D5.3 Prototype description and implementation plan

Dissemination level	PU
Version	1.0 Reviewed
Due date	30.11.2014
Version date	05.12.2014

This project is co-funded
by the European Union

Document information

Authors

Editor: EHU

Contributors:

George Agapiou (OTE), Ivano Cerrato (POLITO), Jokin Garay (EHU), Jon Matias (EHU), Gergely Pongracz (ETH), Fulvio Risso (POLITO), Tobias Steinicke (TP), David Verbeiren (INTEL) and Hagen Woesner (BISDN).

Reviewers:

Fritz-Joachim Westphal (DT), Catalin Meirosu (EAB).

Coordinator

Dr. András Császár

Ericsson Magyarország Kommunikációs Rendszerek Kft. (ETH) AB

Email: andras.csaszar@ericsson.com

Project funding

7th Framework Programme

FP7-ICT-2013-11

Collaborative project

Grant Agreement No. 619609

Legal Disclaimer

Revision and history chart

Version	Date	Comment
0.1	06.05.2014	Initial version (ToC)
0.2	01.07.2014	Prototype phased approach with mapping to deliverables/milestones
0.3	29.07.2014	Summary of UN architecture from D5.2 and mapping to prototype phases
0.4	19.09.2014	ToC reorganization. Additional description of sections
0.5	03.11.2014	Update phased approach and use cases
0.6	17.11.2014	Functionality description and detailed phase content included
0.7	03.12.2014	New ToC and comments from external review addressed
0.8	04.12.2014	Final version for second external review
1.0	05.12.2013	Final version

Table of contents

Executive summary

This document describes the prototype Universal Node (UN) that is developed within the UNIFY project and provides a corresponding implementation plan.

An overview of the UN architecture is first given in Section 1 in order to briefly introduce the components and interfaces defined in D5.2 [1]: the VNF Execution Environments, Virtual Switching Engines and Unified Resource Manager components, on the one hand, and the Virtual Resources management, NF-FG management and VNF Specifications and Images repository interfaces on the other.

Next, Section 2 describes the prototype approach and details the implementation plan which follows a phased structure, in order to gradually build up the functionality required for the Integrated Prototype and provide the input for the subsequent WP5 deliverables and milestones. Phase I is oriented towards providing an initial prototype to demonstrate the feasibility of creating a UN compliant with the UNIFY architecture, with all the basic functions delivered. Phase II will enrich the UN prototype with the functions that are required by the other work packages and build towards the integration in the global Integrated prototype. Phase III will complete the required functionalities for the Integrated Prototype and provide optimisation of the UN for the selected Use Cases through an evolution of the internals of the UN prototype. While Phase I focusses on creating the basic UN prototype structure required by the specificities of the UNIFY architecture and a UN interface centred around NF-FGs, the subsequent phases will provide opportunities to research novel approaches or evolve existing ones in areas such as the local automated optimization of resource usage, the support of multiple VNF execution environments on a single node, and high-performance datapath implementations on commodity hardware. Moreover, the division of the functionalities along the phases has been designed to be aligned with the overall phase objectives and the deliverable and milestone planning, while at the same time ensuring that the cross-dependencies between the functionalities are met.

Section 3 introduces the selected Use Case. For the time being, only a WP5-oriented Elastic NF Use Case is considered, but the implementation plan has placeholders for upcoming use cases, aligned with project wide use cases wherever possible.

Section 4 summarizes the main outcomes from this deliverable and adds a future perspective for the work to be done in relation to other WPs and the implementation plan described here.

1 Introduction

The main objective for WP5 is the design, implementation and performance evaluation of a Universal Node prototype, which aims to be applicable in upcoming communication networks where a large variety of services are deployed on standard hardware.

As a first step towards the objective, deliverable D5.1 [2] gathered the requirements for the Universal Node, related to network virtualization, resource sharing, switching or traffic steering, as well as support aspects like as configuration, performance, monitoring or security.

D5.1 also introduced a first high-level approach to the functional specification that was further refined in D5.2 [1], which defined the UN architecture and interfaces and is summarized in Section 1.1.

Building upon this definition, this document describes the prototype approach, which is based on three phases, including the implementation plan and the detail of the functionalities to be implemented in each phase, in Section 2; the currently selected Use Case in Section 3 and finalizes with some conclusions in Section 4.

1.1 UN Architecture

In order to provide a background to the scope of the prototype phases detailed later in the document, this section contains a brief summary of the UN architecture and interfaces as described in deliverable D5.2 Universal Node Interfaces and Software Architecture [1].

Figure 1.1 shows the three main functional blocks of the UN:

The VNF Execution Environment (VNF EE) consists in one or more compute platform virtualization solutions, including hypervisors or simpler container based approaches (Linux Containers, Docker…).

The Virtual Switching Engine (VSE) implements packet switching on the UN, managing the physical network interfaces (NICs) and the inbound and outbound traffic steering of the deployed NF-FGs, as well as the internal traffic steering between the different NFs deployed as part of the NF-FG.

Managing the VNF EE and the VSE, the Unified Resource Manager (URM) plays the role of a local orchestrator that has complete and detailed view on the resources available on the node, their topology and related usage constraints and limitations. It provides the main UN interfaces and controls the VNF EE and VSE to fulfil the NF-FG deployment and management requests.

Figure 1.1 also shows the three external interfaces of the UN:

Virtual Resource management interface (part of both Sl-Or and Cf-Or): This interface covers the discovery of resources exposed by the node as well as possible updates to those resources (such an update may be the

result of actions performed through the other interfaces, or from a reconfiguration of the node) and also reporting to the upper layers the current availability of resources due to the NF-FGs already deployed in the UN.

NF-FG management interface (part of both Sl-Or and Cf-Or): The Universal Node management interface focusses on deploying and managing Network Function Forwarding Graphs.

VNF Specifications and Images repository interface: When the UN is instructed to deploy a NF-FG, it needs to fetch the detailed specification and the related binaries of the involved VNFs. This constitutes an outbound interface of the UN towards a central VNF repository.

During the architectural discussions with WP2 and WP3 it was established that the Application Control Interface will be internal to the NF-FG, and that Cf-Or will take over the communication of scaling requests. Therefore it was decided to not specify in detail this interface further. The Control App information outlined in [1] was further developed in D3.1 [3] and included in the Use Case described in Section 30.

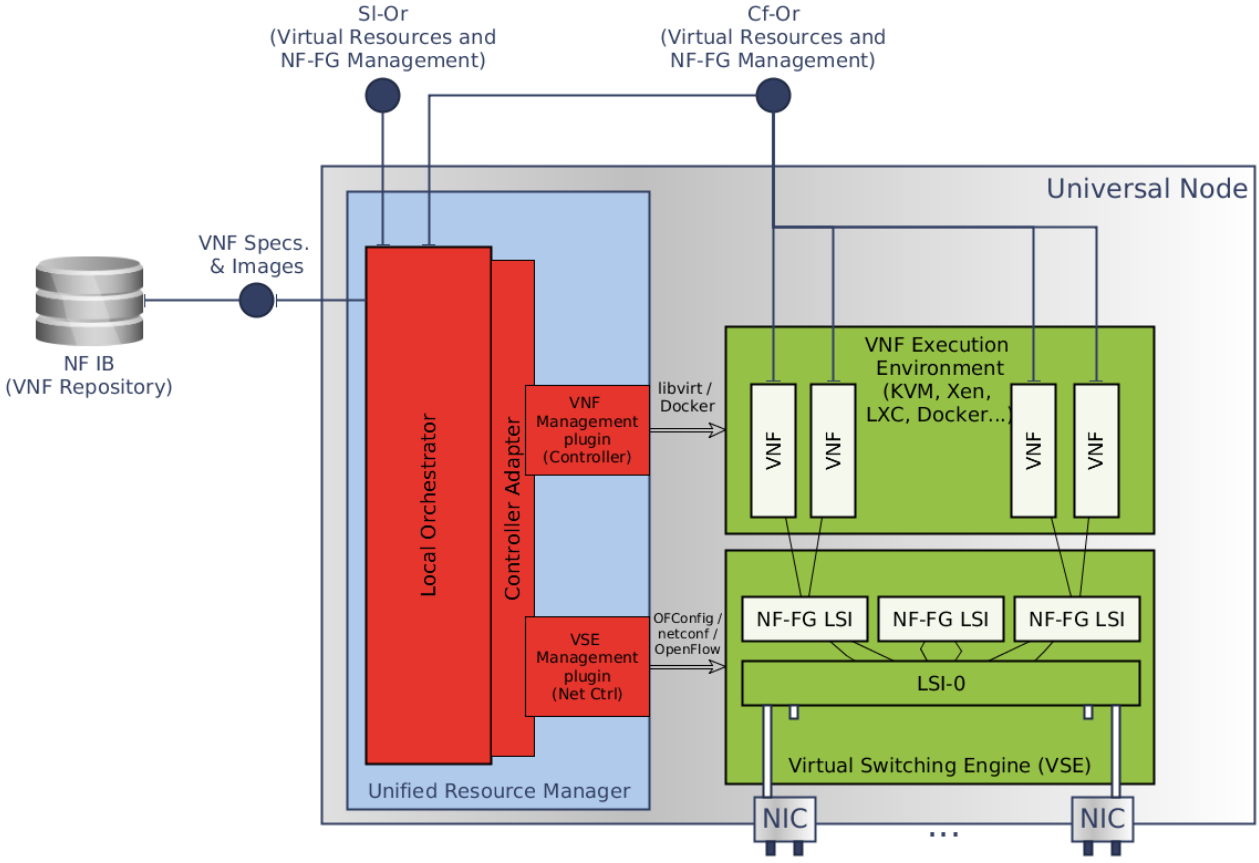

Figure 1.1 – UN Software architecture

2 Prototype approach

This section gives an overview of the prototype plan, which follows a phased approach. Starting with the core components, the phases build towards the integrated prototype and progressive improvement and optimisation of the UN capabilities, performance and developed use cases. The phases defined are aligned with the expected outputs of WP5, with each prototype phase oriented to provide the required content for each of the Milestones and Deliverables for the corresponding period, as summarized in Table 1.

At time of writing, the first phase of the prototype was completed but is described nevertheless so as to provide the basis on which the subsequent phases and deliverables build.

1. Phase I (M11, October 2014): provided the initial prototype to demonstrate the feasibility to create a UN compliant with the UNIFY architecture, with all the basic functions delivered. The resulting prototype is ready for milestone MS5.2, and allows performing an initial set of tests, obtaining results and identifying bottlenecks, which will be documented in D5.4 (in parallel with the phase II development).

2. Phase II (M17, March 2015): will enrich the UN prototype with the functions that are required by the other work packages and build towards the integration in the global Integrated prototype. The updated prototype will be documented in D5.5 and the preparation for the Integrated Prototype reported in MS5.3.

3. Phase III (M29, March 2016): will complete the required functionalities for the Integrated Prototype and provide optimisation of the UN for selected Use Cases through an evolution of the internals of the UN prototype. It is expected to include more sophisticated optimizations oriented to achieve higher performance, while keeping the same external interfaces in order to facilitate the development in the other sibling work packages. The different Use Cases will be targeted at MS5.4, MS5.5 and MS5.6 and the performance data required by WP2 at MS5.7. The final benchmarking documentation after the optimisations will be covered in D5.6, as compared to the initial data provided in D5.4.

Item	Document	Month	Plan Date	Phase
MS5.2	Basic prototype code is finished, test environment is setup, first results are available	M12	November 2014	Phase I
D5.4	Initial Benchmarking documentation	M15	January 2015	Phase I (*)
MS5.3	Report on Universal Node Prototype for preparation of Integration Plan	M17	March 2015	Phase II
D5.5	Prototype deliverable	M17	March 2015	Phase II
MS5.7	Initial performance data available as input to WP2 for TCO	M20	June 2015	Phase III
MS5.4	Use-case One	M21	July 2015	Phase III
MS5.5	Use-case Two	M24	October 2015	Phase III
D5.6	Final Benchmarking Documentation	M26	December 2015	Phase III
MS5.6	Use-case Three	M27	January 2016	Phase III

Table 1: WP5 Milestone and Deliverable mapping to Prototype phases
(*) D5.4 documents results based on the phase I of the prototype but does not belong itself to any phase of the prototype development

2.1 Prototype phases

2.1.1 Phase I

The initial prototype developed in Phase I provides a simplified core of the Unified Resource Manager, allowing the processing of a NF-FG received from the upper layer orchestrator and the VNF specification and do the simplest, most straightforward placement. Once deployed, the NF-FG will not be updated.

The VNFs are deployed in a single UN and run over one VNF Execution Environment. The VNFs are connected through the Virtual Switching Engine inside the UN, following the NF-FG specification.

This first prototype is intended for standalone execution and, as such, the external interfaces are not final.

The applications deployed as VNFs are simple applications (NAT, basic firewall, traffic monitor) statically configured, for the single VNF Execution Environment included in phase I and do not include corresponding Control Apps.

2.1.2 Phase II

The prototype updated in Phase II will enhance the Unified Resource Manager by providing: resource reporting, both for resource capabilities and usage; NF-FG lifecycle management, including deployment, modifications and removal; VNF specification and image retrieval; improved placement and support for basic monitoring primitives.

The VNFs will be deployed and executed over multiple VNF Execution Environments. The VNFs will be connected through the Virtual Switching Engines on the UNs, following the NF-FG specification and deploying the required NF-FG Logical Switch Instances (LSIs).

As the updated prototype is oriented towards supporting the Integrated Prototype, all external interfaces will be included to support the aforementioned operations.

The applications to be deployed as VNFs will be relatively simple applications, including at least one suitable for benchmarking, coming in the different flavours supported in the multiple VNF Execution Environments included in phase II.

2.1.3 Phase III

The final prototype developed in Phase III will provide an optimized Unified Resource Manager including fine-tuned placement. Internal communication primitives between the VSE and VNF EE will also be further developed. Other optimizations detected as a result of Phase I and Phase II will also be analysed and, possibly, included in this phase.

The external interfaces are expected to remain mostly unchanged so the integration with the prototypes of other WPs is not affected.

An attempt will be made to accommodate more advanced applications that make use of the scalability and elasticity features of the platform, as well as trying to achieve the best possible result for the final benchmarking, and will include Control Apps as defined in D3.1 [4] .

2.2 Implementation Plan

As described in section 2.1, each phase increases the functionality of the different components and interfaces of the UN.

Table 2 represents the implementation level of the elements of the UN architecture through the prototype phases, summarizing the implementation level of the corresponding functionalities. Section 2.3 describes which parts of each element are to be implemented and to which extent and Table 3 provides the detail of the implementation level of the individual functionalities that is summarized here.

	1 2 3	4 5 6 7 8 9 10 11 12 13 14	15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
		Phase I	Phase II	Phase III
UN Components				
URM (URM)		NP	NPC	CO
URM: Local Orchestator (LO)		N	N	C
URM: VNF Management (VEM)		NP	C	O
URM: VSE Management (VSM)		NP	P	C
URM: Monitoring (MON)		NP	PC	CO
VNF EE (VEE)		NP	PC	CO
VSE (VSE)		NP	PC	CO
UN Interfaces				
NF-FG Management Interface (NMI)		P	C	O
Resource Management Interface (RMI)		N	NP	C
VNF Template and Images Repository Interface (VTIRI)		P	C	O
Others				
Applications (derived from selected use cases)		P	P	C

N	Functionalities not implemented
NP	Functionalities either partially implemented or not implemented
P	Functionalities partially implemented
NPC	Functionalities completely, partially or not implemented
PC	Functionalities either completely or partially implemented
C	Functionalities completely implemented
CO	Functionalities either optimized (if required) or completely implemented
O	Optimized (if required)

Table 2: Implementation level of the UN architecture elements through the prototype phases

2.3 UN Component and Interface implementation in Prototype Phases

This section describes the functionalities of the UN components and interfaces to be implemented in each of the defined prototype phases. As detailed in section 1.1, Resource and NF-FG management interfaces are part of both Sl-Or and Cf-Or reference points. For each functionality, a general description and expected outcomes per phase is provided and a summary is included in Table 3. During Phase III, those functionalities that are planned to be completed in Phase II and have no specific goals in Phase III, will be involved only if after the benchmarking results are obtained they are identified as requiring optimization.

UN Elements		Functionality	Phase I	Phase II	Phase III
UN Components	URM	URM1 Discover available resources	●	●	●
		URM2 Report available resources	●	●	●
		URM3 Update available resource	●	●	●
		URM4 Local scaling	●	●	●
		URM5 NF-FG lifecycle management	●	●	●
		URM6 NF-FG mapping to specific resources	●	●	●
	URM: Local Orchestrator	LO1 Optimized placement	●	●	●
	URM: VNF Management (VEM)	VEM1 Interface to multiple VEE	●	●	●
		VEM2 Expose VEE capabilities to LO	●	●	●
	URM: VSE Management (VSM)	VSM1 Interface to multiple VSE	●	●	●
		VSM2 Expose VSE capabilities to LO	●	●	●
		VSM3 Configuration of Internal interconnection between VNFs	●	●	●
		VSM4 Configuration of external connection to other nodes	●	●	●
	URM: Monitoring (MON)	MON1 Monitoring support	●	●	●
		MON2 Monitoring data report	●	●	●
	VNF Execution Environment	VEE1 Multiple VEE solutions	●	●	●
		VEE2 VNF mapping to low level resources	●	●	●
		VEE3 Optimized data transfer: VEE-VSE relation	●	●	●
	Virtual Switching Engine	VSE1 Multiple VSE solutions	●	●	●
		VSE2 Dynamic deployment of LSI	●	●	●
		VSE3 External traffic steering:	●	●	●
		VSE4 Internal traffic steering between VNFs	●	●	●
		VSE5 VNF Type 5 implementation	●	●	●
		VSE6 Estimation of resources needed for specific VSE setup	●	●	●
		VSE7 Monitoring support	●	●	●

UN Elements		Functionality	Phase I	Phase II	Phase III
UN Interfaces	NF-FG Management	NMI1 NF-FG interface definition	●	●	●
	Resource Management	RMI1 Report resource usage	●	●	●
		RMI2 Report node capabilities	●	●	●
		RMI3 VNF interface for local scaling	●	●	●
	VNF Specifications and Images Repository	VSIRI1 Interface to the repository	●	●	●

●	Not implemented	●	Partially implemented	●	Completely implemented	●	Optimization depending on benchmarking results

Table 3: UN Component and Interface implementation in Prototype Phases

2.3.1 UN components

2.3.1.1 URM (URM)

URM.1) Discover available resources

a. General description: at the bootstrap of the UN, the URM discovers the resources available on the node. Particularly, these resource includes the number of available CPU/cores, available memory, aspects related both to the virtualization engine (e.g., KVM, XEN, Docker) and to the switching engine (e.g., traffic steering based on OpenFlow 1.2).

   At the moment, the information about the binding of CPU cores to PCI lanes, specific properties of the cores etc. can be retrieved via /proc and /sys filesystems and related tools in a Linux environment. We do not believe that a further abstraction of resources on a generic hardware level is possible, in other words, the URM is where the recursion of orchestrators ends and low-level hardware functions need to be accessed.

b. Phase I: No support.

c. Phase II: Partial. The UN is able to discover the hardware parameters (CPU/cores, available memory) and the main characteristics of the virtualization engine (e.g., KVM, etc.).

d. Phase III: Complete. The UN is able to discover the full list of hardware resources, including the actual memory usage and the possible list of hardware coprocessors (if any), In addition, it is also able to discover the main characteristics of the switching engine and the possible additional native functions supported in it.

URM.2) Report available resources

a. General description: in the boot phase, after that the URM has discovered the resources available on the node, it propagates this information to the upper layer so that it can schedule the NF-FGs on the proper nodes.

b. Phase I: No support.

c. Phase II: Partial. The UN is able to report the parameters that are mentioned in the second phase of the previous section "Discover available resources".

d. Phase III: Complete. The UN is able to report all the parameters that are mentioned in the third phase the previous section "Discover available resources". In addition, the UN is able to report an example of generic capabilities, such as a list of abstract network functions that can be instantiated on the node.

URM.3) Update available resources

a. General description: as a consequence of errors and/or updates of the node (e.g., a new bank of RAM is added), the resources available on the UN could change. Also, deployment, modification and removal of NF-FGs will affect the availability of resources. When this happens, the URM notifies the upper layer so that it can react properly.

b. Phase I: No support.

c. Phase II: Partial. The UN is able to report any information that is related to an upgrade of the hardware characteristics of the UN itself. The actual usage of CPU and memory is reported.

d. Phase III: Complete. The UN is able to dynamically discover and report the full list of parameters that are mentioned in the previous section (e.g., a new bank of RAM added). The UN is also able to report also the usage of any existing hardware coprocessor that is installed within the UN itself.

URM.4) Local scaling

a. General description: For VNF implementations that can scale by modifying their number of parallel threads, the URM will modulate the compute resources allocated to a VNF instance without intervention from the upper level Orchestrator. Once the URM authorizes a resource allocation change, it must interact with the corresponding VNF EE to effectively implement the change and then notify the VNF of the resource allocation change. Specific technologies (hypervisors, user-space packet processing libraries) may lack run-time reconfiguration of allocated resources, in which case the VNF using them may be excluded from this scheme. WP5 will however attempt to

address those shortcomings where realistically achievable without significant impact to the overall project.

b. Phase I: No support.

c. Phase II: No support.

d. Phase III: Complete, given possible shortcomings of used technologies.

URM.5) NF-FG lifecycle management

a. General description: The URM manages the lifecycle of NF-FGs on the node. It receives the requests for deployment, removal or update of an NF-FG from the upper-level orchestrator.

b. Phase I: Partial. Only initial deployment of NF-FGs is supported.

c. Phase II: Complete. Adds support for updates and removal.

URM.6) NF-FG mapping to specific resources

a. General description: When deploying or modifying an NF-FG, the URM must perform several mappings. It must translate from a partially specified VNF to a fully specified one that is compatible with the node architecture and available resources (see section on Service Decomposition in [4]) and that also meets the criteria provided in the incoming NF-FG. The URM must also ensure the VSE will be capable of supporting the required traffic steering and possibly adjust resources allocated to the VSE accordingly.

b. Phase I: Partial: The URM accepts a single NF-FG and maps the VNFs to a fully specified implementation in a very basic manner.

c. Phase II: Complete in terms of the interactions with other components but optimized placement and VSE resource estimation are considered separately. The URM accepts multiple NF-FGs, maps them to specific resources and communication means in terms of VNFs and LSIs, and hands them to the VNF EE and VSE management components.

2.3.1.2 URM: Local Orchestator (LO)

LO.1) Optimized placement

a. General description: The Local Orchestrator (LO) which is the part of the URM that contains most of the intelligence for performing optimized placement of the VNF threads into the UN topology. The optimized placement considers both the compute and networking resources required for the deployment of the NF-FG.

b. Phase I: No support.

c. Phase II: No support.

d. Phase III: Complete. The optimization will be based on the embedding algorithms developed in the WP3 orchestrator [3].

2.3.1.3 URM: VNF Management (VEM)

VEM.1) Interface to multiple VEE

a. General description: In D5.2, we described multiple platform virtualization solutions, like Docker, LXC, KVM and QEMU. Their different interfaces must be somehow abstracted to the LO.

b. Phase I: Partial. In Phase I, the prototype provides basic supports for Docker as VEE and also supports running VNFs as separate DPDK processes alongside the VSE.

c. Phase II: Complete. In Phase II, the prototype extends the support for VEEs to traditional virtual machines (KVM).

VEM.2) Expose VEE capabilities to LO

a. General description: the VEE exposes its capabilities to the LO. In particular, it provides information on the type of VNF Execution Environments it supports, such as KVM virtual machines, Docker containers or DPDK processes.

b. Phase I: No support.

c. Phase II: Complete. The VEE exports the full list of capabilities to the LO.

2.3.1.4 URM: VSE Management (VSM)

VSM.1) Interface to multiple VSE

a. General description: The URM needs information on the physical ports as well as the supported capabilities of the VSE. This information can be provided via the OpenFlow protocol's features_reply and table_features_reply messages from an OpenFlow-capable VSE. The OFConfig protocol that uses netconf/yang to transport largely the same information could also be considered. LO needs to support multi-tenancy between different NF-FGs

b. Phase I: Partial. Netconf to discover ports and switches, creation of LSIs.

c. Phase II Partial. Multi-tenancy, creation of LSI-0 and coarse-grained slicing (per port).

d. Phase III Complete. Multi-tenancy, fine grained slicing (FlowVisor-like).

VSM.2) Expose VSE capabilities to LO

a. General description: the VSE exposes its capabilities to the LO. Particularly, it provides information on the technology used to implement the traffic steering among the ports of the VNFs of the NF-FG as described in [7]. For example, the traffic steering could be based on the OpenFlow 1.0 protocol, on OpenFlow 1.1, etc.

b. Phase I: No support.

c. Phase II: Partial, The VSE exports its capabilities in terms of supported protocol/version (e.g., OpenFlow 1.3, Netconf).

d. Phase III: Complete. The VSE exports its full list of capabilities, including the possible additional supported functions (e.g., native L2 forwarding, hardware functions).

VSM.3) Configuration of external connection to other nodes

a. General description: in the UN the external connectivity of the NF-FGs is implemented in a special LSI (LSI-0), which must be instantiated during the bootstrapping process of the UN. The NF-FG received from CA fully characterizes the endpoints resulting from the scoping process. Based on this information, virtual ports attached to the LSI-0 must be dynamically created (and taken away when the NG-FG is removed) to provide connectivity to the NF-FG, enforcing the traffic isolation (per tenant / per NF-FG) based on different mechanisms (e.g. tunnelling…).

b. Phase I: Partial. During this phase I, the LSI-0 is configured based on the information provided by the NF-FG. A local OpenFlow controller inside the UN is used to properly configure the LSI-0, which is the only LSI allowed access to the physical infrastructure. The LSI-0 is able to classify the traffic coming from the network and to deliver it to the proper VNF based on NF-FG information. A basic isolation between NF-FGs is supported at the LSI-0, which is use-case specific. The LSI is able to dynamically instantiate new virtual interfaces attached to the LSI-0. The implementation is based on the xDPd softswitch, which also supports the dynamic creation of additional LSIs.

c. Phase II: Complete. Extend support to other softswitches (e.g. OVS DPDK) and configuration of additional traffic steering mechanisms providing isolation in LSI-0 (e.g. tunnelling, tagging, etc.).

d. Phase III: Optimization. Optimization of data transfer to allow specifying the mechanism to be used.

VSM.4) Configuration of Internal interconnection between VNFs

a. General description: in the UN the internal connectivity between VNFs of the NF-FGs is implemented in dedicated LSIs (LSI-i) deployed on top of the LSI-0. The LSI-i(s) that are instantiated when NF-FG is deployed (if needed), could be later on modified and are terminated at

NF-FG removal. The NF-FG received from CA fully characterizes the connectivity to be provided between the VNFs. Based on this information, the LSI-i is deployed and configured. As the traffic associated to LSI-i is related to just one NF-FG, isolation is guaranteed and the flow rules from NF-FG definition cannot collide with any other NF-FG.

b. Phase I: Partial. During this phase I, the isolated LSIs per tenant (LSI-i) can be dynamically instantiated on-demand on top of the LSI-0. This support is provided by the xDPd softswitch, which provides the needed mechanisms to request (or remove) additional LSIs. A tenant-related OpenFlow controller is deployed internally in the UN for controlling the instantiated LSIs. The LSI-i can be connected to the VNFs, LSI-0 or other LSIs, and can be used to steer the traffic between them by using standard OpenFlow flow rules. The steering mechanism used in phase I is use-case specific.

c. Phase II: Complete. Extend support to other softswitches (e.g. OVS DPDK) and optimization of data transfer between VNFs (different VNF-types considered).

d. Phase III: Optimization. Further optimization and consolidation with joint LSI-0 and LSI-i strategies e.g.: LSI-i functionalities deployed in LSI-0 (deployment of single NF), other type of DPDK-supported communications not requiring a LSI instance.

2.3.1.5 URM: Monitoring (MON)

MON.1) Monitoring support

a. General description: As part of deploying NF-FGs, the URM also supports the deployment of Observability Points. The URM must facilitate the communication between local observability points and observability points on other UNs or other nodes, for example to implement link monitoring. The deployed OPs will also have the capability to manipulate packets as does any VNF. Additionally, in order to support proactive troubleshooting based on correlation of events, the URM will also record or propagate events such a NF-FG updates, flow rule insertion and updates, etc.

b. Phase I: Partial. The VSE supports a basic set of standard OpenFlow counters. Observability Points are handled as normal VNFs (no specific communication support).

c. Phase II: Partial. Additional counters. Depending on the progress of WP4, support for events reporting and communication between OPs should also be considered for this phase.

d. Phase III: Pluggable monitoring extensions and optimization for the final benchmarking, if required.

MON.2) Monitoring data report

a. General description: The URM provides a conduit for monitoring entities to query or enable reporting of certain events such a NF-FG updates.

b. Phase I: Not supported

c. Phase II: Supported but exact details depend on agreement with WP4.

2.3.1.6 VNF EE (VEE)

VEE.1) Multiple VEE solutions

a. General description: In D5.2, we described multiple platform virtualizations and data plane processing solutions. We also divided the VNF in five types: full virtual machines, containers, processes running in the host environment, VSE plugins and VSE functionalities. For the first three, specific VEEs would be required, whereas the last two would be covered by VSE functionalities.

b. Phase I: Partial. In Phase I, the initial prototype supports Docker as VEE.

c. Phase II: Complete. In Phase II, the prototype extends the support for VEEs by adding solutions for DPDK processes and traditional virtual machines.

VEE.2) VNF mapping to low level resources

a. General description: The URM has to map a VNF to physical resources. DPDK processes need to be pinned to specific CPU cores and so do the hypervisor threads running the virtual CPU cores that execute DPDK application threads in a VM. This is required for resource reservations to handle multiple VNFs. To perform this mapping, the URM relies on the information gathered during the resources discovery procedure (see URM.1).

b. Phase I: None

c. Phase II: Partial. In this Phase, the initial support can pin a particular VNF to a specific CPU core.

d. Phase III: Complete. In Phase III the prototype completes the supports for low level resource mapping. Now the URM is not only capable of assigning a specific CPU core to VNF, but also to move the VNF to another CPU core and remove the VNF from one or more CPU cores. This includes that the URM can handle more than one CPU core for one VNF.

VEE.3) Optimized data transfer: VEE-VSE relation

a. General description – The communication mechanism between the VSE and the VNFs running in the VEE is of critical importance to the overall performance of the UN. From that point of view, a

zero-copy capable mechanism is extremely desirable. However, this is typically not without consequences on manageability of the involved entities and the security of their data in a multi-tenant environment.

b. Phase I: Partial. In this phase, the data transfer (packets, metadata, queues) between the VSE and the VNFs running in VMs or containers is based on standard elements of the Linux environment such as TAP devices. Due to the fact that these rely on kernel support, significant overhead is to be expected in terms of context switches and data copies. Phase I however already provides support for an optimized mechanism for the communication with VNFs implemented as separate processes running alongside the VSE. This mechanism is based on a shared memory area where the data structures storing the packet data and the queues (rings) are directly shared between the VSE and the VNF processes. This mechanism is capable of providing zero-copy packet transfer.

c. Phase II: Complete. The shared-memory mechanism will be extended to VNFs within VMs and containers, using the DPDK IVSHMEM mechanism [8]. At this stage, we will evaluate the practical implications, including security related ones, of deploying and managing VNFs that communicate with the VSE over IVSHMEM, we will evaluate whether this optimal mechanism is acceptable or if more balance between performance and practical aspects is needed. Depending on the conclusions, we may extend the VSE to support an additional more balanced interface like User Space vHost.

2.3.1.7 VSE (VSE)

VSE.1) Multiple VSE solutions

a. General description – In D5.2 [1], we listed various possible candidates to provide the base VSE functionality. Initial evaluation of a subset of these candidates (xDPd, OVS) showed that their performance numbers are in the same order of magnitude but still evolve significantly with important performance variations (positive or negative) when optimization tasks are carried out or features are added.

For this reason, it would be very beneficial to enable support of multiple VSE implementations so that we could keep on comparing them over the course of the project and would not have to make an early choice that may later prove sub-optimal. Additionally, since these candidates also differ in terms of features supported, some use cases may benefit from support provided by one VSE implementation while some other may require different features.

Support for multiple VSE solutions should however be considered a lower priority than the initial implementation with one specific VSE. It should be seen as a desired feature when designing the configuration/LSI management interface to the VSE.

All of the VSE candidate solutions support an OpenFlow interface which is also a good match for the flow rules that an NF-FG includes. However, OpenFlow only covers management of the flow rules, and does not address the configuration and management of LSIs or other configuration actions that must be done prior to creating flow rules on an LSI. Hence, the abstraction of the specific VSE implementation is mostly concerned by this configuration and management interface. OpenFlow already abstracts the control of LSIs.

b. Phase I. Partial. Focus on initial support of a single VSE, namely xDPd.

c. Phase II. Complete. Use learnings from phase I to bring more abstraction in the design of the interface and target the support of an alternative VSE solution: OVS with DPDK support

VSE.2) Dynamic deployment of LSI

a. General description: Logical Switch Instances (LSIs), shown in Figure 2.1, are equivalent to the definition of OpenFlow Logical Switches in the OFConfig 1.2 protocol description and to the concept of virtual bridges within OVSDB.

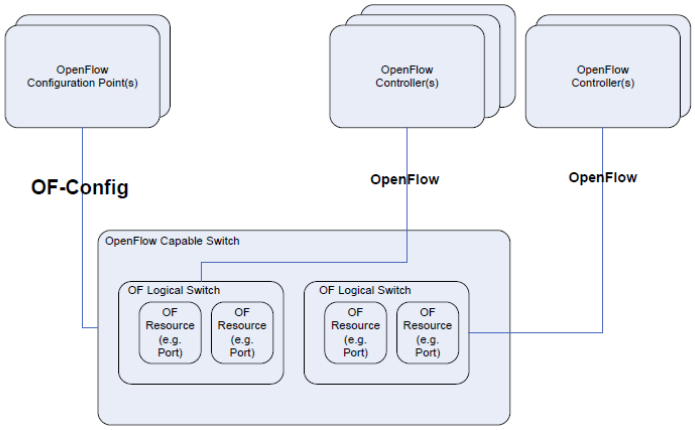

Figure 2.1 – OFConfig is used to instantiate Logical Switch instances each having their own controller(s).

An LSI contains a subset of resources of an OpenFlow capable switch. These resources are typically ports, queues and flow tables. Using the new feature of negotiable datapath modules (NDM), an OF Configuration Point may specify the layout of the tables and specific capabilities to be supported by a logical switch instance.

As well as in other features, the two candidate soft switches WP5 considers (xDPd and OVS-dpdk) also differ in the configuration protocols used (xDPd has an OFConfig plugin). In principle however, the operation of creating an LSI is the same for both candidate switches.

Up until OFConfig v.1.1.1 the creation of LSI was explicitly out of scope and defined a 'management action'. Since OFConfig 1.2, the dynamic instantiation of logical switches is possible, but as of this writing the LSI cannot be "wired" internally, i.e. connected by vlinks. xDPd therefore uses extensions to patch this missing feature. Using ovsctl (to configure ovsdb) however, a wiring of virtual bridges is possible.

b. Phase I: Partial. During phase 1, xDPd could be set up via a simple configuration file. This file specifies the key features of the OF capable switch (physical ports, protocol version) and the logical switches including the respective controller connections.

c. Phase II: Partial. During phase 2, configuration via OFConfig 1.2 allows dynamic creation of LSIs. Dynamic wiring of LSI-0 and the LSI-1,2,3...

d. Phase III: Complete. Integration of QoS in the slicing (providing assignment of bandwidth per slice in the soft switches).

VSE.3) External traffic steering

a. General description – The architecture described in D5.2 [1] considered a special LSI (LSI-0) to manage the physical ports and control the traffic flow into and out from the NF-FGs deployed on the node. As such, the LSI-0 must provide the means to allow the isolation between NF-FGs. One possible manner to implement such isolation could be by using different tunnelling mechanisms. Moreover, the ability to dynamically create and remove virtual interfaces on the LSI-0 towards the NF-FG resources deployed at the UN is mandatory, thus providing the external connectivity for the NF-FG. Also, a zero-copy capable mechanism is extremely desirable for the LSI-0 / LSI-i relationship (VSE-VSE). However, this is typically not without consequences on manageability of the involved entities and the security of their data in a multi-tenant environment.

b. Phase I: Partial. A basic support for external traffic steering is provided for incoming and outgoing traffic from the UN, which is based on the mechanism already supported by any OpenFlow datapath implementing the standard. The mechanism used for NF-FG isolation implemented in this phase I relies on OpenFlow flow rules without any additional tagging mechanism and it is use case specific, rather than a general approach. The implementation is based on the xDPd softswitch and its DPDK-based backend for handling physical interfaces.

c. Phase II: Complete. During this second phase a more general approach for NF-FG isolation is proposed and different technologies and encapsulation mechanisms are considered for their implementation as extensions to the standard OpenFlow support. This means that the selected mechanisms need to be implemented in the VSE and exposed through the OpenFlow interface.

The main goal is to provide a set of basic resources (e.g. encapsulation mechanisms) to the upper layer to define the most adequate mechanisms to isolate the NF-FGs on a network-wide view. Other softswitches (e.g. OVS DPDK) and drivers will be considered.

d. Phase III: Optimization. The phase III is oriented to achieve the optimization of the steering mechanism with some additional or more efficient solutions.

VSE.4) Internal traffic steering between VNFs

a. General description. The NF-FG sent to the UN describes all the resources that must be deployed on the UN associated to the NF-FG. When several VNFs are deployed on the same UN, the internal connectivity is also described in the NF-FG. The architecture described in D5.2 [1] considers a dedicated LSI (i.e. LSI-i) per each NF-FG to deal with the virtual interfaces associated to the VNFs and define/control how the traffic must flow internally to the UN based on the NF-FG description. This LSI-i must be deployed on demand and allow dynamic connection to LSI-0 (for external connectivity). As well as providing mechanism for isolating the traffic associated to one NF-FG from the rest of the traffic from other NF-FGs internal to the UN. For instance, the traffic from each NF-FG can be tagged internally with a different value. As introduced when describing the external traffic steering, a zero-copy capable mechanism is extremely desirable to the LSI-0 / LSI-i relationship (VSE-VSE)..

b. Phase I: Partial. A basic support for the internal traffic steering between VNFs and LSI-0 (or other LSIs), based on isolated LSI-i instances, is provided to isolate the NF-FGs internally at the UN. The internal steering is based on standard OpenFlow mechanisms implemented at any OpenFlow datapath and is use-case specific. The LSI-i implementation is based on xDPd softswitch, which allows to dynamically instantiate additional LSI on-demand and connect it to additional virtual ports dynamically attached to the LSI-0. The DPDK-based backend is used to exchange packets with the VNFs.

c. Phase II: Complete. Extend support to other DPDK drivers and other softswitches (e.g. OVS DPDK).

d. Phase III: Optimization. Support for other types of optimized data transfer between VNFs not requiring LSI-i.

VSE.5) VNF Type 5 implementation

a. General description: Type 5 VNF was defined in D5.2 as functions that can be implemented directly in the switch. Nevertheless, the control function for any of the VNF type 5 functions will need to be deployed as part of the NF-FG. An important prerequisite for type 5 functions is the discovery of hardware acceleration blocks. For instance, many SoCs nowadays come with video codecs or

crypto-blocks that can efficiently implement certain functions, provided they are accessible via an API. OpenFlow provides a basic access to some APIs along the above lines through TTP (Table Type Patterns).

b. Phase I: Partial. Use VNF functions like GRE encapsulation/decapsulation from an external controller that is part of the NF-FG.

c. Phase II: Partial. TTP for specific chips like Broadcom ASIC and SoC with crypto-engine.

d. Phase III: Complete. Negotiable data path model (NDM) to select between TTPs available from an ASIC/SoC.

VSE.6) Estimation of resources needed for specific VSE setup

a. General description: During the deployment or modification of an NF-FG, the URM interacts with the VSE in order to assess the resources needs of the traffic steering associated with the NF-FG.

b. Phase I: No support; static allocation of resources to the VSE.

c. Phase II: Partial: Some relatively simple scheme will be implemented so that the communication mechanisms between the involved components are put in place.

d. Phase III: Complete: A more advanced estimation scheme will be developed covering a few specific cases (e.g. tunnelling) as required for the use cases.

VSE.7) Monitoring support

a. General description: The VSE must support monitoring in the form of standard OpenFlow counters but will also introduce additional and more flexible counters to enable the type of rate and link monitoring that WP4 will investigate, including the aggregation aspects. For example, reporting the sum of the number of bytes transmitted together with the sum of the squared numbers of bytes per incoming packet (variance) can be used to predict link overload without requiring a very high frequency of updates. In order to make it flexible, a pluggable monitoring infrastructure will be developed in the form of lightweight plugins to the VSE that are responsible for gathering such monitoring data and that can be specified with the NF-FG.

b. Phase I: Partial. The VSE supports a basic set of standard OpenFlow counters.

c. Phase II: Additional counters.

d. Phase III: Pluggable monitoring extensions and optimization for the final benchmarking, if required.

2.3.2 UN interfaces

2.3.2.1 NF-FG Management Interface (NMI)

NMI.1) NF-FG interface definition

a. General description: The northbound interface exposed by the UN is related to the SI-Or reference point described in the UNIFY architecture. A high-level view of this interface related to the NF-FG is presented in D2.2 [7], which involves the instantiation, tear down and change of NF-FG. In this regard, the definition of this NF-FG interface must be aligned with WP2 [7] and the NF-FG definition itself must be aligned with D3.1 from WP3 [3]. The NF-FG is processed by the URM to extract fundamental information to perform the mapping to specific resources available at the UN. Based on the defined primitives, the URM is responsible for managing the NF-FG lifecycle, such as deployment, operation, monitoring and removal. Moreover, the NF-FG contains the information needed to implement both the internal interconnection between the VNFs and external connection to other nodes. All this information is embedded in the NF-FG and must be properly parsed by the URM.

b. Phase I: Partial. There is a basic support for a preliminary definition of NF-FG (documented in D5.2 [1] before it was actually formalized in D3.1 [3]). The interface supports the instantiation of a predefined NF-FG used to trigger the subsequent functionalities implemented during this phase. A file-based and REST-based implementation has been developed for the NF-FG management interface.

c. Phase II: Complete. The interface is extended to support the latest defined primitives by WP2 (currently defined in D2.2 [7]), i.e. instantiate (phase I), tear down and change NF-FG. Moreover, the NF-FG definition is updated to the latest definition provided by WP3 (currently in D3.1 [3]). The NF-FG parsing is updated based on this definition. The NF-FG management interface is finalised during this phase.

d. Phase III: Adaptations to possible changes introduced in subsequent deliverables of WP3 after D3.1 [3].

2.3.2.2 Resource Management Interface (RMI)

RMI.1) Report resource usage

a. General description: through the RMI interface, the URM provides information on the global resource availability and current resource usage to the upper layer. This information includes, for instance, available resources discovered during bootstrapping or the consumption of CPU and memory, and can be used by the upper layer to schedule the (complete or partial) NF-FG on the proper node.

b. Phase I: No support.

c. Phase II: Partial. The hardware parameters, main characteristics of the virtualization engine and the actual usage of CPU and memory are reported.

d. Phase III: Complete. The UN is able to report also the possible list of hardware coprocessors (if any) and their usage, the main characteristics of the switching engine and the possible additional native functions supported in it.

RMI.2) Report node capabilities

a. General description: the RMI is also exploited by the URM in order to provide, to the upper layer, information on the capabilities of the node; an example of such information can be the class of VNFs it supports. The upper layer can exploit these capabilities to select the proper node on which instantiate a (part of a) NF-FG.

b. Phase I: No support.

c. Phase II: Partial. The UN reports the list of network function types that can be executed.

d. Phase III: Complete. The UN reports generic capabilities, which include both the network function types that can be supported as well as specific capabilities that can influence the Orchestrator placement decision (e.g. availability of hardware support for cryptographic operations).

RMI.3) VNF interface for local scaling

a. General description: When a VNF that supports scaling by modulating its number of processing threads detects that it is operating close to its maximum capacity or has such headroom that it could release resources, it may signal the Local Orchestrator which would then possibly adapt the VNF resource allocation. An interface between the URM/LO and the VNF is required to support these indications from the VNF as well as to support the control protocol between the URM/LO and the VNF to actually implement the resource change (start/stop thread). The protocol used will be the same as in NMI.1.

b. Phase I: No support.

c. Phase II: No support

d. Phase III: Complete

2.3.2.3 VNF Specifications and Images Repository Interface (VSIRI)

VSIRI.1) Interface to the repository

a. General description: This is the interface that allows the UN to know about available VNF implementations and their specifications and to obtain corresponding images to deploy. The information is used as input to the URM NF-FG mapping process.

b. Phase I: Partial: A very simple initial implementation to support the development of the other parts of the prototype while all WPs converge to a common NF repository approach, if deemed valuable.

c. Phase II: Complete: The interface will be adapted to be aligned with the NFIB interface agreed between the technical WPs in view of the integrated prototype,

2.3.3 Others

2.3.3.1 Applications (derived from selected use cases)

UCA.1) Use case App

a. General description: A lwAFTR application will be developed to support the use case described in Section 3 and to be used as initial target of UN packet processing benchmarking. The applications required for additional Use Cases will be covered In Phase II.

b. Phase I: Partial: The data-plane portion of the application is developed, using DPDK to interface with the network interfaces. Performance when running on "bare-metal" system (no hypervisor, no virtual switch) will be measured as a baseline for later work.

c. Phase II: Partial: The earlier developed data-plane VNF will be made compatible with the complete UN environment and executed in multiple ways: at least as separate DPDK process and in a VM (KVM, IVSHMEM). Control plane side of the application will be implemented to demonstrate the scaling mechanisms.

d. Phase III: Complete. Applications required for additional Use Cases (if any) will be developed.

3 Selected Use Case

This section describes the use case selected for the prototype, as well as the detail of the different functionalities to be implemented in each phase. For the time being, only a WP5-oriented Elastic NF Use Case is considered, the Elastic Lightweight 4over6, but the implementation plan has placeholders for upcoming use cases, aligned with project wide use cases wherever possible.

Elastic Lightweight 4over6

The Lightweight 4over6 (lw4o6) architecture [5] provides support for systems on IPv4 private networks to communicate with a public IPv4 network (the internet) over an IPv6 carrier network. The term 'lightweight' stems from the fact that the solution focuses on a relatively small number of elements, thus providing a way to deploy 4over6 much more lightweight than developing a full-dual stack solution.

The two components involved in this communication, shown in Figure 3.1 are the Lightweight Basic Bridging BroadBand element (lwB4) and the Lightweight Address Family Transition Router (lwAFTR). These two components are directly connected by the carrier IPv6 network. IPv4 packets are transported between lwB4 and lwAFTR by simply encapsulating them within IPv6 packets.

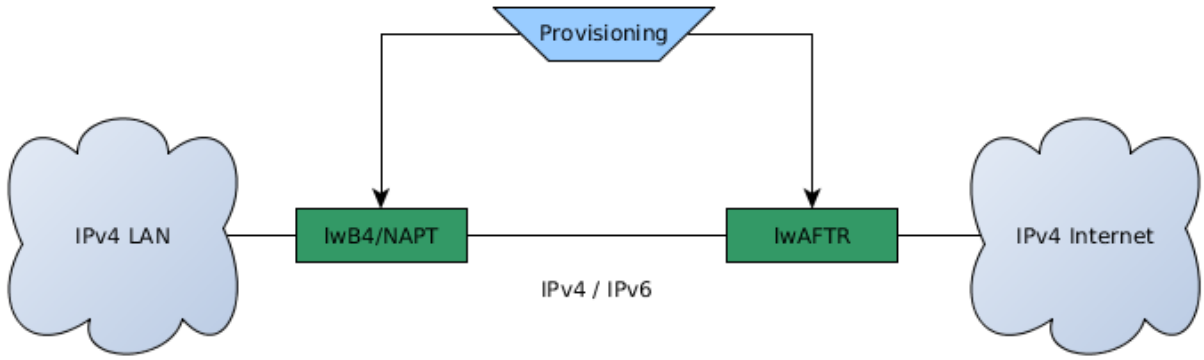

Figure 3.1 – lw4over6 architecture from [5]

Lw4over6 is a variant of Dual-Stack Lite [6] where the Network Address and Port Translation (NAPT) functionality has been relocated from the AFTR to the B4 element. This solves scalability challenges in the NAPT without really adding any new requirement to the Customer Premises Equipment (CPE), since the CPEs have been providing this functionality for a very long time already. Without the NAPT functionality, the lwAFTR is not required to provide per-flow state anymore and hence can scale much more easily. The NAPT44 function is based on the assignment of a Port-restricted IPv4 Address to the lwB4. This includes a public IPv4 address and a range of ports (Restricted Port-Set) that the lwB4 can use as source ports on the public IPv4 address when executing its NAPT44 function.

In addition to the lwB4 and lwAFTR, a provisioning element is responsible for configuring them. At time of deployment of a lwB4, the provisioning system configures it with its assigned Port-restricted IPv4 Address and the IPv6 address of its serving lwAFTR. It also centrally configures the lwAFTR with Port-restricted IPv4 Address and the IPv6 address of the associated lwB4.

For packets originating from a private IPv4 network, the customer's lwB4 performs the NAPT44, substituting the source IPv4 address and port by the assigned public IPv4 address and a port from the assigned Restricted Port-Set. It then encapsulates the packet within an IPv6 packet with its assigned lwAFTR as destination and hands off the packet to the IPv6 carrier network. When the lwAFTR receives the packet, it checks that the packet originated from a valid lwB4, removes the IPv6 header and sends the packet onto the public target IPv4 network (the internet).

Packets coming from the public IPv4 network arrive at the lwAFTR, which matches their destination IPv4 and port with the configured Port-restricted IPv4 Addresses. The corresponding entry also indicates the IPv6 address of the customer's lwB4. The lwAFTR then encapsulates the packet in an IPv6 packet and sends it, over the carrier IPv6 network, to the lwB4. The lwB4 removes the IPv6 header, applies the NATP44 substitution for incoming packets and delivers the packet into the private IPv4 network.

The scale out approach is to split the set of lwB4 between two lwAFTRs. During a scale out operation, a new element is required to perform a L3 matching and forwarding function on the former gateway to the v4-Internet, as shown in Figure 3.2 and Figure 3.3. This allows splitting the IPv4 address range that is assigned to the lwB4s. One efficient way to do so would be splitting in two prefixes that might just differ in one bit length.

This use case illustrates how, depending on the scenario, scaling rules can only be coming from within the service, whereas if the rules for scaling are generic enough, infrastructure base scaling out could be carried out. In this case, following the first approach, the trigger for scaling would come from the Control App overseeing the whole graph. In this use case, just introducing a load balancer would not be enough as the lwB4 will have exactly one uplink lwAFTR. (two uplinks would be needed when there would be a load balancer).

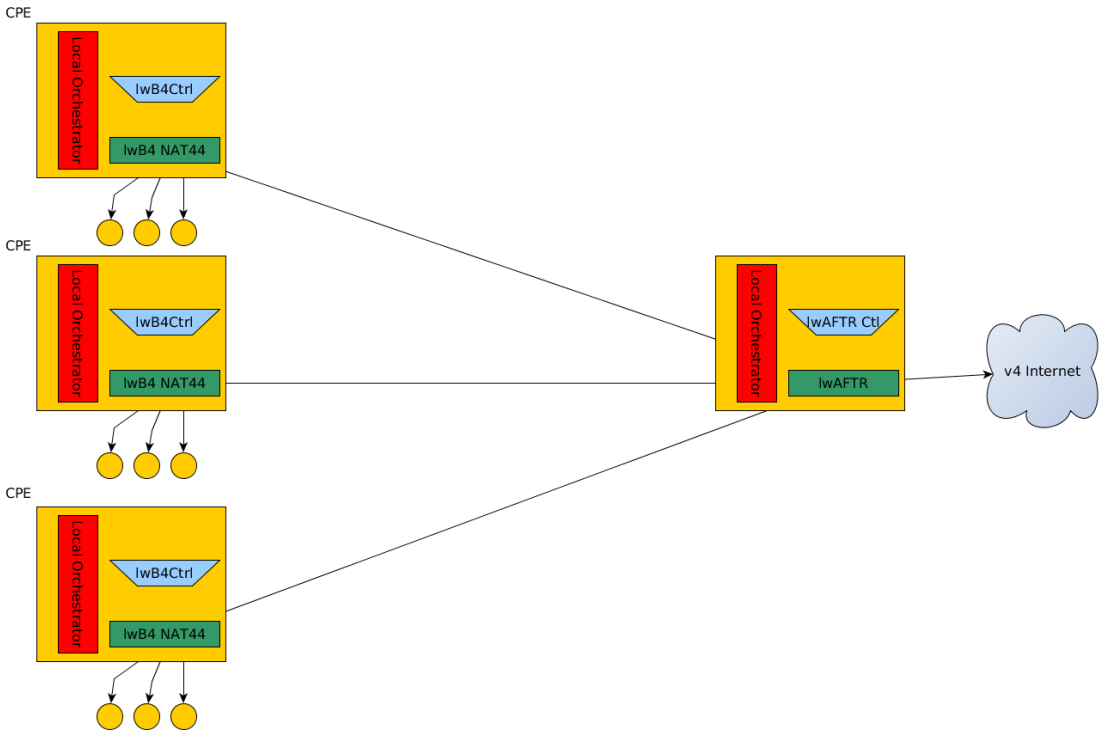

Figure 3.2 – Elastic lw4over6 before update

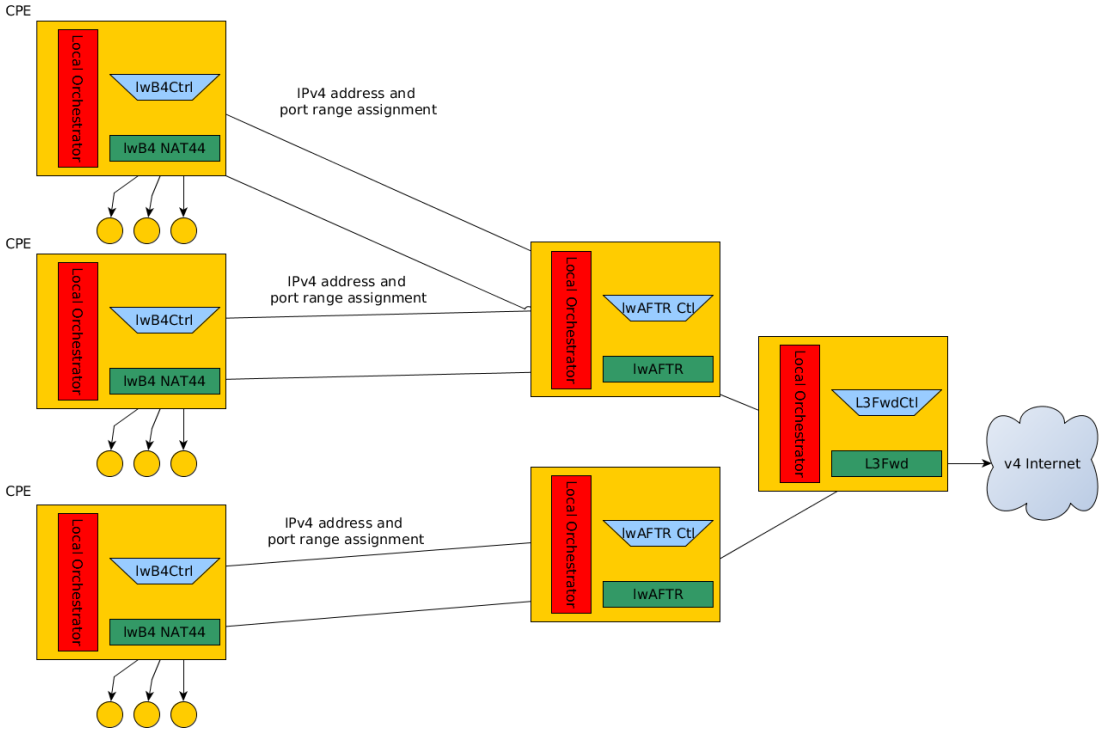

Figure 3.3 – Elastic lw4over6 after update

3.1 Initial assumptions

In order to define the scope of the use case, the following initial assumptions are made:

The UN bootstrapping process has been completed successfully beforehand so the node is available to deploy a Network Function Forwarding Graph (NF-FG).

The UNs depicted in the figures are either directly connected, so there is no need to manage and configure any intermediate network, or connected through an intermediate network that was pre-configured without intervention from the UNIFY layer.

The NF-FG handled to the UN for deployment is fully characterised, including the mechanism used to distinguish the inbound traffic for the NF-FG, and the VNF images to be executed are available at the UN (the process for retrieving the VNF image is not included in the use case).

The process for the different scaling approaches is within the scope of WP3 and described in D3.1 [3], while the UN must provide the means to support the required mechanism. Hence, in the scope of WP5 prototyping the decisions taken by the prototype Control App will be static and related to a pre-defined topology and resource view. That is, the decision process to be implemented by the Control App will be left out of the WP5 efforts.

Requests for scale up will be managed by the Control App of the element to scale up and sent through the Cf-Or to the Local Orchestrator. Requests for scale out will be managed by the Control App of the whole NF-FG and sent through the Cf-Or to the Orchestrator responsible for the whole resource domain.

The NFIB is also within the scope of WP3 and described in D3.1 [4]. For the purpose of the VNF Specifications and Images Repository Interface either a mock NFIB will be used in the WP5 prototype or the WP3 NFIB itself as a step towards the Integrated Prototype.

3.2 Use case process

The use case has been divided in three main processes, each of them aiming to demonstrate a different aspect of the UN capabilities in an incremental manner. The processes are outlined below, and the steps pertaining to each of these blocks are further detailed in the corresponding tables in next subsections. The first process describes the deployment of the Elastic lw4over6 in Table 4 and Figure 3.4, which includes the steps performed by the upper layers of the UNIFY framework for completeness. The second process describes an initial scaling that would be triggered by monitoring upon reaching a certain threshold in Table 5 and Figure 3.5. As an initial approach, the simplest alternative for scaling is used, adding more resources to the NF (scale up). The third process describes another scaling method in Table 6 and Figure 3.6, triggered once the scale up approach can be pushed no further, and that implies deploying additional instances of one of the Elastic lw4over6 elements in other UNs and splitting the load between them (scale out):

1. Initial deployment of the Elastic lw4over6.
    a. Upper layer processing.
    b. Deployment in UN(s).
2. Scale up of the Elastic lw4over6.
    a. Monitoring and scale decision.
    b. lwAFTR scale up.
3. Scale ouf of the Elastic lw4over6.
    a. Monitoring and scale decision.
    b. Deployment of l3Fwd.
    c. Deployment of new lwAFTR.
    d. Modification of old lwAFTR.
    e. Modification of lwB4.

3.2.1 Initial deployment

Step	Description	Input	Output	Actor
1.a	Upper layer processing			
1.1	Handle Elastic lw4over6 SG to deploy to Service layer	N/A	Elastic lw4over6 SG	Service User
1.2	Map Elastic lw4over6 SG to a specific NF type supporting scalability	Elastic lw4over6 SG	Elastic lw4over6 NF-FG with abstract NFs	Service layer
1.3	Fully characterize Elastic lw4over6 NF-FG to deploy and handle to the Orchestration layer	Elastic lw4over6 NF-FG with abstract NFs	Fully Characterized Elastic lw4over6 NF-FG	Service layer
1.4	Select a placement based on the NF types in the Elastic lw4over6 NF-FG, requirements and the available resources	Fully Characterized Elastic lw4over6 NF-FG	Place for deployment (several UNs)	Resource Orchestrator

1.5	Handle Elastic lw4over6 NF-FG to deploy to Controller Adaptation	Fully Characterized Elastic lw4over6 NF-FG Place for deployment (several UNs)	Fully Characterized Elastic lw4over6 NF-FG Place for deployment (several UNs)	Resource Orchestrator
1.6	Split Elastic lw4over6 NF-FG based on the placement	Fully Characterized Elastic lw4over6 NF-FG	Fully Characterized Elastic lw4over6 NF-FG sub-graphs	Controller Adaptation
1.7	Handle Elastic lw4over6 NF-FG subgraphs to the corresponding UN Unified Resource Managers	Fully Characterized Elastic lw4over6 NF-FG sub-graphs	Fully Characterized Elastic lw4over6 NF-FG sub-graphs	Controller Adaptation
1.b	Deployment in UN(s)			
1.8	Determine optimal placement and connectivity for the Elastic lw4over6 NF-FG subgraph	Fully Characterized Elastic lw4over6 NF-FG sub-graphs to deploy	Placement and connectivity determined	URM(s) - Local Orchestrator
1.9	Configure external and internal Elastic lw4over6 NF-FG connectivity	Fully Characterized Elastic lw4over6 NF-FG sub-graphs to deploy Placement and connectivity determined	Elastic lw4over6 NF-FG subgraph connectivity prepared	URM(s) – VSE Management
1.10	Instantiate the Elastic lw4over6 NF-FG subgraph NFs	Fully Characterized Elastic lw4over6 NF-FG sub-graphs to deploy Placement and connectivity determined	Elastic lw4over6 NF-FG subgraph NFs ready	URM(s) – VNF EE Management

1.11	Return Elastic lw4over6 NF-FG deployment and management information up to Service layer	Fully Characterized Elastic lw4over6 NF-FG sub-graphs	Elastic lw4over6 NF-FG deployment and management information	URM(s)

Table 4: Initial deployment of the Elastic lw4over6 NF

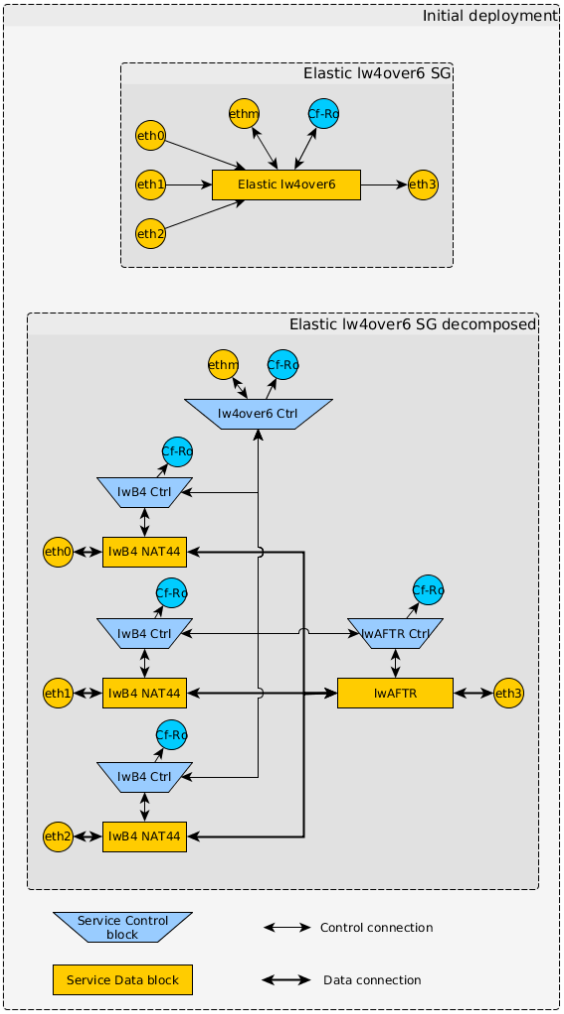

Figure 3.4 – Elastic lw4over6 Initial deployment

3.2.2 Scale up

Step	Description	Input	Output	Actor
2.a	Monitoring and scale decision			
2.1	Monitor lwAFTR performance	lwAFTR operation	lwAFTR performance	lwAFTR Ctrl
2.3	Determine necessity to scale up	lwAFTR performance	Decision to scale up Updated lw4AFTR NF-FG subgraph	lwAFTR Ctrl
2.b	lwAFTR scale up			
2.4	Request scaling up to Unified Resource Manager through Cf-Or	Decision to scale up lwAFTR NF to scale Updated lwAFTR NF-FG subgraph	Request to scale up lwAFTR NF to Unified Resource Manager Updated lwAFTR NF-FG subgraph	lwAFTR Ctrl
2.5	Assign additional resources to lwAFTR NF	Request to scale up lwAFTR NF Updated lw4over6 NF-FG subgraph	Additional resources assigned to lwAFTR NF	URM - Local Orchestrator
2.6	Deploy additional resources for lwAFTR NF	Additional resources assigned to lwAFTR NF	Scaled up lwAFTR NF	URM – VNF EE Management
2.7	Notify requester of the scale up process	lwAFTR scaled up	Notification of lwAFTR scaled up	URM

Table 5: Scale up of the Elastic lw4over6 NF

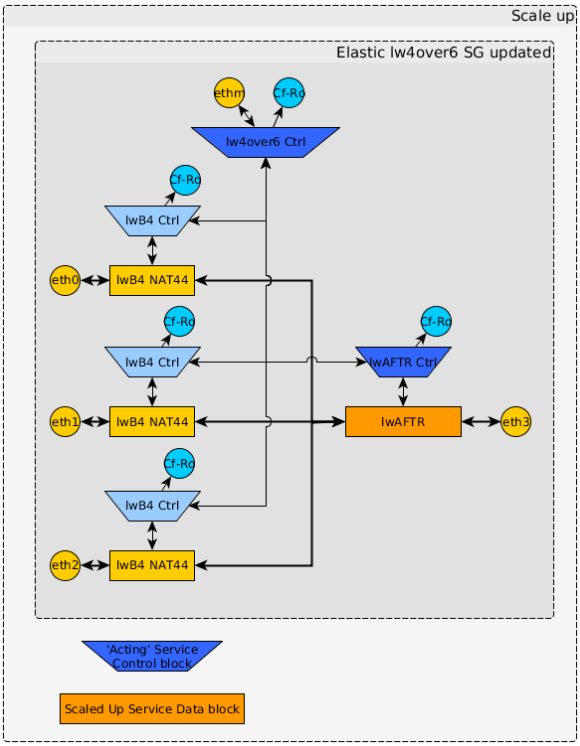

Figure 3.5 – Elastic lw4over6 Scale up

3.2.3 Scale out

Step	Description	Input	Output	Actor
3.a	Monitoring and scale decision			
3.1	Monitor lwAFTR performance	lwAFTR operation	lwAFTR performance	lwAFTR Ctrl
3.2	Notify lw4over6 Control App	lwAFTR performance	lwAFTR performance notification	lwAFTR Ctrl
3.3	Determine necessity to scale out	lwAFTR performance notification	Decision to scale out Updated lw4over6 NF-FG	lw4over6 Ctrl
3.b	Deployment of l3Fwd			
3.4	Request deployment of new l3Fwd NF-FG subgraph to Unified Resource Manager through Cf-Or	Decision to scale out	Request to deploy new l3Fwd NF-FG subgraph to Unified	lw4over6 Ctrl

			Resource Manager New l3Fwd NF-FG subgraph to deploy	
3.5	Determine optimal placement and connectivity for the new l3Fwd NF-FG subgraph to deploy	Request to deploy new l3Fwd NF-FG subgraph New l3Fwd NF-FG subgraph to deploy	Placement and connectivity determined	URM – Local Orchestrator
3.6	Configure external and internal l3Fwd NF-FG subgraph connectivity	New l3Fwd NF-FG subgraph to deploy Placement and connectivity determined	New lw3Fwd NF-FG subgraph connectivity prepared	URM – VSE Management
3.7	Instantiates the l3Fwd NF-FG subgraph NFs	New l3Fwd NF-FG subgraph to deploy Placement and connectivity determined	New l3Fwd NF-FG subgraph NFs ready	URM – VNF EE Management
3.8	Return new l3Fwd NF-FG subgraph deployment and management information to requester	New l3Fwd NF-FG subgraph to deploy	New l3Fwd NF-FG subgraph deployment and management information	URM
3.c	Deployment of new lwAFTR			
3.9	Request deployment of new lwAFTR NF-FG subgraph to Unified Resource Manager through Cf-Or	Decision to scale out Updated lw4over6 NF-FG	Request to deploy new lwAFTR NF to Unified Resource Manager New lwAFTR NF-FG subgraph to deploy	lw4over6 Ctrl

3.10	Determine optimal placement and connectivity for the new lwAFTR NF-FG subgraph to deploy	Request to deploy new lwAFTR NF-FG subgraph New lwAFTR NF-FG subgraph to deploy	Placement and connectivity determined	URM - Local Orchestrator
3.11	Configure external and internal lwAFTR NF-FG subgraph connectivity	New lwATFR NF-FG subgraph to deploy Placement and connectivity determined	New lwATFR NF-FG subgraph connectivity prepared	URM – VSE Management
3.12	Instantiate the lwAFTR NF-FG subgraph NFs	New lwATFR NF-FG subgraph to deploy Placement and connectivity determined	New lwATFR NF-FG subgraph NFs ready	URM – VNF EE Management
3.13	Return new lwAFTR NF-FG subgraph deployment and management information to requester	New lwATFR NF-FG subgraph to deploy	New lwATFR NF-FG subgraph deployment and management information	URM
3.d	Modification of old lwAFTR			
3.14	Request modification of lwAFTR subgraph to update connectivity to Unified Resource Manager through Cf-Or	Decision to scale out New l3Fwd NF-FG subgraph deployment information	Request to update lwAFTR NF-FG subgraph to Unified Resource Manager Updated lwAFTR subgraph	lw4over6 Ctrl
3.15	Update external lwAFTR NF-FG subgraph connectivity	Updated lwAFTR subgraph	Updated lwAFTR subgraph connectivity modified	URM – VSE Management

3.16	Return updated lwAFTR NF-FG subgraph deployment and management information to requester	Updated lwAFTR subgraph	Updated lwAFTR subgraph deployment and management information	URM
3.e	Modification of lwB4			
3.17	Request modification of lwB4 NAT44 subgraph to update connectivity to Unified Resource Manager through Cf-Or	Decision to scale out New lwATFR NF-FG subgraph deployment information	Request to update lwB4 NAT44 NF-FG subgraph to Unified Resource Manager Updated lwB4 NAT44 subgraph	lw4over6 Ctrl
3.18	Update external lwB4 NF-FG subgraph connectivity	Updated lwB4 NAT44 subgraph	Updated lwB4 NAT44 subgraph connectivity modifed	URM – VSE Management
3.19	Return updated lwB4 NAT44 NF-FG subgraph deployment and management information to requester	Updated lwB4 NAT44 subgraph	Updated lwB4 NAT44 subgraph deployment and management information	URM
3.20	Notify lwB4 NAT44 Ctrl of the new assigned lwAFTR	New lwATFR NF-FG subgraph deployment and management information	Notification of updated lwAFTR assignment	lw4over6 Ctrl
3.21	Update lwB4 NAT44	Notification of updated lwAFTR assignment	Updated llwB4 NAT44	lwB4 Ctrl

Table 6: Scale out of the Elastic lw4over6 NF

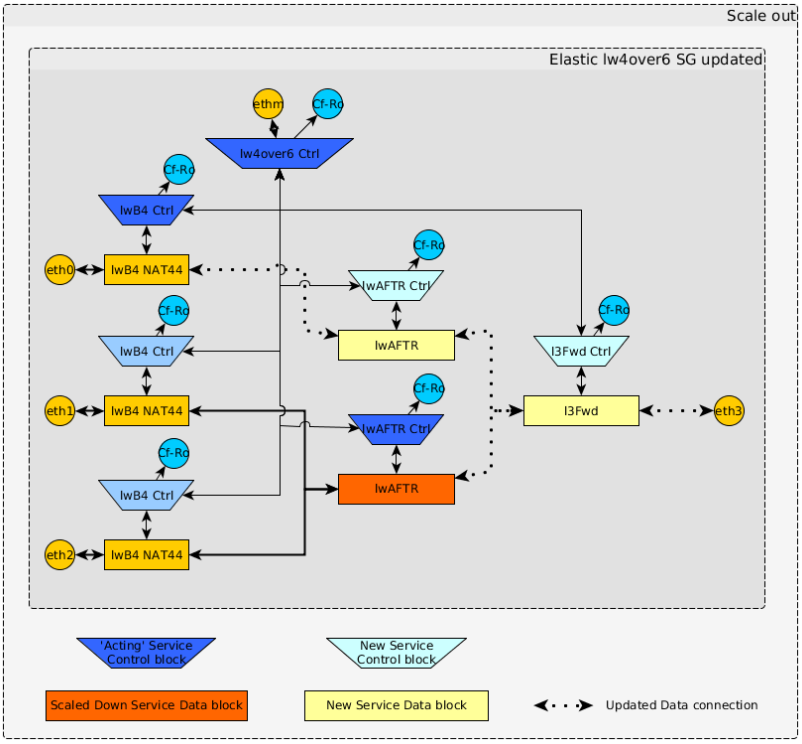

Figure 3.6 – Elastic lw4over6 Scale out

4 Summary

In this deliverable we covered the description of WP5 prototype and define the implementation plan to develop this prototype. We have identified three phases; the first one describes the work that has been done in T5.2 Implementation, which has released the first version of this prototype. The remaining phases describe the work to be done in WP5 related to the prototype implementation.

We have also further detailed all the functionalities from components and interfaces to be implemented in the UN that were previously identified in D5.2. Moreover, we have defined the implementation plan which describes to what extent all these functionalities and interfaces will be developed in the three phase approach. Although the objective is to build a functional prototype that implements all these functionalities, we will focus most of the efforts on those novel aspects that brings innovation to the UN rather than spending efforts in building a full featured prototype.

Regarding the use cases to be implemented, we have described the current target use case, the Elastic Lightweight 4over6, and the different processes to be demonstrated (i.e. initial deployment, scale up and scale out). However, this somehow also overlaps with WP3 functionalities, and as a consequence, we will continue to align our work with the efforts and developments done in WP3 in order to avoid any duplication of work.

Since the T2.3 Integration of prototyping activities is an on-going work and one of our objectives is to prepare the WP5 prototype for inclusion in the Integrated Prototype, we have tried to be flexible enough in our implementation plan to deal with the final use cases being selected for this joint effort between all WPs. Therefore, we will align our work in WP5 prototype with T2.3, since its goal is to coordinate inter-WP developments.

List of abbreviations and acronyms

Abbreviation	Meaning
AFTR	Address Family Transition Router
B4	Basic Bridging BroadBand element
CA	Controller Adaptation
DPDK	Data Plane Development Kit
KVM	Kernel-based Virtual Machine
LO	Local Orchestrator
LSI	Logical Switch Instance
LXC	LinuX Containers
NAPT	Network Address and Port Translation
NAT	Network Address Translation
NF	Network Function
NF-FG	Network Function Forwarding graph
NIC	Network Interface Card (often refers to a physical network port regardless of its presence on a card)
OP	Observability Point
OVS	Open vSwitch
UN	Universal Node
URM	Unified Resource Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNF EE	VNF Execution Environment
VSE	Virtual Switching Engine
xDPD	eXtensible DataPath daemon

References

[1] UNIFY Consortium, „D5.2 Universal Node Interfaces and Software Architecture," 31 August 2014. [Online]. Available: https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY-WP5-D5.2-Universal%20node%20interfaces%20and%20software%20architecture.pdf. [Zugriff am 1 December 2014].

[2] UNIFY Consortium, „D5.1 Universal Node functional specification and use case requirements on data plane," 11 March 2014. [Online]. Available: https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY-WP5-D5.1-Universal%20node%20functional%20specification.pdf. [Zugriff am 1 December 2014].

[3] UNIFY Consortium, „D3.1 Programmability Framework," 14 November 2014. [Online]. Available: https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY_D3.1%20Programmability%20framework.pdf. [Zugriff am 1 December 2014].

[4] UNIFY Consortium, „D3.1 Programmability Framework," 2014. [Online]. Available: https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY-WP3-D3.1-Programmability%20Framework.pdf.

[5] Y. Cui, Q. Sun, M. Boucadair, T. Tsou, Y. Lee and I. Farrer, "Lightweight 4over6: An Extension to the DS-Lite Architecture," 6 June 2014. [Online]. Available: https://tools.ietf.org/html/draft-ietf-softwire-lw4over6-10. [Accessed 4 November 2014].

[6] A. Durand, R. Droms, J. Woodyatt and Y. Lee, "RFC 6333: Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion," August 2011. [Online]. Available: http://tools.ietf.org/html/rfc6333. [Accessed 4 November 2014].

[7] UNIFY Consortium, „D2.2 Final Architecture," 15 November 2014. [Online]. Available: https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY-WP2-D2.2-Final%20architecture.pdf. [Zugriff am 1 December 2014].

[8] Intel, „Intel Data Plane Development Kit (Intel DPDK) - Programmer's Guide," June 2014. [Online]. Available: http://dpdk.org/dpdk/intel/dpdk-prog-guide-1.7.0.pdf. [Zugriff am 1 December 2014].