in C for performance and portability, allowing it to run on systems without scripting language or cron daemon, such as embedded systems. At the same time it includes optimisations and features to handle big amounts of data sets. StatsD [45] is a Node.JS daemon [46] that listens for messages on a UDP to TCP port. StatsD listens for statistics, like counters and timers and then parses the messages, extracts metrics data, and periodically flushes the data to other services in order to build graphs. A tool that can be used to build graphs afterwards is Graphite [47], which is able to store numeric time-series data and render graphs of the data on demand.

vSphere

The vSphere statistics subsystem collects data on the resource usage of inventory objects [48]. Data on a wide range of metrics is collected at frequent intervals, processed and archived in a database. Statistics regarding the network utilisation are collected at Cluster, Host and Virtual Machine levels. In addition vSphere supports performance monitoring of guest operating systems, gathering statistics regarding network utilisation among others.

Amazon CloudWatch

Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications running on AWS [49]. It provides real-time monitoring to Amazon's EC2 customers on their resource utilisation such as CPU, disk and network. However, CloudWatch does not provide any memory, disk space, or load average metrics without running additional software on the instance. It was primarily designed for use with Amazon Elastic Load Balancing and Auto Scaling with load balancing in mind: the service checks CPU usage on multiple instances and automatically creates additional ones when the load increases.

6.3.2. Network Monitoring

Network monitoring is a domain that has attracted significant attention from the research community over the past decades, with well-established technologies and standards with regard to measurement processes (active and passive) as well as the communication of monitoring metrics (SNMP, IPFIX, sFlow etc.).

In the context of T-NOVA, where network management, at least within each NFVI-PoP is based on OpenFlow, the measurement process will leverage OpenFlow’s monitoring capabilities.

OpenFlow provides the capability to report per-flow and per-port metrics, reported by the switch itself. These metrics are then collected by the Controller and communicated to SDN control applications via the northbound API of the Controller it-self (Figure 6.2). Almost all SDN controllers offer the capability to expose monitoring metrics, either via API calls or language bindings. In this respect, the OpenFlow-based architecture provides the capability to monitor all network elements in a uniform and vendor-agnostic manner.

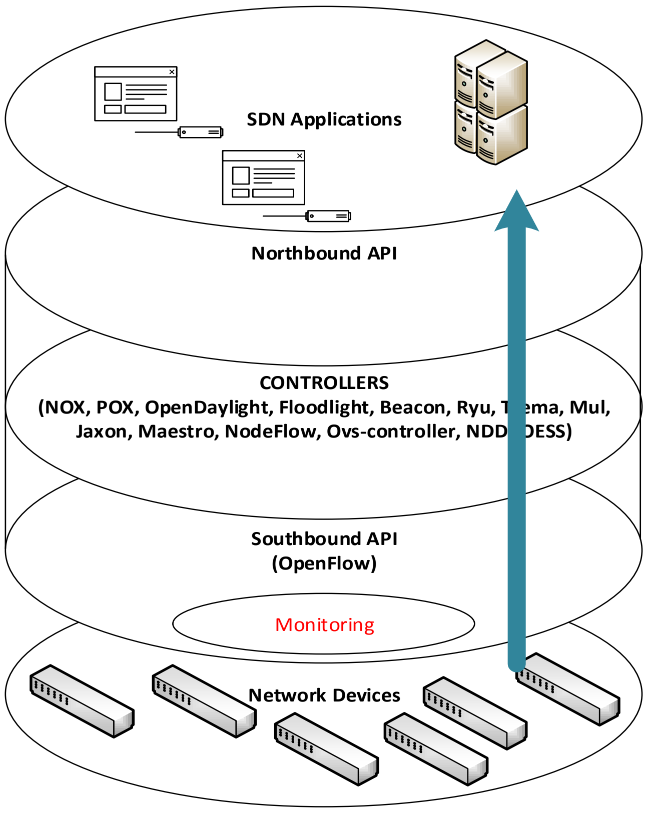

SDN Applications
Northbound API
CONTROLLERS
(NOX, POX, OpenDaylight, Floodlight, Beacon, Ryu, T ema, Mul,
Jaxon, Maestro, NodeFlow, Ovs-controller, NDD DESS)
Southbound API
(OpenFlow)
Monitoring
Network Devices

Figure 6.2 Communication of monitoring metrics in an OpenFlow-enabled architecture

In this context, several monitoring applications have been developed, leveraging OpenFlow capabilities for integrated network management tasks. Some of these applications are presented in Table 6.2.

Monitoring Application	Brief description	Controller Used	Open Source	Available at
OpenNetMon	OpenNetMon [50] continuously monitors all flows between predefined link destination pairs on throughput, packet loss and delay	POX	Yes	https://github.com/TU DelftNAS/SDN-OpenNetMon/
Payless	Payless [51] provides a flexible RESTful API for flow statistics collection at different aggregation levels. It uses an adaptive statistics collection algorithm that delivers highly accurate information in real-time without incurring significant network overhead.	POX, NOX, OpenDayLight	Yes	http://github.com/srcviru s/floodlight.

DCM	DCM [52] allows switches to collaboratively achieve flow-monitoring tasks and balance measurement loads.	None (native OF)	No	Not available
FlowSense	FlowSense [53] achieves a push-based approach to performance monitoring in flow-based networks, where the network informs of performance changes, rather than query it.	None (native OF)	No	Not available

Table 6.2 OpenFlow monitoring applications

In addition, many of the monitoring frameworks described in Section 6.3 for cloud infrastructures can be also used for monitoring OpenFlow infrastructures, via the appropriate plugins.

6.4. T-NOVA VIM Monitoring Framework

The overall architecture of the T-NOVA VIM monitoring framework can be defined by taking into account the technical requirements, as identified in Section 6.2, as well as the technical choices made for the NFVI and VIM infrastructure. The specification phase has concluded that the OpenStack platform will be used for the control of the virtualised IT infrastructure, as well as the OpenDaylight controller for the management of the SDN network elements.

In this context, it is appropriate to leverage OpenDaylight (Statistics API) and OpenStack (Telemetry API) capabilities for metrics collection, rather than directly polling the network elements and the hypervisors at the NFVI layer, respectively.

Theoretically, it would be possible for the Orchestrator to directly poll the cloud and network controllers of each NFVI-PoP and retrieve resource metrics respectively. This approach, although simple and straightforward, would fail to address the challenges outlined in Section 6.3 and in particularly would introduce significant scalability issues on the Orchestrator side. It is therefore appropriate to introduce a mediator/processing entity at the VIM level to collect, consolidate, process metrics and communicate them to the Orchestrator. This entity is called the *VIM Monitoring Manager (VIM MM)*, and acts as a stand-alone software component.

OpenStack and OpenDaylight already provide a rich set of metrics for both physical and virtual nodes, which should be sufficient for most T-NOVA requirements. However, in order to gain a more detailed insight on the VNF status and operation, it would be advisable to be able to connect a rich set of metrics from the guest OS of the VNF container, including information which cannot be obtained via the hypervisor. This enhanced monitoring capability is expected (yet still needs to be assessed) to significantly augment the effectiveness of the VIM MM with regard to status awareness and proactive fault detection.

For this purpose, it is decided to introduce an additional optional entity deployed within the VNF container, namely the *VNF Monitoring Agent (VNF MA)*. The latter will be optionally pre-packaged within the VNF image and will provide enhanced VNF monitoring capabilities. The monitoring agent will collect generic information about guest OS status, processes and resources, rather than VNF-specific information. The latter will be directly communicated by the VNF application itself to the VNF Manager.

Based on the need outlined above, the architecture of the VIM monitoring framework can be defined as shown in Figure 6.3.

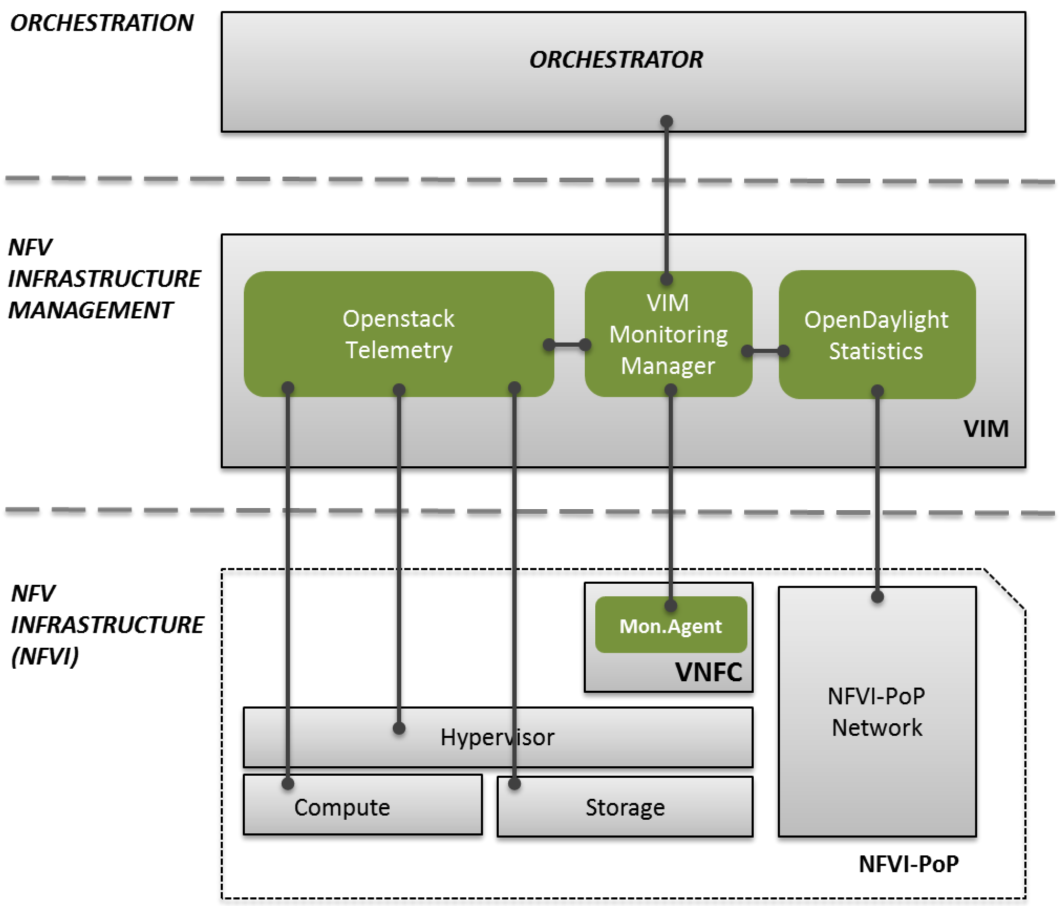


Figure 6.3 Overview of the VIM monitoring framework

The VIM MM aggregates metrics by polling the cloud and network controllers and by receiving additional information from the VNF monitoring agents, consolidates these metrics, produces events/alarms if appropriate and communicates them to the Orchestrator. For the sake of scalability and efficiency, it was decided that metrics will be pushed by the VIM MM to the Orchestrator, rather than being polled by the latter. Moreover, the process of metrics collection/communication and event generation can be partially configured by the Orchestrator via a relevant configuration service to be exposed by the VIM MM. More details on the introduced modules can be found in the following sections.

6.4.1. Overview of Collected Metrics

A crucial task when defining the T-NOVA approach to monitoring is the identification of metrics that need to be collected from the virtualised infrastructure. Although the list of metrics that are available via existing controllers can be extensive, it is necessary, for the sake of scalability and efficiency, to restrict this list to include only the information that is actually needed for the implementation of the T-NOVA Use Cases, as defined in Deliverable D2.1. This is only an initial tentative approach to the list of metrics, which will be continuously iterated on and updated throughout the project.

Domain	Metric	Units	Origin	Relevant UCs
VM/VNF	CPU utilisation	%	VNF Mon.Agent	UC3, UC4
VM/VNF	No. of VCPUs	#	VNF Mon.Agent	UC4
VM/VNF	RAM allocated	MB	VNF Mon.Agent	UC3, UC4
VM/VNF	RAM available	MB	VNF Mon.Agent	UC3, UC4
VM/VNF	Disk read/write rate	MB/s	VNF Mon.Agent	UC3, UC4
VM/VNF	Network Interface in/out bitrate	Mbps	VNF Mon.Agent	UC3, UC4
VM/VNF	Network Interface in/out packet rate	pps	VNF Mon.Agent	UC3, UC4
VM/VNF	No. of processes	#	VNF Mon.Agent	UC4
Compute Node	CPU utilisation	%	OS Telemetry	UC2, UC3, UC4
Compute Node	RAM available	MB	OS Telemetry	UC2, UC3, UC4
Compute Node	Disk read/write rate	MB/s	OS Telemetry	UC3, UC4
Compute Node	Network i/f in/out rate	Mbps	OS Telemetry	UC3, UC4
Storage (Volume)	Read/write rate	MB/s	OS Telemetry	UC3, UC4
Storage (Volume)	Free space	GB	OS Telemetry	UC2, UC3, UC4
Network (virtual/physical switch)	Port in/out bit rate	Mbps	ODL Statistics	UC2, UC3, UC4
Network (virtual/physical switch)	Port in/out packet rate	pps	ODL Statistics	UC3, UC4
Network (virtual/physical switch)	Port in/out drops	#	ODL Statistics	UC3, UC4

Table 6.3 Identification of metrics to be collected

6.4.2. VIM Monitoring Manager

The VIM Monitoring Manager is the core component of the monitoring framework. Taking into account its core functionalities and required features, the VIM MM should be composed of the following modules:

- *Agent connector* (Agent listener), to receive information from the VNF Agents;
- *OpenStack connector*, in order to poll for compute metrics (client for Telemetry API). The connector should periodically poll all deployed physical and virtual nodes for information. Efficiency can be improved if polling frequency is dynamically adjusted i.e. decreased for resources which are found to fluctuate less frequently;
- *OpenDaylight connector*, in order to poll for networking metrics (client for Statistics API). As with the OpenStack connector, the polling is periodic and refers to all network nodes, with an adjustable polling frequency;
- *Orchestrator connector*, which allows communication with the Orchestrator for:
    - Receiving monitoring configuration, e.g. registration to specific metrics, setting of thresholds etc. (server-side),
    - Dispatching (pushing) metrics and events in a push-based approach (client-side);
- L*ocal database* for storing metrics, configuration and any auxiliary information which requires persistence. The typical relational database model is considered adequate;
- P*rocessing engine*, which performs statistical processing on stored metrics, detects events and generates alarms;
- U*ser interface* for presenting collected information. Although a user GUI for the VIM MM does not directly serve any of the T-NOVA use cases, it is considered as a useful tool to monitor and manage the IVM infrastructure.

With regard to implementation, the initial approach for the VIM MM will be developed in JavaScript, under the node.js environment [46]. Node.js is a development platform providing an asynchronous event framework, designed to build scalable network applications. The reason is that the main functionality of VIM MM is data communication and the node.js environment offers several services to facilitate communication, especially via Web services, as well as event-driven networking.

6.4.3. VNF Monitoring Agent

The VNF Monitoring Agent will be pre-installed within the VM image hosting the VNFC. It will be automatically launched upon VNF start-up and run continuously in the background. The agent collects a wide range of metrics from the local OS. For this purpose, it is planned to exploit the collectd daemon, described in Section 6.3.1.2.

The VNF MA will be also assigned with monitoring heterogeneous compute nodes. Heterogeneous compute nodes comprise, apart from the common compute node hardware, also specialised hardware accelerators that can be exploited for some tasks to be performed within a VM. It is reasonable to also include performance metrics related to these devices in the T-NOVA monitoring architecture. Integration of the monitoring agent into the compute node depends largely on the nature of the compute node itself. In D2.31 we distinguished between two types of heterogeneous compute nodes: the first is composed by typical general-purpose processor (x86, ARM, etc.) whereas the second is constituted by nodes with special purpose acceleration resources, in the form, for instance, of GPUs, FPGAs and others. Incorporating the monitoring agent into compute nodes of the former types is notably easier, since the standard agent that will be developed within T-NOVA will be able to be executed on the processor of the system and communicate with the monitoring manager using standard methods. Compute nodes belonging to the latter category however will need to include a custom but compatible monitoring agent through some other means. For example in an FPGA-based system, the agent will have to be implemented in the programmable fabric of the FPGA and is a subject of future research.

It has to be noted that there can be no universal approach for the definition of the performance metrics to characterise the load of the application running on the programmable fabric of an FPGA or FPGA/SoC. The reason is that these metrics depend on the application currently running on the device and may vary from IOPS for a soft processor core to MAC/s for a DSP application. Thus, the monitoring framework has to accommodate also custom metric formats. Moreover, the Orchestrator and VIM have to be aware of these custom metrics upon VNF deployment. It is evident that the implications of deploying heterogeneous compute nodes are far-reaching and involve considerable effort.

In any case, for both generic and platform-specific metrics, the VNF MA will connect to the VIM MM and push measurements periodically. The push frequency will be configurable (either manually or automatically). The set of metrics (selection among all available ones) to be communicated will also be configurable and may vary among VNFs, according to VNF specificities.

6.4.4. Compute, Hypervisor and Storage Monitoring

Monitoring of computing, hypervisor and storage status and resources will be performed directly via the OpenStack Telemetry framework (see Section 6.3.1.1). The VIM MM (OpenStack connector) will periodically poll Telemetry for metrics regarding the currently deployed physical and virtual resources. Although these metrics could be retrieved by directly accessing the Telemetry database, since the scheme of the latter may evolve in future OpenStack versions, it is more appropriate to use the REST-based API offered by Telemetry [54] The VIM MM will issue GET requests to the service referring to a specific resource and meter, and the result will be returned in JSON format.

Fortunately, the Telemetry support for the hypervisor selected for T-NOVA (KVM) offers the widest possible list of available monitoring metrics, compared to other hypervisors, such as Xen or vSphere.

Moreover, collecting metrics via the API allows exploiting additional features of Telemetry such as:

- *Meter grouping:* it is possible to define set of metrics and retrieve an entire set with a single query;
- *Sample processing:* it is possible to define basic aggregation rules (average, max/min etc.) and retrieve only the aggregate instead of a set of metrics;

- *Alarming:* it is possible to set alarms based on thresholds for the collection of samples. An alarm can depend on a single meter, or a combination. The VIM MM may use the API to set an alarm and define an HTTP callback service to be called when the alarm has been set off.

6.4.5. Network Monitoring

Monitoring of network resources (on physical and virtual nodes) will leverage OpenFlow capabilities. Nevertheless, metrics will not be collected via the OpenFlow protocol (i.e. by polling directly the OF-enabled switches). Instead, as outlined previously, it will be achieved by exploiting the Statistics REST API of OpenDaylight. The VIM Monitoring Manager (OpenDaylight connector) will poll the ODL REST API, and in turn ODL will poll the underlying OpenFlow switches via its OpenFlow southbound plugin.

The OpenDaylight Statistics API [55], more specifically the StatisticsNorthbound service, exposes node and flow metrics to high-level network applications.

The VIM MM will issue GET requests to the service, referring to a specific node. The response, structured in JSON, contains metrics for each port, such as:

- number of received packets
- number of transmitted packets
- number of received bytes
- number of transmitted bytes
- receive/transmit drops
- receive/transmit errors (frame/overrun/CRC)

As aforementioned, for scalability reasons, per-flow statistics will not be collected, so the per-flow monitoring capability of ODL (also offered via the StatisticsNorthbound service) will not be exploited.

6.5. Conclusions

The current state-of-the-art of monitoring tools and frameworks for cloud computing environments has been presented. Infrastructure metrics and statistics currently available from the OpenStack and OpenDaylight controllers have been considered. These metrics relating to the resources of an NFVI-PoP must be exposed to the T-NOVA Orchestrator. A proposed VNF monitoring agent has been described which an optional component, responsible for collecting a rich set of metrics from within VNF containers. Finally the solution being developed by Task 4.4 will deliver visibility of the IVM status to the Orchestrator and the Marketplace.

7. TECHNOLOGY SELECTIONS

A key output of the Task 1-4 and in particular Task 4.1 is to identify the appropriate technologies that will be used to implement the T-NOVA IVM in Task 4.5. The following section contains a list of the initial candidate technologies that have been identified and the rational for their selection. These technologies are currently under evaluation in WP4. These selections however may change during the course of the activities within WP4 as technical issues arise or other viable technology options emerge. Final details of the technologies ultimately used to implement IVM will be provided in the next set of deliverables from WP4.

The following tables outline the selected technology for each function, the alternative options available at this point in time, the requirements that each technology satisfies (requirements are referred to the ones in Deliverable 2.31), the trade-offs and the justification (e.g. the rationale behind the selection).

Technology	**Cloud Controller**
Choice	▪ Open Stack Deployment and lifecycle management of the VNF/NS deployed on VMs within the Cloud Infrastructure. Provisioning of a common virtualisation layer across different heterogeneous platforms.
Alternatives	• CloudStack • Eucalyptus • VMware vCloud Suite
Requirements Related	VIM1, 2, 3, 4, 5, 6, 7, 10, 16, 20, 21, 22, 23 C2, 7, 9, 10 H1 N13
Trade-off	Currently no support for enhanced platform awareness External control of scheduling/filtering mechanism not supported. Deterministic behaviour difficult to achieve
Justification	Significant industry support with over 400 companies and organisation backing OpenStack Open source and community led development with active roadmap Adoption into major commercial NFV platforms e.g. Cloudband, NFV Director etc.

Technology	SDN Controller
Selection	▪ OpenDaylight Open-source SDN controller platform backed by the Linux Foundation and developed by an industrial consortium, which includes Cisco, Juniper and IBM, among many others. OpenDaylight is implemented in Java. Provides a flexible northbound interface using Representation State Transfer APIs (REST APIs), and includes support for the OpenStack cloud platform.
Alternatives	• POX • NOX • RYU • Floodlight • Trema • Jaxon • Maestro • ONOS • Contrail • Beacon • Nodeflow • MUL
Requirements Related	VIM3, 5 6, 7, 8, 9, 11, 13, 14, 16, 17, 18, 24 N6, 11, 12, 14
Trade-off	Implemented in Java which may have performance implications
Justification	Supports integration with OpenStack via ML2 plugin Supports OpenFlow, OVSDB, BGP-LS Support for VTN Strong roadmap

Topic	Network Tunnelling Protocols
Choice	• VXLAN – this technique used to encapsulate L2 frames in UDP packets. From a VM perspective, VxLAN offers the abstraction of L2 regardless of the physical location. Compared to VLAN, VxLAN permits a much wider addressing space by using a 24-bit LAN ID.
Alternatives	• VLAN • GRE • STT
Requirements Related	VIM6 8 N5 11 13 14
Trade-off	Small performance downgrade with respect to traditional VLANs, mainly related to encapsulation/decapsulation of VXLAN/GRE frames in the tunnel.

Justification	Tunnelling protocols are fundamental in a NFV environment in order to implement true network virtualisation, where tenants can be provision dynamically in networks (overlay) without any impact on the underlying physical infrastructure (underlay). The VXLAN tunnelling protocol was chosen mainly because it is quickly becoming a de-facto standard, with very large support in the IT industry. Major IT vendors (HP, Cisco, VMware, Red Hat etc.) natively support VXLAN in their products, both at virtualisation/cloud level (i.e. OpenStack and hypervisors) and at a HW infrastructure level (e.g. new HP switches integrate VXLAN gateways in hardware).

Topic	**Network Virtualisation Framework**
Choice	• VTN - OpenDaylight Virtual Tenant Network (ODL VTN) is a framework that provides multi-tenant virtual network on an SDN controller. As such it implements a logical abstraction plane that enables the complete separation of logical plane from physical plane. Networks for applications and end user needs can be deployed without knowing the underlying physical network topology.
Alternatives	• OpenDOVE. • OVSDB+VXLAN
Requirements Related	VIM2 7 8 13 14 N2 4 5 11 12 13 14
Trade-off	Even if the solution composed of OVSDB and the Network Tunnelling Protocol could be considered as a Network Virtualisation Framework, they actually operate at a lower level than VTN and OpenDOVE. The latter present instead a more comprehensive approach in term of multi-tenant network virtualisation. Both VTN and OpenDOVE are included into the OpenDaylight project list, where VTN seems to have a (little) more focus. This may be due to the importance of the VTN Manager implementation, i.e. the VTN component located at the "Controller Platform" level.
Justification	VTN Manager together with the presence of the already developed component at the "Network Application Orchestration and Services" level, i.e. the VTN Coordinator, would make VTN the most obvious choice for the Network Virtualisation Framework. Furthermore it is part of the OpenDaylight package which will ensure a seamless integration with the remaining components selected in this task. Open DOVE implements more modest functionality than ODL VTN and has considerably less industry support being an IBM-driven initiative. OVSDB on the other hand offers very limited functionality in comparison to ODL VTN.

Topic	**Distributed Controller**
Choice	• ODL Clustering Service - supports a cluster based HA model where several instances of ODL controller act as a single logical controller. The global state of the network in maintained through a distributed data store.
Alternatives	• Pratyaastha • ElastiCon
Requirements Related	VIM 3, 10, 17, 18
Trade-off	The adoption of distributed data structures may lead to an inconsistent or stale global network view. Higher rates of control synchronisation and communication overhead can help to achieve consistent state in the global network view. However, this may have an impact on the responsiveness of the system.
Justification	The choice was highly dependent on the selection of the SDN controller platform. The clustering service built in OpenDaylight represents a valid solution to support to high availability and horizontal scaling. Nevertheless, in order to fit the requirements identified in T-NOVA, further extensions and improvements are required.

Technology	**SDN Capable Switch**
Choice	OpenFlow version 1.0 or higher switch with 10GBase-T interfaces, TCAM and VLAN support
Alternatives	OpFlex
Requirements Related	N10, 11, 12
Trade-off	Cost of switch increases with speed OSF+ interfaces provide better environment, but are most expensive More expensive than non-SDN switches
Justification	Commercial 10Gb SDN switch with OpenFlow support, 10GBase-T physical interfaces to reduce connection costs. 10Gb switch considered standard for top of rack switching

Technology	**NIC**
Choice	• Intel X540 Converged Ethernet Adapter PCIe, dual port NIC with RJ45 connectors. Support for DPDK and SR-IOV (up to 64 virtual ports). Backwards compatibility with 1000Base-T networks.
Alternatives	Emulex - oce (OneConnect OCe14000 family) Mellanox - mlx4 (ConnectX-3, ConnectX-3 Pro)
Requirements Related	
Trade-off	Cable length limited to 50M More expensive than non DPDK NICs
Decision	Selection of X540 PCIe provides greater flexibility with workstations and server form factor compatibility. Full DPDK support.

Technology	**Virtual Switch**
Choice	• Intel DPDK vSwitch A virtual switch implemented on top of OvS, coupling the original software switching technology with DPDK in order to improve the performance of OvS, while maintaining its core functionality.
Alternative	Open vSwitch
Requirements Related	H14 N2, 5, 8
Trade-off	Requires DPDK compatible NIC Complex to configure
Justification	Open vSwitch is a production quality, multilayer virtual switch licensed under an Apache 2.0 open source license and it represents the de facto standard technology in terms of vSwitches. The DPDK version improves packet processing performance

Technology	**Compute Platform**
Choices	• Xeon E5 2620 v2/v3 Features 10 cores which can support up to 20 threads. Supports VT-x, VT-d, Extended page tables, TSX-NI, AES Instructions and Trusted Execution Technology (TXT) and 8GT/s Quick Path

	Interconnects for fast inter socket communications.
Requirements Related	C 1
Trade-off	General purpose processor solution for VNFs. Overprovision/under provision will be an issue with some VNFs. Overall performance can be limited by the performance of peripherals devices
Justification	Industry leading performance with extensive virtualisation support. Inclusion of key technologies to accelerate the performance of specific types of VNF e.g. cryptography. Large cache size to minimise context switching. Large number of cores gives greater flexibility

Technology	**Hypervisor and Hypervisor controller**	
Choices	• KVM and Libvirt KVM (for Kernel-based Virtual Machine) is a full virtualisation solution for Linux on x86 hardware containing virtualisation extensions (Intel VT or AMD-V) Libvirt is an open source API, daemon and management tool for managing platform virtualisation. It provides a very useful API for the orchestration layer of hypervisors in a cloud-based solution.	
Alternatives	**Hypervisor** • Xen • VMware ESXi • Microsoft Hyper-V	**Hypervisor Controller** • Xen Centre • VMWare VSphere
Requirements Related	VIM 1, 2 C 5 H 3, 4, 6, 7, 8, 10, 13, 14	
Trade-off	Lack extensive documentation in comparison to commercial products	
Justification	Open Source, Free, It is the default hypervisor within OpenStack, so it is fully implemented and supported, High performance, low overhead, extensive industry adoption.	

Technology	**Operating System**
Choices	• Fedora Fedora is a Linux-based operating system sponsored by Red Hat. Distributed under a free and open source license and aims to be on the leading edge of such technologies. Moreover, it is supported by Intel's Open Networking Platform.
Alternative	• Ubuntu • Windows Server
Requirements Related	
Trade-off	Occasional device driver compatibility issues.
Decision	Fedora better supports natively the virtual switching technology. It is Open Source, Free. It is supported by the ONP specification.

8. Conclusions

The activities in Work Package 4 are focused on developing an understanding of the performance characteristics of technologies to be used in the T-NOVA IVM layer. Task 4.1 is identifying appropriate mechanism for integrating these technologies and developing optimisations that are required to deliver a functional, manageable and performant IVM. A key component of the IVM is management of intra-Data Centre communications based on SDN principles. Task 4.2 is looking in detail at the design of the SDN Control Plane, including service chain traffic management, tenant isolation and the SDN controller architecture. Task 4.3, SDK for SDN task, is focusing on how to alleviate some of the key limiting factors for SDN application developers and data centre implementers through the development of a toolkit. It is envisaged that the toolkit will abstract the differences in popular SDN controllers allowing developers to use their language of choice and to reduce the development lifecycle. Management is an important aspect of the IVM. In order to manage the IVM appropriately, measurements with respect to how the environment is behaving are critical: here dynamic metrics play a critical role in providing the Orchestrator with the necessary information on the physical and virtualised infrastructure environment under its control. In this perspective, Task 4.4 is developing a monitoring solution that will be deployed onto the compute nodes that comprises the IVM layer and will report collected data to the T-NOVA Orchestration layer. Collectively, these tasks have also developed an understanding of their various task dependencies as outlined in Section 2 to ensure they receive appropriate input into their activities and their outputs match the needs of dependent tasks.

Section 3 described the current and planned activities of Task 4.1. The task has selected the initial candidate technologies for the implementation of the VIM functional entity within the IVM. OpenStack has been chosen as the Cloud Controller and OpenDaylight as the SDN Controller. The task has also selected some initial technology components that will be used in the implementation of the NFVI functional entity. These technologies are currently being evaluated in a testbed environment. Initial technology characterisation experiments have focused on DPDK and OVS. The results obtained to date indicate a significant improvement in packet processing performance for the DPDK version of OVS in comparison to the standard version. However the task has also identified that the VNF must natively utilised the DPDK libraries. If the VNF does not have native support the performance benefit is significantly reduced. In this scenario an improvement over OVS was observed in terms of throughput. Task 4.1 has developed an initial version of its workload and technology characterisation protocol which is included in this deliverable. The outputs of this work will play an important role identifying dynamic and static metrics that are most highly correlated with workload or system performance. The task is also developing a set of Best Known Methods (BKMs) for virtualised environment implementation which will be used by other work packages in T-NOVA.

In Section 3 the initial architecture of the SDN control plane based the requirements outlined in previous T-NOVA deliverables is described. In addition Task 4.2 have analysed and identified the functional components and interfaces of the CP. Analysis of a variety of candidate technologies has been carried out in order to identify a suitable solution for the SDN controller implementation based on a balanced view of the available and missing features. OpenDaylight was selected as the Network Controller with Virtual Tenant Network (VTN) as the multi-tenant network virtualisation framework and clustering service for the controller deployment in large-scale environments. Task 4.2 have developed an initial experimental plan to evaluate the performance of the selected technologies under a number of different scenarios and has initiated implementation of the test plan to collective quantitative data to evaluate the Controller architecture options i.e. single instance vs. distributed.

In Section 4 the activities of the Task 4.3, SDK for SDN, which is working on providing common development libraries (with support for multiple SDN controllers) to the SDN application developers were outlined. The task is also undertaking detailed analysis of interactions among virtual and physical network elements in a typical cloud data centre. The aim of this study is to identify bottlenecks in the network stack and to provide requirements for the development of specific libraries to support developers in minimises or removing such bottlenecks. Task 4.3 has created an initial draft architecture for the SDK which over the course of next couple of months will be further refined based on the close interactions planned with other tasks in T-NOVA. The two initial SDN controllers that are targeted by this task are OpenDaylight and Ryu - the final two controllers filtered by the T-NOVA consortium.

Task 4.4 focuses on the development of a monitoring framework for the IVM components. In Section 5 a brief state-of-the-art survey was presented and the architecture of the T-NOVA VIM monitoring framework was specified. Taking into account the use of OpenDaylight and OpenStack as the controller technologies in the VIM, infrastructure metrics and statistics available from these controllers will be collected. These metrics are aggregated and filtered into a centralised Monitoring Manager, which exposes status and resource information of the NFVI-PoP to the Orchestrator, as configured by the latter. A VNF monitoring agent was also introduced, as an optional component, collecting a rich set of metrics from within VNF containers. It is concluded that, with the proposed approach is technical feasible with a goal of delivering an effective, efficient and scalable monitoring solution for the T-NOVA IVM layer. The solution under development will be able to afford the Orchestrator and the Marketplace enhanced awareness of the IVM status and resources, while at the same time keeping the communication and signalling overhead at minimum.

In Section 6 the candidate technologies that will be used in by WP4 in the development, implementation and characterisation of the IVM and its functional entities were described. The rational for the selection of the technology is provided together with a mapping to the T-NOVA requirements that the technology will fulfil.

Each of the tasks will report again on their progress and their key finding/outcomes in task level deliverables which are due between months M21 and M28 of the project. Task 4.5 will also use the outputs of the tasks described in this deliverable to implement a functional IVM will be a realisation of the technology artefacts and key learnings generated by the WP4 tasks.

9. List of Acronyms

Acronym	Description
ACL	Access Control List
AES-NI	Advanced Encryption Standard New Instructions
API	Application Programming Interface
AVG	Average
BCAM	Binary Content Addressable Memory
BGP	Border Gateway Protocol
BGP-LS	Border Gateway Protocol Linkstate
BKM	Best Known Method
CLI	Command Line Interface
CP	Control Plane
CPU	Control Processing Unit
CRC	Cyclic Redundancy Check
DC	Data Centre
DOVE	Distributed Overlay Virtual Ethernet
DoW	Description of Work
DP	Data Plane
DPDK	Data Plane Development Kit
DSP	Digital Signal Processing
DUT	Device Under Inspection
EPT	Extended Page Tables
ETSI	European Telecommunications Standards Institute
FE	Functional Entity
FPGA	Field Programmable Gate Array
Gbps	Giga bits per second
GPU	Graphical Processing Unit
GRE	Generic Routing Encapsulation
HA	Hardware Abstraction
IaaS	Infrastructure as a Service

ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineer
IETF	Internet Engineering Task Force
IRF	Interchange Representation Format
I/O	Input/Output
IOPS	Input/Output Operations Per Second
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information Export
iSCSI	Internet Small Computer System Interface
IVM	Infrastructure Virtualisation and Management
IVSHMEM	Inter-Virtual machine Shared Memory
KVM	Kernel-based Virtual Machine
KPI	Key Parameter Indicator
L2	Layer 2
L3	Layer 3
LAN	Local Area Network
MA	Monitoring Agent
MAC	Media Access Control
MANO	Management and Orchestration
MF	Monitoring Framework
ML2	Modular Layer 2
MM	Monitoring Manager
MPLS	Multiprotocol Label Switching
NC	Network Controller
NF	Network Function
NFaaS	Network Functions-as-a-Service
NFV	Network Functions Virtualisation
NFVI	Network Functions Virtualisation Infrastructure
NFVI-PoP	NFVI-Point of Presence
NIC	Network Interface Cards
NS	Network Service
NUMA	Non-uniform Memory Access

NVGRE	Network Virtualization using Generic Routing Encapsulation
ODCS	OpenDOVE Server
ODGW	OpenDOVE Gateway
ODL	OpenDaylight
ODML	OpenDOVE Management Controller
OF	OpenFlow
ONF	Open Networking Foundation
ONP	Open Networking Platform
OPNFV	Open Platform for Network Function Virtualisation
OS	Operating System
OSGi	Open Service Gateway initiative
OVSDB	Open vSwitch Database Management Protocol
PCIe	Peripheral Component Interconnect Express
PF	Physical Function
PMD	Poll Mode Driver
PPS	Packets Per Second
QoS	Quality of Service
QPI	Quick Path Interconnect
QSFP	Quad Small Form-factor Pluggable
OSGi	Open Service Gateway initiative
RAM	Random Access Memory
REST API	Representation State Transfer API
RDMA	Remote Direct Memory Access
RFC	Request for Comments
RPC	Remote Procedure Call
SAN	Storage Area Network
SDN	Software-Defined Networking
SDK	Software Development Kit
SFP	Small Form Factor Pluggable
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SoC	System on Chip

SOTA	State-Of-The-Art
SR-IOV	Single Root I/O Virtualisation
SSD	Solid-State Disk
SW	Software
TCAM	Ternary Content Addressable Memory
ToR	Top of Rack
TNM	Transport Network Manager
T-NOVA	Network Functions as-a-Service over Virtualised Infrastructures
TXT	Trusted Execution Technology
UDP	User Datagram Protocol
vApp	Virtual Application
VIM	Virtualised Infrastructure Manager
VL	Virtual Link
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMDq	Virtual Machine Device Queues
VMM	Virtual Machine Manager
VMX	Virtual Machine Extension
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
vNode	Virtual Node
VPN	Virtual Private Network
vNIC	Virtual Network Interface Cards
VPN	Virtual Private Network
vNS	Virtual Network Service
VT-d	Virtualisation Technology for Directed I/O
VTEP	Virtual Tunnel End Point
VT-x	Virtualisation Technology for x86
VTN	Virtual Tenant Network
vTunnel	Virtual Tunnel
WAN	Wide Area Network

WP	Work Package
XFP	10 Gigabit Small Form Factor Pluggable
XML	Extended Markup Language
YAML	YAML Ain't Markup Language

10. References

[1] ETSI, "Network Functions Virtualisation; Part 1: Infrastructure Architecture; Sub-part 3: Architecture of Compute Domain," ETSI2014.

[2] DPDK. (2014). *DPDK: Data Plane Development Kit*. Available: http://dpdk.org/

[3] Intel. (2014). *Intel® Open Network Platform Switch Reference Design (Intel® ONP Switch Reference Design)*. Available: http://www.intel.com/content/www/us/en/switch-silicon/ethernet-switch-fm6764-controller.html

[4] Intel. (2014). *Intel® Open Network Platform Server Reference Design*. Available: http://www.intel.com/content/www/us/en/communications/open-network-platform-server.html

[5] QEMU. (2014). *Open Source Processor Emulator*. Available: http://wiki.qemu.org/Main_Page

[6] OpenStack.org. (2014). *Neutron/ML2*. Available: https://wiki.openstack.org/wiki/Neutron/ML2

[7] OPNFV. (2014). *Open Platform for NFV*. Available: https://www.opnfv.org/

[8] ONF. (2014). *OpenFlow*. Available: https://www.opennetworking.org/sdn-resources/openflow

[9] Cisco. (2014). *OpFlex: An Open Policy Protocol*. Available: http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html

[10] B. Salisbury. (2012). *TCAMs and OpenFlow – What Every SDN Practitioner Must Know*. Available: https://www.sdncentral.com/technology/sdn-openflow-tcam-need-to-know/2012/07/

[11] Open vSwitch. (2014). *Production Quality, Multilayer Open Virtual Switch*. Available: http://openvswitch.org/

[12] Intel, "Intel® Data Plane Development Kit (Intel® DPDK)," 2014.

[13] P. Bernier. (2014). *SDN OS from ON.Lab is Now Available for Download*. Available: http://www.sdnzone.com/topics/software-defined-network/articles/394771-sdn-os-from-onlab-now-available-download.htm

[14] The Apache Software Foundation. (2014). *Karaf*. Available: http://karaf.apache.org/

[15] S. Bradner and J. McQuaid. (1999). *Benchmarking Methodology for Network Interconnect Devices*. Available: https://www.ietf.org/rfc/rfc2544.txt

[16] Intel. (2014). *Pktgen version 2.7.7 using DPDK-1.7.1*. Available: https://github.com/pktgen/Pktgen-DPDK

[17] PUC-Rio. (2014). *Lua*. Available: http://www.lua.org/

[18] T. Nadeau and K. Gray, *SDN: Software Defined Networks*. Sebastopol, CA, USA: O'Reilly Media, 2013.

[19] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," presented at the Proceedings of the second ACM SIGCOMM workshop on Hot topics in Software Defined Networking, Hong Kong, China, 2013.

[20] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: towards an open, distributed SDN OS," presented at the Proceedings of the third workshop on Hot topics in software defined networking, Chicago, Illinois, USA, 2014.

[21] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69-74, 2008.

[22] Altor Networks. (2008). *Virtual network firewall for data centres*. Available: http://www.prosecurityzone.com/News_Detail_Virtual_network_firewall_for_data_centres_3584.asp#axzz3KLsGYNm6

[23] M. Ersue, "ETSI NFV Management and Orchestration - An Overview," in *IETF*, Vancouver, BC, Canada, 2013.

[24] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," presented at the Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Monterey, California, 2010.

[25] F. M. Facca, E. Salvadori, H. Karl, D. R. Lopez, P. A. Aranda Gutierrez, D. Kostic, and R. Riggio, "NetIDE: First Steps towards an Integrated Development Environment for Portable Network Apps," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, 2013, pp. 105-110.

[26] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: a network programming language," *SIGPLAN Not.*, vol. 46, pp. 279-291, 2011.

[27] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," presented at the Proceedings of the first workshop on Hot topics in software defined networks, Helsinki, Finland, 2012.

[28] F. Facca, E. Salvadori, H. Karl, D. Lopez, P. Aranda, D.Kostic, and R. Riggio, "NetIDE: First steps towards an integrated development environment for portable network apps," in *The Second European Workshop on Software Defined Networking (EWSDN'13)*, Berlin, Germany, 2013.

[29] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, Lombard, IL, USA, 2013, pp. 1-14.

[30] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," IETF2011.

[31] R. Narisetty, L. Dane, A. Malishevskiy, D. Gurkan, S. Bailey, S. Narayan, and S. Mysore, "OpenFlow Configuration Protocol: Implementation for the of Management Plane," in *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, 2013, pp. 66-67.

[32] OSGi Alliance. (2014). *OSGi™ - The Dynamic Module System for Java™*. Available: http://www.osgi.org/

[33] The Apache Software Foundation. (2013). *Supported Providers*. Available: https://libcloud.readthedocs.org/en/latest/supported_providers.html

[34] ZABBIX. (2014). *The Enterprise-class Monitoring Solution for Everyone*. Available: http://www.zabbix.com/

[35] Nagios Enterprises. (2014). *Nagios Is The Industry Standard In IT Infrastructure Monitoring*. Available: http://www.nagios.org/

[36] J. Gabès. (2014). *Shinken*. Available: http://www.shinken-monitoring.org/

[37] ICINGA. (2014). *ICINGA*. Available: https://www.icinga.org/

[38] Zenoss Inc. (2014). *Zenoss User Community*. Available: http://www.zenoss.org/

[39] Ganglia. (2014). *Ganglia Monitoring System*. Available: http://ganglia.sourceforge.net/

[40] Stacktach. (2014). *Event-based Monitoring & Billing solution for OpenStack*. Available: https://github.com/rackerlabs/stacktach

[41] HP. (2013). *Healthmon*. Available: https://github.com/stackforge/healthnmon

[42] Sealion. (2014). *Quickly Diagnose Problems with you Linux Servers*. Available: https://sealion.com/

[43] Caltech. (2014). *MONitoring Agents using a Large Integrated Services Architecture*. Available: http://monalisa.caltech.edu/monalisa.htm

[44] collectd. (2014). *collectd – The system statistics collection daemon*. Available: https://collectd.org/

[45] StatsD. (2014). *Simple daemon for easy stats aggregation*. Available: https://github.com/etsy/statsd/

[46] Joyent. (2014). *node.js*. Available: http://nodejs.org/

[47] Graphite. (2014). *A Highly Scalable Real-time Graphing System*. Available: https://github.com/graphite-project/graphite-web

[48] vmware. (2014). *vSphere*. Available: http://www.vmware.com/products/vsphere

[49] Amazon. (2014). *Amazon CloudWatch*. Available: http://aws.amazon.com/cloudwatch

[50] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, 2014, pp. 1-8.

[51] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for Software Defined Networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, 2014, pp. 1-9.

[52] Ye Yu, C. Qian, and X. Li, "Distributed and Collaborative Traffic Monitoring in Software Defined Networks," presented at the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'14), Chicago, IL, USA, 2014.

[53] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *Passive and Active Measurement*. vol. 7799, M. Roughan and R. Chang, Eds., ed: Springer Berlin Heidelberg, 2013, pp. 31-41.

[54] OpenStack.org. (2014). *Telemetry API v2 (CURRENT)*. Available: http://developer.openstack.org/api-ref-telemetry-v2.html

[55] OpenDaylight.org. (2014). *StatisticsNorthbound*. Available: https://jenkins.opendaylight.org/controller/job/controlller-merge-hydrogen-stable/lastSuccessfulBuild/artifact/opendaylight/northbound/statistics/target/site/wsdocs/resource_StatisticsNorthbound.html

11. Additional Contributors

CRAT - V. Suraci, F. Cimorelli, R. Baldoni