SUPERFLUIDITY

a super-fluid, cloud-native, converged edge system

Research and Innovation Action GA 671566

Deliverable I2.3:

First functional analysis and decomposition

Deliverable Type:	Report
Dissemination Level:	CO
Contractual Date of Delivery to the EU:	31/12/2015
Actual Date of Delivery to the EU:	08/01/2016
Workpackage Contributing to the Deliverable:	WP2

Editor(s):	Lionel Natarianni (ALBLF), Stefano Salsano (CNIT), Erez Biton (ALU-IL)
Author(s):	Lionel Natarianni (ALBLF), Erez Biton (ALU-IL), Omer Gurewitz (ALU-IL), Giuseppe Bianchi (CNIT), Nicola Blefari-Melazzi (CNIT), Stefano Salsano (CNIT), George Tsolis (CITRIX), Isabel Borges (PTIN), Francisco Fontes (PTIN), Carlos Parada (PTIN), Pedro A. Aranda (Telefónica, I+D), Ignacio Berberana (Telefónica, I+D), Costin Raiciu (UPB), Dirk Griffioen (Unified Streaming), Okung Ntofon (British Telecom), John Thomson (OnApp), Julian Chesterfield (OnApp), Michael J. McGrath (Intel), Pedro de la Cruz Ramos (Telcaria Ideas S.L.), Juan Manuel Sánchez Mateo (Telcaria Ideas S.L.)

Internal Reviewer(s)	
Abstract:	
Keyword List:	

INDEX

List of Figures

List of Tables

Glossary

(TO BE FILLED OUT IN THE FINAL VERSION OF THE DELIVERABLE)

SUPERFLUIDITY DICTIONNARY	
TERM	DEFINITION

Table 1: SUPERFLUIDITY Dictionary.

1 Introduction

Network Function Virtualization (NFV) is emerging a new architectural concept for the design and implementation of communication networks. Running network functions in software is a departure from the current monolithic approach to building and deploying hardware appliances. Such appliances are difficult to customise, upgrade and scale and lead to vendor lock-in. The main benefits of running network functions completely in software are envisioned to be:

1. Ease of developing new functionality
2. Seamless scale-out by adding more resources and load balancing across them.
3. Easy upgrade and more agile bug fixing.

Despite being a relatively recent proposal, the specification / standardization of the NFV paradigm has already achieved significant results ([1] to [4]) and there are already implemented solutions. The current architecture is built around the concept of VNFs (Virtual Network Functions) that can be composed to provide Network Services. The VNFs are instantiated and executed over a hosting environment denoted as VNF Infrastructure (VNFI).

In order to fully exploit the NFV potential, re-thinking the way of designing and modelling the telecommunication services is needed. Before NFV, the focus of modelling was on interfaces between physical boxes, with the identification of reference points and of the protocols exchanged across the reference points. The physical boxes were closed; hence, there was no need to specify their internal structure. With the advent of virtualization technologies and NFV, services can be realized by combining functional blocks that are much like software components. Such components can be deployed and executed over a distributed computing infrastructure, composed a set of big Datacentres and a very large number of distributed resources closer to the access networks (this is referred to as *Fog computing* [5] or *Mobile Edge Computing* [6]). In the 5G context, this scenario is extended to the Radio Access infrastructure, which turns in a *Cloud-RAN* [7].

An optimal allocation of processing components in this highly distributed cloud environment is the key to optimize performances, reduce costs (operational costs and/or equipment buying costs) and achieve higher efficiency. Hence, the desire to decompose functions up to a very high granularity and to extend this approach in a unified manner to Radio Access functionality and to Data Plane "microscopic" forwarding operations. From an ideal perspective, it should be possible to decompose a service in an arbitrary way (from the point of view of the needed resources) and map it in the most convenient way into the set of resource providers (e.g. processing hosts) offered by the infrastructure.

In order to extend the current approach in NFV function decomposition toward including Radio Access functionalities and Data Plane microscopic forwarding operations, a fundamental question is: *what is an appropriate programming language for network processing?* In traditional software development any of the numerous programming languages can be chosen, however there is always a basic set of services that are constant and offered by the operating system (e.g. POSIX) or the virtual

machine (e.g. the JVM). Regardless of the OS, the programmer relies on these basic OS primitives as well as libraries provided by the OS/programming platform to build their own functionality. Underneath the OS, the hardware is fairly uniform: an x86 or ARM processor, main memory and I/O devices such as disk and network. The Posix API and the virtualisation software efficiently multiplexes requests from multiple processes to the same hardware.

There is neither an equivalent of an OS for network processing, nor a specialised API that applications can be built upon; the existing POSIX API is cumbersome to use as it forces all network apps to re-implement basic functionality such as Ethernet/IP header processing/firewalling over and over again. Furthermore, it is highly likely that the hardware used to run network functions will include specialised boards such as TCAMs and NetFPGAs; Robust isolation and multiplexing techniques to allow the sharing of hardware features on the same platform by multiple users are currently lacking. To complicate matters further, the market for network functions is quite complex: it includes deployers (network operators), users of the functionality (network operators and third-parties), but also software and hardware network function vendors.

Superfluidity, aligning with current industry trends, aims to solve this complex issue by defining a set of reusable components that are common across many network-processing functions. These reusable components will serve two main purposes:

- They will allow more complex functionality to be built by combining small, purpose-specific building blocks
- They will allow each building block to be implemented in a variety of ways, including as generic software running on x86/ARM, NetFPGA, specialised ASICs, TCAMs, and so forth.

The element that combines these reusable components is the network programming language. As in general purpose programming languages, it is likely there will be a wealth of languages defined and used. Existing language examples include the configurations used by the Click Modular Router [8] targeted at software routers functionality and GNU Radio [9] targeted at software radio processing.

The reusable components will act as the API offered by network OS to network applications. As in all APIs, there is great freedom in the implementation as long as the API is observed – and this will allow software/hardware vendors to compete in offering implementations of the same functionality and network operators to decide which solutions it wants to deploy based on requirements derived from its own operations or taken from its customers.

This document offers an initial assessment of the reusable components in different domains of the networking landscape including the core network, the Cloud RAN and mobile edge computing. We proceed to decompose existing monolithic network functionality into reusable components, and report our key observations to date. Our goal is not to build a definitive list of reusable components that should be standardised, but rather to start understanding what are the useful functionalities that might be the candidates for standard reusable components.

A fundamental question when decomposing monolithic network functions is to select the right level of granularity. There is a direct trade-off between reusability and performance. On the one hand, fine-grained functions may be easier to reutilise to build complex applications, but have higher instantiation costs and per-flow or global state management across components will become complex and create further overheads. Coarse-grained functions are more efficient, however they will be difficult to reuse. This work does not seek to dictate the appropriate granularity for reusable components; instead, it decomposes monolithic functions using the *appropriate* building blocks for each part of the 5G network, as determined by experts in those areas. The outcome is informative rather than normative, and it provides a concrete starting point for the 5G architecture.

Orthogonal to the granularity of the reusable components, the process of building complex functionality depends on ability of all the involved parties' to ***understand*** what each reusable component is supposed to do: what its allowed inputs are and how the outputs are obtained from the inputs. This is equivalent to the POSIX contract: what are the allowed ranges and formats for input parameters for system calls, and what is the expected output, error conditions and so forth. Another example is the contract between the CPU and the programmer: what each CPU-level instruction does, what the inputs and outputs are, etc.

This API contract is fundamental to decouple implementation from utilisation, yet it is completely missing today in works that aim to virtualise network functions such as ETSI NFV. To understand why this is the case, consider a simple firewall reusable component. At first sight, its semantics are obvious: it will drop all traffic not wanted by the network operator. However, there are many subtleties:

- What type of traffic does it accept beyond TCP and UDP? Does it filter traffic in tunnels, and if so which ones?
- Does it allow stateful processing (e.g. allow outgoing traffic and related incoming traffic)?
- Does it scan TCP options and drops unknown ones?
- Does it parse payloads and remove viruses?

All of these are plausible functionalities of the firewall, and they need to be expressed clearly so that users of the firewall component know exactly what it does, so that they compose it correctly with other reusable components. For instance, using tunnelling before the firewall may completely disable it if the firewall does not deal with that type of tunnel; and using a proxy after the firewall means that any filtering based on source and destination addresses may be rendered useless.

A key contribution of Superfluidity is to recognise the importance of associating semantics to each reusable component in a way that allows to safely compose them into correctly functioning network-wide applications. We believe this is the cornerstone of the 5G architecture, and offer an initial discussion of possible ways in which the semantics may be described in section 4.1. In particular, we identify two main approaches:

- The use of pre and post-conditions in the API to ensure type safety when traffic crosses multiple reusable components.
- The use of a modelling language to describe, at higher level, what each reusable component is doing. We can then use symbolic execution to understand how different components would when applied together to traffic.

2 Modelling of reusable functional blocks

The decomposition of high-level monolithic functions into reusable components is based on the concept of Reusable Functional Block (RFB). A RFB is a logical entity and it is the generalization of the concept of VNF (Virtual Network Function) and of VNFC (Virtual Network Function Component). A Reusable Functional Block can be composed of other RFBs. RFBs can be composed in graphs to provide services or to form other RFBs. In general, a Reusable Functional Block can hold state information, so that the processing of information coming in its logical ports depends on such state information.

A Superfluid RFB is characterized by a set of properties. An important property is the environment on which the RFB can run (or can be hosted), referred to as *hosting_environment*. For example, a RFB that can be mapped 1:1 against a single Virtualisation Container (using the ETSI terminology) corresponds to an ETSI VNFC (VNF Component). In this case, its *hosting_environment* is a Virtualization Container hypervisor. A RFB that corresponds to a Click element, will have as *hosting_environment* a Click router. A logical RFB can have more than one possible hosting environment (i.e. it can be realized using different technologies and frameworks).

RFBs can be characterized by their resource needs (e.g. storage, processing) or load limitations (e.g. maximum number of packets/s or number of different flows). A Reusable Functional Block is characterized by a set of *logical ports* of different types: e.g. data plane ports, control plane ports, management plane ports. A logical port represents a flow of information going in and/or out of the RFB (see below more details and examples on the definition of logical ports).

RFBs characterization can be augmented with formal description of their behaviour, as needed to formally derive the behaviour of a graph composing a set of RFBs.

Figure 1~~Figure 1~~ highlights the difference between the current model considered in the ETSI standardization and the proposed approach. In the ETSI model, there is a fixed hierarchy between VNF Groups (defined in [3] but not considered in [4]). VNFs and VNF Components. VNF Components cannot be further decomposed. In the proposed model, all elements are RFB and the decomposition can be iterated an arbitrary number of time.

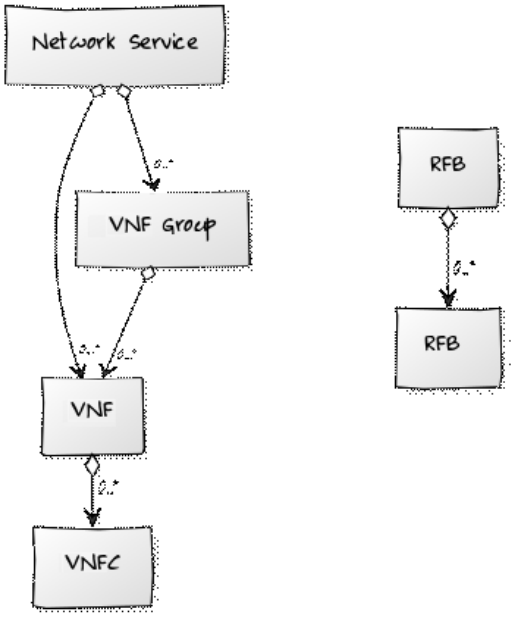


Figure 1. Class diagram for the current approach (left side) and the proposed one (right side).

To make an example of hierarchy, a Click router instance can be a VNF Component in a VNF. Its *hosting_environment* is a VC hypervisor. The Click instance is described by means of a directed graph of *elements* (called *configuration*). Each element is a RFP that has the Click router as *hosting_environment*. As shown in Figure 2~~Figure 2~~, in the current modelling approach it is not possible to further decompose the VNFC.

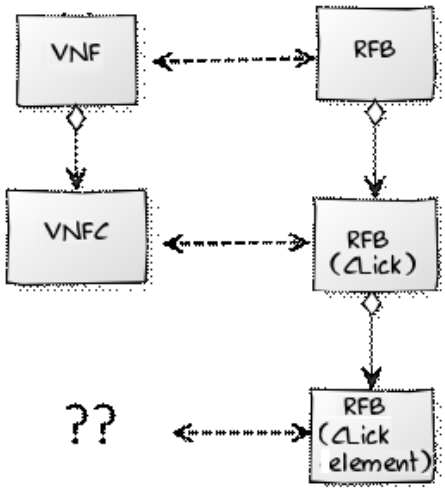


Figure 2. Example of a hierarchy of RFBs

In order to understand how the RFBs can be combined, the consistency between the information exchanged over the ports needs to be checked. Therefore, an appropriate modelling of this information is needed. The logical ports will be mapped into *port instances* when a RFB is deployed

in a concrete configuration. The logical ports can be combined and can be supported by *aggregated* logical ports. For example, an IP router working with the OSPF protocol can be represented as having/offering two types of logical ports: data plane ports for routing of IP packets, OSPF control plane ports for discovering OSFP adjacent entities and exchanging routing information with them. The two logical ports are combined in an *aggregated* logical port, which supports the exchange of IP packets, both data plane and OSPF control plane packets.

3 Functional Decomposition for the wireless access domain

3.1 Models for the description of Functional Elements and their relationships/ interconnections (wireless domain)

Introduction

Today, the mobile network architecture is a composed of many elements: (PGW, eNodeB, etc.) hosting a set of functions implemented as monolithic software or dedicated hardware components. 5G will be more than a new air interface and will consist of a new and distribute network architecture that will provide high flexibility in services and great scalability. However, simply using the latest emerging virtualisation technologies to address 5G requirements is insufficient. It is necessary to re-examine this functional split across functional elements as reusable Wireless Functional Blocks (WFB).

Wireless Function Blocks (WFB)

The first approach to model WFBs is based on functional categorisation and re-aggregation in order to instantiate one example from each function.

For example, in todays' 4G RAN, there are several cases where similar functions are implemented in multiple duplicated elements (especially when considering 3GPP and Fixed):

- Authentication/Authorisation in MME, SeGW and capability sets.
- Security functions in SeGW and MME.
- Selection / Load-balancing in MME and ePDG.
- Routing / bearer management function in SGW, PGW and ePDG.

These categorised WFBs provide the capability to instantiate one example from each function and then give the possibility, thanks to scalability to provision infrastructure in fine-grained manner (Figure 3~~Figure 3~~). This, coupled with an on-demand infrastructure, reduces operational costs (capex/opex).

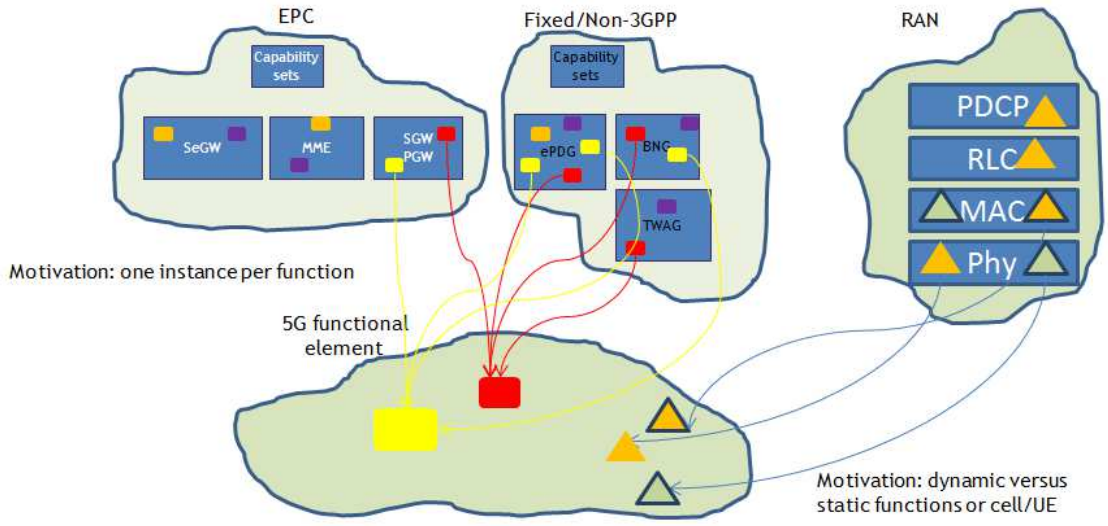


Figure 3. Mapping of 5G functional elements into EPC, Fixed/Non 3GPP and RAN domains

A second approach consists of modelling WFBs according to future 5G requirements and other 5G projects or initiatives (NORMA, IJoin, Crowd, …). The key steps in this approach are as follows:

- Categorise each identified RAN functions into U-plane/C-plane and user/cell related
- Identify dependencies between blocks.
- Categorise RAN functions according to timing requirements for with respect to virtualisation, function inter-dependency and middleware for supporting specific offloading capabilities on x86 or programmable FPGA. As example, at cell level, UE/Cell classification allows to implement "traffic based" scalability by splitting static cell functions and dynamic user functions. Such a split enables "horizontal scalability" where on-demand infrastructure is needed, following the increase of number of users (see Figure 4~~Figure 4~~).

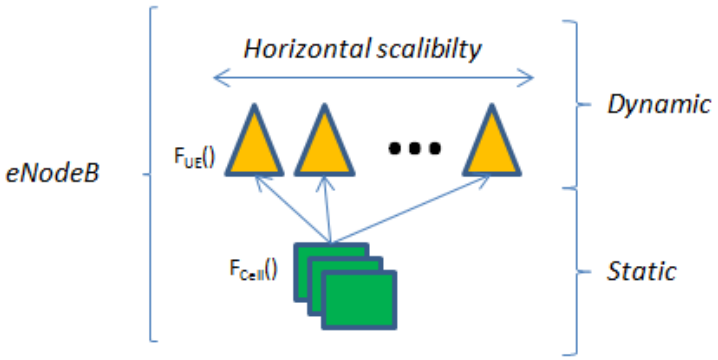


Figure 4. Horizontal scalability approach

Another 5G key challenge is to provide simultaneous support to a variety of services ranging from high throughput like HD video streaming scenario to critical services characterised by low latency

(few milliseconds) and high reliability like autonomous driving, online video game or remote control for robots.

Slicing

Network slicing (introduced in the NGMN white paper [15]) is a key concept in 5G that will provide multi-level and multi-service architecture on the same infrastructure. Slicing allows to split 5G network infrastructure in different functional, logical networks, where each of them offers a specific range of service (IoT, M2M, Multimedia…). WFB can be deployed and chained independently at each slice level. Each slice is composed of WFBs and provides strong functional isolation between logical networks. No WFB communication among different slices is possible.

Flexiblity and New services

To natively support a high level of adaptability, each WFB should be able to support and implement specific 5G enabling technologies. One is “Adaptive decomposition and allocation of WFB”, C-plane and U-plane localisation depends on service requirements. The other is “Joint optimisation”, mobile access WFBs and core network WFBs can be co-localised on Edge Cloud (EC) or Network Cloud (NC). To support these 5G features, WFB should implement a split of c-plane and u-plane in EC and in NC in a way that they can be dynamically instantiated, relocated and grouped with no functional overhead. Main consequence of WFB implementation is that separation of functionality between the core network and the edge cloud is going to disappear. As an example, for low latency services like video games scenario, S-GW & P-GW can be moved from NC to the EC. In a similar way, for high throughput scenario like HD video, S-GW & P-GW WFBs will be instantiated on the network cloud.

Flexibility provided by these 5G enabling features allows the design of a Mobile Network architecture that natively adapts to different type of services, which have different type of requirements in terms of:

- Mobility,
- Latency,
- Traffic Volume,
- Security,
- Power Consumption Optimization.

Multi tenancy

5G will provide new business opportunities to operators and infrastructure providers for developing the concept of multi-tenant mobile network. In this multi-tenant scenario, infrastructure is shared among several tenants via open APIs to support on-demand allocation of edge and network cloud resources.

WFBs in 5G multi-tenant environments are defined per tenant. Each tenant can deploy its own set of WFBs in its own domain. Each tenant represents a software isolation between WFB domains. No WFB can be shared among several tenants.

5G WFB deployment

To address 5G flexibility requirements, WFB will be deployed on a specific, topologically based, cloud architecture composed of:

- An Edge Cloud (EC) closer to the antenna,
- A Network Cloud (NC) or central cloud.

WFBs should embed capabilities to enable them to be alternatively deployed and shipped over the network, from the EC or to the NC according to timing and other 5G flexibility requirements.

WFB implementation

4G RAN network function are traditionally implemented as large monolithic applications with large codebases used to grow as we write code and add new features. Over time, it becomes more difficult to know where change needs to be made because the codebase is so large. Another problematic aspect of monolithic applications is they are not able to scale very well.

To implement 5G WFB we recommend to follow a "Microservice" based design concept [16]. Each WFB can be implemented as a single microservice or as a bundle of microservices (see Figure 5~~Figure 5~~).

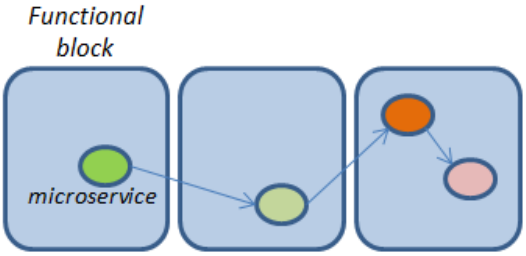


Figure 5. Microservice implementation of a WFB

Microservice is a (re) emergent technology and represents the atomic and independent building elements that compose each WFB.

Microservices can be defined as small autonomous services that work together and are focused in doing one thing in a very efficient way.

Robert C. Martin's Microservice definition [16] gives the "Single Responsibility Principle", which states, "Gather together those things that change for the same reason, and separate those things that change for different reasons".

WFB benefits from Microservices

WFB implementation as microservices brings a number of benefits:

- Autonomous: Each WFB can be a separate entity that can be deployed and managed independently from underlying operating systems, VMs or hypervisor.
- Technology heterogeneity: Network composed of multiple WFBs and microservices can use different technologies inside each of them. If one part of the Network needs to be improved, it is possible to use different technologies inside WFB to achieve the performance level required.
- Resilience: If a WFB fails, failures do not cascade. Problems can be isolated and the rest of the network can carry on. Note that resilience could be complicated to implement in CRAN with synchronous WFB interconnections (see WFB interconnection).
- Scalability: With monolithic services, we have to scale everything together. With smaller WFBs, we just can scale those services that need scaling, allowing other WFBs to run on smaller, less powerful hardware. Scalability is achieved via on-demand provisioning systems and allows us to control costs more effectively.
- Ease of deployment: Monolithic applications require the whole application to be redeployed in order to realise the required changes. With microservices based WFBs, we can make a change to a single service and deploy it independently of the rest of the system. This enables faster deployment of code. If a problem does occur, it can be isolated quickly to an individual service, making fast rollback easy to achieve.
- Agility and organisational alignment: A large codebase shared within large teams is a source of various problems, which are exacerbated when teams are distributed throughout an organisation. WFB approach allows minimising the number of people working on the same codebase, enabling better overall productivity. We reduce the time to change and deploy and we shift ownership of services and keep people working collocated.
- Optimised for replace-ability: As individual services are small in size, the cost to replace them with a better implementation is reduced, or in cases where removal is required it is much easier to manage. The barriers to rewriting or removing services entirely are relatively low.

WFB interconnectivity

Wireless functional blocks are inherently interconnecting services mainly using synchronous communication principles. This characteristic can have significant impact and compromise our capacity to design a fully autonomous and scalable WFB based network in 5G. With synchronous communication, a call is made to a remote server, which blocks the client until the operation completes. This behaviour has the side effect of tying the WFBs to each other thus breaking the WFB

isolation principle. It also drastically reduces the advantage of building a distributed architecture based on WFBs.

With asynchronous communication, the caller does not have to wait for the operation to complete before proceeding. These two different modes of communication enable two idiomatic styles of collaboration styles: request/response or event-based. Clearly, a request/response model is aligned with synchronous communication but can work for asynchronous communication also. It just needs an operation and to register a callback, asking the server to let the client application know when its operation has completed. Event-based collaboration nicely fits our WFB model and microservice communication because it is highly decoupled. In order to avoid synchronous communication for our 5G WFBs network, measures and investigations have to be done in fast event-based processing system domains (like efficient finite state machines) able to replace our synchronous WFB communication with no overhead on our 5G network of WFBs.

4 Functional Decomposition for the “wired” domain (routing / services)

4.1 Models for the description of Functional Elements and their relationships/ interconnections (“wired” domain)

4.1.1 ETSI NFV modelling approach

The NFV architectural model (Figure 6~~Figure 6~~) is based on the concept of Virtualised Network Functions (VNFs) that can be chained with other VNFs and/or Physical Network Functions (PNFs) to realize a Network Service (NS). The environment that hosts the VNFs is called NFV Infrastructure (NFVI). The NFV standardisation in ETSI has considered the use cases for NFV in [1] and the architectural aspects in [2][3]. In particular, the overall architectural framework for NFV is described in [2], while [3] provides further details on the architecture of VNFs.

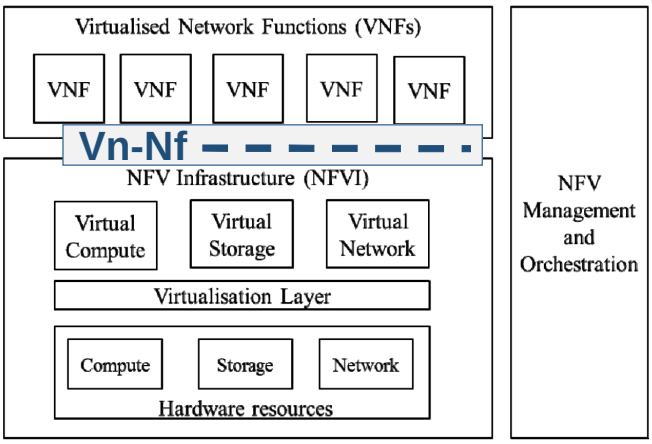


Figure 6. High level NFV framework.

The interface or *reference point* between VNFs and the NFV Infrastructure is called Vn-Nf. The Vn-Nf is a special kind of reference point. In fact, a typical interface or reference points describes the information exchanged between two functional entities and may provide details of the related protocols. On the other hand, the Vn-Nf needs to describe how the NFV Infrastructure can host a VNF, which resources it can provide and several other non-functional characteristics. In short, the Vn-Nf interface needs to describe the execution environment offered by the NFVI to a VNF.

According to [3], VNFs can be decomposed in VNF Components (VNFCs). In this case, as shown in Figure 7~~Figure 7~~, the VNFCs are the deployment units that are deployed on the NFV Infrastructure over the Vn-Nf interface.

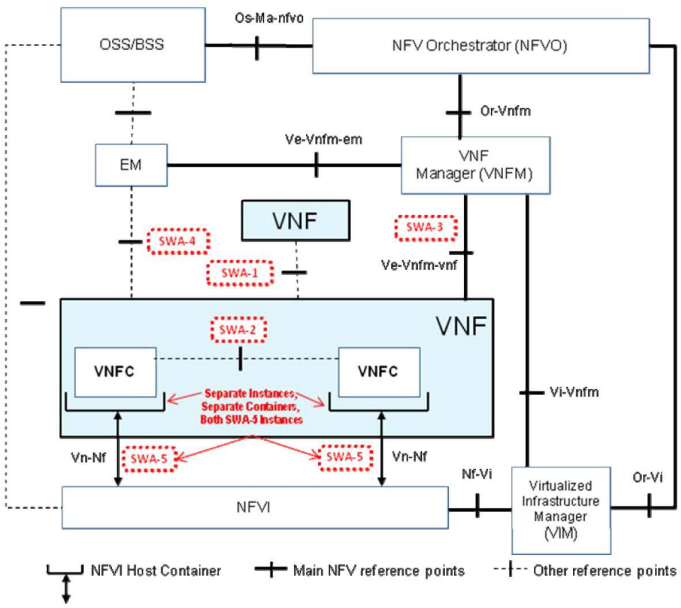


Figure 7. VNF interfaces and NFV reference points

The *orchestration* aspects, i.e. how VNFs can be chained and how their life cycle can be managed are dealt with in [4] (NFV Management and Orchestration, in short *MANO*). It is interesting to note that moving from the overall framework definition ([2]) to the VNF architecture ([3]) and finally to the management and orchestration aspects ([4]), the level of description changes from a general one to a concrete and detailed one. This means that several degrees of freedom that are included in overall framework ([2]) and also in the VNF architecture ([3]) are not present in the MANO specification [4]. In fact, the VNF architecture document [3] provides the conceptual model and a set of requirements that a concrete NFV system should implement, while MANO is much more prescriptive. It includes a set of restrictions in its models so that the specification is much closer to the implementation (an Open Source implementation of MANO, called Open Mano has been developed [10]).

Considering the mapping between VNFs and the NFV Infrastructure (NFVI), the MANO approach is based on a predefined rigid structure. Network Services are obtained by chaining VNFs, VNFs can be decomposed in VNF Components (VNFC). The NFV Infrastructure allows the deployment of VNFs. Concepts like grouping of VNFs to provide other VNFs are not supported. VNFs are implemented with one or more VNFCs and it is assumed that VNFC Instances map 1:1 to the NFVI Vn-Nf interface. The assumption is that the NFV Infrastructure offers a single type of support for running VNFs, that is a Virtualised Container. The main contribution of MANO is the definition of *descriptors* that can be used to characterize Network Services, VNFs, VNF Forwarding Graphs, Virtual Links, and Physical Network Functions in a consistent model. A MANO compliant Orchestrator implementation takes these descriptors in input and it is able to deploy the VNFs and the Network Services over the NFV Infrastructure. The Open Mano project provides a nice explanation and some examples of the most important descriptors in [11].

4.1.2 Definition of a new modelling approach

Defining the semantics of functional elements is crucial to ensure that those functional elements can be composed correctly into more complex network applications but it is a challenging task. We must define an abstract representation for each functional element using some language (whether a regular programming language or a specification language), and then devise a way to check that certain properties such as loop-freedom, reachability, etc., hold in complex configurations of functional elements.

To tackle this challenge there are two related questions that must be answered. First, what is the language that will be used to describe the semantics? Choosing multiple domain specific languages (such as C for x86 code or Verilog for ASICs) is not possible: we must select a single language that expresses the functionality of all functional elements, regardless of how they are implemented. Selecting one of these domain-specific languages, say C, to represent all functional elements is inappropriate: for one, they "tie" the implementation of the functional elements to C semantics.

Secondly, what properties do we want to check with respect to the configurations of several functional elements? The techniques we can use to prove certain properties are listed below in order of their power and decreasing order of speed:

- *Type checking* - checking that the output of a functional element can be used as the input to another functional component. This is very fast and easy to specify. Type checking can for instance, check the tunnel/firewall issues discussed earlier. However, it is also quite limited: it can only flag type violations when chains of functional elements are composed, giving weak assurance of correctness.
- *Dataflow analysis* – allowing reasoning about the ranges of variables at certain program points, tracks how expressions are calculated, etc. It is fast even for complex programs. However it is also inaccurate: it may provide false positives because it does not track individual execution paths, collapsing the instead (e.g. the two branches of an if instruction are assumed to be both executed, and after the if instruction finishes, the results are merged resulting in loss of precision).
- *Pre- and post- conditions* – extends the type of checking by specifying properties that hold on input and are expected to hold on output. Theorem-providers can be used to prove that a given implementation satisfies the post-conditions when the pre-conditions hold.
- *Symbolic execution* – is a very powerful way to test programs and discover bugs such as memory errors; however it does not scale to arbitrary programs when used on C code. Allows tracking the output of a functional element as an exact function of the input.
- *Model checking* – can check arbitrary properties but it is intractable in most practical scenarios.

- *Full modelling* – allows to formally prove the correctness of an implementation by starting from a mathematical model and successively refining it until it is executable. It is not only computationally, but also requires massive modelling effort and is out of reach for most programmers.

There is also a connection between the language used to model and the tractability of the verification tools. For instance, symbolic execution scales poorly on C code but can be made to scale well on domain-specific languages. Type checking and data-flow analysis work well on any language.

A complete analysis of existing verification techniques and their applicability to the 5G context is outside the scope of this deliverable. In the remainder of this section, we discuss about using type checking and symbolic execution in the 5G context.

Type checking

A system of types is a set of rules that are applied to assign a property called type to programming language constructs including variables, expressions and functions. Type inference is the process of deriving the type of a construct based on the types of its components (e.g. the type of an expression is derived by using the semantics of the operator and the types of its operands). Type checking is a static analysis technique that aims to reduce bugs by ensuring that different functions are called in a consistent way (e.g. with the appropriate types at input). Type checking is typically quite fast, but its accuracy depends on the type of programming language used. Weakly-typed languages such as C have some well-defined types (e.g. int, char) but also allow generic types such as (void*). Explicit casts must be used with these types to exact types; in this context type checking should test whether the cast applied is correct (i.e. the underlying type is compatible with the cast). Strongly typed languages ensure type safety by design; in Java, for instance, there is not equivalent of (void*). Finally, dynamic typed languages such as PHP or Python infer the type of a variable from its context; while easy to program for, such an approach leads too hard to detect and debug issues.

In the context of network programming, all variables are packets (or packet headers) and a type system must classify allowed packet formats: for instance, an IP header has a certain layout for its fields, and each of these has a predefined set of possible values (e.g. the protocol number, the TTL, the header checksum, the flags, and so forth).

To implement type checking for 5G networks, we will need for each functional element a description of the packet types expected at its input and the packet types produced at output. Then, given a configuration of such elements, we can statically decide (offline) whether this configuration achieves type safety or not. Furthermore, we can extend type-safety in the dataplane and ensure at runtime that all packets received have indeed one of the allowed types. Defining a type system for packets and implementing a checker for it will be the focus of our future work.

Symbolic execution

If we view packets as variables being passed between different network boxes, understanding a configuration of functional network elements becomes akin to software testing. This is a problem that has been studied for decades, and the leading approach is to use *symbolic execution* [Klee].

Symbolic execution is powerful: it explores all possible paths through the program, providing possible values for each (symbolic) variable at every point. The power of symbolic execution lies in its ability to relate the outgoing packets to the incoming ones: even if all the incoming packet headers are unknown, a symbolic execution engine can detect which parts of the packet are invariant through the network, and can tell *how* the modified headers depend on the input when they are changed. Unfortunately, symbolic execution scales poorly: its complexity is roughly exponential in the number of branching instructions (e.g. "if" conditionals) in the analysed program.

To use symbolic execution, each functional element must be described using a model that captures its high-level behaviour. It is natural to program the models also in C, as previous works do [Dobrescu-NSDI2013, Sekar-ArmstrongTechReport2015], however C is not a good language for symbolic execution as symbolic execution complexity quickly explodes even for modestly-sized programs.

Instead, we have created a language designed specifically for symbolic-network execution that is called SEFL (Symbolic-Execution Friendly Language) and has the following differentiating features:

- A path in the network symbolic execution must be tied to an active packet passing through the network: if a codepath does not result in packets, it should not be symbolically executed.
- Models of network boxes should only focus on the paths that decide the fate of packets, leaving out any logging, reporting, system checks and so forth. Note that the C language does not have this property: a packet is just one of many variables handled by the program, and dropping a packet does not stop the execution of the box.
- Symbolic execution friendly data-structures: SEFL offers a map functionality that is implemented in the symbolic execution engine, avoiding another fundamental problem of C: the poor handling of data structures for symbolic execution. For instance, we have found that symbolic execution complexity explodes even for simple networking code such as parsing TCP options (it takes 1 hour to symbolically execute options parsing when there are 6B of options in total; increasing to 7B takes more than 4 hours, and so forth).

A full description of the SEFL language is outside the scope of this work. However, we briefly highlight the main instructions offered by SELF:

Allocate(v[,s,m]) Deallocate(v[,s]) Assign(v,e)	Allocates new stack for variable v, of size s. If v is a string, the allocation is handled as metadata and the optional m parameter controls its visibility: it can be global (default) or local to the current module. If v is an integer it is allocated in the packet header at the given address; size is mandatory.

	Destroys the topmost stack of variable v; if provided, the size s is checked against the allocated size of v. The execution path fails when the sizes differ or there is no stack allocated for variable v. Symbolically evaluates expression e and assigns the result to variable v. All constraints applying to variable v in the current execution path are cleared.
CreateTag(t,e) DestroyTag(t)	Creates tag t and sets its value e, where e must evaluate to a concrete integer value. Destroys tag t.
Constrain(v,cond) Fail(msg)	Ensures that variable v always satisfies expression cond. The execution path fails if it doesn't. Stops the current path and prints message msg to the console.

If (cond,instr1,instr2)
 	
For (v in regex,instr)
 	
Forward(i)
	Two execution paths are created; the first one executes instr1 as long as cond holds. The second path executes instr2 as long as the negation of cond holds.
	Iterates through all variables that match the name given by regex, executing instruction instr.
	Forwards this packet on exit port i of the functional element.

InstructionBlock(i1,...) NoOp	Groups a number of instructions that are executed in order. Does nothing.

The tags are used to allow indexed addressing for header fields, for instance to access the IP source address field as the beginning of the L3 header + 96 bits.

Symnet

In the previous Trilogy2 FP7 project, partner UPB has built a symbolic execution tool for networks called Symnet. Symnet takes as input SEFL code and performs symbolic execution over it, tracking the propagation of symbolic packets through the network. Symnet can be used to check a range of network properties including:

- Reachability. It is straightforward to check reachability in a network modelled with SEFL. A symbolic packet is injected at the desired source node, and this packet is then propagated through the network by Symnet. At each node reached by the symbolic packet, we can inspect the values of and constraints on the header variables to discover which packets are allowed, what input packets can reach the output, and how the packets look like at the output, on *all the execution paths that make up that node*.
- Loop detection. The loop detection algorithm relies on the reachability algorithm with a twist: when a new node is visited, we save the current execution state. When the same node is

revisited, the current state is compared to all previous states. A loop is detected when the current state is included in a previous state. State A is included in state B if every symbol in state A either a) has the same concrete value as the same symbol in B or b) the range of possible values in state A is a subset of those in state B.

- The algorithm is generic and can capture different kinds of loops. If we apply it to the entire state (including header fields and metadata), the algorithm will not capture traditional forwarding loops because the TTL field will always decrease and thus the state will be different. To capture such loops, we apply the same algorithm but only con- sider destination and source IP addresses in the header.

- Invariants. By checking the value stack of the destination address field, we find that it is bound to the same symbolic value that was set by the client. If the destination address were a constant value, invariance holds only if the variable was bound to the same constant at the origin.
- Header memory safety. When creating or destroying header fields, accesses are indexed through tags. If the tags are set incorrectly, of if the program wrongly assumes the location of headers, the execution path will automatically fail. This allows us to catch various tunnel configuration problems or buggy network models.

Applying SEFL and Symnet to the Superfluidity architecture

To showcase the flexibility of SEFL and the power of symbolic execution we have modelled most of the elements of the Click modular router suite with it.

Click is a programming model for in-network functionality that offers the *element* as the basic building block. Elements perform individual functionality such as decrease TTL, Ethernet encapsulation, filtering, NAT, etc. They are implemented in C++ and compiled as binaries in the Click modular router image; advanced users can implement their own elements and compile them in if the existing elements are not sufficient.

A Click program is a directed graph of elements called a configuration. It typically runs on a single machine, takes packets from network interface, classifies them based on type (e.g. ARP/IP) processes them accordingly and then outputs them on some other interface.

Click elements can be viewed in the context of the Superfluidity architecture as functional elements. They can be implemented by different vendors in software (e.g. C++ running on x86 CPUs as today or C++ running on GPUs) or even run them in dedicated ASICs.

To allow users to understand how their Click configurations work, we use SEFL to manually model each Click element in isolation, and use Symnet as a verification tool to check the correctness of complex configurations.

We have modelled a large subset of the Click modular router elements and wrote a parser that takes a Click modular router configuration and automatically outputs its associated SEFL code. We first note that manually writing models requires expert time and is not easy. We plan to help this process in

the near future in two distinct ways. First, to check that the model is an accurate description of the element implementation we are working on developing automated testing tools that run the model and the code side by side injecting tests in both and comparing outputs. Secondly, we are exploring possibilities to automatically derive the model from the code in certain simple scenarios. Both of these, are outside the scope of this document, and will be described at length in future deliverables.

Finally, we would like to mention that the resulting models can be verified very quickly. We have modelled in Click a CISCO firewall appliance (Application Security Appliance, or ASA) and ran Symnet over the resulting configuration that comprised close to 100 Click elements in less than one second.

5 Analysis of some functional components (ALUIL contribution)

In this section, we present some general functional blocks that may fit for different use cases in both the wired and wireless domains.

5.1 Load-balancer as a functional block

This is a basic example for a functional block – it gets a stream of packets and needs to forward them (in a load balanced way) to a set of servers (services) that do the work on the stream of packets. While this is a very simple example, there are still interesting issues related to this example, which requires a further study.

- The loadbalancer could be realised as a single (physical) component that can be elastic (grow as needed) or could be a single component (we are referring to a loadbalacer as a functional block) that cannot scale.
- The load-balancer functional block interfaces should be defined. Specifically, we should further investigate how to define the incoming stream as well as how to define the number of server and their locations.
- Does it have a state (for example sticky load balancing all packets from the same flow go to the same server)? if so, what is a state and how this state is shared if at all?

5.2 Analytics as a functional block

The analytics functional block is a more complex block, where the input here can be a stream of data items and the output can be an alert when an abnormal characteristic in the data is detected. This can be as simple as a threshold based alert notification, or could utilise a very complex machine-learning algorithm that takes history into account. Again, in this example there are many interesting points to be investigated further:

- How to define the structure of the input (syntax)
- How to define the meaning of the input / output (semantics) – moreover do we need to define this?
- How to describe the algorithms
- Do we specify KPI and resource assignment for this
- How is it implemented

5.3 State repository as a functional block

An important aspect of moving to the cloud is the ability to scale in a timely manner. To that end, it is clear that VNFs should be redesigned and decomposed in (i) stateless workers, and (ii) state repository distributed across the infrastructure. Such an architecture allows fast instantiation of new workers and synchronisation of their role with the state that is available and synchronised to multiple locations.

In the following, we focus on the cloud RAN use case as an example for the usage of the State repository functional block.

The state for the RAN MAC is composed of a control state keeping the channel information for the users and the priorities of the packets (to meet with the required QoS), as well a data state which is composed of the actual pending packets for transmission. Then the MAC scheduler works on the control data to derive the transmission order in the next frame. The scheduler output is fed to the frame builder that takes the actual packet and constructs the frame or the streams to be transmitted from each antenna.

Accordingly, for the wireless MAC scheduler we could identify two state types, namely, control state, and data state.

- Control state
  - Channel states for the different users to the different antennas/cells.
  - Packet priorities (e.g., deadline, MIR and CIR, PF metric, etc.)
- Data state
  - The data queues for each user (organised according to the traffic QoS)

Now, instead of handling this state information by the VNF (i.e., the wireless MAC function), it would be more efficient to utilise a more generic state repository functional block.

Here, the state functional block has various requirements such as:

- Storage volume: the control state consumes small volume, but the data packets may require high volumes.
- Storage traffic: Both the control and data states updates frequently in about every 1msec (read and write)
- Distribution: MAC process may shift between access nodes (e.g., due to handovers) or between the access and the aggregation node (when transmitting data in a multi-cell MIMO techniques). Accordingly, the state should be synchronised and available at the surrounding access node and the nearest aggregation node.

5.4 State machine as a functional block

A state machine has been identified as an important tool for implementation of various functions. For example, in the Flavia EU project [17] it has been shown that the WLAN MAC can be realised as a state machine. State machines can also facilitate the control of network elements. Finally, state machine can serve as an important tool for the life cycle management of VNFs.

Accordingly, it is envisioned that a generic state machine functional block would facilitate many functionalities. However, it is a significant challenge to design such a functional block in a manner that addresses complex tasks on the one hand, and on the other is able to perform in real time scenarios (e.g., at time constraints of a packet transmission periods).

6 Reusable component analysis based on use cases

6.1 Use Case: cloud RAN

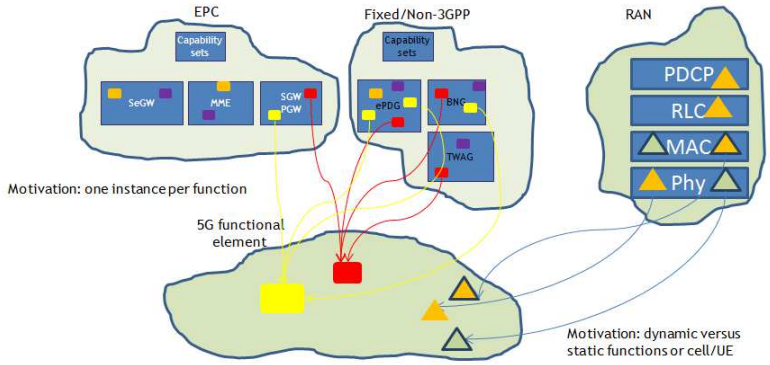
Use case	*cloud RAN*
Motivation for functional split	Today, the network is a network of element: PgW, EnodeB,… hosting a set of functions. However, using virtualization technology to address 5G requirements, it makes sense to re-examine this functional split across elements. There are several cases where functions are implemented in multiple elements or where functions are similar (especially when considering 3GPP and Fixed). • Authentication/Authorisation in MME, SeGW and capability sets • Security functions in SeGW and MME • Selection / Load-balancing in MME and ePDG • Routing / bearer management function in SGW, PGW and ePDG Functional split ➔ instantiate one example from each functions ➔ reduce and transferring of costs capex/opex. Giving the possibility to provision infrastructure in fine grained way. Besides, the flexibility requirements. Enable scalability (especially horizontal scalability in CRAN)
Methodology	1. Identification of the RAN functions: – LTE standard – From other project or 5G initiative (like ijoin, Crowd,…) 2. Categorization of each identified RAN functions into: – u-plane and c-plane, – user- or cell-related, ⇨ Two fine Granularity 3. Grouping of functions into function blocks, 4. identify dependencies between blocks 5. Identify the RAN functions **timing requirements** with respect to virtualisation and the functions inter-dependency.

<table>
<tr><td></td><td>➔ At the end of this step, we are able to specify the cloud area or survival environment of each function groups, example: FFT could not be executed in network cloud where no acceleration is available

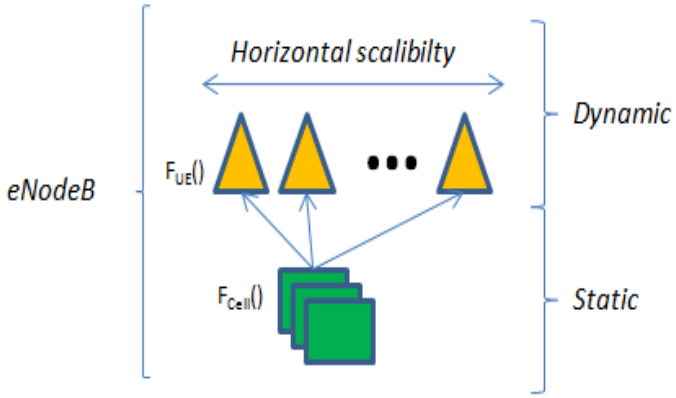
6. Study the impact of integrating AS/NAS functions and their co-location</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Functional bocks</td><td>List all relevant function blocks that will be required to operate a 5G mobile network. A function block should contain functionality from one functional layer (PHY, MAC, … NAS), it should well characterised by a set of features which distinguishes it from other function blocks.

<table>
<tr><th>Block</th><th>Functions</th><th>u/c plane</th><th>Granularity</th></tr>
<tr><td>PHY (RRH)</td><td>A/D; Signal generation; (De)modulation; MIMO precoding/equal.</td><td></td><td>cell</td></tr>
<tr><td>PHY Cell (EDGE)</td><td>Resource mapping; MIMO mapping</td><td></td><td>Cell</td></tr>
<tr><td>PHY UE (EDGE)</td><td>FEC</td><td></td><td>UE</td></tr>
<tr><td>MAC Scheduling, Cell (EDGE)</td><td>ICIC/CoMP; Intra-site; Inter-site; Priority handling; Channel mapping; Scheduling; BS power control; “initial scheduling”; “D2D support”, HARQ, AMC, DRX</td><td>U/C</td><td>Cell</td></tr>
<tr><td>MAC UE (EDGE)</td><td>UE Power control; Padding; Multiplexing of TBs;</td><td>U/C</td><td>UE</td></tr>
<tr><td>RLC TM (EDGE)</td><td>Buffering/transferring of PDCP PDUs</td><td></td><td>Bearer</td></tr>
<tr><td>RLC UM-AM (EDGE)</td><td>Concatenation/Segm; Reordering; Duplicate detection PDU; Reassembly SDU;</td><td></td><td>Bearer</td></tr>
<tr><td>RLC AM (EDGE)</td><td>Error correction ARQ; Re-segmentation</td><td></td><td>Bearer</td></tr>
</table>
</td></tr>
</table>

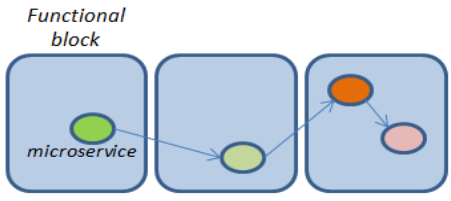
	PDCP U (EDGE)	ROHC, ordered delivery; duplicate detection;	U	Bearer
	PDCP UC (EDGE)	Data transfer; Seq number maintenance; Integrity protection; (De)ciphering; SL-(De)ciphering; discard timer	U/C	Bearer
	RRC Cell (EDGE)	Broadcast SI	C	Cell
	RRC User (EDGE)	RRC connection mgmt.; Paging; RB mgmt.; Connection mobility; QoS management functions; Security, Measurement configuration and reporting	C	User
	NAS UE-U (EDGE)	Mobility anchor mgmt.; packet routing; packet marking; deep packet inspection; DHCP; data forwarding in RAN; lawful interception; UL/DL charging	U	UE or bearer
	NAS UE-C (EDGE)	Inter-node signalling for mobility; NAS signalling to UE; NAS security; NW attachment; Authentication; TA mgmt.; Charging; Paging; RAN info mgmt.; context transfer HO; GTP mgmt.; Bearer mgmt.; packet forwarding	C	UE or bearer
	NAS Event-C (EDGE)	PGW/SGW selection; MME selection for handover; UE reachability procedure; location reporting; S1 UE context mgmt.; ERAB mgmt.;	C	Event/ UE
	NAS CN (CORE)	GTP-C load control; MME load balancing; MME overload control; PDN GW overload control	C	CN/UE
Functional split	1. Function classification & re-aggregation 2. Split between static vs. dynamic functions			



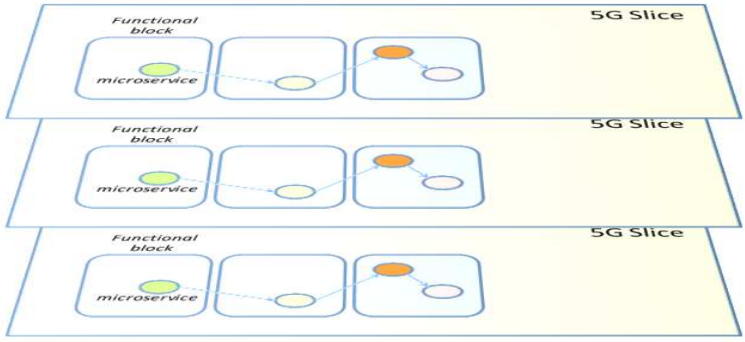
3. Split between static vs. dynamic functions to enable horizontal scalability & elasticity.



4. Model in functional blocks. Functional blocks are chained and composed with microservices



5. Functional block split to support 5G slicing. The Network Slices is a way to split the RAN into different 5G functions (multimedia, IOT, M2M, …)

	
Missing	1. SDN & control part: how the SDN ctrl will interact with the orchestration & functional blocks

6.2 Use Case: On–the-fly Monitoring

Use case	*On –the-fly Monitoring*	
Description in a nutshell	Continuous increase in bandwidth demands means network-wide deployment of DPI will become increasingly expensive and in most cases unsustainable. There is a requirement to enable relatively cheaper monitoring infrastructures. Implementation of DPI like systems with VNF(s) will allow two scenarios (a) dynamic deployment of DPI to monitor selected network segments e.g. specific geographical regions, and (b) implementing multiple DPI deployments for disparate virtual customers sharing the same physical infrastructure	
Functional Blocks	Packet Processor	Service that looks into packet headers to get information on addresses (source and destination), identify protocols, and understands fingerprints of applications
	Load Balancer	Service for distributing packet flows over multiple instances the DPI service chain
	Encryption/Decryption Module	Service to enable analysis of secured packet flows

	Encryption Key Manager	Service that allows encryption keys to be maintained outside of encryption/decryption module. Useful when same key is applied to multiple use cases e.g. same private key for encrypting packet flows & email
	Management Module	Service that manages creation, deletion, and upgrade of patterns and fingerprints of applications
	Policy Control/Enforcer	Service that allows DPI service chain to control access to network
	Switching Logic	Service to switch packet flows in and out of DPI service chain
	Database	Service to store and maintain pattern/fingerprint definitions used for monitoring different applications

6.3 Use Case: S/Gi-LAN Services on the (Mobile) Edge

Use case	*S/Gi-LAN Services on the (Mobile) Edge*	
Description in a nutshell	*In today's mobile networks, services involving traffic management/DPI and transport/content optimization have been traditionally deployed on the Internet side of the GGSN/P-GW, i.e. in the S/Gi-LAN.* *Even though the industry recognizes the utility of these services, always in the context of the Mobile Data Tsunami and the desire of operators to differentiate from their competitors on the basis of QoE, deploying such solutions in a scalable fashion is becoming increasingly challenging/costly.* *Moreover, the lack of accurate visibility on RAN conditions makes it very difficult to deliver traffic management and transport/content optimization in a way that achieves balance between network efficiency and QoE.* *The flattening and "IP-fication" of the network, in the evolution from UMTS to LTE and, eventually, to 5G, provides opportunities of pushing these services into the RAN and towards the Edge of the network.*	
Phases	Instantiate Service	Traffic management and optimization services will be instantiated in the Mobile Edge micro-data centres (DCs) specified by the operator, or determined to require them

	Access User Plane	The traffic handling services will utilise the Mobile Edge Computing (MEC) infrastructure to acquire access to user plane traffic.
	Create Data Session	Metadata about user plane traffic (Packet Data Protocol sessions) will be held in an in-memory Session Database. Information will include subscriber and device identifiers, respective bearers, associated IPv4/IPv6 addresses, etc.
	Enrich with Control Plane Information	The entries maintained in the Session Database will be enriched with control plane information, leveraging the corresponding MEC APIs. Such information will involve radio-access type per bearer, radio resource allocation, cell load, throughput guidance, link quality, etc. Please note that this info may have to be updated regularly (probably asynchronously) during data session lifetime.
	Track Data Flows	The traffic handling services will identify (L4) data flows. Initially the focus will be on supporting the tracking of flows that use the TCP and UDP transport protocols (over both IPv4 and IPv6). The services will need to maintain state (in-memory) for each of the identified data flows, at least for ones selected to receive traffic optimization.
	Inspect Content	The content (i.e. payload of TCP or UDP packets) of the identified data flows will be inspected in DPI-like fashion to determine the content type. Note that this may have to be stateful, i.e. associating packets to logical streams. The analysis applied may have to be more elaborate than simple signature/pattern matching, e.g. to identify video transferred using an encrypted transport (HTTPS/QUIC).
	Identify Application	To determine e.g. whether it is appropriate to apply traffic regulation to ABR video, the services will have to infer that the consuming application is a media player. This occasionally requires logic across data streams of the same subscriber (e.g. when encryption is in effect).
	Evaluate Policies	The services will combine information maintained in the session database with the state kept for each data flow and with the configuration set by the operator to decide which traffic optimization actions must be applied.

	Apply Transport-level Optimization	The services will apply transport (L4) optimization, initially focusing on TCP traffic, to increase the speed, decrease the latency and maximize the efficiency of delivering content to the UE, moreover in a way that takes into account RAN congestion.
	Apply Application-level Optimization	On top of transport optimisation, the services may also apply application-level traffic management schemes. One example is ABR video aware traffic regulation (pacing), which allows the operator to limit the ABR video quality that should be delivered by the mobile network.
	Calculate Metrics, Emit Flow Records	The transport and application layer metrics gathered as part of delivering the above services will be logged and optionally emitted (as IP flow records) to analytics infra.
	State Migration	If the UE moves to a radio service area that is handled by a different MEC DC, the session & data flow state that is maintained on behalf of the UE will need to be migrated, to the extent that the transition is seamless to the user.
	Tear-Down Service	If the traffic optimization services are not required any more in the particular MEC DC, they shall be torn town.
Functional Blocks	Mobile Edge Computing (MEC)	Please refer to 6.11 and 6.12 for more details. Specifically we identify overlaps with the following: • Cloud Infrastructure (NFVI) • Cloud Management (VIM) • MEC Platform (Auth) • MEC Platform (Bus) • MEC Platform Management • MEC Enabler (TRF) • MEC Enabler (TRF API) • MEC Enabler (RNIS) • MEC Enabler (RNIS API) • MEC Orchestrator • MEC Orchestrator (Policy)
	Load Balancer	If traffic needs to be balanced between multiple service instances within the same MEC DC, load balancer must support stickiness on a per sub/UE or data session basis.
	Service Chaining	Since these services will be probably deployed together with other traffic inspection or traffic processing services,

		the Mobile Edge infrastructure needs to provide a traffic steering and/or service chaining function.
	User-Data Plane Interface	Component integrated with the forwarding path, which intends to allow the MEC platform to access the user-data plane traffic (cross-reference to 6.11 and 6.12).
	Session Database	Stored information will include subscriber (e.g. MSISDN) and device (e.g. IMSI) identifiers, associated bearers, allocated IPv4/IPv6 addresses, etc. Extra information will include radio-access type (per bearer), radio resource allocation, cell load, throughput guidance, link quality, … In case of multiple service instances within the same DC, Session Database will have to be shared between them.
	TCP/UDP over IPv4/IPv6	The traffic processing services must be able to process IPv4/IPv6 (L3) headers and TCP/UDP (L4) headers. This can be part of the kernel, but for reasons of throughput and flexibility, a user-plane protocol stack is preferable.
	Ethernet	The underlying (L2) network technology should not be relevant, but Ethernet should be the usual case there.
	Performance Enhancing Proxy	Depending on the transport optimization techniques, the network services will have to modify the timing of packets (traffic regulation/pacing), generate ACKs and handle packet retransmissions on behalf of the content server, or even completely replace the congestion control module (the split-TCP middle-box scenario).
	Application Acceleration	Due to the network I/O heavy workload, the services can benefit from VNF application acceleration technologies, such as SR-IOV, Netmap/Vale or DPDK.
	Connection Tracker	To be able to implement analysis/logic on a per logical flow/stream basis, the network services will include a connection/flow tracker. Depending on stateful-ness of the transport optimization technique, associated state may have to be synchronised across instances, or else service migrations may result into connection resets, etc. For similar reasons, service introduction shall only handle new connections, whereas service tear-down must (reasonably) wait for tracked connections to “starve”.

	Protocol Signatures	Protocol and content detection will depend on DPI-like signatures.
	Content Detection Algorithms	Traffic must be inspected (on the basis of data flows) to identify the type of content being transferred. Focus would be on identifying content that can be optimized (e.g. ABR video, even if it is encrypted).
	Application Detection Algorithms	Traffic must be inspected (on the basis of data flows) to identify the type of application that is consuming the content. Focus would be on identifying applications that can be optimised (e.g. ABR video players).
	Policy Engine	Combining the data-plane and control-plane inputs with the configuration/preferences of the operator to make decisions and apply actions requires a Policy Engine.
	Transport Optimizer	Transport-layer (L4) optimisation, initially focusing on TCP (see also Performance Enhancing Proxy). Objective is to achieve high network speeds, efficient use of network resources, congestion control and low latency.
	TCP Congestion Control Modules	For the case of split-TCP middle-boxes, the congestion control algorithm can be replaced. Congestion handlers are usually delivered in the form of reusable modules.
	Video Optimizer	Regulation/pacing of video flows is a traffic management action that helps conserve network resources. This, for example, allows the operator to specify the ABR video quality that should be delivered by the mobile network. It may require cooperation with the Transport Optimizer.
	Metrics Engine	For monitoring service performance, detailed KPI metrics will be generated, both on a per data flow basis, but also time-windowed aggregates. The former can be emitted to a big data analytics platform (see Flow Record Output below). The latter can be read, in the form of counters, via relevant management plane protocols (e.g. SNMP).
	Instrumentation Gathering	Generation and exporting of instrumentation data is almost always required for traffic handling services, since it enables troubleshooting and root-cause analysis of any issues. This includes the capturing of packet traces, etc.

	Flow Record Output	Detailed per flow metrics can be emitted to network analytics platform in the form of a flow record output.
	State Sync Interface	Interface for synchronizing state between service instances in and across MEC DCs (for service migration).
	Service Manager	Frontend system (CLI/GUI) for managing, configuring, monitoring and maintaining the optimization services
	Service Dashboard	Frontend system (web GUI) for evaluating performance and assuring the benefits of the optimization services
	Big Data Analytics	Composition of functional blocks that receives metrics and counters from the traffic handling services (among other inputs) to implement a Network Analytics solution. Based on the historical metrics, and using trending, forecasting and machine learning techniques, insights extracted can be fed back to the traffic handling services, in terms of policies, towards optimizing them further.
Categorisation	SDX Central	Virtual Network Functions and Applications • DPI • Other Virtual Appliance or Application SDN Software • Network Analytics and Visibility

6.4 Use Case: Dynamic MAC services allocation in Cloud RAN

Use case	*Dynamic MAC services allocation in Cloud RAN*	
Description in a nutshell	*Cloud-RAN aims at moving functionalities from the eNodeB to the cloud by splitting the base station functions to RF implemented in the RRH (Remote Radio Head) and to PHY/MAC virtualized and implemented at the cloud BBU (base band unit).* *The BBU is composed of few major base station functionalities such as, packet processing and fragmentation, MAC scheduler, HARQ, link adaptation and power control, MIMO and Beam Forming, ICIC and SON.*	
Functional Blocks	*Packet processing*	*Assembly and disassembly of packets including fragmentation and defragmentation according to the available transmit resources. In addition the packet processing would also handle the coding and decoding of the packets CRC.*

	Scheduler	*Service for scheduling data transmission to a single or multiple resource blocks in a given sub-frame.* *The scheduler is a statefull function that maintains the queue size of each connection with the arrival time of each packet and the QoS provided for each connection.*
	Link adaptation	*The link adaptation function defines the transmission rate and the transmission power over each resource block. The link adaptation could be realized as a state machine, where the states encapsulates the channel conditions.*
	Handover	*Handover decision between virtual base station and physical base station. Handover decision may impact the number of virtual base stations available and thus the creation and deconstruction of VNFs implementing the virtual base stations.* *Handover decision could also be implemented as a state machine.*
	SON + ICIC	*Coordinates the frequency band and power allocation to the different physical and virtual base stations. The SON aims at optimizing the overall performance of the wireless network via interference mitigation.*
	Virtual MIMO	*Coordination between the transmission of multiple RRH to form MIMO streams originating from a virtual base station.*

6.5 Use Case: Internet of Things (IoT) & SUPERFUIDITY Platform Scenario

Use case	Internet of Things (IoT) & SUPERFUIDITY Platform Scenario	
Description in a nutshell	Superfluidity and interoperability in the IoT: the ability to instantiate IoT services on-the-fly (pub/sub brokers, gateway between different IoT networks), run them anywhere in the network edge (micro server) and shift them transparently to different locations, integrating vertical and/or proprietary platforms and devices. The ability to create on the fly a network (ad-hoc or/and infrastructured) composed by smart objects. The ability to use local communication (LTE Direct, BLE) instead of remote when needed: e.g. not all nodes can reach the internet, e.g. some nodes have to save energy.	
Phases	**Identification of local IoT devices to be connected**:	

	Identification of needed gateways, pub/sub brokers:	
	Realization of interconnection topology	A mix of ad/hoc and infrastructured network.
	Run time phase of the service	Edge processing for data aggregation, protocol gateways
	Reaction to dynamic changes	Adaptation to changes in the number of devices
Functional Blocks	Local discovery mechanism	Combining different radio access technologies, a device could scan for the presence of other devices of access points, an access point could scan for the presence of other devices.
	Gateway discovery mechanism	Needed for the IoT devices to connect to a gateway
	Publish/subscribe	Publish/subscribe primitives for IoT devices as micro services

6.6 Use Case: Context-adapted data delivery

Use case	Context-adapted data delivery	
Description in a nutshell	Ability to adapt the delivery of content based on context information, including location but not restricting to it and including also an estimate of the user behaviour (e.g. user on the move, potential user interests, etc.)	
Phases	**Context information regarding a user is collected**	Collected information may concern (the list is not exhaustive): position, speed/direction, activities (e.g. active applications)
	Context information is transferred toward a "context processing/storage" entity	
	Content to be delivered is adapted to the context	
	Content is delivered	While content is delivered, the same process with the three steps above can be executed in parallel,

		so context can change during the active phase of the content delivery

Context collection phase

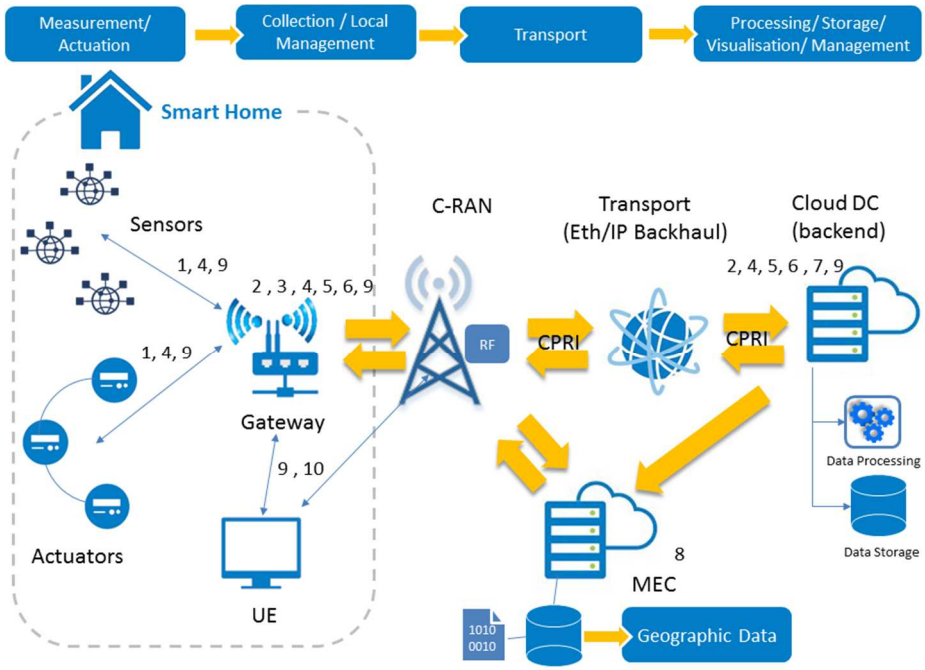
Functional Blocks	The context information can be collected by the terminal and by the access infrastructure. (May be most of the context information can be collected by the terminal.) For the information that will be collected by the terminal, the network should offer some way for the terminal to publish it (periodically or when the terminal notices some change in the context). The decomposition into (micro-)services of the functionality of context collection, transfer of context information towards a context processing/storage entity, and context processing is not trivial. A set of generic operations to collect user context should be defined. A context processing/storage entity should be accessible both in the cloud ran, in the MEC (Mobile Edge Cloud) and in the core, so that other entities interested in knowing the context of a user/terminal can get the information.	

Content delivery phase

Functional Blocks	Video (media in general) transcoding	
	Video (media) caching in the local RAN	
	Media mixing	
	Media replication	

6.7 Use Case: Smart Home – derived from Intel Use Case 3 – Context Aware Smart Living

Use case	*Smart Home – derived from Use Case Context Aware Smart Living*

Description in a nutshell	A key focus for the proliferation of IOT devices is realisation of a connected world where devices and sensors are connected in a seamless manner to support humans in the daily activities of living. The combination of sensor, mobile devices, ubiquitous high-speed wireless connectivity and cloud infrastructures will the support the realisation of a smart society where intelligence is embedded into all aspects of daily life such as smart transport, smart health and wellness etc. A specific application of the Smart Living vision is the realisation of Smart Homes. Sensors, actuators, data collection and processing are used in a seamless manner to support humans in the activities of daily living and to provide management and control of their home environments. Scenario's which can be supported include heating and lighting control, security, environmental control, assisted living for older adults or special needs children/adults etc.
Additional information (e.g. figures)	 High level service architecture for Smart Home use case
Key Service Elements	1. Identification and registration of IOT devices (sensors and actuator) and user equipment (smartphones and smart TV's). 2. Identification and registration of home based IOT Gateway with Smart Home cloud based backend services. 3. IOT Gateway data processing service – provides local data processing and local storage of time window defined data e.g. 1

	week. Sends processed to data to the MEC and cloud backend to big data services.
	4. Provision of secure connections between the sensors/actuators and the IOT Gateway and between the IOT Gateway and the MEC / cloud backend end.
	5. Device Synchronisation service – maintains the correct version of firmware on the gateway and sensors/actuators
	6. Policy Manager – Distributed service which run on the IOT Gateway and Cloud backed end. It responsible for managing the behaviour of the devices deployed in a home such as publication and subscription, metadata lifecycle management, Connectivity and Interoperability etc.
	7. Backend-end data processing and storage.
	8. MEC based service for geographical bounded data collection, processing and visualisation on UE's'. The MEC application can also receive additional summary data and information from the cloud backend which is contextually relevant.
	9. Security – Authentication and provision of secure connections between the sensors and IOT Gateway and between the gateways and the MEC/cloud backend.
	10. UE Service – manages access to local and cloud based data stores and visualisation of data

6.8 Use Case: On-the-fly Ad Removal Offloading

Use case	*On-the-fly Ad Removal Offloading*	
Description in a nutshell	Given the proliferation of online advertisements and the fact that a large number of players based their revenue model on them, it is perhaps unsurprising that ad blockers have become commonplace. While certain useful, they do consume CPU cycles as they scan incoming traffic which results in reduced battery life when talking about mobile devices. In this use case we propose to provide an on-the-fly, virtualized ad blocker service that can be run in edge networks, essentially offloading this functionality from mobile devices in order to increase their battery life.	
Functional Blocks	Strip	Service to strip L2 headers from packets
	IP Checker	Service to filter for IP packets and to check that they are sane (e.g., that the checksum is correct)

	Classifier	Service used to filter for specific traffic flows (e.g., HTTP traffic in case our intent is to block ads in HTTP flows)
	TCP flow reconstructor	Service to reconstruct the TCP flow in case ads span multiple packets
	HTTP parser	Service to parse HTTP traffic
	Ad remover	Service to search for and remove ads from packets.

6.9 Use Case: Transparent web service acceleration

Use case	*SF-ONAPP-1 – Transparent web service acceleration*	
Description in a nutshell	*Re-writing of static content is performed to take advantage of CDN paths that is then synchronised with the Edge nodes.*	
Technical requirements	*CDN platform*	*Requires a system where there are edge nodes and a routing system*
	Hosting of web content – e.g. a web-server	Linux platform that can support a web-server e.g. apache or nginx
	Compression engine	Need a set of compression tools that can compress new content ('in-time').
	Routing system	DNS entries modifier
	HTML content modification	Parser of HTML hrefs and then modify to take advantage of CDN end-points
	Synch engine	Could be rsync or some other form of asynchronous replication
	Security system	Ensure that content is only accessible/modifiable by authorised users at all points in the data-flow. Cannot pre-encrypt the data as this will lose the tag content in the HTML. Encryption after compression is possible for e2e encryption through the channel

6.10 Rapid and massively-scalable instantiation of high performance (virtual) application instances

Use case	*Rapid and massively-scalable instantiation of high performance (virtual) application instances for high performance storage applications*	
Description in a nutshell	*Current VMs are quite large and heavy, requiring a full Linux O/S to be provisioned. We challenge this model to use much more light-weight instances that can be then used for a multitude of applications that are otherwise inaccessible*	
Functional blocks	Command listener	Listens to ATAoE commands
	Driver domain	Helper domain that contains drivers for particular hardware devices
	Router	Routes commands either to/from local VMs or adds them to the network queue for remote processing
	Scheduler	Divides physical resources into time-slices for different domains
	Logging system	Send/receive logs from all guest instances
	ACL system	Authenticate packets coming in and send to authorised users. Maintain privacy for packets not addressed to other users
	Queues/Buffers and handlers	Utilise local storage/memory space for capturing data/commands while the other data is being processed
	Packet reordering	Given TCP/IP connection orientated, sequentialising system is not included, need to re-order packets to generate the original stream
	VM controller	Given the Dom0 logic is moved to the MicroVisor need to distribute the control logic and make each domU responsible for some of the control state.

6.11 Use Case: Local Breakout (LBO)

Use case	***Local Breakout (LBO)***

Description in a nutshell	*Local Breakout intends to avoid user traffic to be sent to the mobile network core, when communication parties are on the same edge network (e.g. eNB). On 3GPP networks, by default, all traffic is terminated on the mobile core (PDP/PDN). However, in many cases, knowing that users are attached on the same edge, communications can be shortcutted, making the connectivity more efficient. A similar concept may also be applied to fixed networks.*	
Functional Blocks	*Cloud Infrastructure (NFVI)*	*Cloud Infrastructure devoted to support Operator and 3rd party Services (can be shared by the NFV infrastructure and the MEC framework itself)*
	Cloud Management (VIM)	*Component dedicated to manage the cloud infrastructure (VIM)*
	User-Data Plane Interface	*Component integrated with the forwarding path, which intends to allow the MEC platform to access the user-data plane traffic*
	MEC Platform (Auth)	*MEC Platform component devoted to Service authentication and authorization, namely regarding the access to APIs*
	MEC Platform (Bus)	*MEC Platform component dedicated to Service communication (Service<–>APIs, Service<->Service)*
	MEC Platform Management	*Component dedicated to manage the MEC Platform, namely, regarding the API exposure, interaction with authentication/authorization modules, etc.*
	MEC Enabler (TRF)	*Service Enabler plugged into the MEC platform, which provides user-data plane features*
	MEC Enabler (TRF API)	*Component that makes an API available for Services to access the user-data plane*
	MEC Enabler (RNIS)	*Service Enabler plugged into the MEC platform, which provides Radio Network Information Systems (RNIS) features*
	MEC Enabler (RNIS API)	*Component that makes an API available for Services to access RNIS features*
	MEC Enabler (Loc)	*Service Enabler plugged into the MEC platform, which provides UE Location features*

	MEC Enabler (Loc API)	*Component that makes an API available for Services to access UE Location features*
	MEC Service LBO	*Operator or 3rd party Service devoted to provide Local BreakOut (LBO) services using the MEC System*
	MEC Service LBO (Mgm)	*Component devoted to manage the Local BreakOut (LBO) Service (associated to the LBO Service)*
	MEC Orchestrator	*Component devoted to orchestrate Services lifecycle (deployment, scaling, migration, disposal, etc.)*
	MEC Orchestrator (Policy)	*Component that hold policies used by the MEC Orchestrator to orchestrate the Services lifecycle*

6.12 Use Case: virtual Convergent Services (vCS)

Use case	***virtual Convergent Services (vCS)***	
Description in a nutshell	*The vHGW use case intends to move traditional functions (e.g. firewall, parental control, NAT, etc.) residing on the customers' home to a virtual HGW (vHGW) in the cloud. In a convergent scenario, these services apply both to fixed and mobile environments, providing a convergent desired behaviour, either when the user is at home or using a mobile device.* *The vSTB component complements the use case, extending the usage of a virtual STB to a multi-screen scenario, simplifying the convergent environment, reducing operator investment and making easier the upgrade and deployment of new services, being accessible to the user from any terminal.*	
Functional Blocks	*Cloud Infrastructure (NFVI)*	*Cloud Infrastructure devoted to support Operator and 3rd party Services (can be shared by the NFV infrastructure and the MEC framework itself)*
	Cloud Management (VIM)	*Component devoted to manage the cloud infrastructure (VIM)*
	User-Data Plane Interface	*Component integrated with the forwarding path, which intends to allow the MEC platform to access the user-data plane traffic*
	MEC Platform (Auth)	*MEC Platform component devoted to Service authentication and authorization, namely regarding the access to APIs*

	MEC Platform (Bus)	*MEC Platform component devoted to Service communication (Service<–>APIs, Service<->Service)*
	MEC Platform Management	*Component devoted to manage the MEC Platform, namely regarding the API exposure, interaction with authentication/authorization modules, etc.*
	MEC Enabler (TRF)	*Service Enabler plugged into the MEC platform, which provides user-data plane features*
	MEC Enabler (TRF API)	*Component that makes APIs available to let Services access to user-data plane features*
	MEC Enabler (RNIS)	*Service Enabler plugged into the MEC platform, which provides Radio Network Information Systems (RNIS) features*
	MEC Enabler (RNIS API)	*Component that makes APIs available to let Services access to RNIS features*
	MEC Enabler (Loc)	*Service Enabler plugged into the MEC platform, which provides UE Location features*
	MEC Enabler (Loc API)	*Component that makes APIs available to let Services to access to UE Location features*
	MEC Service vCS	*Operator or 3rd party Service devoted to provide virtual Converged Services (vCS) using the MEC System*
	MEC Service vCS (Mgm)	*Component devoted to manage the virtual Convergent Service (vCS) (associated to the vCS Service)*
	MEC Orchestrator	*Component devoted to orchestrate Services lifecycle (deployment, scaling, migration, disposal, etc.)*
	MEC Orchestrator (Policy)	*Component that hold policies used by the MEC Orchestrator to orchestrate the Services lifecycle*

6.13 Use Case: Anti NDP Spoofing software implementation

Use case	Use Case - NDP Spoofing
Description in a nutshell	Implementation of a defense against NDP-Spoofing, for example the SEND protocol, by means of VNFs.

Functional Blocks	Idle	Discards packets
	Switch	Sends packet stream to settable output
	Hub	Duplicates packets like a hub
	Queue	Stores packets in a FIFO queue
	EtherSwitch	Learning forwarding Ethernet switch
	CheckIPHeader	Checks IP header
	IPFilter	Filters IP packets by contents
	SEND	SEND protocol implementation

6.14 Use Case: Protection against DDoS

Use case	Use Case – DDoS Attack Protection	
Description in a nutshell	Implementation of a defense against DDoS attacks, by means of VNFs.	
Functional Blocks	Idle	Discards packets
	Switch	Sends packet stream to settable output
	Counter	Measures packet count and rate
	Queue	Stores packets in a FIFO queue
	IPRouteTable	IP Routing Table superclass
	RadixIPLookup	IP lookup using a radix trie
	CheckIPHeader	Checks IP header
	IPFilter	Filters IP packets by contents
	IPRateMonitor	Measures incoming and outgoing IP traffic rates

6.15 Use Case: Late transmuxing

Use case	*Late transmuxing (LTM) on the (Mobile) Edge*	
Description in a nutshell	Instead of using the CDN network only as cache, the CDN (edge) nodes could be used to create requested formats when needed, saving bandwidth and storage within the network, increasing edge resource usage and improving user experience.	
(LTM) Phases	Request parsing	Webserver determines handler based on request type (via extension usually) and maps the request to the local or

		remote location the content (samples and server manifest) is located.
	Upstream sample fetch	(Libfmp4) When needed audio/video samples are not in the local cache from a similar previous request for another protocol, sample have to be fetched upstream.
	Manifest creation	(Libfmp4) Create the appropriate client manifest for the request (HLS, HDS, HSS or DASH).
	Chunk creation	(Libfmp4) Create the appropriate chunk for the request (HLS, HDS, HSS or DASH).
	Encryption/DRM	(Libfmp4) Encrypt chunk and/or signal DRM in the client manifest based on the server manifest settings.
	Output handover	Return created output to Webserver for delivery.
Functional Blocks	Mobile Edge Computing (MEC)	Please refer to PTIN-1 and PTIN-2 for more details. Specifically we identify overlaps with the following: • Cloud Infrastructure (NFVI) • Cloud Management (VIM) • MEC Platform (Auth) • MEC Platform (Bus) • MEC Platform Management • MEC Enabler (TRF) • MEC Enabler (TRF API) • MEC Orchestrator • MEC Orchestrator (Policy)
	Load Balancer	If traffic needs to be balanced between multiple service instances within the same MEC DC, load balancer must support stickiness on a per sub/UE or data session basis.
	WebServer	Linux platform that can support a web-server e.g. Apache or Nginx (or anything that can handle HTTP requests in general), the LTM instance.
	Libfmp4	The USTR muxing software used in the web-server for LTM.
	Storage	Upstream providing mezzanine audio/video samples, accessible over HTTP (accepting range requests, e.g. S3).
	CMS/Policy	Service for LTM instances to fetch configuration (server manifest) from, configuration can change dynamically. The server manifest controls the client/manifest

		generation and can be provided by the CMS on a per request basis, so rules may be applied (for instance in relation with the video optimizer).
	AutoScaling	The setup should be able to add and remove LTM instances based on load or other metric, automated.
	Cache priming	Latency between storage and LTM instance should be low enough so cache priming (setting up local caches with content to be ready for expected load) could be employed. Alternatively 'prefetch' (fetching next chunk before time by the webserver to have it cache already) may be employed as well.
	Video Optimizer	Regulation/pacing of video flows is a traffic management action that helps conserve network resources. This, for example, allows the operator to specify the ABR video quality that should be delivered by the mobile network. It may require cooperation with the Transport Optimizer.
	Metrics Engine	For monitoring service performance, detailed KPI metrics will be generated, both on a per data flow basis, but also time-windowed aggregates. The former can be emitted to a big data analytics platform (see Flow Record Output below). The latter can be read, in the form of counters, via relevant management plane protocols (e.g. SNMP).
	Monitoring	Monitor stream characteristics on LTM instance (bandwidth, sessions, streams).
Categorisation	SDX Central	Virtual Network Functions and Applications
Background	2015 Thesis	http://repository.unified-streaming.com/late-transmuxing-smart-edge.pdf

6.16 Use Case: Static analysis for a network infrastructure

Use case	*Static analysis for a network infrastructure*

<table>
<tr><td>Description in a nutshell</td><td colspan="2">SYMNET has been deployed in order to verify POLITEHNICA's Computer Science Department network infrastructure. The entire topology, including switches and routers, has been modelled in Click. The topology core consists in a Cisco Adaptive Security Appliance, henceforth called ASA.

We use the Click modular router elements as basic building blocks to decompose the Cisco ASA functionality. As Click focuses on the data plane and gives good performance when run in kernel mode or over netmap, this choice ensures both ease of decomposition and performance.

In order to model the ASA in Click, an ASA configuration parser and Click code generator was developed. The tool identifies: access control lists (ACLs) and interfaces where they are applied, routing rules, NAT rules, and security-levels (numerical values assigned to interfaces with the purpose of limiting traffic from interfaces with higher to lower security level). The tool relies on the components illustrated in the "pipeline elements" section below to generate the click configuration. The list of the most important Click elements used in the generation process is given in the "Functional blocks" section.</td></tr>
<tr><td rowspan="3">Pipeline elements</td><td>ACL generator</td><td>For each set of ACL rules and each interface on which they are applied, we generate a Click IPClassifier element.</td></tr>
<tr><td>Routing rule generator</td><td>For routing, we generate a set of IPClassifiers which implement ASA routing rules, as well as destination-based forwarding, based on the subnet assigned to each interface.</td></tr>
<tr><td>NAT rule generator</td><td>ASA allows NAT rules to be applied to arbitrary IP packets. We treat TCP packets and all others differently. For the former, we implement an IPRewriter which manages state. For the latter, we do prefix-based IP translation (also using IPRewriter), but we ignore state.</td></tr>
</table>

	Security level generator	We use a Paint element to assign, to each interface, the appropriate security level.
	Pipe-line assembler	For each ASA interface, we build a Click pipeline, which ends with the routing phase (i.e. the unique IPClassifier element which implements routing). The routing phase also includes security level filtering. The pipeline consists in: (i) checking if traffic is stateful (e.g. belonging to an existing TCP connection) (ii) applying filtering rules (ACL), (iii) checking for (and if applicable, applying) NAT rules for non-TCP packets, (iv) dynamic NAT (for TCP packets), (v) routing and security-level filtering.
Functional Blocks	IPRewriter	The Click element is used to keep dynamic NAT state, as well as state for all TCP connections. The element is also used to modify source/destination IP addresses for static NAT.
	IPClassifier	The Click element is used to: separate TCP connections from other traffic, implement ACLs, separate traffic subject to static NAT from other traffic, implement routing.
	Paint	The Click element is used to mark the security level of the ingress interface, on current traffic.
	PaintSwitch	The element is used to implement security-level filtering, in the routing phase.

7 References

[1] “Network Functions Virtualisation (NFV); Use Cases” ETSI GS NFV 001 V1.1.1 (2013-10)

[2] “Network Functions Virtualisation (NFV); Architectural Framework”, ETSI GS NFV 002 V1.1.1 (2013-10)

[3] “Network Functions Virtualisation (NFV); Virtual Network Functions Architecture”, ETSI GS NFV-SWA 001 V1.1.1 (2014-12)

[4] “Network Functions Virtualisation (NFV); Management and Orchestration”, ETSI GS NFV-MAN 001 V1.1.1 (2014-12)

[5] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, “Fog computing and its role in the internet of things”, 1st SIGCOMM workshop on Mobile cloud computing (MCC), August 2012, Helsinki, Finland

[6] “Mobile-Edge Computing – Introductory Technical White Paper”, https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf

[7] K.Chen et al., “C-RAN: The Road Toward Green RAN”, White Paper, China Mobile Research Institute (2011).

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti and M. F. Kaashoek, "The Click Modular Router Project," ACM Transactions on Computer Systems, vol. 18, no. 3, pp. 263-297, 2000.

[9] GNU Radio – The free and open software radio ecosystem, Available: http:/gnuradio.org/

[10] Gerardo Garzia, “Introduction to OpenMANO”, presentation on slideshare.net http://www.slideshare.net/movilforum/introduction-to-open-mano (Published on Jun 16, 2015)

[11] openmano descriptors (produced by the Open Mano project) https://github.com/nfvlabs/openmano/wiki/openmano-descriptors

[12] UCLA CS Read, "The Click Modular Router Project," 17 January 2012. [Online]. Available: http://www.read.cs.ucla.edu/click/elements. [Accessed 20 October 2015].

[13] UCLA CS Read, "The Click Modular Router Project," 01 December 2009. [Online]. Available: http://www.read.cs.ucla.edu/click/packages. [Accessed 07 October 2015].

[14] SDNCentral LLC, "SDX Central," [Online]. Available: https://www.sdxcentral.com/nfv-sdn-products-directory/. [Accessed 05 October 2015].

[15] NGMN alliance, “NGMN 5G white paper”, 17 February 2015, Available online: https://www.ngmn.org/uploads/media/NGMN_5G_White_Paper_V1_0.pdf

[16] Sam Newman, “Building Microservices”, February 2015, O’Reilly] Media

[17] FLAVIA EU project (FLexible Architecture for Virtualizable future wireless Internet Access), http://www.ict-flavia.eu/