Document Properties

Document Number:	D 5.2
Document Title:	**Use Case Design : 1st release**
Document Responsible:	Kévin Phemius (THALES)
Document Editor:	Kévin Phemius (THALES)
Authors:	Kévin Phemius (THALES) & Diego López (TID) Pedro A. Aranda (TID) & Bernhard Schräder (FTS)
Target Dissemination Level:	PU
Status of the Document:	Final
Version:	1.0

Production Properties:

Reviewers:	Elisa Rojas (IMDEA) & Carmen Guerrero López (IMDEA)

Document History:

Disclaimer:

This document has been produced in the context of the NetIDE Project. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2010–2013) under grant agreement n° 619543.

Abstract:

This deliverable presents the first iteration of NetIDE's Use Case design defined in the Work Package 5 (Use Cases and Evaluation). This work aims at presenting NetIDE's three Use Cases in two aspects : their definition and their evaluation. The definition of the Use Cases will present in detail the motivation behind the necessity of NetIDE to solve real industrial issues and the evaluation will present how we will proceed to test the solution brought by the project. The deliverable expose our approach to rationalize the choice of the Use Cases and presents the methodology used to describe the Use Cases and their evaluation throughout the life of the project.

Keywords:

clouds, networking, Information-centric, future internet

Contents

List of Figures

List of Tables

List of Acronyms

DC Data Centre

DMZ Demilitarised Zone

DoW Description of Work

IDE Integrated Development Environment

IRF Intermediate Representation Format

NetIDE NetIDE

SDN Software Defined Networking

UC Use Case

API Application Programming Interface

NFV Network Functions Visualization

IS Integrated System

EHM End Host Mode

PoC Proof of Concept

Executive Summary

This deliverable presents the Use Cases defined in Work Package 5 (Use Cases and Evaluation) of the NetIDE project. The industrial partners have structured their practical work on the the following Use Cases (UCs):

- Vendor Agnostic Data Centre (DC) Evolution
- Integrated IT System
- Hybrid Environment Application Development

We present the design of the Use Cases and provide the evaluation criteria for the implementation later in the project. We discuss the methodologies used for the definition of the Use Cases and for the evaluation. The design cycle of the Use Cases is handled in an agile way and may vary any of them at a later stage of the project as a result of a change in the orientation of the work due to the evolution of the SDN scene, compatibility issues, change industrial target, etc.

As an appendix, we include a detailed State of the Art analysis of SDN techniques applicable and currently applied to Data Centres. The techniques analysed therein are directly impacting the implementation of UC1 and are also significant in the design and evaluation of UC2.

1 Introduction

This deliverable presents the Use Cases defined in the Work Package 5 (Use Cases and Evaluation) of the NetIDE project. Following the work in deliverable "5.1 - Use Case Requirements" [1], the industrial partners presented three preliminary Use Cases extracted from the DoW to evaluate NetIDE assertion in solving their issues. The three UCs presented in the Description of Work (DoW) are :

- Vendor Agnostic DC Evolution
- Integrated IT System
- Hybrid Environment Application Development

This deliverable will expose the design of the Use Cases as well as how they will be evaluated at a later point of the project's life. We will present the methodology used to describe the Use Cases and their evaluation. The deliverable is based on D5.1 for some of the concepts (e.g. the Use Case Requirements). The Use Cases may be modified during the next part of the project for various reasons (compatibility issues, changed industrial target, ...); the justification for such changes will be reported in the milestone M5.2 (Use Case Redesign) at M25 and a second version of this deliverable will be edited as D5.4 (Use Case Design - 2nd release).

The rest of the deliverable is structured the following way :

- Section 2 describes the methodology used to present the Use Cases in two parts :
  - Section 2.1 presents the methodology of the definition of the Use Cases.
  - Section 2.2 presents the methodology of the evaluation process of the Use Cases.
- Section 3 details each Use Case and their specific evaluation.
- Section 4 concludes this deliverable.
- Annex A contains a presentation of the State of the Art of SDN in data centers.
- Annex B provides an updated list of the Use Case Requirements.
- Acronyms are provided at the beginning of the document.

2 Methodology

This section describes the methodology used to gather NetIDE's Use Case definition and to evaluate the Use Cases.

2.1 Use Case Definition

The NetIDE development cycle is Use-Case driven, meaning that the Use Cases have had a strong influence on the definition of the architecture (Work Package 2) which in return had an effect on the definition of Use Cases (this process was called the "feedback loop" in previous documents). It is thus very important to ground the Use Cases in real world situations, and not abstract concepts. Along these lines, we ask each of our industrial partners to produce a Use Case demonstrating an industrial necessity of NetIDE's ideas and realizations. The resulting Use Cases hence fit different points of view on how NetIDE could be used to solve real and concrete industrial issues. These Use Cases encompass small and large scale deployments (data centers, company networks, ...), heterogeneous hardware (routing elements, firewalls, load-balancers, ...), and usage (regular network, mission critical networks, ...). To ease the comprehension of the Use Case, each partner had to describe it in three steps: a *rationale*, a *presentation*, and a *story*.

2.1.1 Rationale

The *rationale* presents the industrial necessity of the Use Case according to the partner supporting it. It will present a problem or a situation that was encountered, how the state of the art could not resolve the issue (or the shortcomings that prevented it from being applied), and how NetIDE would be the ideal tool in this context. The idea will be to show one or more aspect of NetIDE particularly suited to help the partner progress without investing in costly solutions (in capital/operational expenditure or manpower).

2.1.2 Presentation

The *presentation* section will outline the general view of the Use Case. It will be crucial to give a clear description of the Use Case without being too technical about the actual implementation, otherwise this can create confusion when trying to explain the idea behind the Use Case.

2.1.3 Story

The *story* will contain the actual, step by step description of the Use Case. It will for example contain the exact topology used, the configurations, the network applications and how each part of NetIDE will be used: the IRF, the simulator, the App. Engine, etc.

2.2 Use Case Evaluation

The evaluation section of each Use Case will show how they will be evaluated and how each attached Use Case Requirement will apply; as described in deliverable D5.1, each Use Case presented a list of requirements that NetIDE will need to address. In order to be validated, the Use Case will

need to provide *metrics* with quantifiable values that will assess if NetIDE effectively attained its objectives. It will also try to evaluate non-critical shortcomings: for example if a Use Case required from NetIDE to simulate 1000 network nodes in the emulator but for some reason NetIDE could only manage to get to 700 nodes, there need to be a possibility to accept this result without being caught in a pass/fail situation (see D5.1 for more information).
With these information, each Use Case will have a ballot similar to a "scorecard" that NetIDE will try to fill.

2.2.1 Metrics

Because each Use Case is different, they will have their own specific metrics to be validated. As described in deliverable D5.1, there are two possibilities to validate part of a Use Case:

- If the requirement is mandatory, the system *need* to achieve the condition required for success. This is a pass/fail situation with no recourse.
- If the requirement is optional, a trade-off is acceptable; it will be necessary to bound the performance achieved by NetIDE (e.g. establishing a satisfactory threshold to reach).

2.2.2 Use Case Requirement

Deliverable D5.1 established a list of requirements derived from the Use Cases (an updated list is available in Annex B). This section will go through each requirement with some specificity related to the Use Case.

3 Use Cases

This chapter presents NetIDE's Use Cases definition and method of evaluation. After a description of the Use Case and its motivation, we will introduce how it will be evaluated using metrics, quantifiable elements, and the relation with the Use Case Requirements (see Annex B).

3.1 Use Case 1: Vendor Agnostic Data-Centre Evolution

3.1.1 Definition

This Use Case explores how NetIDE can help Network Operators streamline their DCs by minimizing vendor lock-in and simplifying management and operational aspects in scenarios where DC-based infrastructures become pervasive and there is a need to deal with heterogeneity in both the temporal (technology migration) and spatial (distributed DCs) axes.

3.1.1.1 Rationale

DC-based infrastructures are becoming more and more relevant for network operators. They were initially applied to perform computing-intensive management tasks (Customer Relationship Management, corporate databases, ...) and soon became employed to implement services for their network control planes, like the Evolved Packet Core in mobile networks, their Internet support services (DHCP, RADIUS, ...) and value-added services like IPTV. With the advent of Network Functions Visualization (NFV), these infrastructures are becoming essential for the core business of network operators. SDN is an essential component in these infrastructures.

In this context, network operators have to necessarily manage heterogeneous and distributed DC infrastructures, and at the same time the criticality of the services supported by the these infrastructures requires a specific agility in migration procedures. NetIDE is in the position of supporting both aspects by decoupling the policies to be applied in the network implementation via the SDN program from the particular technology used to support it, irrespectively of being an open-source or vendor solution. Furthermore, NetIDE provides additional abstraction to support different configurations over a distributed DC infrastructure.

3.1.1.2 Presentation

A general architecture for a future network operator infrastructure heavily relying on network virtualization mechanisms is presented in the figure below. The figure depicts the essential vision of a converged architecture able to support the challenges of the evolution towards 5G networks.

All access mechanisms share a common and elastic IP edge that connects them to local points of presence, global service centres, and the whole Internet at large. All services, either localized or global, at any ot the network segments are provided by DCs deployed according to the network topology and the service provision patterns.

It is obvious that DCs will follow different infrastructure deployment patterns according to its goals, and it can be foreseen that even in the same segment different DCs may be at different stages of technology evolution. What is more, technology updates at any segment will require a strongly coordinated action.

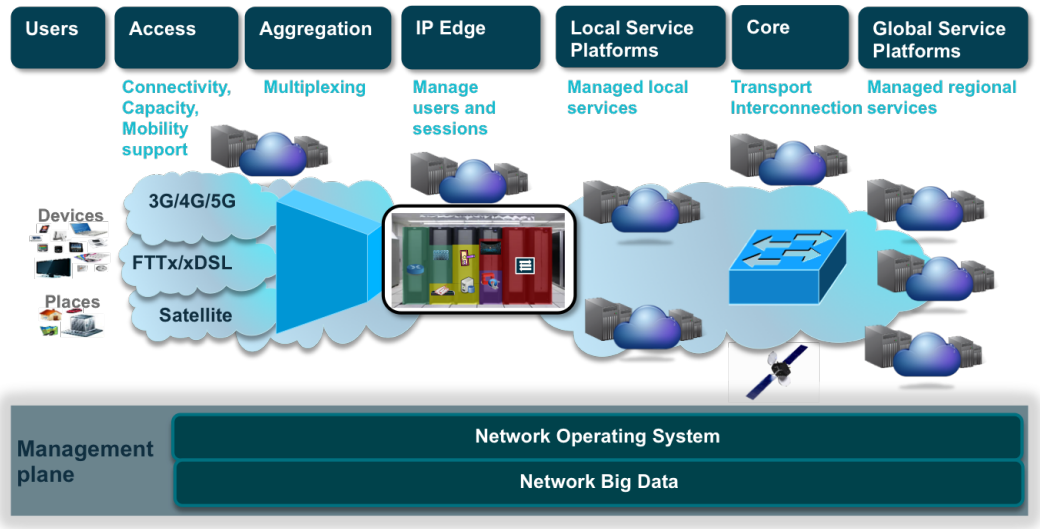


Figure 3.1: A converged virtualized network infrastructure

This Use Case intends to demonstrate how a common infrastructural network policy can be expressed according to a high-level specification of the topology and participation elements plus the corresponding SDN programs, expressed in the appropriate supporting formalism by means of the NetIDE input methods, and deployed over different SDN technology solutions after the necessary verification in a controlled environment provided by NetIDE. The deployment mechanisms provided by NetIDE will support further management and potential evolution decisions in the scenarios under consideration.

3.1.1.3 Story

For simplicity, the Use Case demonstrator will be circumscribed to the migration of a single DC, and consider a *canonical* topology with an unprotected open zone, a Demilitarised Zone (DMZ) and a protected intranet zone.

In an initial stage, the DC operator uses the NetIDE environment on a closed SDN-based DC. The main network application is provided by NetIDE. The switching infrastructure in conjunction with the SDN controller are assumed to provide a level of protection similar to that offered by a simple firewall.

In a second phase, the DC operator has to migrate the vendor-specific SDN network gear to an infrastructure based on an open-source SDN framework. The migration needs to be completely seamless. The user decides to start with an emulator-based environment and migrate it to a POX-based environment in a second stage.

In a last stage, the SDN environment must be rolled back to its initial state (because some problem has been detected in the migration, for example) and the DC operator uses NetIDE deployment facilities to recover the original state in a seamless way.

3.1.2 Evaluation

The use case will be evaluated in the TID experimental Software Networks lab, able to replicate several DC configurations, and to base its network infrastructure on virtual and/or physical OpenFlow switches. A change in the switches and the control framework will be applied to implement the story described above.

3.1.2.1 Metrics

The most relevant metrics for this use case are related to deployment times. In particular:

- The time required to validate the design in the new technology environment by means of the simulation facilities, and to deploy it.
- The time required for rolling back the DC network infrastructure to the original state.

3.1.2.2 Use Case Requirements

- **UCR.1**: The use case will use two different SDN control frameworks.
- **UCR.2**: The use case will make use of the abstraction capability to express the DC infrastructure network behavior to be preserved across migrations.
- **UCR.3**: The use case will require the capability of closely reproducing the production environment to accomplish the second phase described in the story section.
- **UCR.5**: The use case will rely on NetIDE deployment capabilities for its basic goals.
- **UCR.6**: The use case will require the integration with external frameworks used to define and operate the whole DC infrastructure.
- **UCR.7**: The use case will address multi-vendor coexistence and migration.
- **UCR.8**: The use case will use at least two different SDN control frameworks.
- **UCR.9**: The use case will require support for modular deployment and incremental testing.
- **UCR.10**: The use case will require large-scale simulation of the DC infrastructure network.
- **UCR.13**: The use case will attempt to reuse the maximum amount of code possible in the different technology scenarios to be migrated.
- **UCR.14**: The use case will use debugging and tracing capabilities across the migration phases.
- **UCR.15**: The use case will require different platform coexistence to support seamless migration in any direction.
- **UCR.16**: The use case will use at least two different SDN control frameworks.
- **UCR.17**: The use case will need to match policies against an abstracted network model.
- **UCR.18**: The Intermediate Representation Format (IRF) will be a crucial aspect of the project.
- **UCR.19**: The use case will rely on IRF definition to guide its deployment tasks.
- **UCR.20**: The use case will require, as an essential aspect, to avoid conflicts with different choices of controllers.

3.2 Use Case 2: Integrated IT System

3.2.1 Definition

This use case describes an Integrated System (IS) which will typically be deployed within a larger data centre. Looked at from the outside, from an infrastructure management perspective, the IS behaves as one computer with high processing capabilities. Looked at from the inside the IS is a network of closely collaborating computers.

3.2.1.1 Rationale

The use case addresses the customer need to rapidly introduce highly scalable computing infrastructure within an existing data centre environment. The solution described in the use case is encapsulated from sensitive network management processes in the rest of the data centre and can therefore be deployed without causing network management conflicts. Most existing server solutions expose more of their network management to the surrounding environment. It is hard to rapidly integrate many of them without disrupting the existing environment.

The solution described in the use case has its own internal network which must be designed, tested, deployed and maintained. The solution benefits from the clear separation of the control and forwarding planes which underlies SDN. NetIDE promises to deliver a development environment for SDN which should ease the development of this solution.

3.2.1.2 Presentation

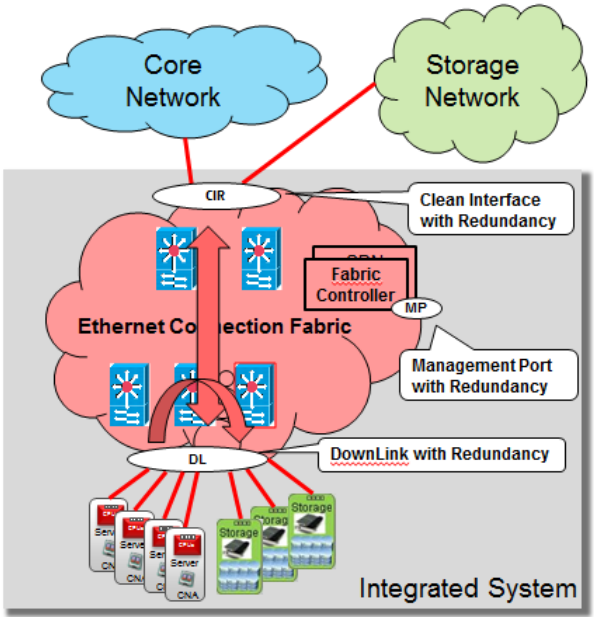


Figure 3.2: Integrated System-Overview

The IS will be used for general purpose computing under the cloud operational model and will support multiple tenants (separate users whose data must be strictly segregated). The IS has some key characteristics:

- It is very scalable. We currently see the sweet-spot configuration as having 128 physical servers. This can scale to over 1000. Smaller configurations are possible. Each physical server can support multiple virtual servers.

- The IO performance of the IS matches that of the individual servers. Dedicated bandwidth can be allocated to the servers ensuring that their IO can run at full speed. To achieve this IS manages internal congestion and load balancing.
- The IS automatically provides full redundancy of servers and their connectivity thus ensuring high availability of the system.
- The IS is easy to deploy and operate within an existing data centre infrastructure. In particular, using L2 connectivity there is no conflict between the control plane management of the overall data centre network and the control plane management of the IS internal network. Using a technology called End Host Mode (EHM) we isolate the IS network from the data centre network. The IS appears to the outside network as a computer with its own network end points. There are no spanning tree conflicts and there is no Link Aggregation Group (LAG) alignment. The IS can be managed separately from the rest of the data centre network, even by a separate team (e.g. the server management team). EHM is described in more detail below.
- IS is agnostic to the switching hardware and management systems. It can be implemented using hardware and software from any vendor. We believe that an implementation based on SDN controllers and switches can offer investment cost benefits and ease development. We believe that IS development supported by NetIDE can reduce Fujitsus development time, speed time to market and increase our flexibility to choose the best suited networking infrastructure.

IS as described here captures essential qualities Fujitsu wishes to implement within a simple design. This design is well suited to Proof of Concept modeling. Depending on further requirements, for instance integration into Cloud Management Platforms like OpenStack, the design of a commercial offering from Fujitsu may be different. However, we believe that the learnings from the design described here will still be valid.

3.2.1.3 Story

The Integrated System is a blade/rack level IT infrastructure which includes server systems, edge and aggregation switches and may include storage systems as well. Requirements of the Integrated Systems Network:

- Redundant clean uplink interface to the core network by behaving like a single host with multiple network adapters, thus no inter-switch protocols required, no switching between uplink ports.
- Optimized flow forwarding between the infrastructure components and external components (north/south traffic).
- Optimized flow forwarding between the infrastructure components (east/west traffic).
- Utilizes all active connections.

Fabric Management:

- Initial Fabric Setup, automatic discovery of network architecture
- Support of VM mobility
- Rules for default and alternate routing

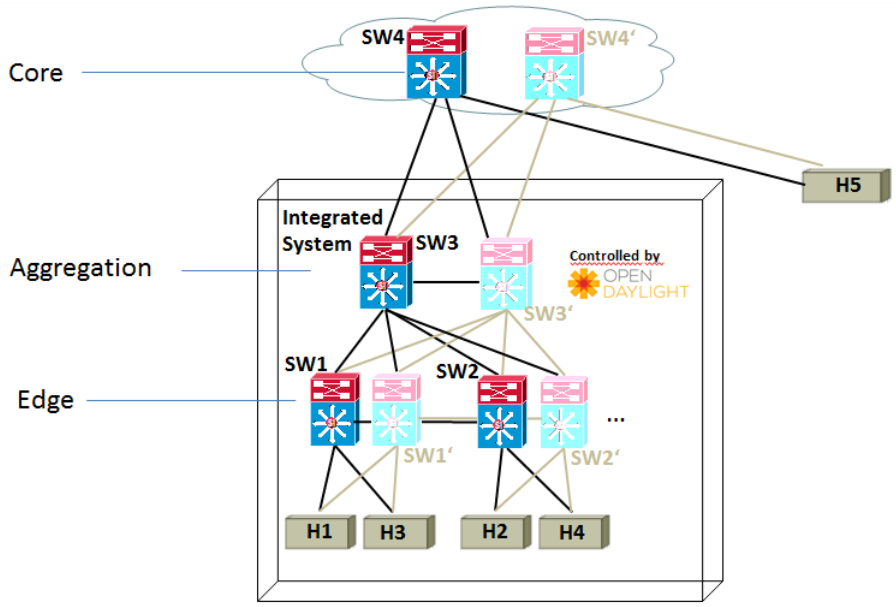


Figure 3.3: Principle of Integrated System Layout

- Prepared for uplink, downlink and switch device fabric redundancy

In order to ensure interoperability with any network vendor, the external network ports of the Integrated Systems does not represent / works like typical switch ports, but behave like network adapter ports. By that for the external data centre the IS is like a large server with multiple network adapters (End Hosts).

From data centre point of view the Integrated System shall look like a single Host System, it shall operate from data centre point of view operate in "End Host Mode".

End Host Mode: This means the integrated system is very easy and secure for L2-integration in any data centre networking environment. EHM does not interfere with the customer Spanning Tree Protocol (STP) domain nor does it introduce data loops. It has no STP implemented and avoids loops inherently as there is no packet forwarding between uplink ports. EHM does not burden additional effort for Link Aggregation (LAG) deployment by MAC-binding (see below).

Spanning Tree Protocol (STP) Avoidance: In EHM, STP is not implemented. No STP BPDU packets are sent by the Integrated System. This avoids STP interference with the customer network domain and enables using any network vendor specific protocols outside the IS. Data loops are not possible as there isnt any packet forwarding between Integrates System uplink ports. STP BPDUs sent by the upstream (customer) switch are disregarded and not passed-through to Integrated System servers.

Source MAC binding: By binding the server source MAC address to an uplink port, there is no LAG to the upstream customer LAN switch required anymore. MAC-binding automatically pins server MAC addresses to dedicated uplink ports. That means that all traffic from servers (MAC address) is sent through its dedicated pinned uplink port to the customer network. Source MAC-binding means that there is no LAG configuration required on the connections between Integrated System and the upstream (customer) switch. To make that work, the Integrated System filters traffic from servers (ingress downlink port) returning on its uplink ports to avoid duplicate broadcast/multicast/unknown unicast traffic to all other servers.

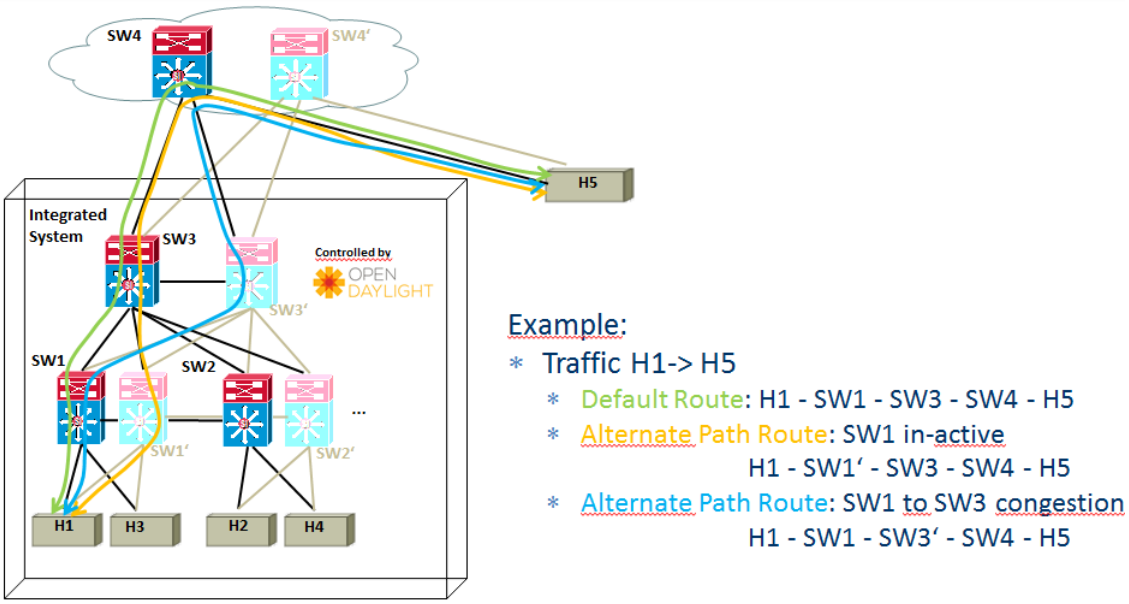


Figure 3.4: Behaviour of Elements (1)

3.2.2 Evaluation

The IS Proof of Concept (PoC) as described here serves three purposes:

- The PoC helps Fujitsu evaluate whether the IS design is a suitable basis for a commercial Fujitsu server offering.
- Implementing the PoC with SDN controllers and switches helps Fujitsu evaluate the suitability of SDN technology for the network implementation.
- Implementing the PoC with SDN technology using NetIDE tools lets us assess the productivity and portability benefits of NetIDE when applied to our use case.

3.2.2.1 Metrics

Congestion problems caused by Link overload or in-active becoming switches will impact the Integrated System fabric bandwidth and latency significantly. In case of load caused congestion problems on certain Integrated Fabric links or in case of in-active becoming Fabric switches, alternate path routing has to be setup to keep fabric bandwidth and latency impacts low. Congestion/routing issues shall be identified as quick as possible e.g. by switch health and switch in/out latency measurements. The setup of new routing paths shall become as fast as possible effective after congestion/routing issues have been identified. For a few corner case host initiated network traffic loads (Figure 4.4 and 4.5) it shall be shown that a network application for UC2 can quickly solve the packet routing issue in the fabric.
The identification of congestion/routing issues with the following setup of new routing path setup shall become effective in less than five seconds.

3.2.2.2 Use Case requirements

This section focuses on Fujitsus evaluation of NetIDE in the usage context of the IS PoC. The following topics should directly benefit from NetIDE as they are important application characteristics. Improvements brought by NetIDE are welcome but the primary evaluation criteria is that

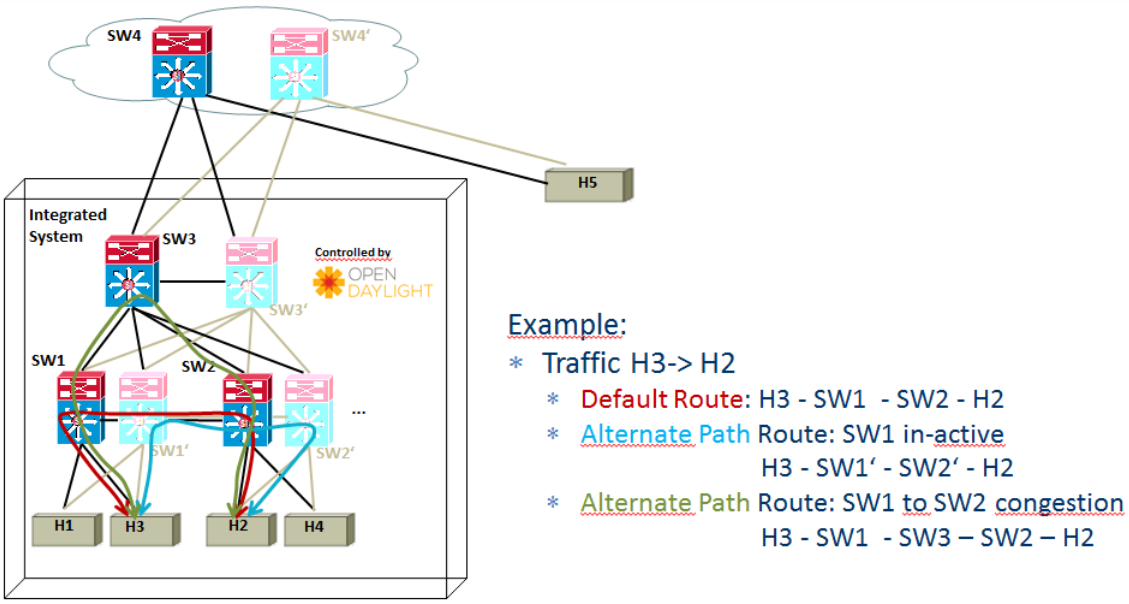


Figure 3.5: Behaviour of Elements (2)

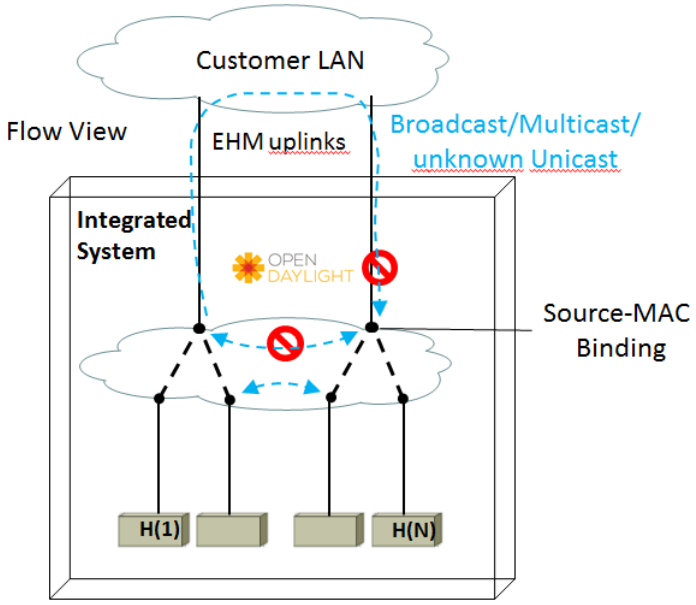


Figure 3.6: End Host Mode (EHM)

there is no degradation in *Performance*, *Scalability*, *Reliability*, and *Security*. We also expect to see improvements in *Coding*, *Debugging*, and *Troubleshooting*.

- **UCR.2**: Is more or less complex coding required in order to formulate the behavior of the application when using NetIDE vs alternative development environments?
- **UCR.3**: The use of the NetIDE development and runtime environment should not degrade the reliability of the application's behavior.
- **UCR.3**: Does NetIDE support and ease troubleshooting in live production environments?
- **UCR.5**: Does NetIDE ease the installation and deployment of applications?

- **UCR.9**: Is the testing of multiple version of modules supported?
- **UCR.11**: Will Ethernet fabrics be supported by NetIDE.
- **UCR.14**: Is debugging easier in the NetIDE environment due to its design advantages?
- **UCR.14**: Can we check the application latency or use of resources such as memory to not throttle the network.
- **UCR.14**: The use of the NetIDE development and runtime environment should not compromise the security of the application (E.g. by making it easier to inject malicious code into a benevolent application).
- **UCR.17**: Does NetIDE provide a full network topology picture (i.e. map, links, load, events, etc.)
- **UCR.18**: Will NetIDE provide a single internal representation form which covers the high-level ones?
- **UCR.19**: Will NetIDE support model checking and symbolic execution tools?
- **UCR.20**: Does NetIDE provide an abstraction for different controller interfaces, allowing to support writing applications that can run on different controllers?

3.3 Use Case 3: Hybrid Environment Development

The Hybrid Environment Application Development UC demonstrates how NetIDE can help developers to build, test and run network applications seamlessly in a single environment while keeping the user's preference in terms of language specificity or network equipment use.

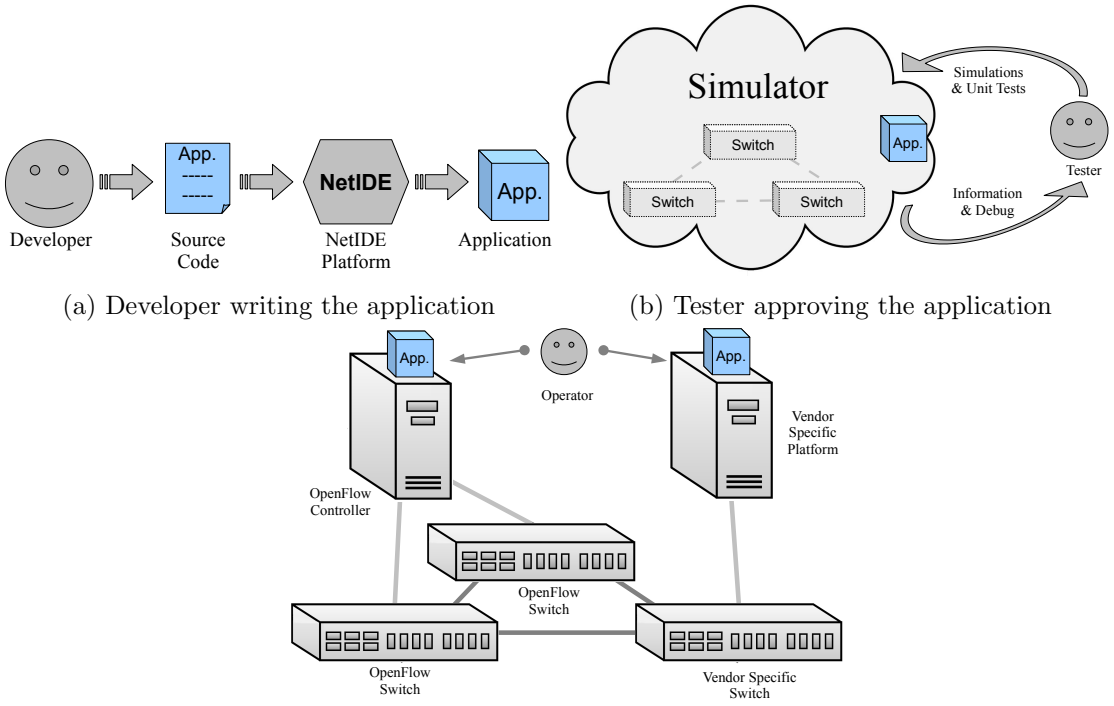
3.3.1 Definition

3.3.1.1 Rationale

In the current State of the Art, whenever a network application needs to be written and deployed, we have to face multiple languages, implementation and platforms for the code to run. The easy workaround would be to have uniform system throughout the network. But nowadays, companies may use by choice, or to reduce costs, have a heterogeneous environment made up of network equipment from multiple vendors.
The current solution is to train the workers on multiple systems or have them specialized in a single system (limiting their reach). In case an application need to be developed, we either have to write a complex, multi-system application or write a simple application but additional porting cost must be taken into account (one port per system).
NetIDE could help by having developers create the application only once, or even use legacy code, and use it directly on the target system. Furthermore, its testing environment could limit the number of bugs in production. NetIDE is thus a great solution in this scenario.

3.3.1.2 Presentation

The Use Case presents a process in which a company need to develop a network application and run it in production a two different network systems. As presented in the rationale, the Developer will need to write a complex application or a simple one. Using the NetIDE application development process, the Developer will write the application by either :



(a) Developer writing the application (b) Tester approving the application

(c) Operator running the application

Figure 3.7: Hybrid environment application development steps

- Using his development platform of choice.
- Using legacy code.

The multiplicity of choices is one of the key strength of NetIDE, providing ample freedom to the Developer.
After completing the application, the Tester will run simulation with NetIDE's simulator, using the reports from the debugger to correct errors.
The final step will be to run the application using NetIDE's App. Engine on multiple SDN platforms. By writing the code only once, the company is expected to reduce its expenditure and ease the deployment of application.

3.3.1.3 Story

The Use Case 3 will involve the development of a network monitoring application. In this story, the developer already wrote the application using the Floodlight SDN platform for the main office branch and the company wants to deploy it in a remote branch which uses OpenDaylight as a platform. Instead of rewriting the code from scratch, we will use NetIDE to run the application on OpenDaylight.
The Monitor application was written in java with Floodlight and in consequence uses primitives from the Floodlight library. The role of NetIDE will be to intercept these calls (as well as the OpenFlow calls) and translate them to the target platform (here, OpenDaylight).

The Use Case will not focus on the network forwarding but on the monitoring. The application allows a network administrator to check statistics such as link status, bandwidth use, latency, ... to perform traffic optimization.
The topology used in this Use Case if presented in Fig.3.8. We can see both the company Headquarters (HQ) and the Remote Branch (RB). The HQ network is driven by a Floodlight SDN

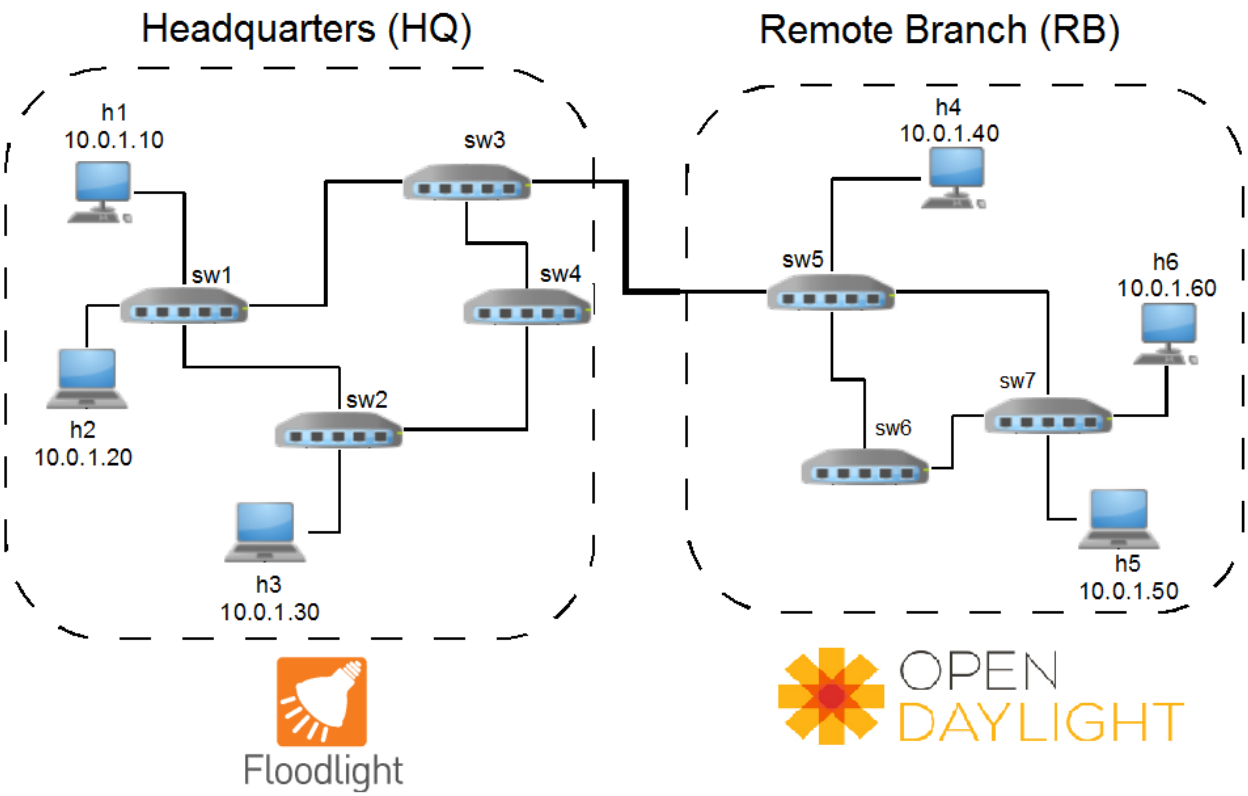


Figure 3.8: Topology used in the Use Case Hybrid Environment Application Development

platform while the RB is used with OpenDaylight.
The first step will be to use the code and adapt it to the NetIDE process. Here, we already have some code so the Monitor app. will be treated like a legacy application. Next, we will use the editor to mimic the network topology with mininet. The simulator will have the topology and the controllers matching the Use Case so that we can test the application behavior and get feedback from the debugger. If issues are reported, we will be able to correct them. The final step will be to deploy the application on a real network, seamlessly.
If all these steps can be validated, NetIDE could be used to deploy other legacy applications from the company as well as developing native, IRF-based code.

3.3.2 Evaluation

3.3.2.1 Metrics

The most relevant metrics for this use case are related to development time and complexity. In particular:

- 1) The time taken by the application development cycle to be completed.
- 2) The amount of code needed to be produced using the NetIDE ecosystem (IRF, definition files,...).
- 3) If any, the amount of modification in the legacy application source code.

3.3.2.2 Use Case Requirements

This section shows how the Use Case Requirements will be used to validate the Use Case, specially the mandatory ones (using words like "must" or "necessary"). It will also try to evaluate non-critical shortcomings by providing an explanation on the leeway we can accept.

- **UCR.1** : The Use Case 3 must operate on Floodlight and OpenDaylight.
- **UCR.2** : This component is essential so that the statistics and network state can be accessible to the Monitor application. The list of parameters required by the application is only a subset of the total normally available list. Only the useful parameters are required.
- **UCR.3** : This requirement is necessary for the testing part of the Use Case.
- **UCR.4** : The Developer will write the application with Floodlight. It's NetIDE's role to make the application run on Floodlight and OpenDaylight.
- **UCR.5** : The application will be deployed on a physical network. We must ensure that there are no damaging interaction with the network traffic.
- **UCR.8** : The homogenization is crucial for this Use Case. For example, in case OpenDaylight uses another southbound protocol than OpenFlow, the application must still have access to the same primitives.
- **UCR.9** : Testing multiple version would allow quicker development. For example by adding features in a development branch and being able to revert easily. If we can't do concurrent testing, we can still expect an easy way to swap application version within NetIDE.
- **UCR.12** : Depending on the controller platform, we could use multiple southbound protocols. For example, OpenDaylight can provide this functionality natively.
- **UCR.13** : This requirement is the key to the rationale of the Use Case. The application developed must be able to be deployed in the future on another branch of the company with another SDN platform seamlessly.
- **UCR.14** : The debugger is a must have aspect of application development during the testing phase to ensure an environment devoid of critical bugs.
- **UCR.17** : These levels of abstraction are necessary to allow the developer to focus on writing generic, portable code, instead of an application specific to a network topology or dataplane hardware.
- **UCR.20** : This requirement is important to ensure code portability and prevent conflicts.

4 Conclusion

This deliverable provided a clear and synthetic view of the design of the three Use Cases defined in the Work Package 5 (Use Cases and Evaluation) of the NetIDE project.
The deliverable presented our methodology to describe the Use Cases and build the approach to evaluate NetIDE through them. The Use Cases presented in the DoW :

- Vendor Agnostic Data-Centre Evolution
- Integrated IT System
- Hybrid Environment Application Development

are now clearly defined as is their evaluation method and metrics.

The main achievements reported in this deliverable include:

- Underlining the necessity to employ "real world" scenarios provided by the industrial partners in the development of NetIDE.
- Defining and using our methodology to gather and validate the Use Cases designs.
- Providing a detailed description of the Use Cases and the evaluation process.

The document also provided a detailed State of the Art on Software Defined Networking in data centres as well as an update on the Use Case Requirement list. There is a clear link between this document and some of the following deliverables because of the Use Case driven approach of NetIDE including: D2.2 for the IRF which will be used by the Use Cases implementation, D3.2 and D4.1 for the Developer toolkit and the App. Engine. D6.3 is also relevant because the partners adapted their exploitation plans with the modifications done in the Use Cases.
The next step will be to start implementing the Use Cases using the advances and the tools from the other Work Packages to start the evaluation. The result of this work will be reported in deliverable D5.3. We will also use the feedback from this work to adapt (if necessary) the Use Cases to a new direction. The result of the possible changes will be reported in milestone M5.2 and deliverable D5.4.

A State of the Art of Software Defined Networking in Datacentres

The cloud computing paradigm has allowed a new model for service delivery where the Information Technology (IT) resources deployed in Data Centres (DCs) form a pool able to attend multiple service demands by means of a dynamic assignment of resources, like CPU or storage capacity, either physical or virtual (by using some abstraction mechanisms). The computing resources can be provided on-demand depending on the user requests.

This elasticity on the resource consumption allows an agile adaptation to the business requirements and efficient resource utilization. In this multi-tenant model, the sharing of resources among users reduces costs and maximizes utilization, leveraging the economy of scale. Such computing resources are used not only for providing cloud-based service offerings (like storage, processing capabilities, etc) to external customers, but they are also dedicated to internal purposes, for instance, to explore the capabilities of virtualizing and distributing existing network functionalities.

Furthermore, the virtualization concepts are being extended to the network side. The virtualization in the IT side implies ubiquity, service independence of the real location, and then flexibility. The pure connectivity have to be revisited to consider new flexible infrastructures that can accommodate the dynamic demands required by the cloud by building virtual network infrastructures provided by third parties. These new infrastructures will be composed of distinct technologies built on top of a high capacity transport based solution able to provide enough bandwidth capacity in a dynamic manner. To address these challenges, DC infrastructures connecting to high-performance network infrastructures can adopt orchestration mechanisms allowing the convergence of DC (storage and processing) and network at both infrastructure level and service level.

Cloud services are changing telecom operator infrastructure. On-demand virtual machine creation and new services in the cloud enable the reduction of IT resources, but require to dynamically configure the network elements to optimally use their resources. The versatile consumption of resources and the distinct nature of the applications running on it will produce very variable traffic patterns on the connections reaching the data centres. The transport network becomes the key point to efficiently connect users to services and applications, which are now consumed independently of where either the resource or the user is located.

The flexibility provided by the cloud computing dynamically changes both the overlay service topology and the corresponding traffic demand, affecting the traditional planning and dimensioning rules of network operators. The location of the services is not any more tightly bound to a small number of nodes, topologically and timely changing. The network utilization becomes then time-varying and less predictable. The efficient integration of cloud-based services among distributed DCs, including the interconnecting network, becomes then a challenge to provide performance guarantees, localization and high availability properties.

A.1 Infrastructure Virtualisation

Server virtualization is a technology that partitions the physical machine into multiple Virtual Machines (VMs), each capable of running applications just like a physical machine. The virtualization technology allows a flexible management of IT resources, distributing them as needed for a certain

service either among distinct servers into a data centre, or even spreading them across several data centres connected to the network. By separating logical resources from the underlying physical resources, server virtualization enables flexible assignment of workloads to physical machines. This not only allows workload running on multiple virtual machines to be consolidated on a single physical machine, but also enables a technique called VM migration, which is the process of dynamically moving a virtual machine from one physical machine to another.

The software that controls the virtualization is called hypervisor. A VM then simulates the behavior of a physical, dedicated server (i.e., like a machine within the machine). It accesses to processing units, RAM memory, hard disks and devices. However, those elements are provided by the hypervisor, which is executed in the physical machine hosting the VM. Examples of hypervisors are Xen[2], KVM[3], VMware[4], etc. Any operating system can be installed on a VM, and the operating system is unaware that it is being executed on a virtual rather than a physical machine. The user programs interact with the operating system as usual. The hypervisor ensures that the VMs are isolated among them and have the necessary resources.

The virtualization of IT resources, which encompass processing units, storage devices, and even working memory, implies an on demand creation and configuration of virtual machines that provide a full computerization capability. In fact, service providers have to cope with cloud services delivered by more and more geographically distributed data centres, ever-increasing requests by users and DC providers for very high throughputs and low latencies, resource dynamicity and elasticity (i.e. flexible storage and computing on-demand, even from different DC) and seamless resource/service migration.

Cloud computing challenges related to performances of delivered services are heavily depending on network management and virtualization. Scalability is also an issue since, in contrast to a small number of warehoused-sized data centres (DC) commonly used in public clouds, telecom cloud must support a large number of small, distributed DCs to reduce traffic in the core network. A distributed DC architecture brings many benefits for network operators. By encapsulating workloads in VMs a cloud resource manager can migrate workloads from one DC to another looking for improving the perceived quality of experience (QoE), reducing energy consumption, or even in response to situations such as network failures or high-demand events. In addition, placing DCs closer to end-users enables the development of services and applications that can take advantage from very low latency.

A.1.1 Network Infrastructure Virtualisation

Existing ways of handling the network based on a huge number of manual interventions are not more valid, neither in the internals of the DC nor in the network providing the DC interconnects. Higher levels of automation and adaptation to connectivity demands are needed.

Apart of the mentioned flexibility required to adjust the network to the real conditions (e.g., in terms of bandwidth for tenant, or connection per physical destination), the DC interconnects need to scale in terms of supporting massive tenant demands. This is consequence of the increasing virtualization capabilities offered by the computing world, which allow for an efficient sharing of resources.

Multi-tenancy in the DC allows for the sharing of information resources among users. However the access to those resources should be implemented in such a way that it is achieved a similar level of privacy, isolation and reliability as if they were part of a per-user dedicated infrastructure. Additionally, the network providing the connectivity to the computing resources requires protection from external applications in the sense of not interfering with the operational procedures and the transported traffic of the rest of services (e.g., by not having IP address dependencies from the rest of the other services in the network).

There exist several methods and technologies for providing a logical separated network per tenant.

The basic concept is to set up an overlay network to separately transport the information between data centres, or from an access network to a data centre. The overlay network can be basically layer-3 or layer-2 based. This overlay network will typically interconnect nodes in the border of the DCs given access to an IP/MPLS (or even optical) wide area network. A virtualized networking environment then supports the coexistence of multiple virtual networks over the same physical infrastructure. The concept of multiple coexisting networks was supported first, in the context of Virtual Local Area Networks (VLANs) and Virtual Private Networks (VPNs).

A VLAN is a group of hosts logically brought together under a single broadcast domain. VLANs have become a widely used standard with well-defined use cases. The idea behind is that the traffic from the VMs of the different tenants within the DCs is segmented by using VLANs, in the more simplistic scenario, that later connect to a VPN. Unfortunately, VLAN-based solutions cannot meet the requirements for dynamicity, flexibility and scalability, needed both for configuration and proper operation of virtual networks. In addition, some researchers and industry vendors are attempting to adapt and extend existing network paradigms to accommodate new requirements brought by the virtualized use cases. Examples of such are vendor products, like Cisco's VN-Link, and network standards amendments, such as TRILL and 802.1Qbg.

A virtual private network is a dedicated communications network of one or more enterprises that are distributed over multiple sites and connected through tunnels over public communication networks, forming an overlay network. The VPN guarantees private communication between the end-points, constituting a Closed User Group (CUG). These VPN connections can serve complex connectivity requiring not only connectivity among the participants in the VPN but also external connectivity (e.g., to the Internet). Those scenarios of complex connectivity are typically built on top of layer-3 VPNs (L3VPN) where specific virtual routing functions (VRFs) are defined per each of the border routers (or provider edges, PEs) participant in the VPN service to properly route the IP packets interchanged within the VPN. The IP addressing within the VPN has to be unique (i.e., not duplicated), but different customers can use the same addressing scheme in different VPNs. On other hand, when simple layer-2 connectivity is needed in point-to-point or point-to-multipoint, layer-2 VPNs (L2VPN) can be configured, working in base of the MAC addresses instead of the IP addresses.

However, VPNs are too rigid, and cannot support a network virtualization environment where dynamics, flexibility and scalability are highly required attributes. Managed network VPN (e.g., BGP/MPLS), which represents a widely deployed network service for enterprises, is a significant example. This type of services has been conceived to work in a relatively stable network environment (which is the case with most enterprise networks today), but is not appropriate to cope with the typical dynamics of cloud services. The traditional VPN model is not able to handle essential cloud properties such as elasticity and self-provisioning, which means that those properties should be also extended to network resources. Quite often, expanding or reducing cloud resource capacity, or provisioning new cloud resources, requires a corresponding reconfiguration of network resources, e.g., bandwidth assigned between two data centres, whether they are in the same geographical place or not, or between the data centre and the end user. In order to cope with the cloud, future network services will certainly require on-demand and self-provisioning properties.

Today the network can provide static connectivity to cloud resources, to what we call conventional networking. The next evolutionary step is to make the network elastic and adaptable according to the cloud dynamics. It has become clear that the overlay based approach is the correct answer for achieving independency from the physical networking infrastructure. An overlay network can be created on top of an existing network, by generating logical communication links between hosts within the service domain. Overlay networks enable the design of modular networking protocols and services in which logical functions are separated from the underlying physical infrastructure. Multiple vendors have put significant efforts in order to achieve efficient overlay solutions built

upon different tunneling protocols. Recent solutions building upon overlays to achieve benefits of scale in multitenant virtual networks are VXLAN[5], NVGRE[6] and STT.

Among them VXLAN emerges as the most used technology. The basic concept behind VXLAN is the encapsulation of an original Ethernet frame on top of an UDP packet sent between two corresponding Network Virtualization Edges (NVEs). The role of NVE is played by the virtual switches where VMs are attached for internal communication in the DC. When it is required to get connectivity between DCs, the node in the border of the DC will play the role of NVE as well, and stitching that traffic to the inter-DC overlay network. VXLAN header includes a 24-bit long field named VXLAN Network Identifier (VNI) that allows per-tenant network differentiation. Then with VXLAN it is possible to create a virtualized end-to-end layer-2 network on top of a layer-3 overlay transport.

The major improvement provided by overlay networks is their separation from the underlying infrastructure, and from each other. This separation facilitates independent address spaces, ensures isolation, and allows different virtual networks to be managed by different administrators.

The conventional approach for connecting computing resources to the network uses network segmentation based on VLANs to separate end-user or tenant services. This way of segmenting the network provides a limited number of configurable networks per DC (determined by the theoretical 4096 available different VLAN tags), leading to a careful planning of resources either locally or remotely, when more than one DC is involved in a service. This approach has the burden of having to manually reconfigure multiple switches and routers every time a new service has to be deployed, creating inefficiencies and costs.

Multisite L2 services can be built under the concept of virtual patch-panels. In each location, ports on multiple switches (spread across the network) can be programmatically connected among them to set up extended point-to-point connections, in a dynamic and automated way. It is also important to implement mechanisms capable of providing a dynamic on-demand scaling (up and down) of the resources offered to those services, and capable of automatically propagating any network change that could affect those services.

The virtual patch panel function will consist on a stitching of the per-user generated VLAN in a DC with the corresponding pre-provisioned connection that connects with the remote DC. The same is done in the other end to build the end-to-end point-to-point layer2 connection.

A.1.2 Programmable and Software-Defined Networking (SDN)

The term *Network Programmability* refers to the capability provided by L2/L3 physical network elements to arbitrarily program their switching, forwarding and routing logic on-demand. In general, network programmability requires the separation of forwarding and control modules within the network elements, allowing the control logic to be developed as a set of separate custom software components.

Currently, the most popular paradigm for vendor-neutral network programmability is Software Defined Networking (SDN), a model for network control which separates the control and forwarding logic, migrating the traffic handling decisions from the network elements themselves to centralised software controllers. Conceptually, in SDN networks forwarding (physical) devices have minimal intelligence, while the control logic is implemented on top of a so-called SDN controller. The controller is a logically centralised entity which is responsible for a set of tasks, including the extraction and maintenance of a global view of the network topology and state, as well as the instantiation of forwarding logic appropriate to a given application scenario. In practice, the controller manages connections to all substrate network elements and installs, modifies and deletes forwarding entries into the forwarding tables of the connected switches by using protocol specific control messages.

This communication between controllers and network elements in SDN is commonly based on the OpenFlow protocol, which originated in Stanford university and is currently maintained by the Open Networking Foundation (ONF)[7]. To date, OpenFlow[8] is the dominant driving standard for SDN. Using OpenFlow, the Controller can dictate specific rules to SDN-enabled switches. These rules define whether flows which match specific characteristics should be forwarded, re-routed, altered, dropped or QoS-shaped.

While the OpenFlow protocol itself is quite low-level, several controller Application Programming Interfaces (APIs) have been made available in order to facilitate high-level programming of networking applications. What these controllers do, is to abstract the OpenFlow protocol to a programming language that the network application is written in. In this context, management applications for cloud networking can be easily developed using a common set of architectural patterns as well as common means to query data flows from one or more network elements and supporting framework functions.

The NOX[9] controller was the first widely available OpenFlow controller. Due to its early availability and its simplicity, NOX quickly become the de-facto reference design for OpenFlow controllers. As a result, it has been used to test new OpenFlow features, novel controller ideas and it has been employed extensively in research and feasibility studies. NOX applications called modules are implemented using the C programming language. NOX is event based; each module essentially consists of a collection of callback functions, triggered by the arrival of specific OpenFlow protocol messages. A spin-off of NOX called POX enables the use of Python for programming modules. While NOX/POX is extremely versatile, it is not primarily aimed for production use, as it is not optimised for performance and stability and lacks resilience features. Other controller frameworks aimed at deployment in production environments, include Beacon, Maestro, and Floodlight, all of which are implemented in Java.

Apart from the aforementioned frameworks, there also exist SDN management platforms, which are more complete in terms of the services they offer, so that they can be considered as integrated stand-alone solutions for the management of SDN infrastructures. Most of them also leverage the SDN capabilities in order to offer network virtualisation services, supporting multi-tenancy (often called *Network-as-a-Service*).

A.2 Management of Virtualised Resources

Cloud management platforms are integrated tools that provide management of cloud environments. These tools incorporate self-service interfaces, provisioning of system images, enabling metering and billing, and providing some degree of workload optimization through established policies. Through the self-service interface the user can request virtual infrastructure. This request is issued to a Cloud Controller, which provisions this virtual infrastructure somewhere on available resources within the DC. The Cloud Controller provides the central management system for cloud deployments

The most popular cloud management platforms include open source solutions such as OpenStack[10], CloudStack[11] and Eucalyptus[12] and commercial solutions from companies such as Microsoft and VMware. These solutions are briefly introduced here below, though OpenStack constitutes by far the most widely adopted one, close to a *de facto standard*.

A.2.1 OpenStack

OpenStack is a cloud OS that controls large pools of compute, storage, and networking resources throughout a DC, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. As an open source solution, OpenStack is developed and supported by a global collaboration of developers and cloud comput-

ing technologists. The project seeks to deliver solutions for all types of clouds by being simple to implement, scalable, and feature rich. The technology consists of a series of interrelated projects delivering various components for a cloud infrastructure solution. All OpenStack source code is available under an Apache 2.0 license.

OpenStack has a modular design that enables integration with legacy and third-party technologies. It is built on a shared-nothing, messaging-based architecture with modular components, each of which manages a different service; these services, together to instantiate an IaaS Cloud.

OpenStack has many different components/services. All this components/services includes an API to interact with each other. OpenStack APIs are open-source Python clients, and can run on Linux or Mac OS X systems. These components are:

- *Horizon*: service and administration Web-GUI dashboard.
- *Keystone*: identity (authentication and authorization) service for users, tenants and roles and also for the platform services.
- *Nova*: compute management service, which provides virtual servers upon demand. Includes nova-api as end-point for all API queries (EC2/OpenStack). It initiates most of the orchestration activities (such as running an instance) and also enforces some policy (mostly quota checks).
- *Neutron*: provide network connectivity between interface devices managed by other OpenStack services and network isolation too. It has different agents depending on the service to be provided (Neutron-L3-agents, Neutron-dhcp-agents, Neutron-metadata-agent, ...)
- *Glance*: provides a catalogue and repository for virtual disk images.
- *Heat*: provides orchestration services into the platform.
- *Ceilometer*: collects infrastructure and instances measurements and provides a monitoring system. It aims to deliver a unique point of contact for billing systems to acquire all of the measurements they need to establish customer billing, across all current OpenStack core components with work underway to support future OpenStack components
- *Cinder*: manages persistent block storage (data volumes) that could be attached to VM instances.

There are other components that are not part of OpenStack, but are part of the architecture:

- *MySQL*: it is not itself a logic element of the OpenStack platform but many OpenStack services make use of it (Keystone, Glance, Neutron, Nova, etc.). It stores most of the build-time and runtime state for a cloud infrastructure. This includes the instance types that are available for use, instances in use, networks available and projects.
- *QPID*: queue service implementation used by multiple OpenStack components for message exchange. The queue provides a central hub for passing messages between components.

A.2.2 Eucalyptus

Eucalyptus (Elastic Utility Computing Architecture Linking Your Programs To Useful Systems) is an open-source Cloud that provides on-demand computing instances and shares the same APIs as Amazons EC2 cloud.

Eucalyptus was designed as a highly-modular framework in order to enable extensibility with minimal effort. The Cloud Controller (CLC) in Eucalyptus acts as the Cloud entry-point by exposing and managing the virtualised resources. The CLC offers a series of web services oriented towards resources, data and interfaces (EC2-compatible and Query interfaces). In addition to handling incoming requests, the CLC acts as the administrative interface for cloud management and performs high-level resource scheduling and system accounting. The CLC accepts user API requests from command-line interfaces like euca2ools or GUI-based tools like the Eucalyptus Management Console and manages the underlying compute, storage, and network resources.

A.2.3 CloudStack

Apache CloudStack is open source software designed to deploy and manage large networks of virtual machines, as a highly available, highly scalable Infrastructure as a Service (IaaS) cloud computing platform. CloudStack is used by a number of service providers to offer public cloud services, and by many companies to provide an on-premises (private) cloud offering, or as part of a hybrid cloud solution. CloudStack is a turnkey solution that includes the entire "stack" of features most organisations want with an IaaS cloud: compute orchestration, Network-as-a-Service, user and account management, a full and open native API, resource accounting, and a first-class User Interface (UI).

CloudStack is a framework that allows pooling of computing resources in order to IaaS cloud services that can be used to provide IT infrastructure such as compute nodes (hosts), networks, and storage as a service to the end users on demand. CloudStack Management Server is the main component of the framework, consisting of managing resources such as hosts, storage devices and IP addresses. The Management Server runs on a dedicated host in a Tomcat container and requires a MySQL database for persistence. The Management Server controls allocation of VMs to hosts and assigns storage and IP addresses to VM instances. This component also controls or collaborates with the hypervisor layers on the physical hosts over the management network and thus controls the IT infrastructure.

A.2.4 VMware vCloud Suite

VMwares vCloud Suite is a comprehensive, integrated cloud platform for building and managing cloud environments. Tools for cloud management are delivered through VMware vCenter Server, a centralised and extensible platform for managing virtual infrastructure.

The tools included in the vCenter Server framework support: configuration of ESX servers and VMs, performance monitoring throughout the entire infrastructure, using events and alerts. The objects in the virtual infrastructure can be securely managed with roles and permissions.

A.2.5 SDN-based Network-as-a-Service Platforms

A.2.5.1 FlowVisor

FlowVisor[13] is the ON.LAB network slicer, which allows multiple tenants to share the same physical infrastructure. A tenant can be either a customer requiring his own isolated network slice; a sub-organisation that needs its own slice; or an experimenter who wants to control and manage some specific traffic from a subset of endpoints.

FlowVisor acts as a transparent proxy between OpenFlow switches and various guest network operating systems. It supports network slicing and allows a tenant or an experimenter to control and manage some specific traffic from a subset of end points. This approach enables multiple experimenters to use a physical OpenFlow network without interfering with each other. FlowVisor enables network virtualisation by dividing a physical network into multiple logical networks

ensuring that each controller touches only the switches and resources assigned to it. It also partitions bandwidth and flow table resources on each switch and assigns those partitions to individual controllers.

A.2.5.2 OpenVirteX

OpenVirteX[14] is a network hypervisor that can create multiple virtual and programmable networks on top of a single SDN-based physical infrastructure. Each tenant can use the full addressing space, specify their own topology, and deploy the network OS of their choice. Networks can be reconfigured at run-time, and OpenVirteX can automatically recover from physical failures.

OpenVirteX is actually a network hypervisor that enables operators to provide networks whose topologies, management schemes, and use cases are under the full control of their tenants. More specifically OpenVirteX builds on OpenFlow as protocol and FlowVisor for design. In this respect they share some common properties i.e. act as proxies between tenants and the underlying physical infrastructure. Unlike FlowVisor however, OpenVirteX provides each tenant with a fully virtualised network featuring a tenant-specified topology and a full header space.

A.2.5.3 Neutron

OpenStack Neutron[15], historically known as Quantum, is an OpenStack project focused on delivering Networking as a Service (NaaS), especially tailored for cloud environments. Neutron provides a way for organisations to make it easier to deliver networking as a service in the cloud and provides REST APIs to manage network connections for the resources managed by other OpenStack services.

It is designed to implement a "plugin" mechanism that will provide an option for network operators to enable different technologies via the Neutron API making it technology agnostic. However, currently Neutron is able to deliver all its core features only above an SDN-enabled infrastructure. Neutron provides native multi-tenancy support (isolation, abstraction and full control over virtual networks), letting tenants create multiple private networks and control the IP addressing on them, and exposes vendor-specific network virtualisation and SDN technologies.

As a result of API extensions, administrators and users have additional control over security and compliance policies, QoS monitoring and troubleshooting, the ability to build sophisticated networking topologies, as well as the ability to easily deploy advanced network services, such as a firewall, L2-in-L3 tunnelling, end-to-end quality of service support intrusion detection or VPN.

The core Neutron API includes support for Layer 2 networking and IP Address Management (IPAM), as well as an extension for a Layer 3 router construct that enables routing between Layer 2 networks and gateways to external networks. It is based on a simple model of virtual networks, subnet, and port abstractions to describe networking resources. Network is an isolated layer-2 segment, analogous to a VLAN in the physical networking world. More specifically, it is a broadcast domain reserved for the tenant that created it or explicitly configured as shared. Neutron includes a growing list of plugins that enable interoperability with various commercial and open source network technologies, including routers, switches, virtual switches and SDN controllers.

A.2.5.4 OpenNaaS

OpenNaaS[16] (Network-as-a-Service) is an open-source framework, which provides tools for managing the different resources present in any network infrastructure, particularly focusing on SDN infrastructures. The software platform was created in order to offer a neutral tool to the different stakeholders comprising an Open Access Network (OAN). It allows them to contribute and benefit from a common NaaS software-oriented stack for both applications and services. It is based on a

lightweight, abstracted, operational model, which is decoupled from actual vendors specific details, and is flexible enough to accommodate different designs and orientations.

A.2.5.5 OpenDaylight

OpenDaylight is currently the newest and also largest SDN controller/management platform. It is backed by the Linux Foundation and developed by an industrial consortium, which includes Cisco, Juniper and IBM, among many others. OpenDaylight includes numerous functional modules which are interconnected by a common service abstraction layer. Further, OpenDaylight provides a flexible northbound interface using Representation State Transfer APIs (REST APIs), and includes support for the OpenStack cloud platform.

OpenDaylight is built upon four *layers*:

- Technology-specific plug-ins, for managing SDN and non-SDN devices with various network configuration protocols.
- A Service Abstraction Layer, unifying the capabilities of the underlying technology-specific plug-ins.
- A core of basic network services, such as topology management, host tracking etc.
- A set of northbound APIs (REST-based) for communicating with network management applications.

A.3 Federated Resource Management and Orchestration

There are a variety of deployment models where a number of DCs interact in a distributed fashion. The typical cases are the private and the public cloud. In the private cloud the computing resources existing in the DCs are operated by an enterprise for its own purposes. In contrast, the public cloud is operated by a cloud service provider which offers computing capabilities to end users for commercial purposes. In both cases the resources are managed by a single organization which retains the control across the different DCs. Sometimes the private cloud can complement its assets by interacting with a public cloud When both models are simultaneously in place emerges the concept of hybrid cloud in which both private and a public cloud temporary cooperate for providing a given service.

Another model is the one offered by the federation of smaller DCs, named as community or federated cloud. The information resources are distributed in geographically dispersed locations, and are administrated separately, constituting different infrastructure domains. However, the federated DCs provide a management and control system common to the federation. Then, the creation and provision of a service from that system result in a number of control and management actions in each of the involved domains, interfacing with the specific control mechanisms present in each of the administrative domains.

The federation of DCs is a very relevant case because of the need for coordination to provide a consistent service between the heterogeneous environments existing in each of the DCs. For instance, the conventional mechanisms of segmenting the network are hard to scale considering the administrative silos defined for each infrastructure domain. The time required for providing a service increases when several actions need to be coordinated between domains. Additionally, every change to the service impacts on the separated administrative domains.

The conventional approach for connecting computing resources normally involves the burden of requiring a manual reconfiguration every time a new capability is deployed within a domain,

creating inefficiencies and extra costs in the chain. There is an evident deviation between business-level requirements (in terms of service connectivity and associated Key Performance Indicators) and the provision of the capabilities to be offered.

Although the situation is approachable when considering a domain controlled by a single infrastructure administrator, the methodology turns unfeasible in the case of configuring and operating a distributed environment, where multiple administrative entities are enclosed in a common federation, providing cross-domain services. Such heterogeneous environments can be highly benefitted from the existence of automated control mechanisms which can coordinate autonomously the resources present in every DC of the federation. This can allow performing unified service operation, control and management as it was operated by a single organization.

From the DC federation point of view, a common control point based on a single SDN controller is hard to achieve. The heterogeneity of the resources and the underlying technologies can compromise the scalability of such controller. Also the fact of having different administrative domains per DC prevents from delegating all the control of the infrastructure to an external entity.

However, it is yet required to keep an end-to-end view of the community cloud resources. The way of doing that is by means of orchestration capabilities on top of separated SDN control domains. The orchestration will allow conjugate resources from different infrastructures composing true end-to-end services while keeping the local complexities being handled by the SDN controller in each domain. The orchestrator will maintain the service awareness and will interact with each of the SDN controllers in the federation responding to the customers demands.

SDN controllers and cloud hypervisors should provide the orchestrator with an abstract view of the infrastructure. Technological details should be hidden and, in turn, the offered capacity should be characterized by Service Level Agreements parameters. All this would allow the orchestrator to efficiently allocate the available resource independently from the underlying technology Dynamic and automatic harmonization of heterogeneous domain that allows efficiently using the network resources. The effect of being software driven means that the underlying hardware resources are not directly visible or addressable, rather the resources are described and presented using abstract virtual elements.

Within a single administrative domain, the modules and components will have specific well-defined functionality, interacting with the other modules and components using task specific APIs. For the inter-administrative domain activity, there will be a set of these components that support various negotiation, control, and information exchange and functions migration operations between administrations. In most cases the cost of transport resources is more critical than the cost of computing and storage, thus the optimization of the resources should be considered.

From a networking point of view, four key services should be provided by the federation orchestrator:

- **Provisioning**. This capability enables the set-up, release and modification of connections in the network. Its most basic feature is to set up a point-to-point connection between two locations. However, there are other characteristics that a client interface can have such as excluding or including some nodes, defining the protection level, defining its bandwidth, or defining its disjointness from another connection.

- **Topology discovery**. This functionality requires unique identifiers for the exported network topology information. Network identifiers (such as IPv4 or datapath-IDs) help to carry out path computation and to integrate the nodes for an end-to-end scenario. Further, the local controllers can provide information about the links in the domain (physical or virtual) or even their utilization. It is clear that the more information is shared, the less abstracted the network appears. So it is important to highlight that the Northbound Interface (NBI) of the orchestrator should be more abstract than the NBI of the federated controllers.

- **Monitoring**. To collect the status of the connections that have been created is very useful in a multi-controller scenario, where after a failure in one domain, the domains controller may request another connection.
- **Path computation**. Global path computation is an important feature because individual controllers in each domain are only able to share abstracted information that is local to their domain. An orchestrator with its global end-to-end view can optimize end-to-end connections that individual controller cannot configure. Without a path computation interface, the orchestrator is limited to carrying out a crank-back process.

B Use Case Requirements List (Updated)

Following the recommendations from the M6 review, the Use Case Requirement list was updated to reflect the perceived shortcomings of the D5.1 document. This new list was internally reviewed by the partners. The Use Case Requirements were gathered in the fist phase of the project. Each of them has an identifier that will be used consistently throughout every deliverable. Due to the nature of the Use Case driven approach of NetIDE, some of them are bound to change (be it addition, modification or deletion). These changes will be reflected at the beginning of the subsequent deliverables.

UC Requirement	UCR.1
Role	Operator
Necessity	High
Description	As an Operator, I need NetIDE to be able to operate on multiple SDN platforms.
Short explanation	This requirement ensures that NetIDE application can be deployed on multiple platforms and Network Operating Systems such as Floodlight, OpenDaylight, OnePK... A minimum of 2 or 3 platform is necessary.
Affected Component	Network App Engine (Network Drivers Application Programming Interface (API))
Relevant UC	UC1, UC3

UC Requirement	UCR.2
Role	Application Developer
Necessity	High
Description	As an Application Developer, I need a high-level abstraction of every component of the network, such as switches, statistics, links, ports, flows,...
Short explanation	This requirement makes sure that the developer can have a correct view of the network to program it efficiently. The model must be clear enough so that every element can be accessed logically (like in a tree form).
Affected Component	Development Toolkit (IRF Transformer API, Integrated Development Environment (IDE))
Relevant UC	UC1, UC2, UC3

UC Requirement	UCR.3
Role	Systems Tester
Necessity	High
Description	As a Systems Tester, I need NetIDE to be able to closely reproduce a production environment either by emulating or by simulating the network conditions.
Short explanation	This requirement is crucial for testing. The simulated/emulated environment must reproduce networks conditions as real as possible. This includes, but is not limited to, virtual switches implementing OpenFlow, legacy switches, fake traffic/data, scalability tests ,... This requirement enables an Application Developer to thoroughly test his application, improving the application's reliability.
Affected Component	Network App Engine (Emulator / Simulator)
Relevant UC	UC1, UC2, UC3

UC Requirement	UCR.4
Role	Application Developer
Necessity	Low
Description	As an Application Developer, I would like NetIDE to make sure that I don't have to worry about the environment in which the application will run (similar to the Java Virtual Machine).
Short explanation	In the Hybrid Environment Application Environment Use Case, the developer should (ideally) not know on which platform the code will be used if the abstraction is in a level high enough.
Affected Component	Network App Engine (Network Driver API) Developer Toolkit (IRF Transformer API)
Relevant UC	UC3

UC Requirement	UCR.5
Role	Operator
Necessity	High
Description	As an Operator, I need NetIDE to let me safely deploy and run the application in a production network.
Short explanation	When deploying a new application, the Operator shouldn't have to worry about compatibility or reconfigure the entire system. Ideally, it would be something akin to a zero-configuration deployment.
Affected Component	Network App Engine (Network Drivers API)
Relevant UC	UC1, UC2, UC3

UC Requirement	UCR.6
Role	Module Developer
Necessity	High
Description	As a Module Developer, I need NetIDE to provide me a software development framework which is modular, extensible and provides well designed interfaces in order to link my module with external frameworks (such as REST).
Short explanation	This requirement enables a Module Developer to easily start the implementation of a new module by "inheriting" the required interfaces and implementing new, independent functionality.
Affected Component	Network App Engine , Developer Toolkit (IDE)
Relevant UC	UC1

UC Requirement	UCR.7
Role	Module Developer
Necessity	High
Description	As a Module Developer, I need NetIDE to provide the mechanism to build generic and reusable network service functions.
Short explanation	This means that my function (e.g. Statistics collector from Switches) will be able to talk many vendor languages exploiting (but not being restricted by) the appropriate drivers or standard protocols.
Affected Component	Network App Engine , Developer Toolkit
Relevant UC	UC1

UC Requirement	UCR.8
Role	Operator
Necessity	High
Description	As an Operator, I need NetIDE to enable the interaction of all the core network services with more than one Controller platforms in a unique way.
Short explanation	This requirement is crucial to enable the coexistence of an OpenFlow and a proprietary SDN Controller under the same umbrella, which is important for network operators towards trying to homogenize the legacy hardware with the new generation devices.
Affected Component	Network App Engine (Network Driver API)
Relevant UC	UC1, UC3

UC Requirement	UCR.9
Role	Systems Tester
Necessity	Low
Description	As a Systems Tester, I would like NetIDE platform to enable a modular deployment of services for testing purposes that will be also able to support multiple versions of the modules under testing.
Short explanation	This requirement implies that the tester will be able to deploy only the required context modules in order to test a particular unit/module. Also, while constantly updating a module, it will be able to coexist with previous versions of the same one.
Affected Component	Network App Engine (Simulator)
Relevant UC	UC1, UC2, UC3

UC Requirement	UCR.10
Role	Operator
Necessity	Normal
Description	As an Operator, I would like NetIDE to provide large scale simulations for SDN-based DCs that will guide the development phase of real software.
Short explanation	In the Vendor Agnostic DC Evolution Use Case, this is an important phase before the actual deployment. The proprietary DC solution currently in use should be translated into different SDN controller platforms. The simulator should aim to replicate a small DC sized network in the order of a thousand nodes.
Affected Component	Network App Engine (Simulator)
Relevant UC	UC1

UC Requirement	UCR.11
Role	Application Developer
Necessity	Normal
Description	For Network Application Developers, NetIDE should support the development of network applications that implement Ethernet fabrics, which are the central connecting element in integrated systems consisting of networks, servers and storage.
Short explanation	In the Integrated IT System Use Case, we need to develop IT solutions based on SDN concepts for integrated environments. Sometimes knowledge of the physical infrastructure can lead to optimization.
Affected Component	Network App Engine , Developer Toolkit
Relevant UC	UC2

UC Requirement	UCR.12
Role	Module Developer
Necessity	Normal
Description	As a Module Developer, I would like NetIDE to provide me a set of drivers to translate the IRF code stemming by the application layer into at least two SDN controller configurations (OpenFlow + Proprietary SDN controller).
Short explanation	In the Hybrid Environment Application Environment Use Case, we need to add a new application X in a dual domain infrastructure (OpenFlow + Proprietary SDN controller) without coding two different platforms.
Affected Component	Network App Engine (Interpreter)
Relevant UC	UC3

UC Requirement	UCR.13
Role	Application Developer
Necessity	High
Description	As an Application Developer, I need NetIDE to provide me an easy way to reuse code of my applications and run them in as many platforms as possible.
Short explanation	Map the same IRF code to different platforms. This improves the productivity, avoiding wasted time in rewriting applications for every used SDN platform. “Develop once, deploy everywhere”.
Affected Component	Network App Engine (Interpreter, Network Drivers API)
Relevant UC	UC1, UC3

UC Requirement	UCR.14
Role	Application Developer
Necessity	High
Description	As an Application Developer, I need NetIDE to provide me tools like debuggers and profilers to see the behaviour of the applications.
Short explanation	This requirement enables an Application Developer the possibility to check the application behaviour. The performance of the application can be tracked (e.g. to see memory usage, complexity of a program, etc.) with the profiler and with the debugger the different variables can be visualized to determine some problems that may occur. Those tools should work both in simulator and production.
Affected Component	Network App Engine (Debugger)
Relevant UC	UC1, UC2, UC3

UC Requirement	UCR.15
Role	Module Developer
Necessity	High
Description	As a Module Developer, I need NetIDE to provide me an easy way to extend the network app engine to deploy on other platforms and to write drivers.
Short explanation	This requirement is a must in order to different platforms can coexist in the same environment, which is important towards homogenize different OpenFlow and proprietary SDN Controllers to be as universal as possible.
Affected Component	Network App Engine (Network Drivers API)
Relevant UC	UC1

UC Requirement	UCR.16
Role	Operator
Necessity	Normal
Description	As an Operator, I would like NetIDE to provide me an abstraction to run my app in two different controllers in the same context/application.
Short explanation	This requirement implies that network app engine should know how to distribute parts of code among the different controllers. It is necessary that two controllers coexist in the same application and the IRF logic should know all to distribute the different parts. Everything should be transparent for the developer. It's only necessary to write an application that uses two controllers.
Affected Component	Network App Engine
Relevant UC	UC1

UC Requirement	UCR.17
Role	Application Developer
Necessity	High
Description	As an Application Developer, I need NetIDE to provide me a single and generic interface to interact with a single network controller entity.
Short explanation	This requirement provides the appropriate level of abstractions to Application Developers in order to build generic and reusable network management components for different platforms. The interaction should be bidirectional; an interface that gives to the Application Developer the full network topology picture (i.e. map, links, load, events, etc.) and an interface that enforces high level management tasks to the controller (i.e. apply a new, big policy, update a network domain).The physical topology structure and the diversity of dataplane hardware should be abstracted.
Affected Component	Network App Engine
Relevant UC	UC1, UC2, UC3

UC Requirement	UCR.18
Role	Module Developer
Necessity	High
Description	As a Module Developer, I need NetIDE to provide me a single representation form (IRF) of all the different programming languages and styles that Application Developers use.
Short explanation	This requirement abstracts the programming choices of Application Developers in the Core Controller platform. The modules should be able to understand a single language that covers the high-level ones.
Affected Component	Developer Toolkit
Relevant UC	UC1, UC2

UC Requirement	UCR.19
Role	Module Developer
Necessity	Normal
Description	As a Module Developer, I would like NetIDE to provide me a single language (IRF) to develop model checking and symbolic execution tools.
Short explanation	This requirement is crucial to avoid building debugging tools that are tailored to specific languages and controllers.
Affected Component	Developer Toolkit
Relevant UC	UC1, UC2

UC Requirement	UCR.20
Role	Operator
Necessity	Normal
Description	As an Operator, I would like NetIDE to provide me an abstraction to avoid any conflicts with controllers.
Short explanation	This requirement implies that network app engine should know for example that we can have two controllers than run the same application, but if for any reason one of the controllers fails, the network app engine should be able to switch to the another controller to try working. This is very useful in deployment when I have my solution and I want change to another platform. So if a moment of the deployment the new app goes wrong, the 'live network' doesn't break because the network app engine changes to my old application that works for me. Network App Engine should differentiate applications: application that can map with different controller (configure priority) but it's the same application and application that uses two controllers but they are complementary, which means that the application needs to use two controllers to work.
Affected Component	Network App Engine
Relevant UC	UC1, UC2, UC3

Bibliography

[1] The NetIDE Consortium. NetIDE deliverable 5.1 - Use Case Requirements, Jun 2014.

[2] The Xen project hypervisor. =http://www.xenproject.org/.

[3] kvm hypervisor. =http://www.linux-kvm.org/.

[4] VmWare hypervisor. =http://www.vmware.com/.

[5] M. Mahalingam. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. =http://tools.ietf.org/html/rfc7348, aug 2014.

[6] M. Sridharan. Nvgre: Network virtualization using generic routing encapsulation. =https://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-00, sep 2011.

[7] Open Networking Foundation. =https://www.opennetworking.org/.

[8] OpenFlow protocol specification. =https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[9] NOX controller. =http://www.noxrepo.org/.

[10] OpenStack Cloud Plateform. =http://www.openstack.org/.

[11] CloudStack Cloud Plateform. =http://cloudstack.apache.org/.

[12] Eucalyptus Cloud Plateform. =https://www.eucalyptus.com/.

[13] FlowVisor controller. =https://github.com/OPENNETWORKINGLAB/flowvisor/wiki.

[14] OpenVirteX Network virtualization Plateform. =http://ovx.onlab.us/.

[15] OpenStack Neutron Plugin. =https://wiki.openstack.org/wiki/Neutron.

[16] OpenNaaS Plateform. =http://opennaas.org/.