SUPERFLUIDITY

SUPERFLUIDITY

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

Research and Innovation Action GA 671566

Deliverable D3.1:

Final system architecture, programming interfaces and security framework specification

Deliverable Type:	Report
Dissemination Level:	PU
Contractual Date of Delivery to the EU:	1/12/2016
Actual Date of Delivery to the EU:	1/12/2016
Workpackage Contributing to the Deliverable:	WP3

Editor(s):	Stefano Salsano (CNIT)
Author(s):	Carlos Parada (ALB), Francisco Fontes (ALB), Isabel Borges(ALB), Omer Gurewitz (BGU), Asaf Cohen (BGU) Philip Eardley (BT), Andy Reid (BT), Giuseppe Bianchi (CNIT), Nicola Blefari Melazzi (CNIT), Luca Chiaraviglio (CNIT), Pierpaolo Loreti (CNIT), Stefano Salsano (CNIT), George Tsolis (CITRIX), Christos Tselios (CITRIX), Michael J. McGrath (Intel),

	Filipe Manco (NEC), Felipe Huici (NEC), Lionel Natarianni (NOKIA-FR), Bessem Sayadi (NOKIA-FR), Erez Biton (NOKIA-IL), Danny Raz (NOKIA-IL), Yaniv Saar (NOKIA-IL) John Thomson (OnApp), Julian Chesterfield (OnApp), Michail Flouris (OnApp), Anastassios Nanos (OnApp), Haim Daniel (Red Hat), Livnat Peer (Red Hat), Pedro de la Cruz Ramos (Telcaria Ideas S.L.), Juan Manuel Sánchez Mateo (Telcaria Ideas S.L.), Raúl Álvarez Pinilla (Telcaria Ideas S.L.), Pedro A. Aranda (Telefónica, I+D), Costin Raiciu (UPB), Radu Stoenesccu (UPB)
Internal Reviewer(s)	Andrea Enrici, Bessem Sayadi (Nokia FR)

Abstract:	This deliverable provides the specification of the architecture, APIs and the security framework designed by the Superfluidity project.
Keyword List:	Architecture, reusable functional blocs, programmable network functions

VERSION CONTROL TABLE			
VERSION N.	PURPOSE/CHANGES	AUTHOR	DATE
V1	First version	Stefano Salsano	28/11/2016
V1.1	Final version	Nicola Blefari Melazzi	01/12/2016

INDEX

List of Figures

List of Tables

List of Abbreviations

3GPP	3rd Generation Partnership Program
API	Application Program Interface
ASA	Adaptive Security Appliance
BBU	Baseband Unit
CDF	Cumulative Distribution Function
CDN	Content Distribution Network
CIR	Centralised Infrastructure Repository
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
CRAN	Cloud RAN
CSI	Channel State Information
CTL	Computation Tree Logic
DC	Data Centre
DDoS	Distributed Denial of Service
DOS	Denial of Service
DPI	Deep Packet Inspection
eNB	Evolved Node B
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
EVNF	Emergency VNF
GS	Group Specification
HW	Hardware
IETF	Internet Engineering Task Force
IS	Identification Server
ISG	Industry Specification Group
KPI	Key Performance Indicator
LCM	Life Cycle Management
LTE	Long Term Evolution
MAC	Medium Access Control
MANO	MANagement and Orchestration
MEC	Mobile Edge Computing
MIMO	Multiple-Input Multiple-Output
MISO	Multiple-Input Single-Output
MoC	Model of Computation

NB	Node B
NDP	Neighbour Discovery Protocol
NE	Network Element
NEMO	NEtwork Modeling
NF	Network Function
NFPD	Network Forwarding Path Descriptor
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
NIC	Network Interface Card
NS	Network Service
ONF	Open Networking Foundation
OPP	Open Packet Processor
OS	Operating System
OSS	Operational Support Systems
RAN	Radio Access Network
RDCL	RFB Description and Composition Language
REE	RFB Execution Environment
RFB	Reusable Functional Block
RRH	Remote Radio Head
SDN	Software Defined Networking
SDO	Standard Defining Organization
SEFL	Symbolic Execution Friendly Language
SLA	Service Level Agreement
SNCG	Secure Network Coding Gossip
SW	Software
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TOF	Traffic Offloading Function
TOSCA	Topology and Orchestration Specification for Cloud Applications
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNE	Virtual Network Element
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
VNFD	Virtual Network Function Descriptor
VNFFG	VNF Forwarding Graph
VNFFGD	VNF Forwarding Graph Descriptor
VNFM	Virtual Network Function Manager
XFSM	eXtended Finite State Machine
YAML	YAML Ain't Markup Language

1 Introduction

1.1 Deliverable Rationale

This public deliverable represents the final result of the Work Package WP3 “Cloud-Native Edge System Architecture” of the Superfluidity project. It builds on the work related to the analysis of the use cases and requirements (Deliverable D2.1 [25]) and to the functional analysis and decomposition (Deliverable D2.2 [34]) of a 5G system. This deliverable provides the overall system architecture and a high level identification of the APIs (Part A), allowing the technical Work Packages (WP4, WP5, WP6) to further progress with the detailed design of the APIs that will be implemented and integrated in the demonstrations. It also provides the analysis of security aspects and the definition of a security framework (Part B), which will be further developed in WP6.

1.2 Executive summary

This deliverable presents the Superfluidity architecture. With a focus on the softwarization of network functions, the primary objective of the document is to promote a “divide et impera” approach for the design and operations of network services, with the notion of Reusable Function Blocks (RFB) as elementary primitives. RFBs can be integrated and composed together in a programmatic manner to devise more comprehensive (composed) network functions in a rapid and cost-effective way. RFBs can be “orchestrated” over their *execution platforms* (REE – RFB Execution Environments) in order to achieve an optimal efficiency in the utilization of the needed resources (e.g. computing, networking, storage). In this document, we specifically highlight the general (abstract) concept of RFB and discuss the underlying issues and the necessary architectural support, raising the need (and role) for a RFB Execution Environment. We consider that the architecture defined by Superfluidity can be recursively applied to solve network design and operations issues that can be found at multiple levels of abstraction (e.g., from network wide abstractions dealing with the interaction of different physical or virtual appliances to device level abstraction covering the interaction of hardware and software local components). Therefore, in this document, we define a generic approach that can be applied at all abstraction levels, regardless of their specific characteristics. At each abstraction level, we identify a REE Manager and a REE User and define in general terms the programming interfaces (APIs) that characterize a RFB Execution Environment. We then proceed by casting this general approach to a variety of more concrete scenarios, ranging from traditional NFV frameworks, where composability and reusability of comprehensive network functions is already widely established, to more advanced data plane programmability environments where more fine-grain decomposition of network services and tailored execution environments play a key role.

Any architecture that relies on real-time cloud infrastructures is susceptible to security hazards. All the more so when the architecture relies on virtualization of network functions. The Superfluidity project is facing the challenges of identifying and addressing these security hazards. In Part B of this report we describe the specific security challenges and hazards that the Superfluidity architecture will need to address and we propose solutions we intend to adopt. In particular, we discuss the key security challenge of understanding the functionality and operations of a given processing block before it is instantiated. To solve this issue, we suggest countermeasures that rely on symbolic execution. We further discuss two solutions aiming at identifying abnormal behaviour. Specifically, we first suggest a novel anomaly detection mechanism which is able to learn the normal behaviour of systems and alert for abnormalities (unusual requests, unusual code execution, illegitimate requests, etc.), without any prior knowledge of the system model (e.g., without assuming any prior knowledge on the type of requests nor any knowledge on the system normal operation). In addition and complementary to this, we propose an approach which relies on static analysis of RFBs to verify that the expected output of incoming traffic to any "black-box" complies with the legitimate behaviour of the black-box. In addition to the above-mentioned security issues related to software processing, in this report we also analyse the vulnerability of the C-RAN architecture to eavesdroppers. In particular, we analyse the secrecy rate and outage probability in the presence of large number of legitimate users and eavesdroppers. Last but not least, we discuss novel physical layer schemes to secure communications with minimal rate loss and complexity.

PART A – SYSTEM ARCHITECTURE AND APIs

2 Overall architectural framework for Superfluidity

Current cellular networks are facing significant challenges. In 2015, mobile data traffic grew by nearly 70% with video streaming services contributing over 50% to that growth. Smartphones have been a key driver in this growth in combination with the rollout of LTE networks. By the end of 2015, there were nearly a billion LTE subscribers globally, belonging to nearly 500 commercial networks. In the US, mobile network traffic for AT&T network grew by 100,000% from 2007 to 2014 and video traffic doubled from 2014 to 2015. The challenge lays, therefore, in supporting the increasing mobility and increasing bandwidth requirements of end-users.

The explosion in the Internet of Things (IoT) is rapidly driving up mobile network traffic. It is expected that there will be over 30 billion connected devices by 2020, divided across massive IoT and critical IoT. To address the explosion in mobile traffic and connected devices, service providers need to perform better capacity management, in order to react dynamically to the network conditions and device's traffic. Additionally, there is also a need to handle the heterogeneity of mobile networks themselves, in terms of access technologies and deployment [1].

5G is expected to support multiple industries/user business cases, multiple access technologies, multiple services and multiple tenants. 5G systems will rely on advancements in the radio connectivity (e.g., massive MIMO, beam-forming, etc), in the deployment options (e.g., small cell, multi-RAT, etc), in the network capability (e.g., mobile Edge Computing) and in the latest advancements in the SDN and NFV domains.

One can classify the 5G use cases under consideration by the various Standard Defining Organizations and stakeholders fora into three broad categories [2], namely: (i) Enhanced Mobile Broadband (eMBB), (ii) Massive IoT, and (iii) Critical IoT. eMBB covers a variety of new challenging use cases including: Immersive Internet, Augmented Reality, Virtual Reality, Ultra High Definition (UHD) Content Delivery, to cite a few. Critical IoT covers use cases like eHealth, Vehicular communication, etc. It is important to highlight, here, the stringent requirements that 5G should fulfil for these use cases in terms of ultra-low latency (~1ms) and reliability (`5 nines'). The Superfluidity project has analysed a set of use cases to help in the process of gathering requirements for the definition of its architecture. The outcome of this activity has been reported in Deliverable D2.1 [25].

The Superfluidity project aims to build 5G networks addressing the above-mentioned use case requirements and unifying existing network design approaches that support

1. Agility: Quick and flexible service creation and deployment.
2. Mobility: Flexible placement and rapid migration of services.

3. Abstraction and efficient exploitation of network heterogeneity
4. Functional composition/decomposition of services by means of the *Reusable Functional Block* (RFB) concept.
5. Efficient RFB implementation, e.g. by means of hardware (HW) and software (SW) acceleration when needed.
6. Security by design: RFBs are verified to be secure before deploying them.

The Superfluidity architecture will rely on a proper combination of a set of emerging technologies in different areas. Beyond the utilization of multiple technologies, the big innovation brought by the project lays in how those technologies can be integrated into a single conceptual environment, taking advantage of synergies among them. Superfluidity leverages the works conducted individually in many forum, namely ETSI NFV, ETSI MEC, ONF, 3GPP and TMForum and advances them. Figure 1 depicts a high-level view of the overall architectural framework considered in SUPERFLUDITY. In this section, we assume the reader is familiar with the different involved technologies. In Section 6 we provide some basic tutorial information and a short report on the state of the art of these technologies.

The top layer of the Figure 1 includes the different components involved (CRAN, MEC, virtual core and Data Centres - DCs), while in the bottom layer the different types of physical DCs are shown (namely Cell-site, Local, Regional and Central). This classification is somehow arbitrary and the infrastructure of different operators can be structured in different ways. The next layer down is a traditional Operational Support System (OSS), whose main goal is to deal with all the components in order to create services for end-users.

On the bottom, an Extended-NFVI represents an evolution of the ETSI NFVI concept, that considers the additional heterogeneity of the infrastructure that we want to model (e.g. including specialized hardware for radio or packet processing, hypervisors, and other execution environments), and the federation of DCs at different geographies and different types. This extended-NFVI is common to all components, easing resource management and allowing an agile ("superfluid") orchestration of services. The mapping of the components into different physical DCs (Cell, Local, Regional) that is shown in Figure 1 and described hereafter is the one considered for the Superfluidity testbed implementation. However, we stress that the flexibility of the architecture, together with the concept of an Extended-NFVI, allow support for different mappings of components into the physical DCs, derived by considering the various trade-offs between performances and efficient utilisation of resources.

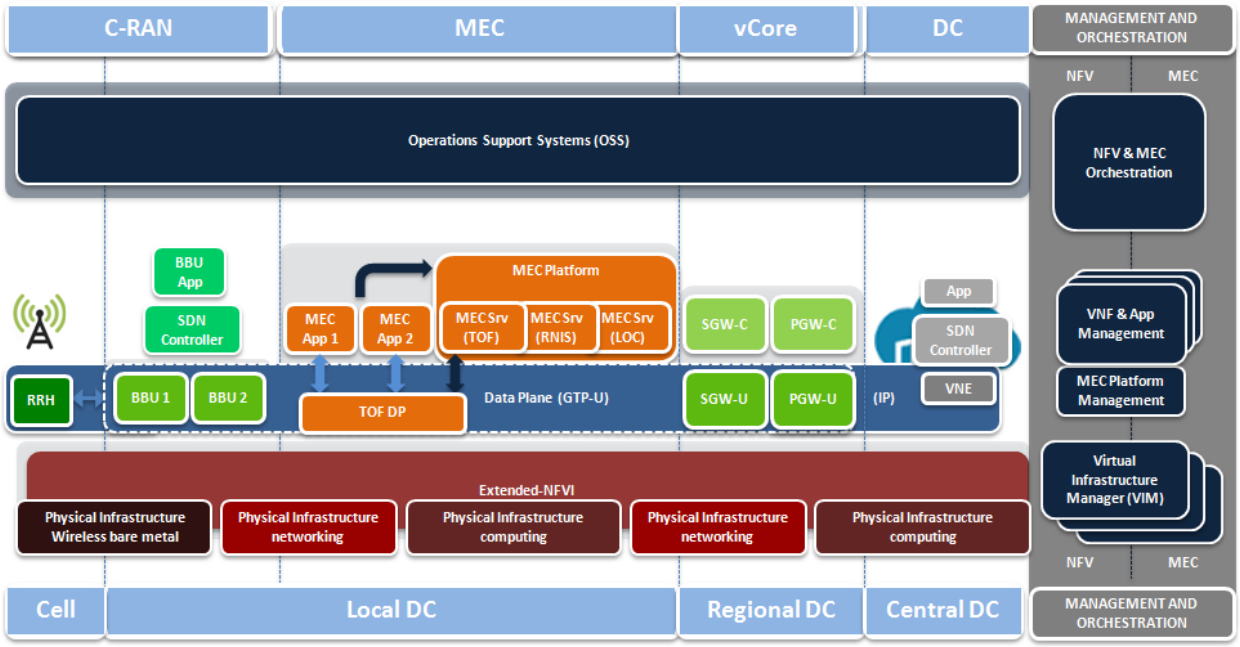


Figure 1: Architectural framework for Superfluidity with an example mapping into the physical Data Centres

Starting from the left of Figure 1, the CRAN component is split into two blocks, corresponding to the Remote Radio Head (RRH) and the Base Band Unit (BBU) components. The RRH is placed in the cell-site, while the BBU is located in a specialised local DC, adapted for telecommunications. It is assumed that the local DCs will control a small number of cell-sites and that are geographically distributed, but in general proximity to the cell sites. The BBU functionalities will be virtualised and thus they will benefit from their centralisation to scale in/out, according to the load and to be fully re-programmable by a holistic NFV/SDN controller.

The MEC component, in Figure 1, appears next and can be deployed in the same location (DC) as the C-RAN component. As C-RAN concentrates in the local DC a large number of RANs, this is the right place to run MEC applications (*note: Other options are also possible, e.g. closer to the core; however, this architecture would result in higher latencies*). The MEC environment uses the same infrastructure as the C-RAN or other VNFs, making the architecture very convenient and efficient.

The next component shown in Figure 1 is the virtual Core (vCore), that comprises the central nodes of a cutting-edge mobile network. The vCore runs on the common E-NFVI, usually located in regional DCs. In this way, both agility and fluidity of the overall architecture are improved, especially when live nodes need to be migrated and/or scaled. Figure 1 shows the expected evolution of mobile core networks towards the SDN model, where the control plane and data plane components are completely separated, with the former fully controlling the latter on data processing tasks. In particular, the components shown are the ones resulting from splitting the 4G/LTE Core elements. The DC component corresponds to the traditional datacentre segment, where a large number of services are deployed. These services are located at central points and deal with significant compute/storage/network resources. Beyond traditional services, central

DCs also implement NFV and SDN technologies, in order to control the network components inside the DC. For this reason, a generic VNE (Virtual Network Element), a SDN controller and a generic App element are included in the architecture.

The last part of the architecture in Figure 1 is the common management and orchestration vertical layer (right part of the Figure), which is responsible for providing the system intelligence. It includes a NFV-like set of functions, which supports an integrated view of the overall architecture, including networks, services and DCs. In this way, it is possible to take advantage of a common extended-NFVI and achieve an end-user centric view of the ecosystem. The management and orchestration layer is responsible for resource management over the different DCs illustrated at the bottom of Figure 1, thus building a federated environment. Moreover, it orchestrates VNFs to create complex services, taking the best decisions for services to be deployed, while considering customer needs. In the bottom of this vertical box, a VIM-like set of components manages the resources on the different DCs, interacting with them to build a federated environment. In the middle, a set of VNF and App Managers handle the lifecycle details of particular VNFs and Applications. Also, a Platform Manager is used to manage service APIs and applications access to those APIs. Finally, orchestrators are able to orchestrate VNFs to create complex services and take the best decisions for Apps to be deployed and serve customers.

3 Superfluidity Architecture

The vision of the Superfluidity project is to move from the current architectural approaches based on monolithic network components/entities and their interfaces, to an approach where network functions can be programmatically composed using "building blocks" that are dynamically deployed, allowing a continuous real-time optimization of the network.

The idea of decomposing relatively complex network functions in more elementary parts is certainly not novel by itself, and has been explored in many scenarios. These scenarios range from higher level Network Function Virtualization frameworks [3] [26], to low level programmable switches/routers based on modular blocks (e.g. Click routers [17]) to even lower level approaches entailing the identification of very elementary primitives (e.g. OpenFlow actions [4]). However, while discussing such different approaches, we realized that most of the work in this area has remained quite focused on the specific domain scenario, and only a limited attempt to find out similarities and general principles among the different approaches has been carried out to date. At the cost of appearing perhaps abstract (but we will cast such general ideas into more concrete scenarios in the next sections), the goal of this section is to try to understand if there are some common architecture elements and principles which are shared by composition-based approaches.

3.1 Overview

We propose to base the decomposition of high-level monolithic functions into reusable components on the concept of ***Reusable Functional Blocks*** (RFBs). A RFB is a logical entity that performs a set of functionality and has a set of logical input/output ports. In general, a Reusable Functional Block can hold state information, so that the processing of information coming in its logical ports can depend on such state information. RFBs can be composed in graphs to provide services or to form other RFBs (therefore a Reusable Functional Block can be composed of other RFBs). The RFB (de)composition concept is applied to different heterogeneous environments in which the RFB can be deployed and executed. We call these environments ***RFB execution environments*** (REE).

In order to achieve the vision of superfluid composition, allocation and deployment of "building blocks" (RFBs) the Superfluidity project focuses on the following pillars:

Decomposition of monolithic functions into Reusable Functional Blocks (RFB) in different heterogeneous execution environments (REE) - Superfluidity aims to decompose network services into basic functional abstractions; node-level functions into more granular packet processing primitives; heterogeneous hardware and system primitives into block abstractions.

Measurements based RFB operations - This pillar consists in the definition and realization of tools for real time monitoring of RFB performances and of the status of the REE environments (e.g. resource utilization). The information provided by such tools can be used to drive the RFB operations (e.g. allocation of resources to RFBs).

Semantic specification of Block Abstractions – A crucial aspect in composition-based approaches is to provide the "programmer" with a clear and unambiguous understanding of what each block is expected to do, and how it processes input data and produces an output.

High Performance Block Abstractions Implementation - This pillar consists of deriving block abstractions for each of the underlying hardware components, and to optimize their performance.

3.2 RFB heterogeneity: different abstraction levels and granularity

An important peculiarity of the Superfluidity approach is that the (de)composition is applied to different heterogeneous execution environments (REEs), from network-wide composition of (virtual) network functions to node-level composition of packet processing blocks, to composition of signal processing blocks. Actually, it is possible to operate recursively on the REEs, for example network-wide services can be decomposed in node-level RFBs, which in turn can be decomposed in smaller packet processing RFBs. For this reason, we can refer to the RFB Execution Environments as different ***abstraction levels***.

An important property is the formal detail that characterizes the description of an RFB. In some RFB Execution Environments, the RFBs can be seen as *small*, elementary building blocks to which it is possible to associate a formal description of their behaviour using some adequate language. Therefore, starting from the formal description of the RFBs functionality, it is conceptually possible to compose RFBs and derive the properties of the composed RFB. In this case, we can define models that describe the RFBs with high granularity and the languages that describe the RFB are able to capture the functional "input/output" behaviour of the RFB with all details. Examples of this type of REE can be modular software routers or Software Defined Radio (SDR) processors.

In other environments, like for example a NFV Infrastructure, the RFBs are high-level components, which are expected to respect some functional specification (e.g. a large number of protocol specification documents). In this case, it is not possible to have a complete description of the RFB functionality using some formal language. The languages that model the RFB composition operate on RFBs with coarse granularity (e.g. on high-level service building blocks). The RFB descriptions, in this case, are not able to completely capture the "input/output" functional behaviour of the RFBs and are mostly used to describe non-functional properties like requirements on the underlying infrastructure supporting the RFB execution.

Given this heterogeneity and considering the target framework for Superfluidity (see Figure 1) it is not possible to build from scratch a *clean slate* 5G system that homogeneously follows a new Superfluidity architecture design. Rather, the Superfluidity architecture provides a unified framework, combining, adapting and extending existing models and languages.

3.3 RFB Operations and RFB Composition views

In the definition of the Superfluidity architecture and models, the RFB concept is used in two complementary ways. On one hand, it is used to model the *allocation* of service components to an execution platform, with the proper mapping of resources. We refer to this approach as *RFB Operations*. On the other hand, it is used to explicitly model the *composition* of RFBs to realize a service or a component. We refer to this approach as *RFB Composition*. The two approaches are not mutually exclusive and in some scenarios they can be combined.

Considering these two approaches and the heterogeneity of the target environments that we have discussed in the previous section, the Superfluidity architectural modelling based on RFBs can support the definition of different types of frameworks:

Operation/Orchestration frameworks: These frameworks concern the deployment of high-level RFBs onto the supporting execution environments in an optimal way, taking into account the real time status of resources (e.g. computing, networking, storage). As an example, we can refer to the current ETSI NFV approach, with the allocation of VNFs (Virtual Network Function) to their execution Environment (NFVI).

Composition frameworks: These frameworks concern the automatic composition of RFBs to build other functional components or services. They can be seen as “compilers” or “builders” that produce executables for composed RFBs, or they can deal with the formal correctness check of RFBs chains. The features of these frameworks depend on the type of RFBs and on the characteristics of the RFB execution environments. As an example, we can refer to a provisioning framework that matches node-level function abstractions (e.g. a firewall) with the available hardware/block abstractions (e.g. packet processing primitives) in order to *automatically* derive high performance and meet end-user SLA requirements, i.e., without forcing developers to understand low-level system details.

3.4 Superfluidity Architecture

In the previous section, we have introduced the concept of Reusable Functional Blocks and proposed their use in Operation and Composition frameworks. Let us further proceed in the Superfluidity architecture modelling that will support such frameworks.

RFBs need to be characterized and described: we need platform-agnostic node-level and network-level “programs” describing how the RFBs interact, communicate, and connect to each

other so as to produce (macroscopic) node components, network functions and services. A language that supports the description and the interaction of RFBs is referred to as ***RFB Description and Composition Language (RDCL)***. At the state of the art, we believe it is not possible to have a single language that is capable to describe the different types of RFBs in the heterogeneous environments and properly capture the needed features to support the RFB Operations and Composition views. Therefore, the Superfluidity architecture will deal with a set of RDCLs, targeted to different environments. The Superfluidity project will extend the existing RDCLs where needed to support the dynamicity of the allocation/deployment on the underlying infrastructure and to support the coordination of operations in the different heterogeneous environments.

The heterogeneous computational and execution environments, supporting the execution (and deployment) of the RDCL scripts and the relevant coordination of the signal/radio/packet/flow/network processing primitives are referred to as RFB Execution Environments (REE). A set of REEs considered in the context of Superfluidity is described in Section 5.

The Superfluidity architecture shown in Figure 2 describes the relation among the RFB, RDCL and REE concepts. The figure highlights that the model (with due technical differences) recursively applies at different levels of abstraction.

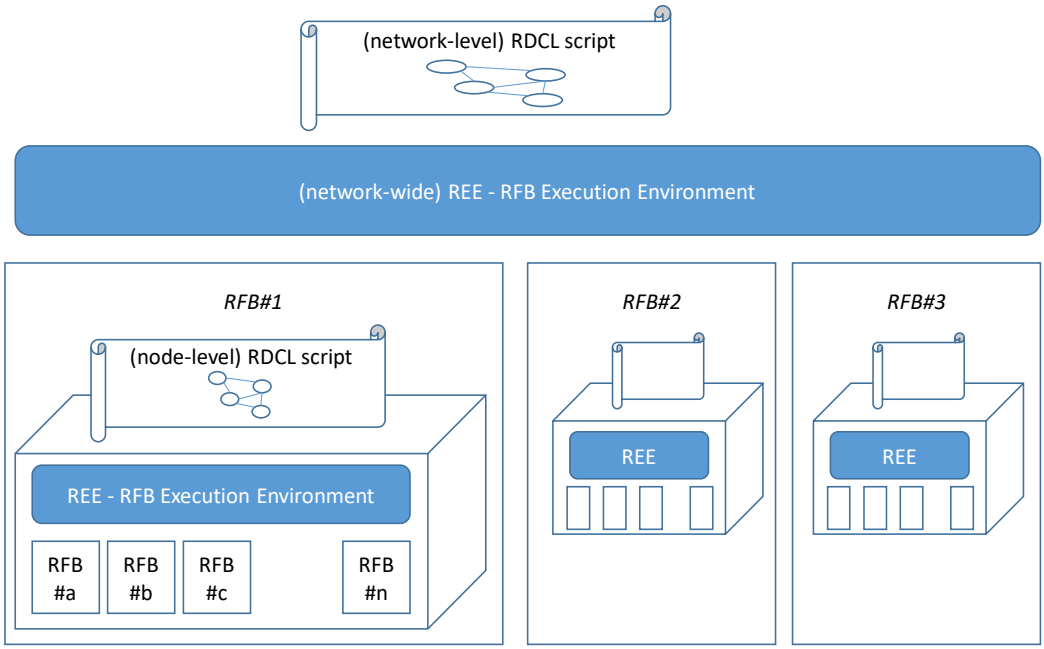


Figure 2: Superfluidity architecture

For example, let us consider the abstraction level indicated as “network-level” in Figure 2. It represents the typical NFV environment (NFVI) of the ETSI model, where RFBs are meant to be

relatively large functions (e.g. a load balancer, a firewall, etc., represented by the RFB #1, #2, #3) and the REE is a network-wide orchestrator (usually complemented by an SDN controller). In this context, a specific high-level network function can be implemented over a programmable network node using in turns a decomposition into more elementary sub-blocks (e.g. packet processing primitives of a programmable router or OpenFlow-like elementary forwarding and processing actions, described as RFB #a, #b, #c).

Under this vision, operators can formally describe a desired network service as a composition of platform-agnostic, abstract elementary processing blocks (RFBs). Vendors will be in charge of providing efficient software implementations of such elementary blocks, possibly casting them to underlying hardware accelerated facilities. The (cloud-based) infrastructure will provide the brokerage services to match the design with the actual underlying hardware, to control and orchestrate the execution of the RFBs comprising the designed service, and to permit dynamic and elastic provisioning of supplementary dedicated computational, storage, and bandwidth resources to the processing components loaded with peak traffic. As long as different platform vendors expose the same set of RFBs, and as long as they are clearly specified in terms of semantics (i.e. as long as RFBs implemented by different vendors produce the same output for a same input), how they are specifically implemented (e.g., in HW or in SW) is not nearly a concern for the network programmer. A standardization of such set of RFBs (again, think to the OpenFlow analogy in terms of standardized actions) is all needed to guarantee that an application which suitably composes RFBs running on a given platform (e.g. a SW switch) can be migrated on a different platform (e.g. a bare metal HW switch) which supports the same set of RFBs.

The fundamental concepts of the Superfluidity architecture have been presented and discussed in [11]. This deliverable contains an updated and more detailed specification of the architecture with respect to the content included in [11] and it is fully aligned regarding the overall vision.

3.4.1 Reusable Function Blocks (RFBs)

As anticipated in the overview section 3.1, an RFB is a logical entity that performs a set of functionality and has a set of logical input/output 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. 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). 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.

To make an example of RFB Execution Environment let us consider the NFV Infrastructure (NFVI) under standardization by ETSI. In this context, the RFB can be mapped into the concepts of VNF (Virtual Network Function) and of VNFC (Virtual Network Function Component). For example, a RFB that can be mapped 1:1 against a single Virtualization Container (using the ETSI terminology)

that corresponds to an ETSI VNFC (VNF Component). In this case, its execution environment is a Virtualization Container hypervisor.

Another example of RFB Execution Environment is a modular programmable router architecture like Click [17]. In the Click architecture, a set of components called Click elements can be composed by means of a directed graph (called configuration). In this context, a RFB corresponds to a Click element and will have as execution environment a Click router.

Figure 3 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 [5]). VNFs and VNF Components. VNF Components cannot be further decomposed. In the proposed model, all elements are RFBs and the decomposition can be iterated an arbitrary number of times.

To make an example of hierarchy, a Click router instance can be a VNF Component in a VNF. Its execution environment is a VC hypervisor. The Click instance is described by means of a directed graph of elements (called configuration). Each element is a RFB that has the Click router as execution environment. As shown in Figure 4, in the current modelling approach it is not possible to further decompose the VNFC.

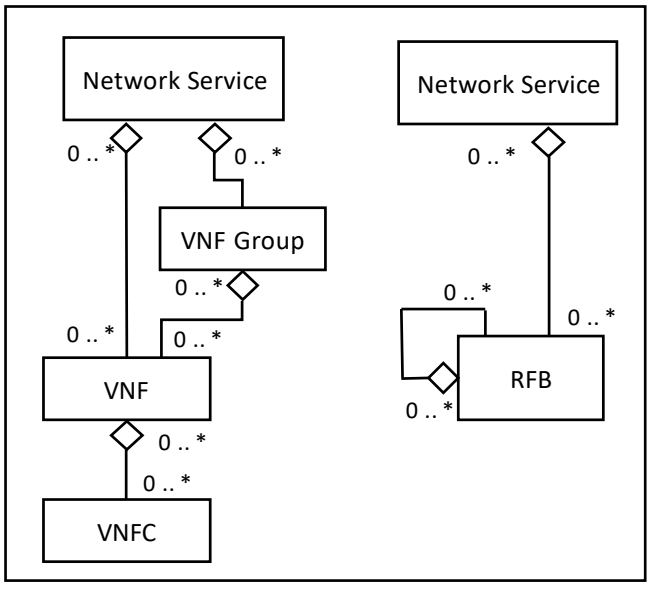


Figure 3: Class diagram for the current approach (left side) and the proposed one (right side).

RFBs can be composed to provide high-level functions and services but also to form other RFBs, as a RFB can be composed of other RFBs. The decomposition of RFBs into other RFBs can be iterated an arbitrary number of times, unlike VNF Components that cannot be further decomposed.

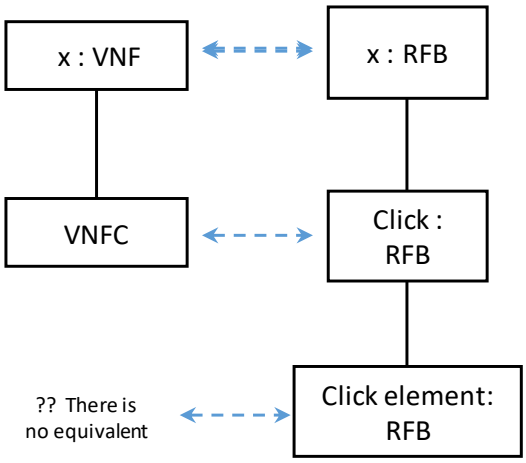


Figure 4: Example of a hierarchy of RFBs.

3.4.2 RFB Description and Composition Language (RDCL)

The concept of RFB lends itself to different usage scenarios. An RFB *Composition* framework may consider that logical RFBs can have more than one possible execution environment (i.e. they can be realized using different technologies and frameworks). Elementary radio, packet, and flow processing primitives and events can be formally described as Reusable Functional Blocks independently of the specific underlying hardware. In this case, these RFBs can be automatically instantiated to match the underlying hardware facilities while taking advantage of the relevant accelerators (e.g., GPUs or FPGAs) if/when available. On the other hand, we can have RFBs bound to a specific RFB Execution Environment, and an *Orchestration* framework that only cares about coordinating their execution and allocating them to resources in an optimal way within the specific Execution Environment.

The different scenarios and the heterogeneous RFB Execution Environments imply that a set of different tools are needed to describe the RFBs and their composition. We refer to these tools as RFB Description and Composition Languages (RDCLs).

Considering the NFV Infrastructure under standardization by ETSI, some of the features that VNF, Virtual Machine and Compute Host Descriptors should contain for the appropriate deployment of VNFs on Virtual Machines over Compute Hosts in a telco data center are described in [7].

An approach towards a recursive definition of RFBs has been recently proposed in [24].

An important usage scenario for the RFBs composition is the formal correctness check of a configuration before its deployment. This requires RFB Description and Composition Languages that are able to express the functional properties of a RFB with a formal (semantic) language.

Coming to RFBs that describe packet processing operations, a key to portability is the ability to describe a potentially complex and stateful network function as the combination of RFBs. The

goal is to obtain a platform-agnostic formal description of how such RFBs should be invoked, e.g. in which order, with which input data, and how such composition may possibly depend (and change!) based on higher level "states". Indeed, writing a high-level network function as a composition of blocks that are pre-implemented in the underlying platform(s), facilitates migration of such function from a platform to another. As an example, an RFB could be re-allocated from a software node to a more capable hardware accelerated node, or from a node placed in the core to an edge node. In general, the reallocation can happen in different phases of the service life cycle and at different time scales. For example, considering the phases in the service life cycle, the simpler approach is that the allocation is done only at the deployment time. Even in this simple case, different options are possible with respect to the time scale of the allocation process, which could range from static approaches (deployments in the time scale of days) to dynamic approaches in which the RFBs are dynamically allocated following real time requests (e.g. in the order of few seconds).

While the idea is simple, its actual specification may be all but trivial. Different languages may be more appropriate to different network contexts (e.g. for node-level programmable switch platforms opposed to network-wide NFV frameworks). And, to the best of our knowledge, as of now there is not a clear cut candidate standing out. Indeed, the typical approach, used in block-based node platforms (such as the Click router, or even in Software Defined Radio platforms) is to formally model a composition of blocks as a Direct Acyclic Graph. However, this description is typically not sufficient, as it does not permit the programmer to introduce the notion of "application level states" and hence dynamically change (adapt) the composition to a mutated stateful context. Languages in the IFTTT family (If This Then That), recently introduced in completely different contexts (such as web services or Internet of Things scenarios), may find application also in the networking context. This may be possible thanks to their resemblance with the matching functionality that is frequently used in forwarding tasks, although, again, stateful applications may require extensions. It is worth to mention that the Superfluidity project is specifically addressing novel languages formalized in terms of eXtended Finite State Machines. These appear suited to generalize the widely understood OpenFlow's match/action abstraction, and at the same time appear to retain high-speed implementation feasibility.

3.4.3 RFB Execution environment (REE)

Having a set of blocks (i.e., the RFBs), and having a language which describes how they are composed does **not** conclude our job. In the proposed architectural framework, we **also** need an entity or a framework in charge of "executing" such a composition. For an extreme analogy with ordinary computing systems, in addition to the processor instruction set and the programming (machine) language, we also need a Central Processing Unit which is in charge to execute the workflow and directives formally specified by the programmer using the language. Generally

speaking, we refer to such framework as “RFB Execution Environment” (REE). The role of the REE is to concretely instantiate a desired RDCL script instance, to control its execution, to maintain and update the application-level states, to trigger reconfigurations, and so on. In traditional network-wide NFV scenarios, this role is generally played by an orchestrator. However, in VNFs implemented in software within a container, or a Virtual Machine, the role of the REE is played by the VM itself. Conversely, the case of high-speed bare metal switches where configuration and per-flow state maintenance must occur at wire speed is a bit more problematic: in most architectures this role may not be clearly identified. Delegating this role to the overlying software/control stack may not be the most performance effective solution and it may lead to performance bottlenecks and slow-path operation.

The REE can be seen as a generalization and enhancement of the NFVI (NFV Infrastructure). In the current model under standardization by ETSI [3] the infrastructure (NFVI) provides resources and an external orchestrator coordinates them (but it only instantiates VMs and connects them according to the graph that describes the network services). In the approach proposed here, the enhanced infrastructure REE provides the means to execute arbitrarily complex RDCL scripts operating at different abstraction levels.

3.4.4 Recursive architecture

As mentioned in Section 3.4.1, there are some differences between the current ETSI documents and the Superfluidity architecture. We have been pursuing updates at ETSI in order to promote a fundamentally recursive architecture, as seen in Figure 3, into the standards. It is being adopted as the key technique for describing integration of existing systems and NFV, and a single comprehensive framework that incorporates NFV and SDN. Superfluidity experts are leading ETSI’s “End to end process descriptions” Work Item within the NFV EVE (Evolution and Ecosystem) group. This section acts as a status report on progress.

The architecture needs to fulfil a number of requirements, including:

- The need to interwork with existing systems (and their OSS/BSS). We want an existing service to be able to incorporate an NFV component, and we want an NFV service to be able to incorporate an existing service (WAN connectivity, for instance) as a component.
- The need for future system evolution. We want to be able to upgrade the individual components of an NFV network service, at different times.
- The need for NFV services between network operators. We want a network operator to be able to use NFV-related components from other operators, for example NFVI-as-a-Service or Network Service-as-a-Service.

In order to meet these requirements, we have proposed in ETSI EVE a layered architecture. There are an arbitrary number of layers, with each layer having the same three sets of capabilities and the same three interfaces. A layer here corresponds to an *abstraction level* or RFB Execution

Environment as defined in the previous section. Hence, the architecture is fundamentally recursive.

Each layer must have a full set of administration functions, which support the full life-cycle of the layer service. Taken at an abstracted high level, these layer administration functions fall into three groups of capabilities, which are used at different phases of operations:

- the development and definition of the service, to create a template which describes the service and is uploaded to a catalogue of service/function types which can be instantiated within the layer;
- the Instance Life-Cycle, to instantiate an instance of the service/function and request resources from the underlying layers, and also remove instances and free the resources when no longer required;
- the 'In-Life' of the service, to monitor and maintain the instance.

Each of these is described in more detail below.

Together with these three sets of capabilities, there are three interfaces with the functionality needed to support these interfaces:

- a northbound service API, which offers services to the layer(s) above;
- a southbound API, which interacts with the service API of the layer(s) below;
- a GUI or human interface for anything which needs manual intervention (note that such actions are not done via the service API)

The high-level structure of a layer admin is illustrated in Figure 5:

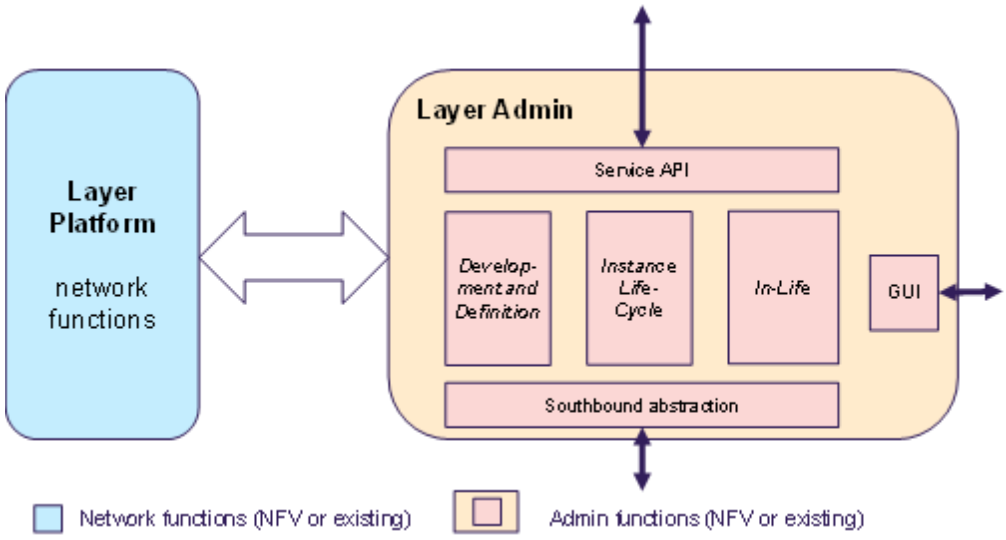


Figure 5: Basic entities of a layer admin

Figure 6 shows an example with three layers. The 'top' layer has a service that consists of three component functions. One of these is actually a service provided by the 'middle' layer; in turn, one of its components is provided by the 'bottom' layer (where it is seen to consist of three

component functions). Hence, when the top layer's admin receives a request for service, it calculates how best to meet it – this involves instantiating a composite function of three components, one of which necessitates a request for resource from the middle layer. The middle layer's admin handles this request in a similar fashion, and has request resource from the bottom layer. In summary, a layer utilises services provided by underlying layers; they provide it with resources. In a complementary fashion, a layer, in order to be useful, offers services to overlying layers and it provides them with resources.

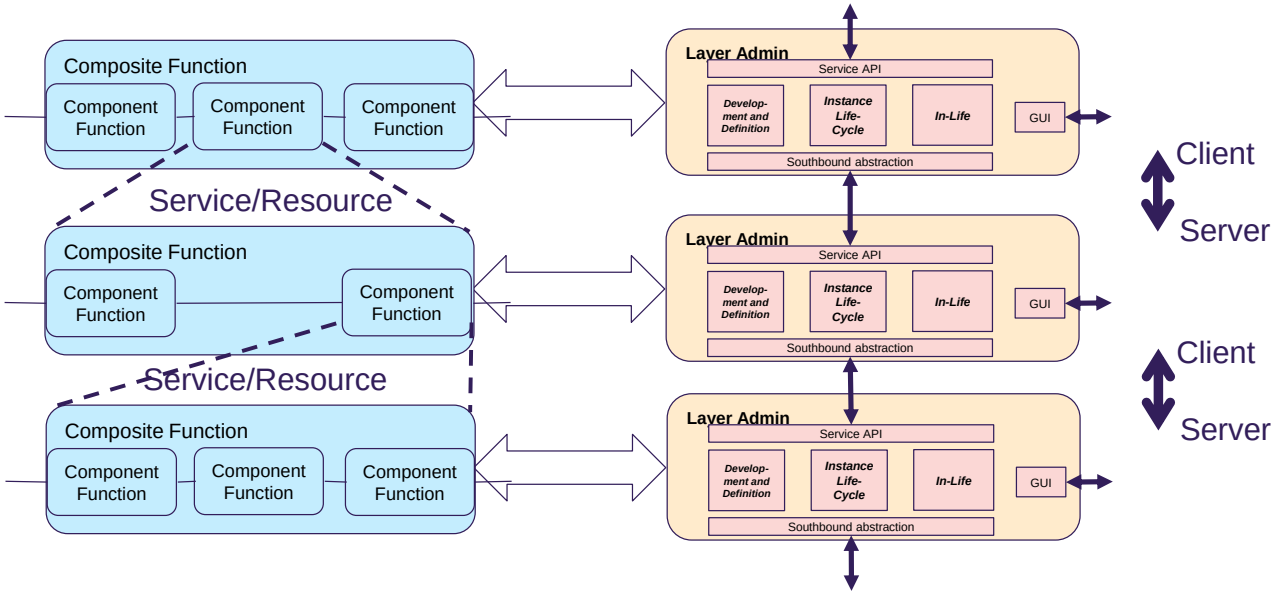


Figure 6: Example with three layers, to illustrate recursive architecture

The details of each layer are hidden from the other layers, and so each layer can be implemented in different ways or its implementation changed as long as the service presented at the interfaces is maintained unchanged. The layers may be implemented by different vendors or even owned by different parties. For example, an NFV service could incorporate an existing WAN to connect the NFV nodes. On the other hand, the same software code could be used to achieve a capability at more than one layer. In this case, the same code would be operating on different data, thereby implementing the same logical functionality at the different layers.

The logical administration of each layer is also self-contained, in that it administers full control of the services it provides. Therefore, each and every layer needs to have a full set of capabilities that administer the layer.

Finally, at system level, each layer can be seen as providing the same type of service and the same type of interfaces. This identification of a common abstract type for the services and their interfaces supports the requirements for efficient interoperability in a heterogeneous environment.

We now look in more detail at the three groups of groups of capabilities within each layer's administration functions.

The **Development and Definition phase** (illustrated in Figure 7) results in templates which can construct the services/functions offered by the layer. The templates are stored in a catalogue which can then be accessed by the Instance Life-Cycle phase for the creation of service/function instances.

The development and definition phase also includes the processes by which the templates are constructed and tested/validated. This development environment has a development toolkit, libraries of draft templates and resource types, and an isolated test and validation 'sandbox' execution environment.

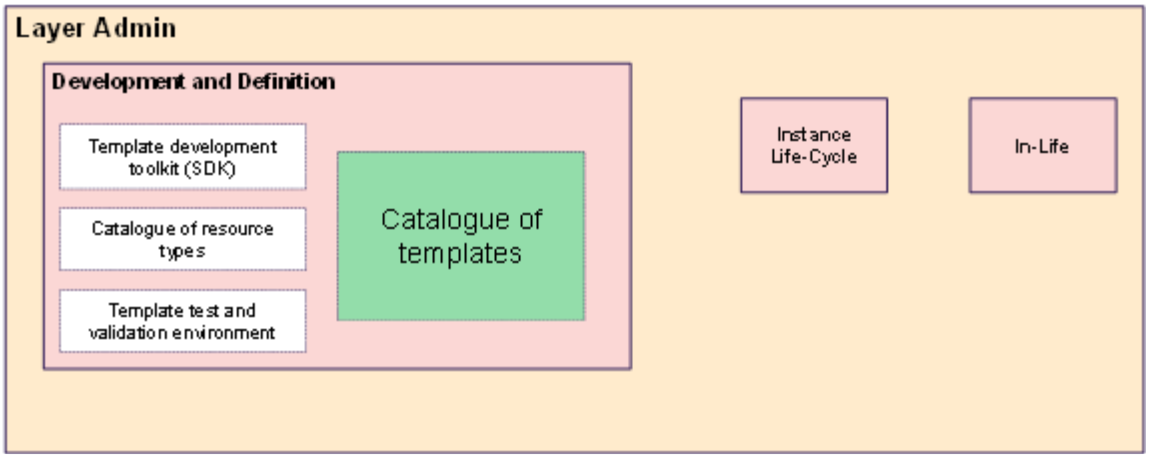


Figure 7: High level view of the development and definition phase

The **Instance Life-Cycle phase** (illustrated in Figure 8) results in instances of services/functions which, for management purposes, are registered in an inventory database of instances together with the mapping to the resources used to support each instance.

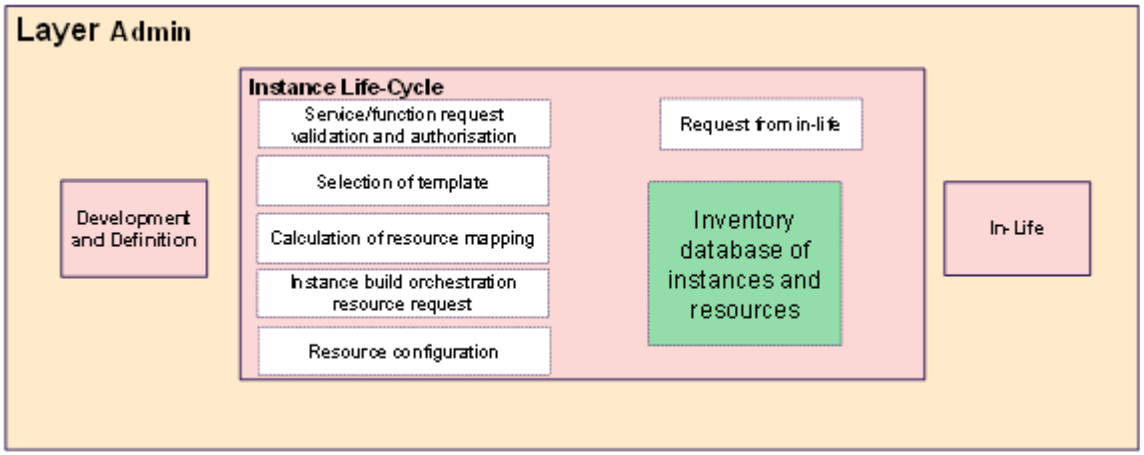


Figure 8: High-level view of the instance lifecycle phase

The phase starts with a request from a client for a service. The purpose of this phase is to validate and authorise the request, select a template from the template catalogue, calculate the resources required to instantiate the service, and then actually perform the actions required to achieve the instantiation.

The phase also maintains a full inventory of instances together with the mapping to the resources supporting each instance. It can also accept requests from the in-life phase.

The In-Life phase (outlined in Figure 9) is concerned with maintaining the health and performance of instances (which were created by the in-life phase). This consists of three main activities:

- Monitoring of instances;
- When monitoring detects one or more faulty or underperforming instances, then it calculates what action is needed and then performs the required restoration or scaling activity (which typically involves a request to the Instance Life-Cycle phase)
- Reporting the status of services to clients.

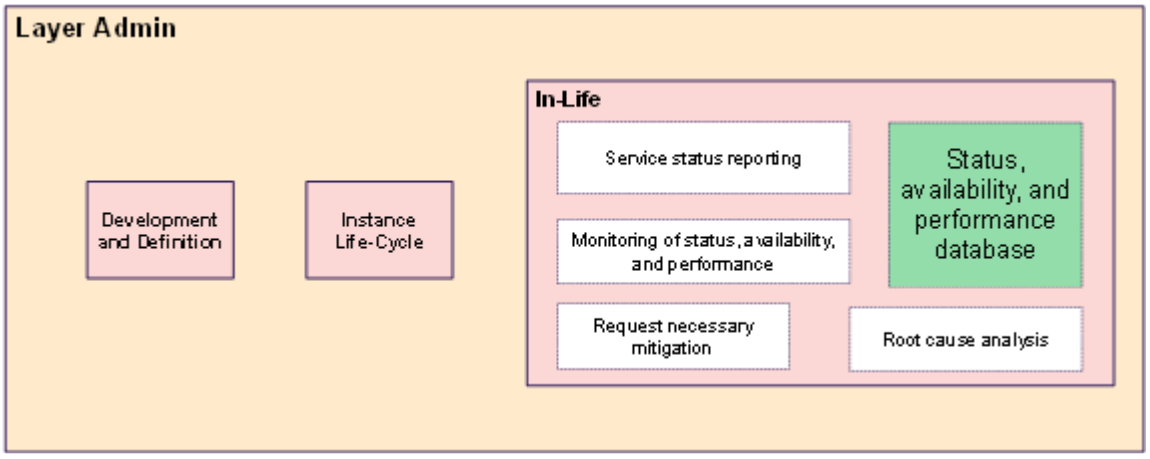


Figure 9: High-level view of the in-life phase

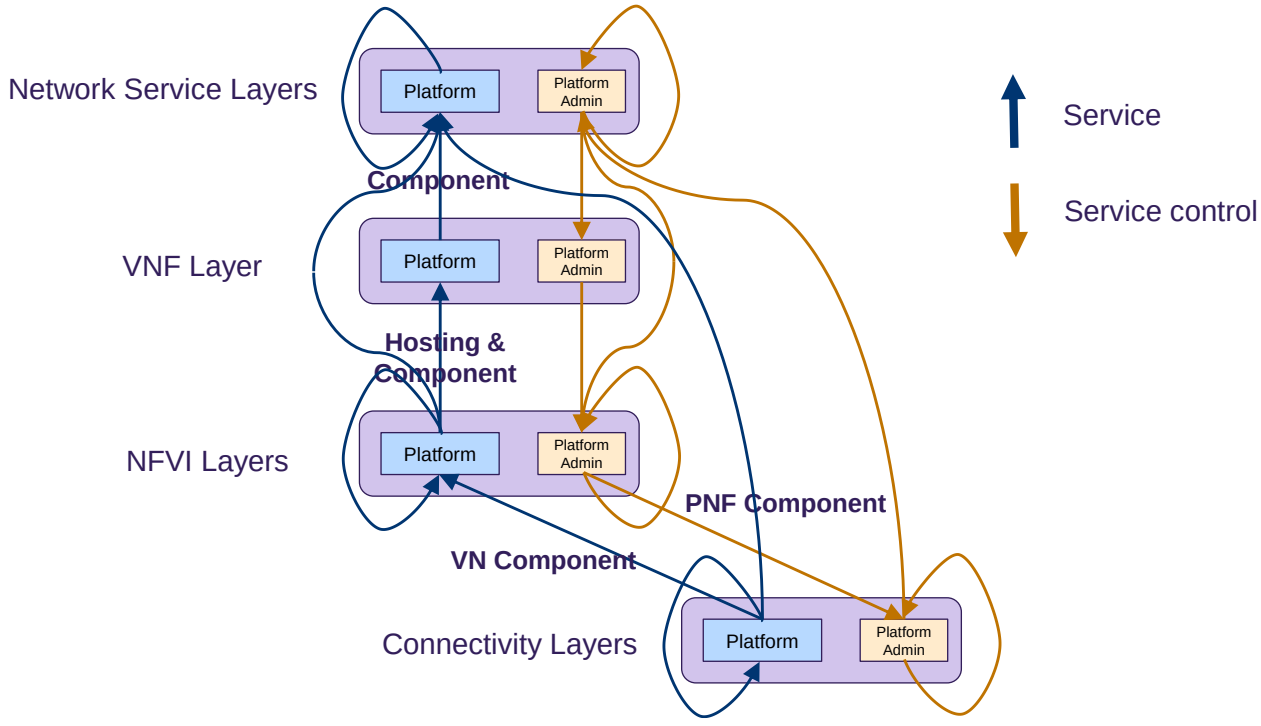


Figure 10: NFV specific example illustrating our recursive, layered approach

Finally in this sub-section, we provide an example in Figure 10 of how our architecture can be seen in some specific logical layers of NFV. The figure illustrates the inclusion of some non-NFV connectivity layers, and also that layering need not be a strict linear line of layers, but can be an interconnected graph of layers. Note also that the NFV Orchestrator is effectively split between the Network Service layer and the NFVI layer (NS and resource orchestrator parts).

- Network Service (NS) layer – this logical layer offers NFV network services. The NS layer uses VNFs from the VNF layer, NSs from other NS layers, and uses virtual networks (VNs) from one or more NFVI layers for external virtual links (eVLs) as resource components. It could also include physical network functions (PNFs) from connectivity layers as resource components in a network service.
    - A network service layer can recursively use another network service layer thus allowing a network service to be included in a wider network service.
- VNF layer – this logical layer offers VNFs as component functions which can be used by NS layers. This VNF layer uses one or more NFVI layers for virtual machine (VM) and virtual network (VN) hosting resource to host VNFCs and also to provide internal virtual links (iVLs).
    - As the definition of a VNF is based on the scope of a deployable VNF as an implementation, the concept of a VNF having a VNF as a component does not arise. The VNF layer is therefore not inherently recursive.
- NFVI layer – this logical layer provides virtual machines (VMs) interconnected by virtual networks (VNs) for the hosting of VNFs and providing the interconnectivity between VNFs. A NFVI layer can use resources from other NFVI layers as well as the resources of connectivity layers.
    - Any specific NFVI layer will have a defined scope (for instance, a network operator's administrative domain or the domain of a specific administration system such as a VIM or a hypervisor).
    - A NFVI layer can recursively use another NFVI layer thus allowing resources of one NFVI layer to be included as part of a wider NFVI.
- Connectivity layer – this layer provides virtual networks (VNs) which provide interconnectivity between VMs, VNFs, PNFs, NSs, depending on the context. It may also provide PNFs direct to a network service layer.
    - A connectivity layer can recursively use another connectivity layer thus allowing resources of one connectivity layer to be included as part of a wider connectivity service.

4 Programming interfaces

Considering the architecture proposed in Figure 2, in this section, we identify the logical entities that are involved in the realization of services. We consider that the architecture in Figure 2 can recursively be applied to a number of abstraction levels (RFB Execution Environments - REEs). We define a generic approach for the identification of the entities involved in the operations of a REE that can be applied at all abstraction levels. At each level, we identify a REE Manager and a REE User. The REE User requests the realization/deployment/execution of a service / service component described using a RDCL script to the REE manager. The REE Manager is in charge of deploying/executing the RDCL script using the resources in its REE. Within a REE, the REE Manager interacts with REE Resources Entities that are needed to support the realization of the RDCL script.

Hence, we can identify two main APIs. The first one is the API used by a REE User that wants to deploy a service or a component into an RFB Execution Environment. We refer to it as the User-Manager API (UM API). The second one is the API used by a REE Manager to interact with the resources in its REE. We refer to it as the Manager-Resource API (MR API). The difference between a UM API and a MR API is blurred, because in some cases a REE Manager can directly interact with a REE User of an underlying REE.

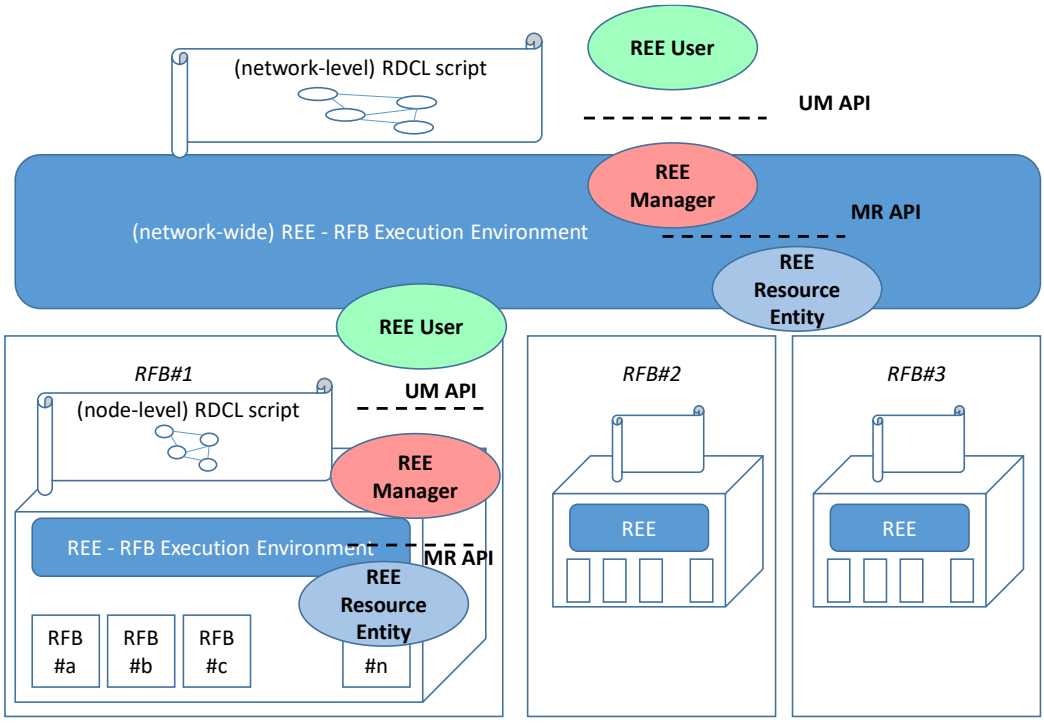


Figure 11: Superfluidity Architecture APIs

Figure 11 illustrates these concepts, showing how the APIs can be mapped to different levels. Note that we refer here to the APIs in general terms, assuming that the information exchange

can be initiated in both directions. The REE user will need to make requests to the REE Manager. The REE Manager may need to notify information to the REE user. The specific details needs to be clarified in the detailed design of the APIs for a given RFB Execution Environment.

The APIs for the different abstraction levels may use different modelling approaches, languages and tools, considering that Superfluidity will not build them from scratch but will rely on existing work. Wherever possible, the project will try to relate the APIs at different abstraction levels and proceed towards a harmonized vision.

4.1 API for collecting performance information from the NFV Infrastructure

The NFVI is responsible for providing the execution environment for RFB's and will have a significant influence on both the performance and behaviour of RFB's depending on the execution environment's level of heterogeneity. Heterogeneity within the NFVI appears in at least three major forms. Firstly, the virtualised infrastructure environment will mostly likely be comprised of different generations and types of X86 CPU's. Additionally, the compute nodes of the NFVI will have different PCIe devices such as accelerators, network cards with different features sets, chipsets, storage types etc. Secondly, the state of the compute nodes will vary significantly, depending on what types of workloads are running and on the number of workload instances. Finally, the physical location of resources, may in some cases be important for some RFB's, where the close proximity to users is required to deliver the best possible user experience. Therefore a detailed description of the characteristics of the workload and a detailed description on the potential execution environments are required. The descriptions will include some static characteristic information and also some contextual information for the current status of the workloads running on the system. These descriptions are needed in order to allow the orchestration framework to make either informed initial placement, migration or scaling decisions. According to our architectural model, the ability to perform actions by the orchestration framework corresponds to a Manager-Resource API.

Information about the physical and virtualised resources may be available from the Virtualised Infrastructure Manager (VIM) used to manage the infrastructure resource. For example, in an OpenStack cloud environment infrastructure information is available from the various service databases such as NOVA and Neutron, via standardised REST API's. However, the granularity of information available may be insufficient in the cases where the information is insufficiently modelled depending on the explicit needs of the Orchestrator and algorithms used to make RFB placement decisions. It is therefore important to match the information that can be modelled with the actions that can be performed. The information available is also dependent on the

implementation and may vary in other VIM environments such as Mesos[1] or Kubernetes[2] and this can also be significant. In an SDN environment, network information from the SDN controller also needs to be considered in order to complete the network topology picture. For example, OpenDaylight[3] exposes various types of topology information via REST based API's.

The mixture of different types of information and orchestration system is further complicated in multi-VM environments where commonality (type, format, quantity) in resource information across in the various VIMs does not exist (no consistent data model). It is therefore apparent that Centralised Infrastructure Repository (CIR) is necessary. This CIR will collate the infrastructure information from the various VIMs and expose the information via a standard set of REST API's. Having the APIs defined will be advantageous as it simplifies the required interactions between the orchestrator and NFVI. The CIR can also be used to store additional attributes and meta-data, e.g. the physical location of resources could use some sort of geo tagging or other form location specific identifier that is consumable at an orchestration level. Collectors designed for each VIM environment can ensure that the information being retrieved from each environment is consistent and concurrent. The information in the repository could be supplemented with additional information, e.g. network information from SDN controllers. Secondly, this approach also provides greater flexibility in representing the topology of workloads and their allocated resources (physical and virtual).

Telemetry plays a key role in monitoring the state and utilisation of the infrastructure resources within the NFVI. Different levels of monitoring of resources may be available as a service within varying VIM environments. For example, in OpenStack the Ceilometer [33] service provides data on the utilisation of the physical and virtual resources that comprise clouds deployed via standard API. However, the granularity of the available monitoring information is limited to metric such as CPU and memory utilisation, which may or may not be sufficient depending on the type of workload to be monitored. A standalone metrics platform, which can support fine-grained monitoring of resources across different virtualised (VMs, containers, unikernels) and non-virtualised environments is necessary. Collection of network metrics from SDN controllers, as expressed earlier, can be used to determine which flows are connected to which ports on either physical or virtual switches. The monitoring information collected can be written to various common databases such as MySQL, MariaDB, OpenTSDB. These can then be exposed at an orchestration level either through predefined queries or preferentially through an API such as REST. Using APIs enables abstraction from the underlying telemetry collection mechanisms,

[1] http://mesos.apache.org/
[2] http://kubernetes.io/
[3] https://www.opendaylight.org/

which may change over time. The metrics collected may also processed in some predefined manner and results stored in the database repository. For example, the CIR could be interrogated for SLA or KPI violations and then used to trigger corrective actions by the Orchestrator when necessary.

4.2 API toward the 3rd parties that want to deploy applications over Superfluidity

Deployment of applications on the Superfluidity architecture must utilise a well-defined API. Associating this to Figure 11 (described in the introduction of Section 4), the interface of interest corresponds to the top-level REE User-Manager (UM) API. Moreover, if we map this to the NFV Architectural Framework [3], the Manager of interest becomes the NFV Orchestrator (NFVO). Consequently, the UM API under consideration shall build on the functions and reference points of the NFVO, augmenting them to expose capabilities unique to Superfluidity.

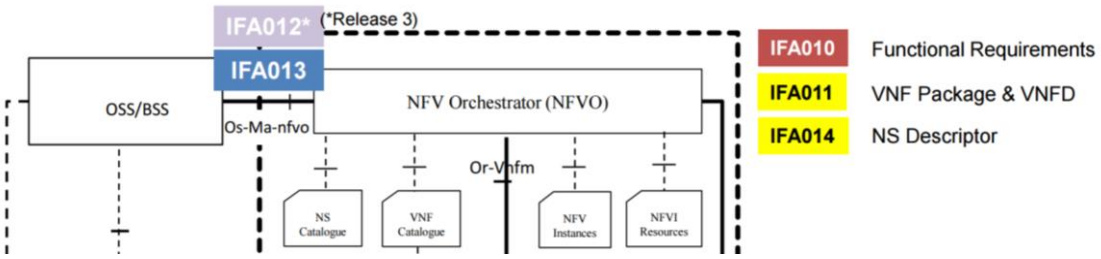


Figure 12: NFV Release 2 Specifications Relevant to the NFV Orchestrator

The only external reference point of NFVO, in the scope of the NFV, ISG is the Os-Ma-Nfvo [40]. This makes the functionality of NFVO available to OSS/BSS systems, implementing the relevant requirements of NFV Management and Orchestration (MANO) [38] to manage:

- Network service descriptors: on-board, enable, disable, update, delete, query, subscribe/notify (most of these functions also apply to physical network service descriptors)
- Network service life-cycle: create, instantiate, scale, update, query, terminate, delete, heal, get status, subscribe/notify
- Network service performance: create job, delete job, subscribe/notify, query job, create threshold, delete thresholds, query threshold
- Network service faults: subscribe/notify, get alarm list
- Virtual network function packages: on-board, enable, disable, delete, query, subscribe/notify, fetch (on-boarded) package, fetch package artefacts, abort package deletion

The top-level User-Manager API of Superfluidity must implement most of the above functions. Most importantly, some of them will be enhanced to leverage the innovations of Superfluidity. However, given that the Os-Ma-Nfvo specification was just published, and the corresponding protocol is still in the works, there is no NFVO that implements this specification in a standards-compliant way. Therefore, these functions will be implemented by the toolchain of the NFVOs of choice.

The functions exposed by the Os-Ma-Nfvo reference point make use of a multitude of elements from the NFV Information Model [39][4]:

- NS metadata, deployment flavour, monitoring information and lifecycle management scripts
- Service Access Point Descriptor (SAPD)
- Virtual Link (VL) Descriptor (VLD) and deployment flavour
- Virtual Network Function (VNFD)
- VNF Forwarding Graph Descriptor (VNFFGD)
- Physical Network Function Descriptor (PNFD)
- Network Forwarding Path Descriptor (NFPD)

In addition to managing Network Service Templates [41], stored in the NFV Service Catalogue, NFVO is also responsible for managing VNF Packages, stored in the VNF Catalogue. Details are published in the VNF Packaging Specification [42], but at a high level they contain:

- TVNF descriptor that defines metadata for package on-boarding and VNF management
- Software images needed to run the VNF
- Optional additional files to manage the VNF (e.g. scripts, vendor-specific files etc.)

Based on our detailed review of the NS template, VNF descriptor and VNF packaging specification docs, the information model entities support decomposition up to the VNF component (VNFC).

However, as we have described previously (see Section 3.4.1 and Figure 3), the RFB model proposed by Superfluidity is more general than the NFV model, since it can recursively support an arbitrary depth of sub-classing and decomposition. As a consequence, we will have to amend the NFV information model to support the RFB model of Superfluidity. Moreover, similar changes will have to be introduced to the MANO reference points (APIs) (or the corresponding toolchain of the NFVO).

Something else that is important to consider, in the context of specifying the API to third-parties, is the concrete textual representation of the information model elements. Practically speaking, this will specify the syntax of the files that will represent the network service templates, VNF descriptors, VNF forwarding graph descriptors, etc.

Based on the current practice of NFVOs and VNFMs, which seem to also be adopted by the ETSI standardisation efforts [43], the elements of the NFV Information Model are represented using a YAML syntax, and increasingly compliant with the OASIS TOSCA Simple Profile for NFV [44].

[4] Please note that the complete NFV Information Model has not been officially released yet, but the subset used by Os-Ma-Nfvo is published as part of the reference point specification [40], one of the NFV Release 2 documents [37]

Based on the current discussions, and under the assumption that the NFVOs of choice are aligned with this decision, Superfluidity will adopt YAML-based file formats, and, to the extent possible, TOSCA-based. But given the abovementioned requirement for a recursive composition of RFBs, the top-level RDCL is envisioned to utilize the NEMO (Network MOdeling) language instead (please see Section 4.6).

Finally, the descriptors of RFBs, for which formal analysis of correct behaviour is available, can be extended to include the relevant representation. In that context, we aim to extend them to optionally embed the domain-specific language adopted by Superfluidity, SEFL (please see Section 4.5). This will facilitate automatic execution of the symbolic execution engine, SymNet [114], for the subset of NS/VNF life-cycle transitions that would require (re-)verifying the composition of RFBs.

4.3 Optimal function allocation API (for packet processing services in programmable nodes)

The *optimal function allocation problem* consists in mapping a set of network services (decomposed into RFBs) to the underlying, available hardware block implementations in the RFB execution environment (REE). It aims at minimizing resource usage while meeting individual SLAs.

Solving the optimal function allocation problem is still a work-in-progress, and its API is by no mean definitive (final API will be in deliverable D5.1). However, we expect its API to comply with the following.

At each level (as explained in the introduction of section 4 and illustrated in Figure 11), we need a way to efficiently map the user desired service onto the available underlying REE resources. To this aim, the optimization system will need as input:

- Information about the user desired service, describing:
  - The semantics of the network function to be deployed, most likely under the form of a RDCL script. Not all semantic information is useful for allocation purpose, but some might be (*e.g.* can a pipeline be safely parallelized and under which conditions?).
  - The required performance (*e.g.* in terms of throughput, delay, *etc.*) for the whole service, and/or for specific RFBs;
  - Information about the expected workload, if relevant and available.
- Information about the implementations available for the RFBs used by the user on the resource entities available in the REE. This information should comprise both:
  - Which implementations are available, *i.e.* which REE resource entities can be used to deploy a given RFB (or part of the RFB, if it is to be decomposed into sub-RFBs);
  - How the performance of a given implementation can be estimated, according to an expected workload and given current REE conditions. This information might be

provided, but we will investigate if it might be learned automatically by the system using unsupervised, on-line machine learning techniques.

This information could be pre-computed and is not required to be part of the API for every allocation (but the API must allow to update it, *e.g.* to make the system aware of new RFB implementations, or update the performance models).

- Information about the current state of the REE:
  - Available resource entities;
  - Statistics about resource and network use;
  - Currently allocated functions (including their expected performance requirements).

  This information is to be gathered from the VIM and other sources as described in section 4.1. For the currently allocated functions, the information can either be kept in the internal state of the function allocation module, or passed around through the API (*i.e.* through function arguments and return values).

As output, the allocation functions will give:

- A list of REE resource entities to use to deploy the user service;
- Configurations for those resource entities, in a form suitable for easy deployment (see sections 4.4 and 5);
- Migration instructions, in the case where some currently used functions should be migrated from some resource entities to other resource entities.

For example, a network-wide RDCL script could lead to the selection of a number of available nodes, provide node-level RDCL scripts for them, and maybe ask for some functions to be migrated from one node to another.

4.4 Configuration/deployment API for OpenStack VIM

NFV requires the support of multiple domains. The set of domains manifests an abstraction chain where on one end stands the most abstract business and service orchestration, and on the other end stands the concrete low level Virtual Infrastructure Manager (VIM), that implements the low level RFBs execution environment.

Each domain in the abstraction chain requires its own unique support in the form of a designated Domain Specific Language (DSL). Generally, the DSL shall support several language features and properties, that are required for managing NFV resources:

- Event notification – enabling flexible triggering of procedures
- Workflow engine – enables implementing procedure flows
- Catalog driven – versatile and robust catalogue that implements lower level RFBs
- Hierarchical aggregation – each layer aggregates more information and functionalities
- Standard APIs – with a domain specific information

- End-to-end information view – must be available on demand

TOSCA (Topology and Orchestration Specification for Cloud Applications, [44]) is an OASIS (Organization for the Advancement of Structured Information Standards) language to describe a topology of cloud based services, their components, relationships, connectivity, and the processes that manage them. TOSCA's goals (and advantages) are as follows:

- Enable portability and automated management across cloud providers, regardless of the underlying cloud platform or infrastructure.
- Enable the interoperable description of application and infrastructure cloud services
- Policies for distributed placement and application lifecycle operations
- Extensible LCM (Life Cycle Management) event definitions

TOSCA aims at interoperable description of application and infrastructure cloud services, the relationships between parts of the service, and the operational behaviour of these services (e.g., deploy, patch, shutdown). TOSCA also aims to make it possible for higher-level operational behaviour to be associated with cloud infrastructure management. Accordingly, TOSCA satisfies the required properties for a language suitable for NFV, particularly, TOSCA's, simple profile for NFV. However, OpenStack VIM requires a refined and concrete DSL that is aware of native resources that are relevant in the context of OpenStack. Therefore, TOSCA is suitable for the higher layer (abstract) application domains, but falls short in low level (concrete) virtualized resources domains, such as OpenStack VIM, due to several reasons:

- Weak notion of resources – no clear mapping to OpenStack, i.e. resources defined explicitly, rather than via OpenStack flavours
- No support for general purpose workflow, left to actual implementation
- Limited commissioning support, no ability to statically invoke external scripts and no inventory propagation

We conclude that TOSCA is not sufficient for OpenStack VIM, since: ***(i) deploying VNFs is requires detailed resources specification, while TOSCA is not suited for low level specification; (ii) it requires costly adoption of new cloud infrastructure capabilities into the domain specific language.*** Accordingly, we take two measures to enrich TOSCA for VNF descriptors: use TOSCA in conjunction with HEAT templates; extend TOSCA scope (e.g., for networking support) to support VNFs description.

In addition, it is clear that TOSCA is suitable for network service descriptors since it is: ***(i) aiming at high-level DSL that is suitable for NSs; (ii) cloud agnostic; (iii) open-source project amenable to changes; (iv) following ETSI NFV MANO stage1 recommendation.***

OpenStack VIM requires a refined and concreate domain specific language that is aware of the native resources that are relevant in the context of OpenStack. Thus we conclude that the API for OpenStack VIM is the set of native LCM tools that OpenStack has to offer in order to manage its infrastructure, namely:

- OpenStack/HEAT – an orchestration engine to launch multiple composite cloud applications. Heat provides both an OpenStack-native ReST API and a CloudFormation-compatible Query API.
- OpenStack/Mistral – a workflow service that implements multiple distinct interconnected steps that need to be executed in a particular order in a distributed environment
- OpenStack/Murano – an application catalog to OpenStack, implementing categorized cloud-ready applications.

This set of OpenStack projects satisfy the required properties for a language suited for NFV (following the set of properties discussed above for a language suitable for NFV), and has appealing properties of: ***(1) rich set of features; (2) constantly being evolved by community; (3) better inter-operability.***

4.5 An API for static verification of packet processing services

Software verification is not possible without understanding the properties the software product should meet. One of the techniques of specifying these properties is to build a specification of the program through *program annotations*. Annotation means that programs are augmented with conditions imposed on variables, usually expressed using computational logic. These conditions are usually placed prior and posterior to function calls, denoting conditions imposed on the function input parameters (preconditions) and the values returned by the function (postconditions). The verification process is carried out during runtime - whenever a call to a function that carries annotations is encountered the preconditions are checked, and then, upon return from the function, the postconditions on the result are checked. A program is thus considered correct with respect to a set of annotations if at any time during its execution, all conditions hold true. There are two approaches to prove these conditions indeed true: static verification (prior to deployment) and/or runtime checking.

Network processing pipelines are composed of *network functions* that operate at a *per-packet* basis. Specifying correct behaviour of such a network function means describing its processing and adding pre and postconditions on the fields of the *packet header* at various points on the packet's path. To express both the network processing and the annotations, we need a language that should: 1) be expressive enough to cope with protocols of different layers in the network stack 2) allow its users to impose conditions that amount to the complete specification.

We use the SEFL (Symbolic Execution Friendly Language), developed by PUB [114], for both expressing the processing of network functions and for expressing annotations. In SEFL any header field can be accessed using either the numeric offset of the header inside the packet or using a string alias identifying the header (IPSrc, MACDst etc). To specify conditions on fields SEFL provides support for the usual boolean operators (and, or, not etc), arithmetic comparison operators (>, <, !=, == etc). Our conclusion is that SEFL allows a user to specify correctness

conditions of a network function. SEFL specifications can be verified both with runtime checking, but also via static checking using symbolic execution. Below we provide a brief overview of the SEFL Syntax.

SEFL description	*Instruction Description*
Allocate(v[,s,m])	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.
Deallocate(v[,s])	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.
Assign(v,e)	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)	Creates tag t and sets its value e, where e must evaluate to a concrete integer value.
DestroyTag(t)	Destroys tag t.
Constrain(v,cond)	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.
Fail(msg)	
If (cond,i1,i2)	Two execution paths are created; the first one executes i1 as long as cond holds. the second path executes i2 as long as the negation of cond holds.
For (v in regex,instr)	Binds v to all map keys that match regex and executes instruction instr for each match.
Forward(i)	Forwards this packet to output port i.
Fork(i1,i2,i3,...)	Duplicates the packet and forwards a copy to each output port i1, i2, ...
InstructionBlock(i,...)	Groups a number of instructions that are executed in order.
NoOp	Does nothing.

Symnet verification needs a model of the data plane in SEFL format and the packet that needs to be inserted. In more detail, the input to Symnet is a folder that contains all the data plane elements, as follows:

- One SEFL file for each network element, where each port can have associated instructions.
- A file containing links between the ports of different elements.
- A list of ports where packets should be injected
- The type of packet to be injected, including the headers included and their values (concrete or symbolic)

The output of Symnet is a list of packets resulting after the packets are injected, in json format. Each packet has the list of network hops together with constraints and concrete values where this is the case (i.e. for non-symbolic fields). This output can be used to check reachability, isolation, loops as well as see how packets are changed.

To allow the operator to easily specify and check network-wide properties, we are also developing a high-level specification language based on CTL (Computation Tree Logic). This will act like human-readable version of the properties the network should obey, and will be used to drive the symbolic execution engine. The difficulty here is to reduce the complexity of symbolic execution while ensuring that the desired properties hold.

4.6 NEMO: a recursive language for VNF combination

Currently, there is a lot of activity going on to use NFV in the network. From the point of view of the orchestration, Virtual Network Functions are blocks that are deployed in the infrastructure as independent units. They provide for one layer of components (VNF components – VNFCs), i.e. a set of VNFCs accessible to a VNF provider that can be composed into VNFs. However, there is no simple way to use existing VNFs as components in VNFs with a higher degree of complexity. In addition, VNF Descriptors (VNFDs) used in different open source MANO frameworks are YAML-based files, which despite being human readable, are not easy to understand.

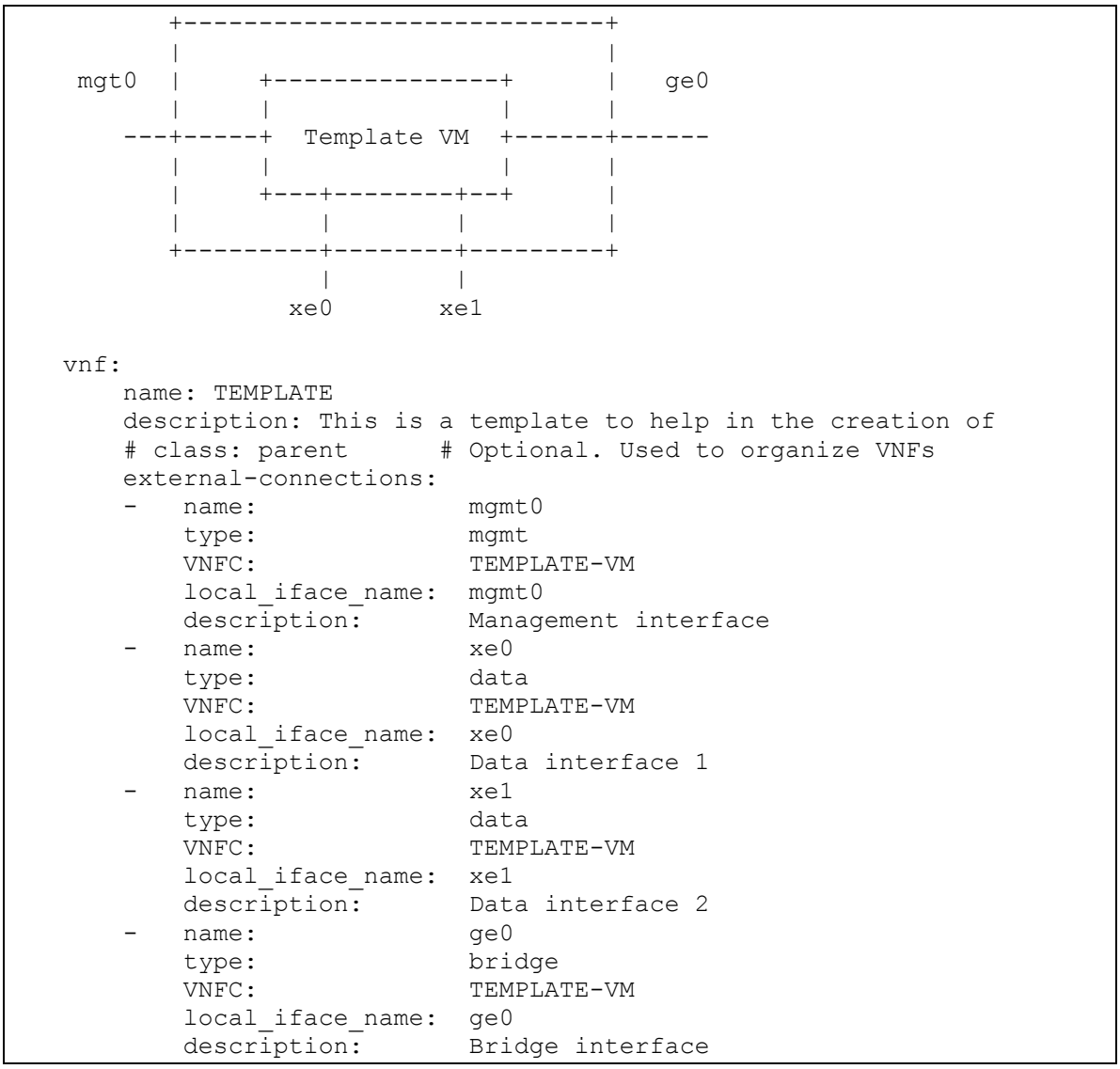
On the other hand, there has been recently an attempt to work on a modelling language for networks (NEMO). This language is human-readable and provides a NodeModel construct to describe nodes that supports recursion. In this draft, we propose an extension to NEMO to make it interact with VNFDs supported by a NFV MANO framework. This extension creates a new language for VNFDs that is recursive, allowing VNFs to be created based on the definitions of existing VNFs.

Hereafter, we use OpenMANO and OSM descriptor references as an example for the lowest level descriptors that are imported into NEMO. Conceptually, other descriptor formats like TOSCA can also be used at this level.

4.6.1 Virtual network function descriptors tutorial

Virtual network function descriptors (VNFDs) are used in the Management and orchestration (MANO) framework of the ETSI NFV to achieve the optimal deployment of virtual network functions (VNFs). The Virtual Infrastructure Manager (VIM) uses this information to place the functions optimally. VNFDs include information of the components of a specific VNF and their interconnection to implement the VNF, in the form of a forwarding graph. In addition to the forwarding graph, the VNFD includes information regarding the interfaces of the VNF. These are then used to connect the VNF to either physical or logical interfaces once it is deployed.

There are different MANO frameworks available. For this draft, we will concentrate on the example of OpenMANO [26], which uses YAML [27]. Taking the example from the (public) OpenMANO github repository, we can easily identify the virtual interfaces of the sample VNFs in their descriptors, see Figure 13:




```
vnf:
    name: TEMPLATE
    description: This is a template to help in the creation of
    # class: parent      # Optional. Used to organize VNFs
    external-connections:
    -   name:              mgmt0
        type:              mgmt
        VNFC:              TEMPLATE-VM
        local_iface_name:  mgmt0
        description:       Management interface
    -   name:              xe0
        type:              data
        VNFC:              TEMPLATE-VM
        local_iface_name:  xe0
        description:       Data interface 1
    -   name:              xe1
        type:              data
        VNFC:              TEMPLATE-VM
        local_iface_name:  xe1
        description:       Data interface 2
    -   name:              ge0
        type:              bridge
        VNFC:              TEMPLATE-VM
        local_iface_name:  ge0
        description:       Bridge interface
```

Figure 13: Sample VNF and descriptor (source: OpenMANO github)

4.6.2 NEMO tutorial

The Network Modeling (NEMO) language is described in [28]. It provides a simple way of describing network scenarios. The language is based on a two-stage process. In the first stage, models for nodes, links and other entities are defined. In the second stage, the defined models are instantiated. The NEMO language also allows for behavioural descriptions. A variant of the NEMO language is used in the OpenDaylight NEMO northbound API [29].

NEMO allows to define NodeModels, which are then instantiated in the infrastructure. NodeModels are recursive and can be built with basic node types or with previously defined NodeModels. An example for a script defining a NodeModel is shown below:

```
CREATE NodeModel dmz
  Property string: location-fw, string: location-n2,
    string: ipprefix, string: gatewayip, string: srcip,
    string: subnodes-n2;
  Node fw1
    Type fw
    Property location: location-fw,
      operating-mode: layer3;
```

Figure 14: Creating a NodeModel in NEMO

4.6.3 Proposed enhancements to NEMO

In order to integrate VNFDs into NEMO, we need to take into account two specifics of VNFDs, which cannot be expressed in the current syntax/semantics of NEMO language. Firstly, we need a way to reference the file that holds the VNFD provided by the VNF developer. This will normally be a universal resource identifier (URI). Additionally, we need to extend the information model underlying the NEMO language to become aware of the virtual network interfaces.

Referencing VNFDs in a NodeModel

As explained in the introduction, in order to easily integrate VNFDs into the NEMO language we need to reference the VNFD as a Universal Resource Identifier (URI) as defined in RFC 3986 [30]. To this avail, we define a new element in the NodeModel to import the VNFD:

```
CREATE NodeModel NAME <node_model_name>
  IMPORT VNFD FROM <vnfd_uri>
```

Referencing the network interfaces of a VNF in a NodeModel

As shown in Figure 13, VNFDs include an exhaustive list of interfaces, including the interfaces to the management network. However, since these interfaces may not be significant for specific network scenarios and since interface names in the VNFD may not be adequate in NEMO, we propose to define a new element in the node model, namely the ConnectionPoint.

```
CREATE NodeModel NAME <node_model_name>
  DEFINE ConnectionPoint <cp_name> FROM VNFD:<iface_from_vnfd>
```

An example

Once these two elements (VNFDs and Connection Points) are included in the NEMO language, it is possible to recursively define NodeModel elements that use VNFDs in the lowest level of recursion. Firstly, we create NodeModels from VNFDs:

```
CREATE NodeModel NAME SampleVNF
    IMPORT VNFD from https://github.com/nfvlabs/openmano.git
/openmano/vnfs/examples/dataplaneVNF1.yaml
    DEFINE ConnectionPoint data_inside as VNFD:ge0
    DEFINE ConnectionPoint data_outside as VNFD:ge1
```

Figure 15: Import from a sample VNFD from the OpenMANO repository

Then we can reuse these NodeModels recursively to create complex NodeModels:

```
CREATE NodeModel NAME ComplexNode
    Node InputVNF TYPE SampleVNF
    Node OutputVNF TYPE ShaperVNF
    DEFINE ConnectionPoint input
    DEFINE ConnectionPoint output
    CONNECTION input_connection FROM input TO InputVNF:data_inside
        TYPE p2p
    CONNECTION output_connection FROM output TO ShaperVNF:wan
        TYPE p2p
    CONNECTION internal FROM InputVNF:data_outside TO ShaperVNF:lan
        TYPE p2p
```

Figure 16: Create a composed NodeModel

This NodeModel definition creates a composed model linking the SampleVNF created from the VNFD to a hypothetical ShaperVNF defined elsewhere. This definition can be represented graphically as follows:

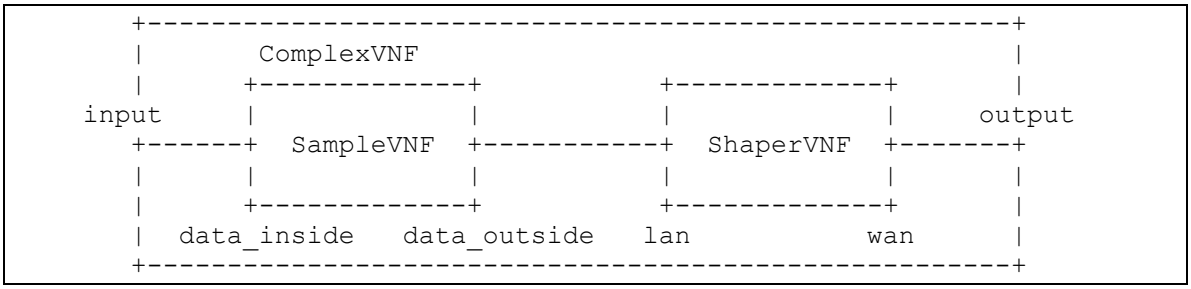


Figure 17: Representation of the composed element

In ETSI NFV, a network service is described by one or more VNFs that are connected through one or more network VNFFGs. This is no more than what is defined in the composed NodeModel

shown in Figure 17. By using NEMO, we provide a simple way to define VNF forwarding graphs (VNF-FGs) in network service descriptors in a recursive way.

5 Instantiation of the RFB concept in different environments

5.1 Management and Orchestration in a "traditional" NFV Infrastructure

The cloud infrastructure is the basis of the emerging cloud technology. It allows the creation of virtual entities, with compute, storage and networking resources, appearing as if they were physical. The use of hypervisors (e.g. KVM) is still the most common virtualization technology; however, container-based technologies (e.g. Docker [31]) are getting momentum. A variant of the hypervisor technology is represented by the Unikernel approach. In this approach the traditional "heavy-weight" VMs running a full OS (Linux, Windows) are replaced by tiny specialized VMs running customized Operating System obtained by cutting out unused functionality such as unused drivers, services or even the memory management system, which is common in modern OS's but pre-computed for unikernels at compile-time (for further information see Section 5.5). The ETSI NFV architecture globally refers to these execution environments (hypervisors, containers) as NFV Infrastructure (NFVI).

In the Superfluidity architecture the NFVI infrastructure is used to support the Reusable Functional Blocks (RFBs), which together compose the workloads and services that are derived from the use-cases. Two types of RFBs are considered by Superfluidity: network functions (e.g. eNB, EPC) and application oriented RFBs (e.g. Augmented Reality, Video Analytics). A goal of the Superfluidity project is to manage the two types of RFBs as much as possible in a unified way, allowing to share physical and logical resources in the most efficient way.

Independently of the virtualization technology in use, the NFVI needs to be managed with a high degree of agility, in order to allow rapid changes in the virtual entities, taking advantage of the virtualization paradigm. At the same time, as services are often very complex an orchestration function must build coherent end2end services using virtual entities. Although significantly different, management and orchestration functions are referred by ETSI NFV together as the MANO block. There are many ways to organize the MANO macro block, along different administrative domain and locations.

5.1.1 NFVIs and Services

Today, different cloud infrastructures (NFVIs) are usually devoted to run specific services (see Figure 18-a). However, the utilization of a common NFVI for multiple services (see Figure 18-b) can provide a more efficient use of resources, taking advantage of synergies and reducing costs. The reusability of RFBs across different services is also facilitated by the utilization of a common NFVI. The Superfluidity vision targets the unified solution, with a single enhanced NFVI infrastructure capable of executing an inclusive set of RFBs. On the other hand, designing and building a holistic platform capable to deal with the heterogeneity of the infrastructure is beyond

the reach of the project. Therefore, a pragmatic approach will be used and the architecture needs to support the coexistence and combination of different NFVIs.

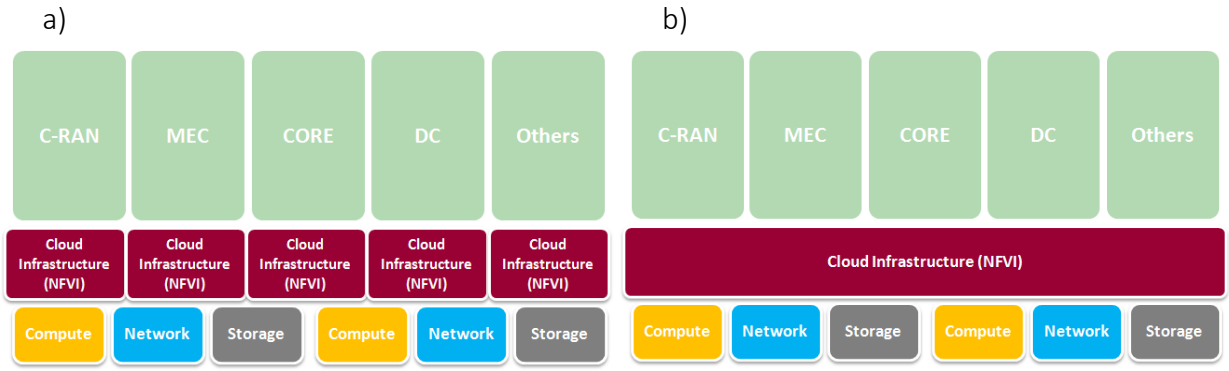


Figure 18: NFVIs per Service.

5.1.2 VIMs and NFVIs

A Virtualized Infrastructure Manager (VIM) function is responsible for managing the cloud infrastructure (NFVI). The VIM provides and manages the virtualized resources (e.g. VMs, containers) that support the execution of RFBs. In case more than one NFVI will need to be integrated, a single VIM can manage an NFVI, which is probably a simpler approach than having (see Figure 19) a single centralized VIM can also manage all NFVIs (see Figure 20). This latter option can face some limitations, depending on the heterogeneity and size of the infrastructure. For this reason, a hybrid approach can be also a good compromise solution (see Figure 21). The use of a single VIM for different NFVIs provides a good opportunity to present a unified environment for the execution of RFBs to the orchestrator elements.

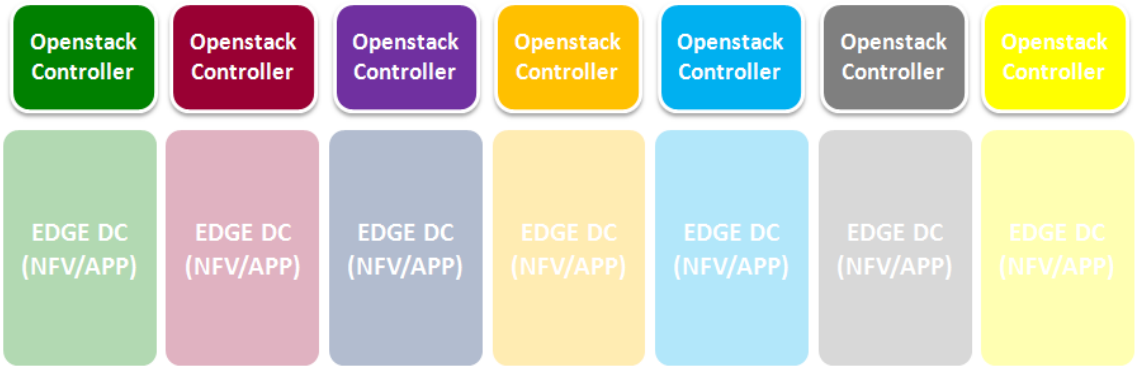


Figure 19: VIMs per NFVI.

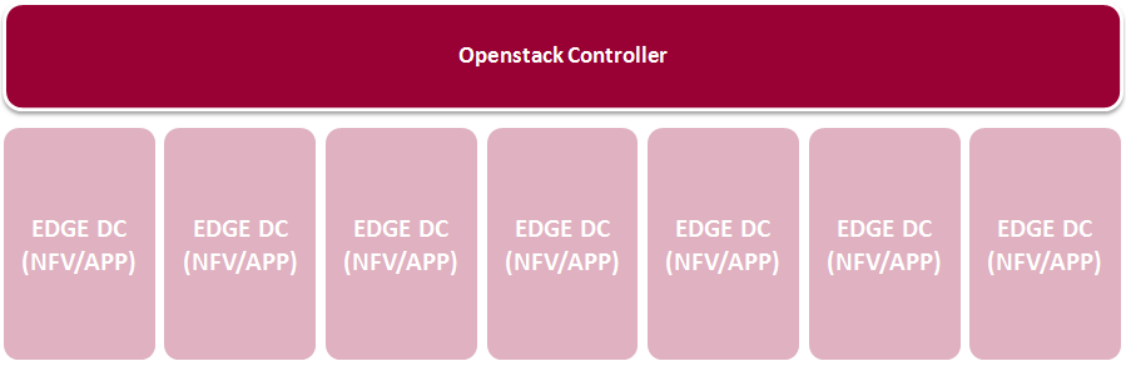


Figure 20: Single VIM for all NFVIs.

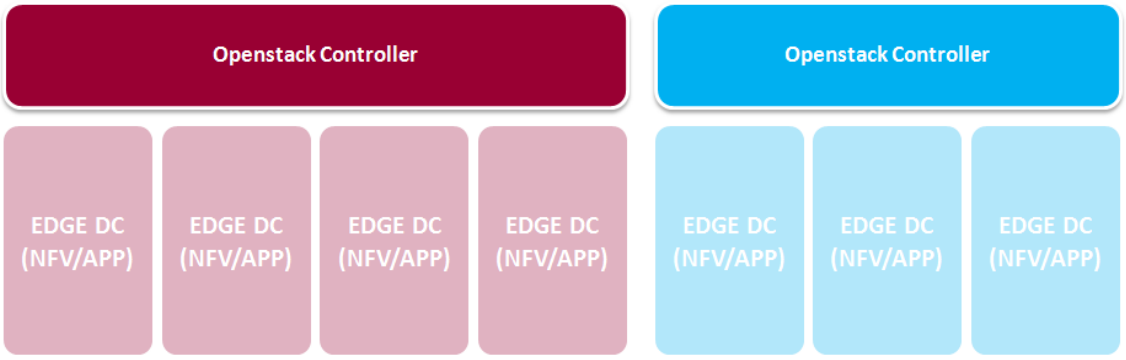


Figure 21: Hybrid; multiple NFVIs per VIM.

5.1.3 Orchestration and Services

The orchestration of complex services can be performed using different approaches. One solution is to have a top-level Orchestrator, capable of managing all the services in all locations (see Figure 22). However, this is, in the view of the Superfluidity consortium, unlikely given that the orchestration frameworks are built to be very specific and, in the real world, many service providers will bring their own orchestration framework. For this reason, a more reasonable approach should be the use of multiple Orchestrators, one per service (See Figure 23).

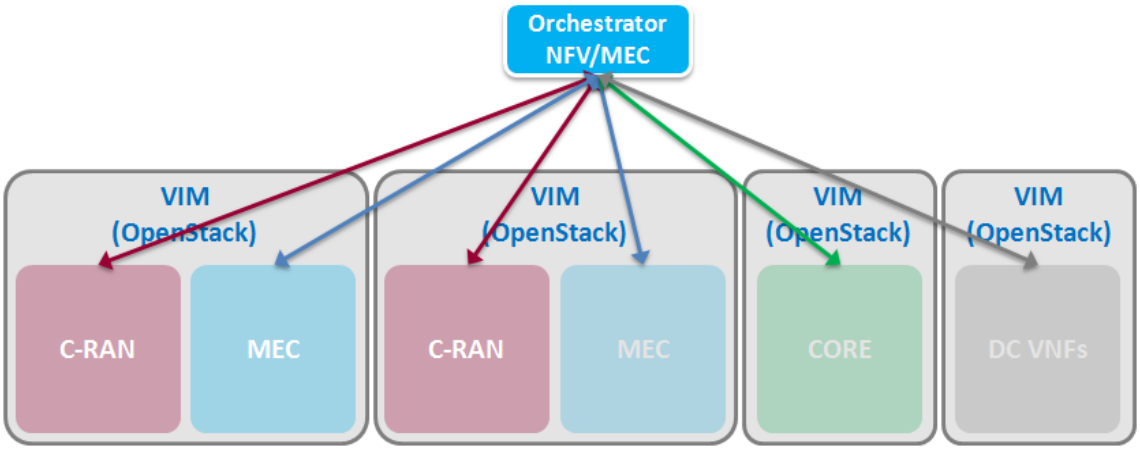


Figure 22: One generic Orchestrator for all Services and locations.

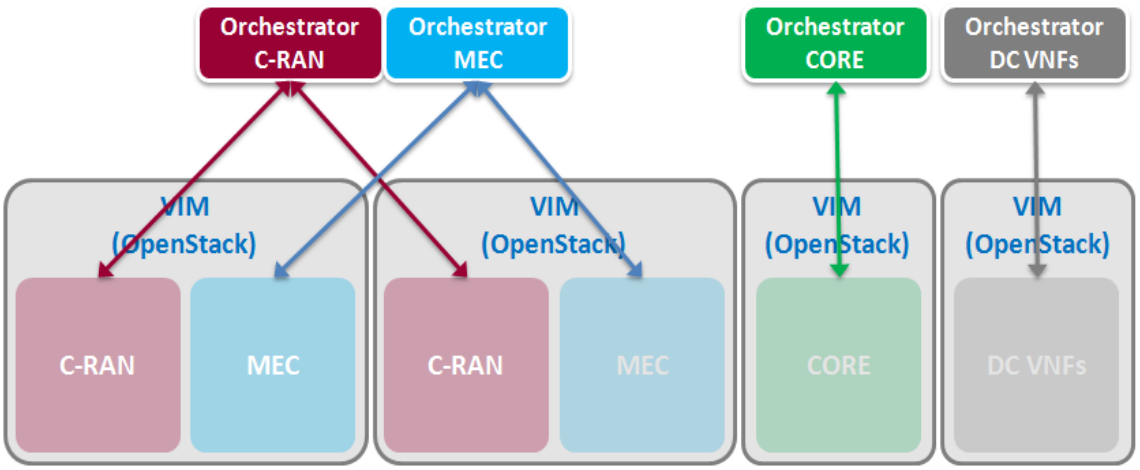


Figure 23: One specialized Orchestrator per Service (for multiple locations).

5.1.4 Orchestration Integration

In case where multiple service Orchestrators exist, one per service, there is a need for some integration between them and, eventually, an additional (top) level of Orchestration. Figure 24 depicts three integration models: (1) in the North-South integration, all service Orchestrators communicate only with a top-level Orchestrator, which simplifies integration as service Orchestrators only need to integrate with the top-level one; (2) in the East-West integration, service Orchestrators communicate with each other, making integration more complex. There is also a hybrid approach, which comprises both models altogether (1+2).

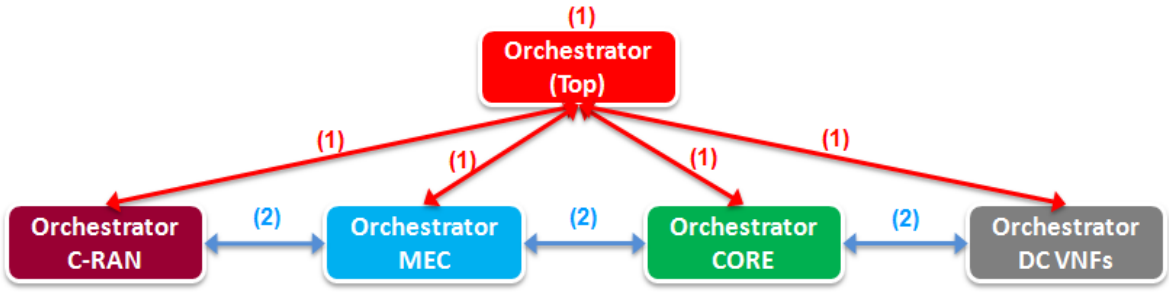


Figure 24: North-South Integration(1), East-West Integration(1) and Hybrid Integration (1 and 2).

5.1.5 Mapping onto the Superfluidity architecture

With reference to the Superfluidity architecture and APIs depicted in Figure 2 and Figure 11 (copied below for ease of reading), the NFVI (as a single entity or as a set of NFVIs) represents a REE (RFB Execution Environment). The Orchestrator, in this model, is mapped in a REE manager, offering services to a REE user. An Orchestrator uses the VIMs to manage the resources in its REE, so that the Orchestrator – VIM interface can be mapped in a Manager Resource API. With a different modelling, a VIM can be seen as a manager of a separated RFB Execution Environment, offering services to the Orchestrator. In this case the Orchestrator – VIM interface can be mapped to a User Manager API.

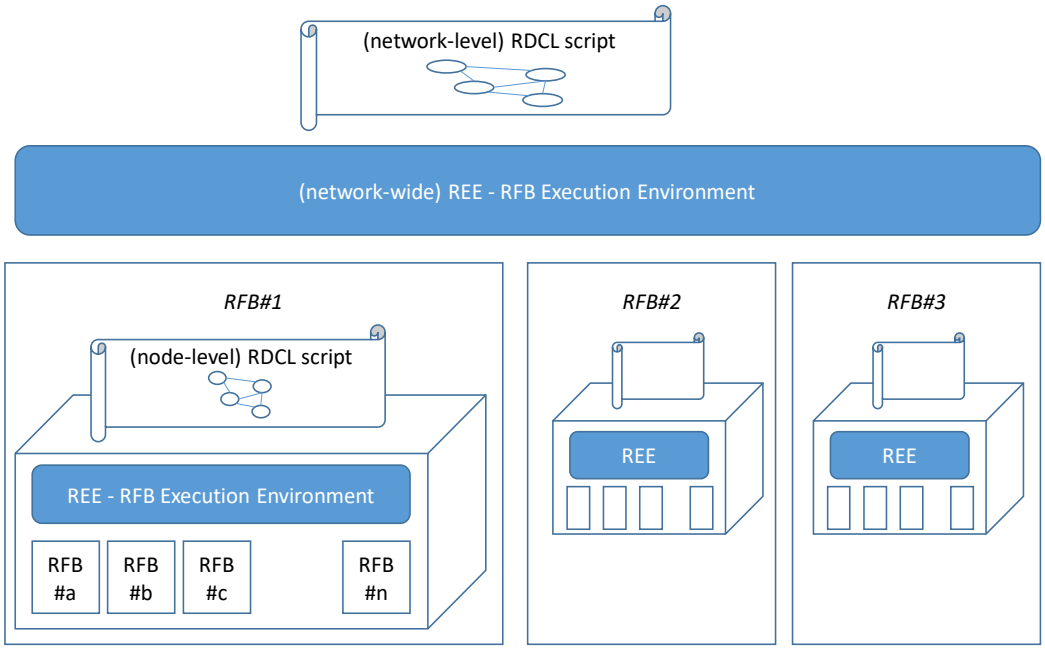


Figure 25: Superfluidity architecture (copy of Figure 2)

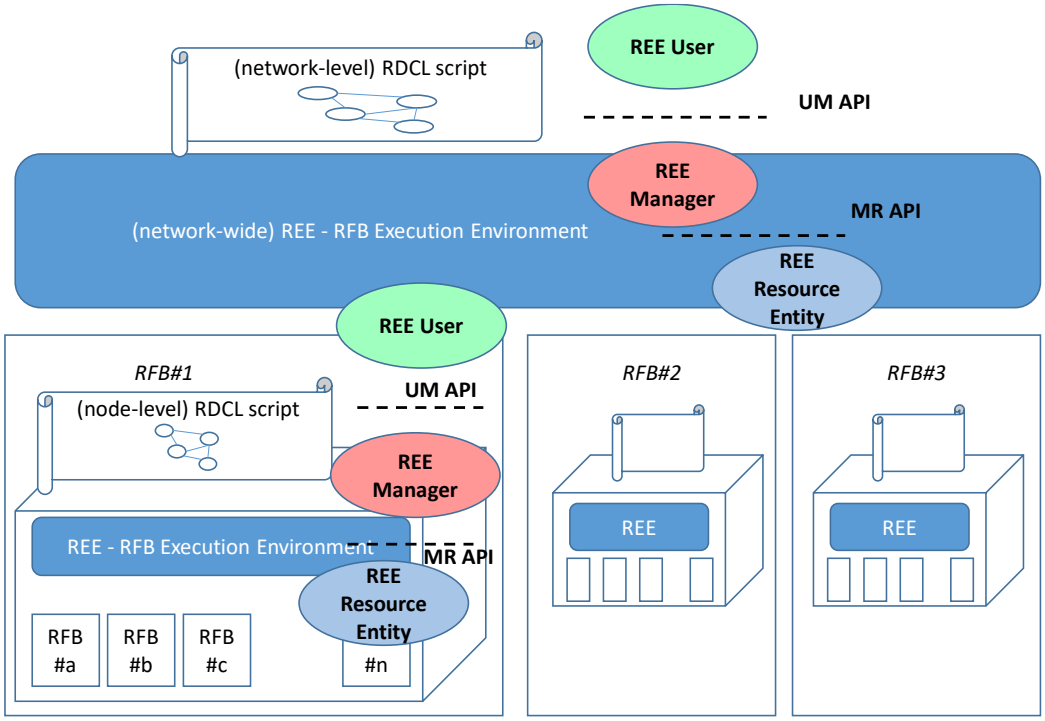


Figure 26: Superfluidity Architecture APIs (copy of Figure 11)

5.2 "Traditional" NFVI infrastructure: Telemetry driven VNF Scaling

The process of collecting and publishing metrics related to the infrastructural components within the NFVI and the services running on top is known as telemetry. A variety of mechanisms can be

used to collect the metrics ranging from parsing log files, reading of system counters, to the implementation of highly customised probes to collect specific system metrics under explicit conditions such as a service failure event. In the context of telemetry to support scaling decisions it is necessary to collect and process the relevant data available and present the data to an Orchestrator in near real-time in order to react in a timely manner. Telemetry provides the Orchestrator with the necessary data in order to sufficiently reason over how to manage both the resources and services under its control.

The data available in a VNF host environment is usually much finer-grained than what is available to a VIM. A VIM virtualisation layer such as Xen or KVM will only be able to look at a VNF virtualised environment as a black box and at best expose the aggregate performance counters. The performance as measured by Xen can be done through timers and counters using a combination of software and/or hardware techniques. The Xen hardware performance monitoring technique is to use Xen OProfile, whereas XenPerf, XenTop, XenTrace, XenAlyze are part of the Xen tool stack for software performance measurement. In KVM, libvirt can be used to monitor performance counters. Both virtualisation techniques suffer though from not being able to take advantage of the hardware performance units (PMUs) unless the data is passed through to a particular guest or a virtual performance counter system.

As VNFs are broken down into RFBs as proposed by Superfluidity, with less overhead provided to the virtualisation sub-systems, this leads to a black box approach providing a closer approximation to the real-world performance. In the interim period though, where there are still monolithic functions and services that are performing multiple roles, the approximation is less adequate. This can be resolved by introducing agents into the virtualised environments as described next.

From a service monitoring perspective, telemetry information may need to be collected directly from the virtualised host (VM, Container, Unikernel), which provides the execution environment for the VNF. This is achieved by deploying distributed telemetry agents or configurable monitoring services throughout each of the virtualised hosts and on the host infrastructure. The telemetry data collected can then be relayed to a centralised location (typically a database) accessible to the Orchestrator via a set of defined API's such as REST APIs. These data are then consumed by an Orchestration subsystem dedicated to VNF monitoring, which can decide, based on predefined scaling rules, whether to request a scaling action from the Orchestration scaling subsystem.

In OpenStack there are a number of performance and metric gathering telemetry frameworks. Currently the leading framework is Ceilometer [33]. The goal of Ceilometer is to efficiently collect, normalise and transform data produced by OpenStack services. However, the performance of Ceilometer does not scale efficiently with large numbers of metrics across large numbers of service instances. To address this limitation a new OpenStack project, Monasca (monitoring as a service at scale) has been developed.

While it is possible to collect an extensive range of telemetry data from platforms such as Collectd and SNAP, a 'collect everything' approach is inefficient and can negatively affect VNF performance in a worst-case scenario due to the consumption of CPU cycles. The telemetry platform should only collect data in relation to the metrics that have been a-priori identified by the VNF vendor or during acceptance testing by the service provider as having a significant influence on the performance of the VNF. Typically, thresholds will be established for the metrics of interest e.g. memory utilisation, CPU utilisation, network I/O, disk I/O etc. In an operational context, when the metrics of interest are reporting within their acceptable thresholds the VNF is functioning as required. Outside of these thresholds, the behaviour and performance of the VNF may not be compliant with the associated SLA and KPIs for the VNF. Temporal conditions will also typically apply i.e. the metrics of interest will exceed the defined threshold for a specified period of time before an Orchestration scaling action is initiated.

In order for telemetry information to be useful, the system has to be able to actuate changes that in turn provide a feedback loop for changing the environmental conditions i.e. did the actuated change positively or negatively impact the performance of the system. Recording information at a rate that is mismatched from the actuation (orchestration) system will lead to either over sampling or under sampling. In some cases, it may also not be possible to make changes, which actually improve system performance. It is therefore very important to match the telemetry system with the orchestration system to ensure that what is being monitored can be reasoned upon within a given time frame and then acted upon in order to ensure that the overall SLAs and KPIs are met. The control system's feedback loop is very important to keep in mind for the NFVI.

The key implications from an architecture perspective for telemetry-driven scaling are as follows:

- Distributed telemetry agent/services run on VNF virtualised hosts and/or host physical infrastructure. Telemetry agent/services configurable in order to define/select only metrics of interest/relevance. Agent/service level definition of metrics thresholds to minimise telemetry traffic.
- Centralised storage of telemetry of data. Possible processing of data to derive temporal trends for predictive scaling. Data accessible by Orchestrator via standard APIs.
- VNF monitoring subsystem which has required the logic capabilities to reason over the telemetry data using simple thresholds or models to determine when a scaling action is required.
- VNF monitoring subsystem has interface to the VNF provisioning subsystem.

5.3 Container-based NFVI infrastructure for Cloud RANs

Today, the network is a network of elements (e.g., an e-NodeB) hosting a set of functions. However, when using virtualization technology to address 5G requirements, it makes sense to decompose these virtualized network elements into RFBs (Reusable Function Block) to support

and implement 5G flexibility. In addition, a given functionality, such as authentication/authorization, can currently be implemented in different network elements (i.e., packet data network gateways and mobility management entities) to ensure vendor compatibility. Through the decomposition principle of Superfluidity, each functionality (authentication, authorization, radio resource management, turbo decoding, fast Fourier transform, etc.) will be associated to an RFB, thus allowing a modular design of 5G. Thanks to this modularity, an update of any specific RFB can be done without interfering with other RFBs.

In Deliverable D2.2 [34], a full analysis of the different processes supported in each layer of the RAN stack (Physical, MAC, RLC, RRC, PDCP) has been conducted. A classification of the identified processes is proposed, based the nature of the process (control or data plane), synchronization to low layer time (TTI), scope of the process (user based or Cell based). Based on that classification, a set of RFBs are proposed.

Due to real time requirements in the RAN part (for example, the scheduler has to decide each 1ms on the way the radio frame is filled), we select the container technology as an environment of execution of RFB of the RAN. This choice is motivated by the performance of the container technology compared to Virtualization Machine (using Xen or KVM as hypervisor) in terms of boot time, RTT (ping flood) and throughput (computed using iperf [47]) as it is measured in [22]. The structure of each container technology is depicted in Figure 27.

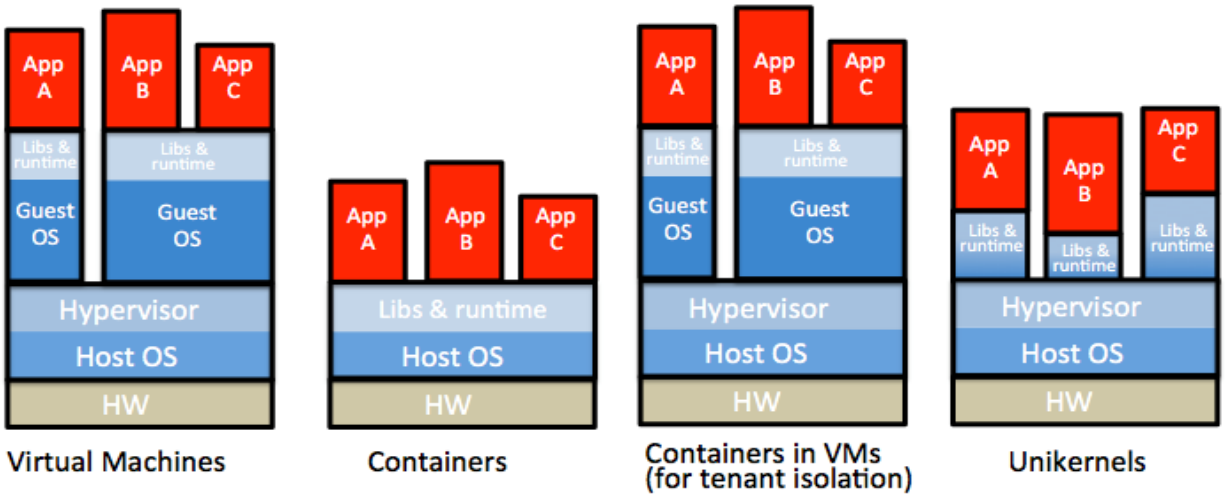


Figure 27: Virtual Machine, Container and Unikernel (source [23])

While virtual machines provide an abstraction at the hardware level, the container model virtualizes operating system calls so that typically many containers share the same kernel instance, libraries and runtime (Figure 27). Containers are attractive because they enable complete isolation of independent software applications running in a shared environment. They can also greatly mitigate traditional bare-metal performance through isolation of the network, file system and resource consumption of each container.

Docker [31] is a new leading container based technology that offers a more efficient and lightweight approach to application deployment. Its eco-system offers a simple way to define a service chain as a multi-container application by including all the dependencies in a single descriptor file thanks to docker compose command.

Docker represents a natural evolution in the container technology space. The Docker platform integrates and extends existing container technologies by providing a single, lightweight runtime and API for managing the execution of containers as well as managing the container images themselves. The main reason Docker has seen such pervasive adoption is because it allows software applications to prepackage their dependencies into a lightweight, portable runtime requiring less overhead to run than a KVM based hypervisor.

Docker ecosystem proposes a variety of tools for containers management; they can be used to reproduce complex systems across different environments. Docker "Compose" is the composition tool for Docker-based containers. This tool is used to define and run multi-container Docker applications into a service chain environment.

The service chain structure and configuration is held in a single place, which makes spinning up services simple and repeatable.

The Compose file is a YAML file defining services, networks and volumes. YAML stands for "*YAML Ain't Markup Language*" taking concepts from programming languages such as C. It shares a set of similarities to XML.

We generalized the Docker compose model so that it can represent an RFB Execution Environment (REE). RFBs are the key components of an REE. They are described with composition files (RDFiles.yaml) which implement semantic specification for the RFB structure in a human readable language. In Figure 28, we give a detailed representation of an RFB structure. The "Metadata" section is used for the characterization and description of the RFB. It is composed of metadata sub-sections each of them describing a specific behaviour, capability or requirements of the RFB. Meta1, as an example, describes interaction behaviour, inbound and outbound interfaces, type of communication and supported protocols, access points and connectors. The "Meta2" section describes requirements for composition, affinities (and anti-affinities), capabilities and execution environment.

The "Composition" section, in Figure 28, provides a reference (a software image) of the composed RFB and describes the embedded network functions or service graph composition. It describes the logical connectivity and association between the different RFBs and how the component RFBs are connected to each other.

In the Superfluidity architecture and models definition, the RFB concept is used in two complementary ways:

- The RFB Operations mode: is to model for the *allocation* of service components to an execution platform, with the proper mapping of resources.

- The RFB Composition mode: can be used to explicitly model the *composition* of RFBs to realize services, network functions or components.

The two approaches are not mutually exclusive. As described in Figure 30 with the CRAN RFB, RFB Operation and RFB Composition can be combined to deploy a full (infrastructure, resources & services) NFV deployment.

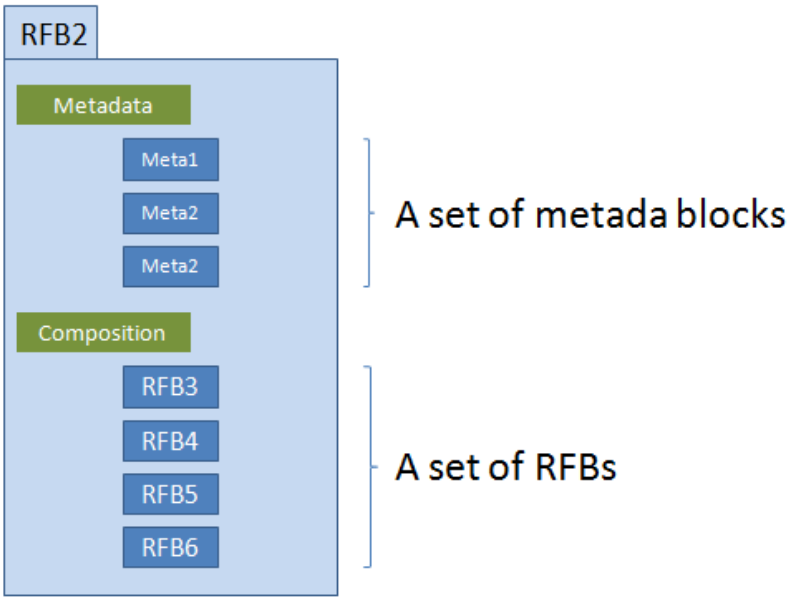


Figure 28: RFB composition file structure

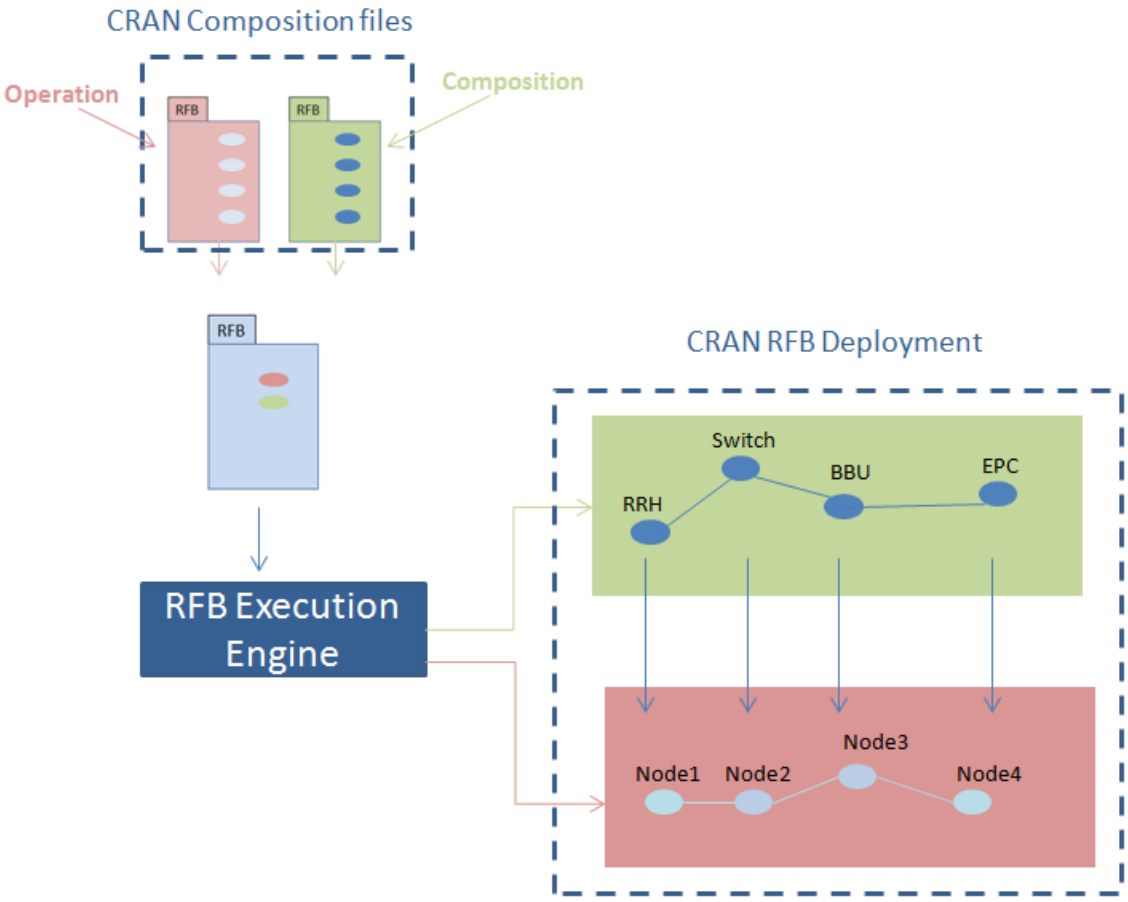


Figure 29: RFB deployment with RFB execution engine

5.4 "Traditional" NFVI infrastructure : the MEC scenario

A key enabler for low-latency scenarios of 5G networks is the concept of Mobile Edge Computing (MEC) [19], which main idea is to deploy computing capabilities near to end users. ETSI has specifically devoted an ISG called MEC (*Mobile Edge Computing*), to standardize the way application developers and content providers can run their services from the edge MEC supporting Radio Access Network (RAN) processing and third party applications. The MEC technology brings a set of advantages: i) ultra-low latency, ii) high bandwidth, iii) real-time access to radio network information and iv) location awareness. As a result, operators can open the radio network edge to a third party, allowing them to rapidly deploy innovative applications and services towards mobile subscribers, enterprises and other vertical segments. One of the goals of Superfluidity is to integrate MEC in the overall architecture, so that the MEC platform can rely on the same physical resources of an Extended-NFVI.

By introducing the notion of RFB in MEC, it becomes easier to: i) scrutinize constituent elements of the aforementioned components, ii) distinguish those that can be virtualized, implemented

and re-utilized in the most efficient way, and iii) provide a network capable of handling its resources with a higher degree of granularity.

Applications are the most dynamic entities of the MEC Architecture. The whole architecture is devoted to provide an agile environment to ME Apps (Mobile Edge Applications), making them available closer to end users, even when those are moving along different attachment points. The utilization of reusable functional blocks (RFBs) in MEC ecosystems takes advantage of the fact that Applications are usually built based on multiple components, e.g. load balancers, webservers, databases, back-ends, typically associated to different environments. The reutilization of these components permits a more flexible management of ME Apps. Namely, it allows the deployment of the appropriate computation capacity for each environment, and can easily accommodate ME Apps changes (scale in or scale out) to the performance requirements along the time.

Other functional blocks (other than ME Apps) can also benefit from the concept of RFBs. The MEC architecture uses a network function (NF) to inspect and offload certain traffic towards the appropriate ME App to serve the users from the edge. This NF can be virtualized (VNF) or not. Although this NF is out of the scope of the MEC standardization work, it is usually referred to as TOF (Traffic Offloading Function), since this was the name used at the beginning of the ETSI MEC activity. Note that the TOF name has been removed from the final MEC architecture picture. When implemented as VNF, the TOF can also be deployed as a set of RFBs, as it is composed of smaller components such as, GTP encap/decap, Traffic Filter, Router, or D/SNAT, among others. TOF can take advantage of this modularity by deploying and scaling this VNF according to the performance requirements.

Finally, the Management and Orchestration components of the MEC Architecture can also be decomposed into smaller and reusable blocks. However, as those components are not expected to be deployed or scaled dynamically (at least in our scenarios), the utilization of the RFB concept is more limited and its application could be just a matter of software development good practices.

5.5 Unikernel based virtualization

Common lore says that when needing virtualization we need to pick our poison between virtual machines, which are heavyweight but provide strong isolation, and containers, which are lightweight but come with less-than-optimal isolation properties. In Superfluidity, being able to support multi-tenancy, and thus having strong isolation, is a basic requirement, and so the crucial question is whether we can have the best from both worlds: is it possible to have the strong isolation that hypervisors provide along with the lightweight properties (e.g., small boot times, low memory footprints and the ability to run large numbers of instances simultaneously on a single server) commonly found in containers?

The work described in this section is a step in in answering to the above question, optimizing not only the virtual machines themselves (through the use of VMs tailored to specific applications, also known as Unikernels) but also re-architecting the virtualization platform they run on.

5.5.1 Optimizing Virtualization Technologies – A Brief Background

In order to optimize a virtualization platform, three main components come into play: the application, the virtual machine's operating system and the virtualization platform itself (i.e., the hypervisor and any tools it depends on); these components are shown graphically in Figure 30.

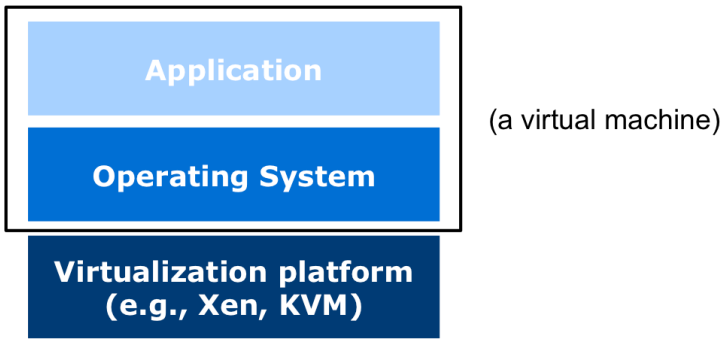


Figure 30: Main components of a virtualized system running a virtual machine

The work we have been carrying out in Superfluidity consists of improvements to the bottom two layers of this stack (we also work on optimizations to applications upon request).

Before we can explain what these technologies are, we must first give a bit of background about the Xen hypervisor.

5.5.2 A Quick Xen Primer

The Xen hypervisor is a type-1 hypervisor, meaning that it is a thin layer running directly on the hardware and performing basic operations like managing memory and CPUs (see Figure 31). Once the hypervisor boots, it creates a privileged virtual machine or domain called dom0 (typically running Linux) in charge of managing the virtualization platforms (e.g., an administrator can log in to dom0 via ssh and create and destroy VMs). In addition, Xen includes the notion of a driver domain VM which hosts the device drivers, though in most cases dom0 acts as the driver domain. Further, Xen has a split-driver model, where the so called *back half* of a driver runs in a driver domain, the front-end in the guest VM, and communications between the two happen using shared memory and a common, ring-based API. Xen networking follows this model, with dom0 containing a netback driver and the guest VM implementing a netfront one. Finally, event channels are essentially

Xen inter-VM interrupts, and are used to notify VMs about the availability of packets.

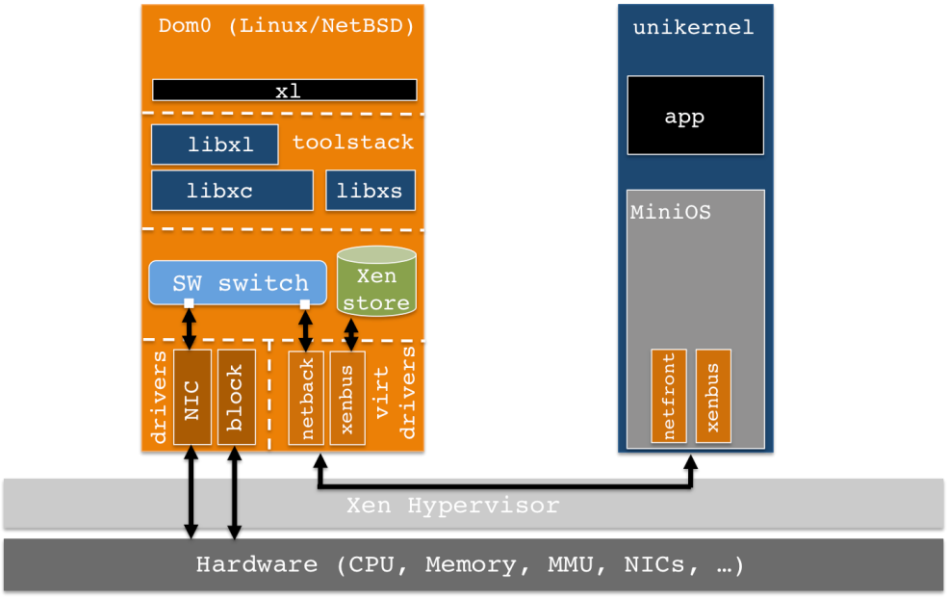


Figure 31: The Xen architecture

In addition, dom0 includes the *xl* command that allows a system administrator to manage the platform (e.g., to create/destroy/migrate VMs, checkpoint them, monitor them, access their consoles, etc.). The xl command sits on top of a number of libraries which together comprise the *toolstack*. The toolstack is fundamental to the basic operations as VM creation and migration and strongly influences their efficiency and performance (e.g. latency).

Further, dom0 includes the *xenstore*, a proc-like database used to share control information between dom0 and other VMs. The *xenstore* is also key to the performance of the operations mentioned above (VM creation and migration).

5.5.3 Xen Optimizations

The first optimization (see Figure 32) has the purpose of improving the efficiency, throughput and delay characteristics of Xen's network I/O. This includes replacing the standard Open vSwitch back-end switch with a high-performance software switch called VALE; replacing the netback driver in dom0 and the netfront driver in the virtual machine with optimized versions of them; and placing the software switch directly in the hypervisor to further improve performance.

The second optimization focuses on replacing the Xen's inefficient toolstack with a lean, purpose-built toolstack we call *chaos*. Such a change helps up improve the performance of a large number of operations including boot and destroy times.

The third optimization consists of removing the xenstore from Xen. This requires multiple modifications, since several Xen components depend on it (the toolstack and the virtualized drivers both in dom0 and in the VMs).

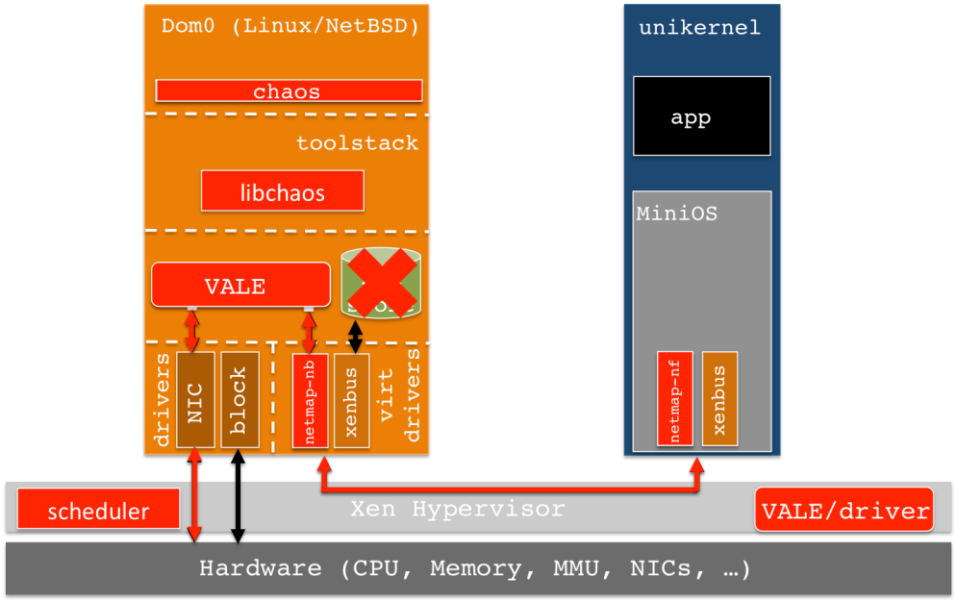


Figure 32: Xen platform optimizations

The fourth optimization targets dom0 itself. Dom0 is a full Linux distribution (e.g., Ubuntu) and so consumes significant amounts of memory and CPU cycles, especially on a resource-constrained single board computer. We are in the process of building a minimalistic dom0 either based on a minimalistic Linux distribution or a minimalistic OS (MiniOS, please refer to the next section).

5.5.4 Unikernels

So far, all of the optimizations described have dealt with the virtualization platform layer (refer back to Figure 30). In this section, we will discuss optimizations to the VM's operating system layer, that is, the layer above. The purpose of our research is to optimize the performance parameters like for example VM startup latency and VM memory footprint.

In the VM's operating system layer, our research has focused on the notion of *unikernels*: virtual machines based on a minimalistic OS (i.e., not a general-purpose OS such as Linux) and tailored to a single application (e.g., a web server).

Figure 33 illustrates the difference between a standard VM based on a general-purpose operating system, having separate address spaces for kernel and user level and running multiple applications; and a unikernel which has a single address space, a single application. Because unikernels are specialized, they consume much fewer CPU cycles, require less memory and help in reducing things like boot times and migration times.

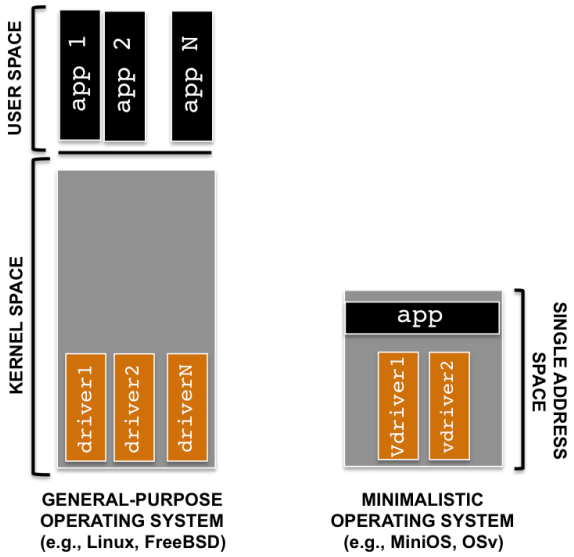


Figure 33: A standard virtual machine with separate address spaces for kernel and user space along with multiple applications (left) versus a unikernels consisting of a single memory address space and a single application

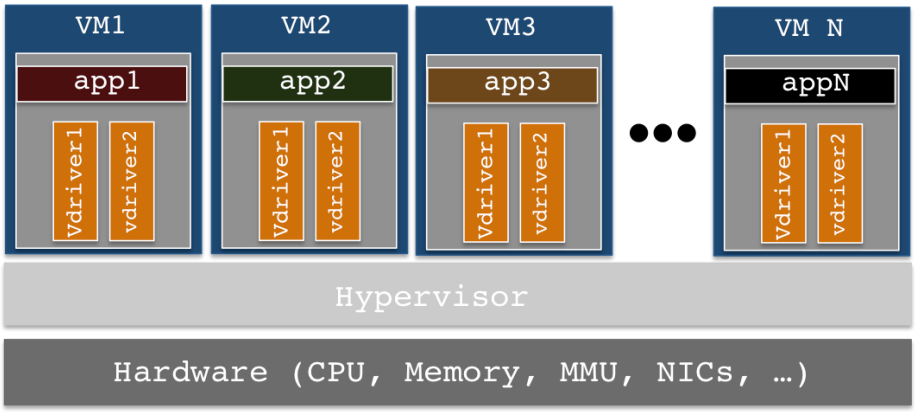


Figure 34: Multiple unikernels running different applications concurrently

It is worth noting that running a single application is not a limitation: whenever a system requires multiple applications, it is easy to have multiple unikernels, one per application, running on the same server (Figure 34).

Unikernels provide the best results in terms of performance, at the cost of having to potentially port applications to the underlying minimalistic operating system. To mitigate the issue of porting the application to the unikernel, we are working on an alternative we call *Tinyx*, a tool that allows automatic creating of minimalistic Linux distributions (i.e., VM images) tailored to specific applications.

5.5.5 A Few Numbers

To conclude this section, we give a few performance numbers run on an x86 server. While we don't yet have equivalent numbers on SBCs, we do have our technologies running on several of

them (e.g., the ARM-based CubieTruck and the Intel-based NUC). Figure 35 below gives a summary of these numbers.

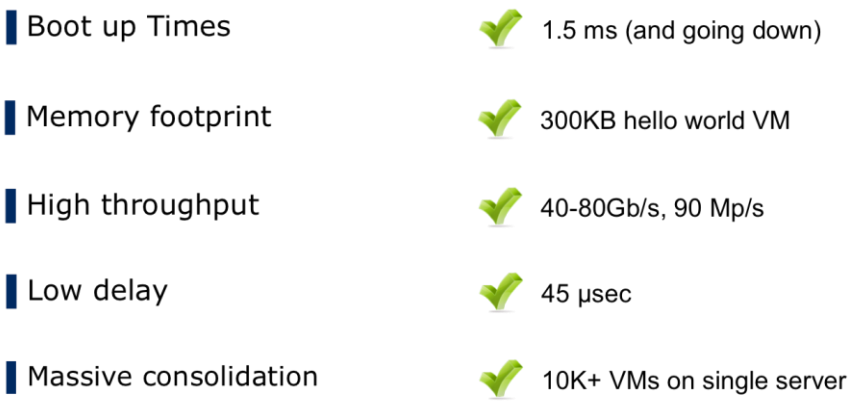


Figure 35: Summary of performance numbers obtained when running Xen with Superfluidity's optimization technologies

These are still early days, but the preliminary numbers are promising and proof that is it possible to have lightweight virtualization along with strong isolation. We are currently in the process of further optimizing boot and migration times. As future work, we will look into developing tools for automatically building high-performance, custom unikernels.

5.6 Click-based environments

Click [17] is a software architecture that decomposes the functionality of a packet-forwarding node into elements that are interconnected to implement arbitrarily complex functions. The Click elements operate on packets (e.g. at IP or Ethernet level) by performing inspections and/or modifications of the different packet header fields. A large number (hundreds) of built-in Click elements are available, including for example Packet Classifiers, Rate Monitor, Bandwidth Shapers, and Header Rewriters. The interconnection graph of Click elements is called a *configuration*, which is described by a declarative language. The language allows the definition of compound elements composed of other elements. In Superfluidity, Click is one of the RFB Execution Environments. In our vision, a single Click element represents a RFB and a configuration that combines Click elements represents again a (composed) RFB. In [21], it is shown how a Click configuration can be executed in minimalistic virtual machines that runs in a specialized hypervisor called ClickOS. ClickOS Virtual Machines can be seen as RFBs and can be composed to implement the desired packet processing services.

To give an example of the Click capacity to build complex forwarding behaviour, the functionality of a Cisco ASA (Adaptive Security Appliance) [8] in a real environment (University Politehnica of Bucharest's Computer Science Department network) has been analysed. The corresponding Click configuration has been reconstructed. Figure 36 shows a simplified "Out-In" traffic pipeline of the ASA model. The ASA model contains over 200 Click elements. It performs VLAN switching, routing,

traffic filtering, static and dynamic NAT. To generate an ASA model, we have developed a tool, which parses a subset of ASAs' configuration file, and builds its corresponding Click model. We have validated our generation tool using "black-box" testing – by crafting various input packets and observing the outputs. We have sent TCP, UDP, raw-IP and ICMP packets through a real ASA and its model, and compared the results. A detail of the actual Click elements used by the TCP traffic classification block is shown in Figure 37.

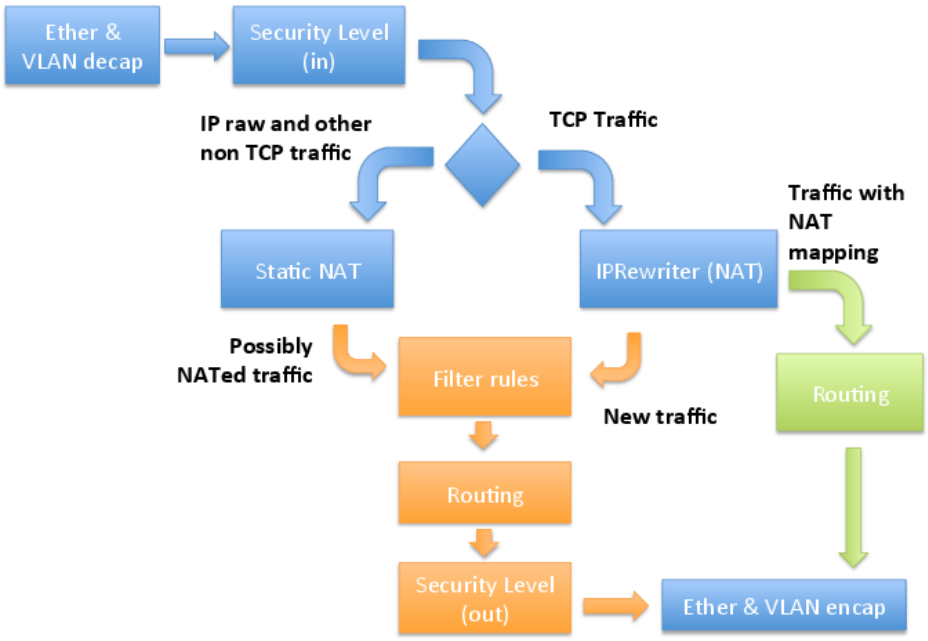


Figure 36: High-level decomposition of Cisco ASA

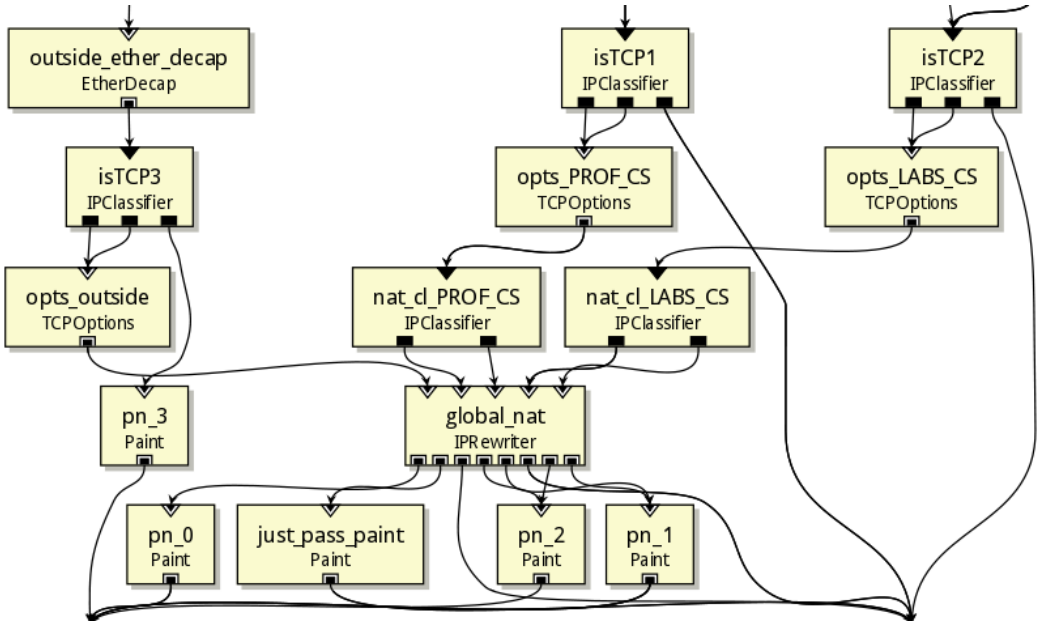


Figure 37: detail around TCP traffic classification block.

5.7 Radio processing modules

The radio processing modules implement radio access functionality, enabling the transformation of user bit-streams at network layer (Layer 3) into the waveforms at radio transceiver interface (Layer 1) and vice versa. This part of work addresses primarily data plane functionality of virtualised baseband unit as illustrated in Figure 38.

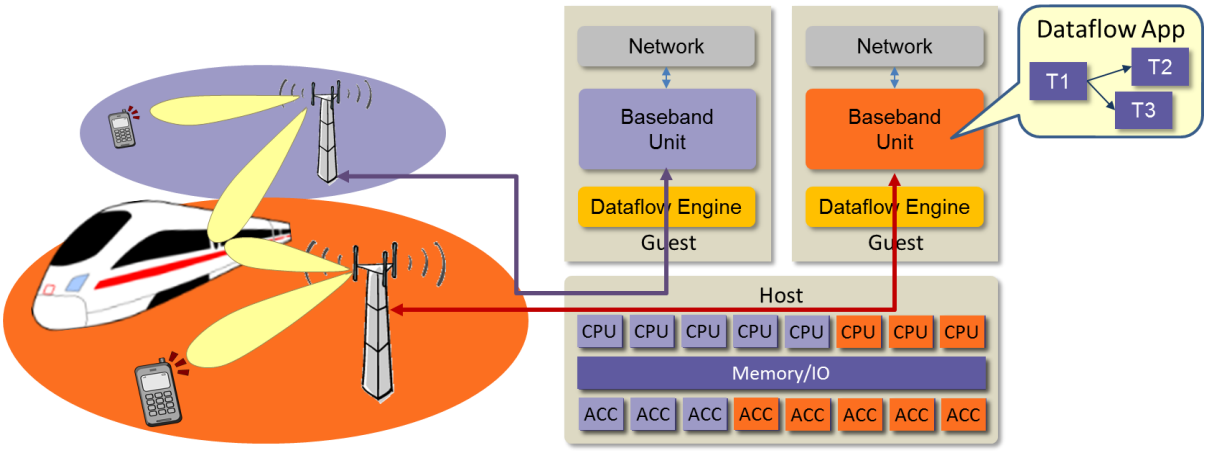


Figure 38 Principle of C-RAN architecture employing virtualised baseband units employing dataflow model of computation

The baseband unit is decomposed into the graph of functional elements (actors) each of them consuming data tokens on input ports produced by its predecessors while generating result tokens on output ports for its successors (Figure 39). The tokens produced by source actor and consumed by destination actor are stored temporarily into buffers implementing FIFO policy. Hence, the execution of actors is determined by input data availability. This kind of processing is well represented by the dataflow Model of Computation (MoC) [46]. The associated dataflow engine provides execution runtime environment abstracting HW resources and enabling efficient mapping of baseband elements to underlying processing resources. In order to allow programming of dataflow applications, the dataflow language or API are necessary in order to express the dataflow elements and relations and translate it to associated commands of dataflow engine. Specific dataflow MoC, engine, API and associated development tools are called dataflow framework.

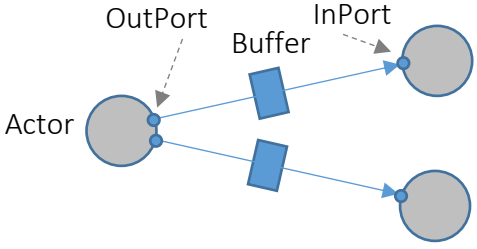


Figure 39 Principle of dataflow application comprising actors (circles) and buffers (rectangles). Buffers implements typically FIFO policy and stores temporarily the tokens produced by source actor until consumed by destination actor

In contrast to the network functions employing general Actor Model (AM) of Computation allowing out of order token processing, the radio data and signal processing adopt more restricted models in order to increase the determinism and predictability of dataflow application. Kahn-process networks (KPN) provide both the determinism and predictability due to the restrictions regarding in-order token processing and single token consumption-production on each actor invocation. However, the schedulability and dead-lock free operation is guaranteed only for unbounded buffers. Synchronous Dataflow Model (SDF) is a special case of KPN with state-less actors and predefined constant number of produced/consumed tokens in each actor iteration. Unfortunately, due to the limited expressiveness of SDF, these are not well suited to represent realistic dynamic baseband applications. Direct Acyclic Graph (DAG) is the simplification of SDF with buffer size one. Nevertheless, the dynamically generated DAGs may well represent time-varying dataflow applications, as it appears in the subframe-centric LTE baseband processing. The relations of the mentioned dataflow Models of Computation is illustrated in Figure 40. In the Superfluidity project, the main focus is in the adoption of KPNs and dynamic DAGs for the implementation of signal processing baseband algorithms.

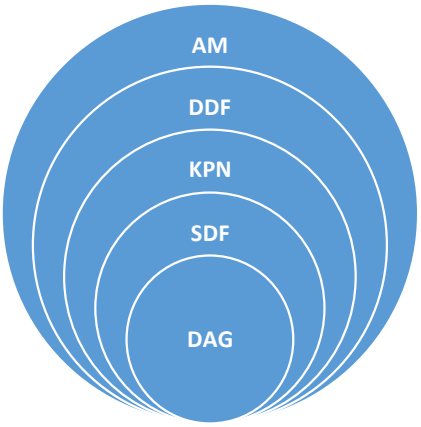


Figure 40 Dataflow models of computation

Baseband signal processing in dataflow MoCs is modelled by interconnecting functional elements i.e. tasks into dataflow graph as depicted in Figure 41. Tasks are stateless atomic units of computations realising specific function on input tokens produced by predecessor tasks. Tasks are completely characterized by their interface (i.e. in/out ports, token types, interconnect) and parameters (e.g. execution time, priority, resource affinity, deadline). Therefore, a task is a RFB according to the concept of Superfluidity. The tasks can by aggregated into systems (composed RFBs) for the purpose of implementing more complex functionality.

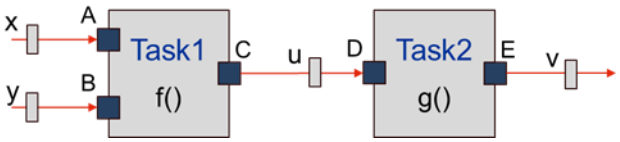


Figure 41 Implementation of dataflow model using task actors

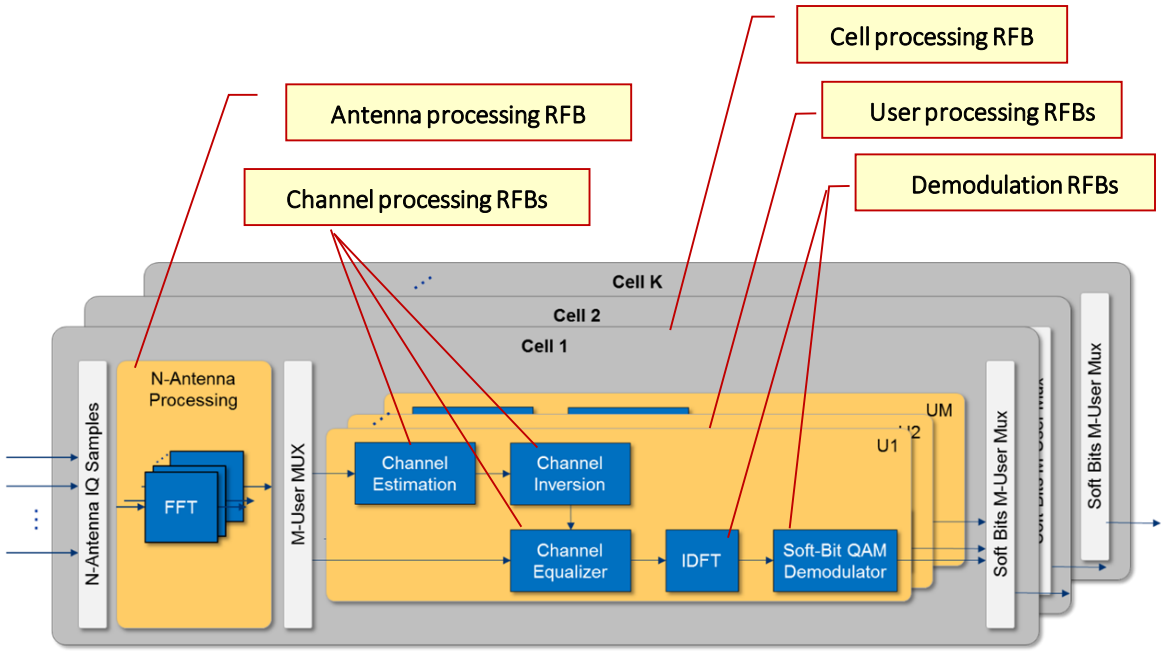


Figure 42 Flexible LTE baseband receiver as RFB

In order to evaluate the dataflow framework developed within the Superfluidity project, a flexible baseband RFBs have been developed comprising transmitter and receiver modules. The modules are composed of antenna, channel and user specific processing elements allowing to configure the RFBs for different 4G/5G deployment scenarios. An example of LTE UL (Up Link) receiver RFB based implementation is illustrated in Figure 42. The receiver implements RFBs for antenna specific sync and time-to-frequency transformation based on FFT, channel specific processing (channel estimation and inversion) and user specific processing (channel equalization, frequency-to-time transformation, demodulation). The baseband module is highly configurable and adaptable in terms of subframe structure and resource block structure, number of antennas, number of MIMO layers, number of users, modulation size. Moreover, multi-cell scenario is also possible in order to support C-RAN functionality.

5.8 Programmable data plane network processor using eXtended Finite State Machines

In Deliverable D2.2 [34], section 4.5, we have introduced a convenient platform agnostic programming abstraction for data plane network processing tasks (e.g. switching, routing, traffic processing, etc), based on eXtended Finite State Machines (XFSMs) [35]. With respect to the architecture terminology introduced in this deliverable, in D2.2 we have listed:

1. The relevant (nano) RFBs, i.e. the set of (in this case very) elementary primitives the programmer can use as "instructions" to program a network processor node.
2. The RDCL (RFB Description and Composition Language), which in this case was conveniently found to be an XFSM, being an XFSM an abstract and platform agnostic formal model which permits to combine elementary actions, events, conditions, and ALU-like arithmetic and logic micro-instructions into a stateful behavioural formal model.

It remains to specify how an XFSM can be "executed" by a software or hardware platform, i.e. it remains to specify how, concretely, an RFB Execution Environment (using Section 3's notation) for a high performance data plane network processor should be engineered. In the remainder of this section we complete this final step, and propose an initial reference architecture for a programmable network processor, which we call Open Packet Processor (OPP) [36] which supports processing tasks formally described in terms of XFSMs.

Unless otherwise specified, we will refer to OPP in terms of reference HW design. The reasons is twofold. The first one is technical: of course an HW design is more challenging and restrictive over a SW one, as it needs to show that the proposed reference RFB Execution Environment is capable by design to process each packet in a limited and bounded number of clock cycles per packet. The second reason is more strategical. We believe, and we will argument in the next "digression" Section 5.8.1, that virtualization should not restrict to SW executed on commodity CPUs (e.g. x86 or ARM) but should also entail domain-specific processors. Our OPP is a concrete proposal for such domain-specific processor - which, despite being tailored to a specific domain (in our case flow processing) still permit a broad range of functions to be programmed. The reader just interested in the OPP architecture may skip section 5.8.1, and directly move to the next section 5.8.2.

5.8.1 Digression: network function virtualization on domain-specific HW

The specification currently in progress within the ETSI Network Function Virtualization (NFV) ISG retains an high level of generality in terms of Infrastructure specification (namely, the underlying NFVI, Network Function Virtualization Infrastructure). Still, reliance on commodity processors such as x86 or ARM as platforms for running Virtual Machines implementing virtualized network functions is often implicitly assumed. While such a choice of commodity processors seems of

course optimal in terms of (low) cost and (maximum) flexibility, it brings about obvious performance concerns, especially when dealing with network functions which require intensive packet-level processing at (ideally) line rate. Take for instance very basic network functions such as switching. Despite the extensive effort carried out by companies and research groups specialized in software acceleration on commodity computing platforms, the gap between HW-based switching chips and SW-based ones is still significant and not going to decrease in the future. (see Figure 43, taken from a presentation of Nick McKeown, Stanford University, 2014).

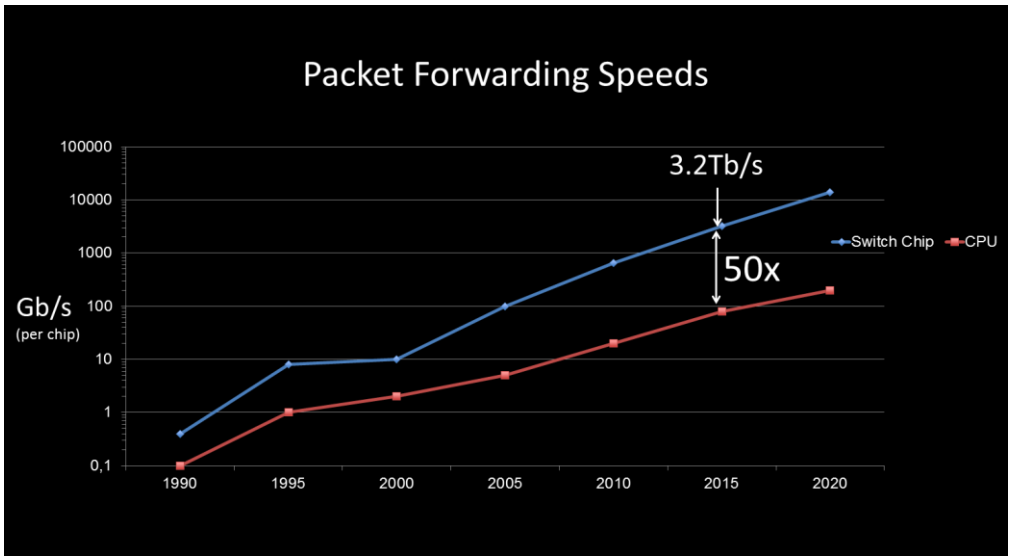


Figure 43: HW vs SW switches forwarding speeds

Indeed, massive I/O parallelization is possible only in an HW switching chip. Additionally, some very basic match/lookup primitives, such as wildcard matching, are extremely efficient when implemented in HW (e.g., on commodity TCAMs), unlike their SW implementation counterpart (at least in the most general wildcard match case, so far there is no algorithm which can perform at O(1) speed/complexity). The situation becomes even worse when dealing with flow/packet-level functions such as traffic classifiers or policers, traffic control algorithms (say an Active Queue Management algorithm), packet schedulers, packet-level monitoring tasks, security-related functions (such as DPI or stateful firewalls), and so on. Running such functions directly on the switches' fast-path is today unaffordable with commodity CPUs, and in fact 'pure' SW switches usually divert such flows over the slow path, whereas commercial brand-name switches rely on custom HW.

Domain-specific platforms.

Interestingly, networking is a field where general-purpose commoditized domain-specific platforms are somewhat still lagging behind. In fields such as graphics and video/image processing, general purpose commodity programmable platforms, namely Graphical Processing Units (GPUs), have taken up since long time. They do provide the applications' developers with a dedicated HW platform which retains full flexibility and programmability. At the same time, they

are also tailored and performance-optimized towards the very special type of processing needed in this field. The same holds in fields such as signal processing (with DSP platforms), and to some extent also in the field of cryptography (with crypto accelerators). Conversely, in the networking field, bare metal switches are just starting to emerge, and (besides the pioneering work in OpenFlow which however is functionally limited to only support programmable Flow Tables, and besides the more recent work on P4 where, however, a reference architecture for a P4 target chipset – albeit already claimed to be manufactured by Barefoot - is still not made public) a clear and neat interface between hardware functions and their software configuration and control is still missing.

A closer look at the issue of a lack of domain-specific network processors reveals that the real crux of the matter is not only technological, but it also (and mostly) revolves around the identification of what should be the network and traffic processing "primitives" or blocks supported by a platform, and how such blocks should be combined to produce meaningful and realistically complex network functions. In other words, the problem is not (only) related to the feasibility and viability of a programmable platform for developing network functions. It is also necessary to consider i) the appropriate functional decomposition of an higher-level network function into smaller and reusable network and processing blocks, and ii) the identification of how to formally describe (and run-time enforce) how such blocks combine together to form a desired higher level function. An enlightening example of what has been done so far in this field, and also what is still missing, is OpenFlow. On one side, OpenFlow has clearly identified elementary "actions" which are pre-implemented in the switch and which can be therefore exploited by a third party "programmer", and has left full room for extensions in the supported set of actions. On the other side, the OpenFlow match/action abstraction that the programmer can use to configure a desired Flow Table is largely insufficient to formally describe more complex (e.g. stateful) flow processing tasks. As a result, most of the today's network programming frameworks circumvent OpenFlow's limitations by promoting a "two-tiered'' programming model: any stateful processing intelligence of the network applications is delegated to the network controller, whereas OpenFlow switches limit to install and enforce stateless packet forwarding rules delivered by the remote controller.

In the next sections, we describe our proposal for an Open Packet Processor. It starts from the OpenFlow model of decoupling actions from matches, but it further permits the programmer to describe the switch operation via a behavioural XFSM-based model which can be thus used as network processor's programming language.

5.8.2 The Open Packet Processor Architecture

In this section, we discuss into details a novel switch reference architecture which permits to concretely support a **decomposition approach, which we descriptively refer to as "nano-**

decomposition" (see Deliverable D2.2, Section 4.5). Rather than employing relatively high level functions, the idea is to define a set of extremely elementary and stateless actions and primitives as building blocks. In such model, we would not handle, say, a "load balancer function" but we would rather handle much more elementary forwarding and processing instructions (for instance, send packet from flow X to port Y, increment a counter, etc). To make a concrete example, a load balancer would not anymore be treated as a basic function block, but might be in turn "programmed" using a suitable combination of such very elementary primitives.

Challenges and requirements

In order to properly exploit "nano-decomposition", we wish to emerge with a packet processing architecture which holds the following four properties.

(1) Ability to process packets directly on the fast path, i.e., while the packet is traveling in the pipeline (nanoseconds time scale). More specifically, we would like to emerge with a solution which, ***if implemented in HW***, would meet the requirement of *performing packet processing tasks in a deterministic and small (bounded) number of HW clock cycles*. Indeed, as duly discussed in the introduction, network functions should not be restricted to be implemented on commodity CPUs, but should be eventually able to exploit, when high performance is mandated, hardware platforms. The reason is that a software program running on a commodity x86 or ARM platform might not sustain network functions which require intensive packet-level processing at multi Gbps line rate.

(2) Efficient storage and management of per-flow stateful information. As anticipated in Deliverable D2.2, section 4.5, we envision nano-decomposition as based on very elementary stateless actions or instructions. Hence, stateful management of flows is now mandated to the platform, opposed to the function block. Stateful management is required to permit the programmer to further use the ***past*** flow history for defining a desired per-packet processing behaviour (for instance, if a threshold number of packets has arrived for a flow, reconfigure the forwarding decision). As shown next, this can be easily accomplished by *pre-pending* a dedicated storage table (concretely, an hash table) that permits to retrieve, in O(1) time, stateful flow information. We name this structure as Flow Context Table, as, in somewhat analogy with context switching in ordinary operating systems, it permits to retrieve stateful information associated to the flow to which an arriving packet belongs, and store back an updated context the end of the packet processing pipeline. Such (flow) context switching will operate at wire speed, on a packet-by-packet basis.

(3) Ability to specify and compute a wide (and programmable) class of stateful information, thus including counters, running averages, and in most generality stateful features useful in traffic control applications. It readily follows that the packet processing pipeline, which in standard

OpenFlow is limited to match/action primitives, must be enriched with means to describe and (on the fly) enforce conditions on stateful quantities (e.g. the flow rate is above a threshold, or the time elapsed since the last seen packet is greater than the average inter-arrival time), as well as provide arithmetic/logic operations so as to update such stateful features in a bounded number of clock cycles (ideally one).

(4) Platform independence. A key pragmatic insight in the original OpenFlow abstraction was the decision of restricting the OpenFlow switch programmer's ability to just *select* actions among a finite set of supported ones (opposed to permitting the programmer to develop own custom actions), and associate a desired action set (bundle) to a specific packet header match. We conceptually follow a similar approach, but we cast it into a more elaborate eXtended Finite State Machine (XFSM) model. Indeed, an XFSM abstraction permits us to formalize complex behavioural models, involving custom per-flow states, custom per-flow registers, conditions, state transitions, and arithmetic and logic operations. Still, an XFSM model does not require us to know how such primitives are concretely implemented in the hardware platform, but "just" permits us to combine them together so as to formalize a desired behaviour. Hence, it can be ***ported*** across platforms which support a same set of primitives.

eXtended Finite State Machines

We recall, from Deliverable D2.2, section 4.5, that an eXtended Finite State Machine is an abstraction which is formally specified (see Table 1), by means of a 7-tuple **<I,O,S,D,F,U,T>**. Input symbols **I** (OpenFlow-type matches) and Output Symbols **O** (actions) are the same as in OpenFlow. Per-application states **S** permit the programmer to freely specify the possible states in which a flow can be, in relation to his/her desired custom application (technically, a state label is handled as a bit string). For instance, in a heavy hitter detection application, a programmer can specify states such as `NORMAL`, `MILD`, or `HEAVY`, whereas in a load balancing application, the state can be the actual switch output port number (or the destination IP address) an already seen flow has been pinned to, or `DEFAULT` for newly arriving flows or flows that can be rerouted. With respect to a Mealy Machine, the key advantage of the XFSM model resides in the additional programming flexibility in three fundamental aspects:

(1) **D: Custom (per-flow) registers and global (switch-level) parameters.** The XFSM model permits the programmer to explicitly define his/her own registers, by providing an array **D** of per-flow variables whose content (time stamps, counters, average values, last TCP/ACK sequence number seen, etc) shall be decided by the programmer himself/herself. Additionally, it is useful to expose to the programmer (as additional registers) also switch-level states (such as the switch queues' status) or ``global'' shared variables which all flows can access. Albeit practically very important, a detailed distinction into different register types is not fundamental in terms of abstraction, and therefore all registers that the programmer can access (and

eventually update) are summarized in the XFSM model presented in Table 1 via the **array D of memory registers**.

(2) **F: Custom conditions on registers and switch parameters.** The sheer majority of traffic control applications rely on *comparisons*, which permit to determine whether a counter exceeded some threshold, or whether some amount of time has elapsed since the last seen packet of a flow (or the first packet of the flow, i.e., the flow duration). The **enabling functions f_i:D→{0,1}** serve exactly for this purpose, by implementing a set of (programmable) boolean comparators, namely conditions whose input can be decided by the programmer, and whose output is 1 or 0, depending on whether the condition is true or false. In turns, the outcome of such comparisons can be exploited in the transition relation, i.e. a state transition can be triggered only if a programmer-specific condition is satisfied.

(3) **U: Register's updates**. Along with the state transition, the XFSM models also permits the programmer to update the content of the deployed registers. As we will show later on, registers' updates require the HW to implement a set of **update functions ui:D→D**, namely arithmetic and logic primitives which must be provided in the HW pipeline, and whose input and output data shall be configured by the programmer.

Finally, we stress that the actual computational step in an XFSM, i.e. the step which determines how the XFSM shall evolve on the basis of an arriving packet, resides in the transition relation **T:S×F×I→S×U×O**, which is ultimately nothing else than, again, a ``map'' (albeit with more complex inputs and outputs than the basic OpenFlow map), and hence is naturally implemented by the switch TCAM, as shown next. In other words, what makes an XFSM a compelling abstraction is the fact that **its evolution does not require to resort on a CPU**, but can be enforced by an ordinary switch's TCAM.

XFSM formal notation		Meaning
I	input symbols	all possible matches on packet header fields
O	output symbols	OpenFlow-type actions
S	custom states	application specific states, defined by programmer
D	n-dimensional linear space $D_1 \times \ldots \times D_n$	all possible settings of *n* memory registers; include both custom per-flow and global switch registers.
F	set of enabling functions $f_i: D \rightarrow \{0,1\}$	Conditions (boolean predicates) on registers
U	set of update functions ui:D→D	Applicable operations for updating registers' content
T	transition relation T:S×F×I→S×U×O	Target state, actions and register update commands associated to each transition

Table 1: Formal specification of an eXtended Finite State Machine (left two columns) and its meaning in our specific packet processing context (right column)

Open Packet processor: reference architecture design

To our view, what makes the previously described XFSM abstraction compelling is the fact that it does not necessarily mandate for a SW implementation (which of course is clearly a possibility), but if needed **it can be directly executed on the switch's fast path using off the shelf HW and without resorting on any CPU**. To support this claim, the next Figure 44 shows a possible reference implementation of an architecture which we refer to as "**Open Packet Processor**", in short **OPP**. Such architecture is on purpose described in terms of building blocks, which can therefore be implemented in hardware; when commenting about implementation issues, by default, we will refer to a possible HW implementation as this is the most challenging one since it requires to guarantee a bounded number of clock cycles per block/operation.

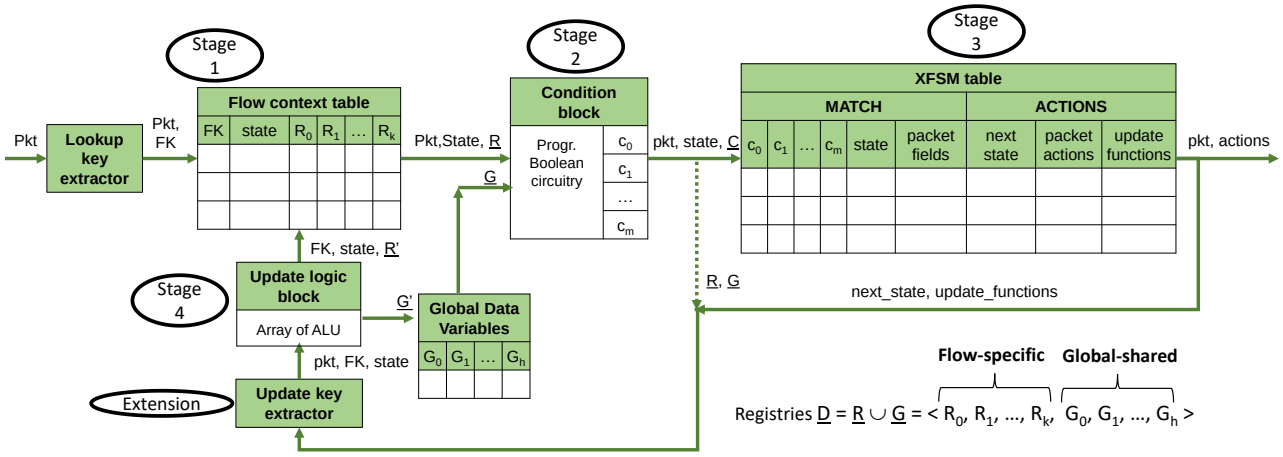


Figure 44. Open Packet Processor Architecture

The packet processing workflow for a packet traveling across the architecture depicted in Figure 44 is best explained by means of the following *stages*.

Stage 1: flow context lookup. Once a packet enters an OPP processing block, the first task is to extract, from the packet, a **Flow Identification Key** (**FK**), which identifies the entity to which a state may be assigned. The flow is identified by a unique key composed of a subset of the information stored in the packet header. The desired FK is configured by the programmer (an IP address, a source/destination MAC pair, a 5-tuple flow identifier, etc) and depends on the specific application. The FK is used as index to lookup a **Flow Context Table**, which stores the *flow context*, expressed in terms of (i) the **state label s_i** currently associated to the flow, and (ii) an array **$\underline{R}=\{R_0, R_1, ..., R_k\}$** of (up to) k+1 registers defined by the programmer. The retrieved flow context is then appended as metadata and the packet is forwarded to the next stage.

Stage 2: conditions' evaluation. The goal of the **Condition Block** illustrated in the above Figure (and implementable in HW using ordinary Boolean circuitry) is to compute programmer-specific conditions, which can take as input either the per-flow register values (the array $\underline{R}$), as well as global registers delivered to this block as an array **$\underline{G}=\{G_0, G_1, ..., G_h\}$** of (up to) h+1 global variables

and/or global switch states. Formally, this block is therefore in charge of implementing the **enabling functions** specified by the XFSM abstraction. In practice, it is trivial to extend the assessment of conditions also to packet header fields (for instance, port number greater than a given global variable or custom per-flow register). The output of this block is a boolean vector $\underline{C}=\{c_0, c_1, ..., c_m\}$ which summarizes whether the *i*-th condition is true (c_i=1) or false (c_i=0).

Stage 3: XFSM execution step. Since boolean conditions have been transformed into 0/1 bits, they can be provided as input to the TCAM, along with the state label and the necessary packet header fields, to perform a wildcard matching (different conditions may apply in different states, i.e. a bit representing a condition can be set to "don't care" for some specific states). Each TCAM row models one transition in the XFSM, and returns a 3-tuple: (i) the next state in which the flow shall be set (which could coincide with the input state in the case of no state transition, i.e., a self-transition in the XFSM), (ii) the actions associated with the transition (usual OpenFlow-type forwarding actions, such as `drop()`, `push_label()`, `set_tos()`,etc.), and (iii) the micro-instructions needed to update the registers as described below.

Stage 4: register updates. Most applications require arithmetic processing when updating a stateful variable. Operations can be as simple as integer sums (to update counters or byte statistics) or can require tailored floating point processing (averages, exponential decays, etc). The role of the **Update logic block** component highlighted in Figure 44 is to implement an **array of Arithmetic and Logic Units (ALUs)** which support a selected set of computation primitives which permit the programmer to update (re-compute) the value of the registers, using as input the information available at this stage (previous values of the registers, information extracted from the packet, etc). The updated registry values are then stored in the relevant memory locations (flow registries and/or global registries). More into details, since we are not relying on any general purpose CPU, but we are free to design in an HW implementation our "own" specific array of ALUs, the above mentioned computational primitives shall be most conveniently implemented as a domain-specific (i.e., specific for packet processing task) instruction set, which includes either micro-instructions typically found in a standard RISC processor (logic, arithmetic and bitwise instructions), as well as dedicated more complex microinstructions useful to traffic control applications (e.g. compute running averages, standard deviations, exponentially weighted running averages, etc). A detailed overview of the candidate microinstructions supported by the array of such ALUs has been reported in Deliverable D2.2, section 4.5.

Extension: Cross-flow context handling. There are many useful stateful control tasks, in which states for a given flow are updated by events occurring on *different* flows. A simple but prominent example is MAC learning: packets are forwarded using the *destination* MAC address, but the forwarding database is updated using the *source* MAC address. Thus, it may be useful to further generalize the XFSM abstraction, i.e. by permitting the programmer to **use a Flow Key during**

lookup (e.g. read information associated to a MAC destination address) and employ a possibly different Flow Key (e.g. associated to the MAC source) for updating a state or a register. The possible Flow Key differentiation between lookup and update phases is highlighted in Figure 44 by the usage of two different Key Extractor blocks.

6 State of the art

This section reports on different state of the art activities relevant to the Superfluidity architecture definition process.

A set of ongoing architecture definition activities in SDOs and industry associations are covered in Sections 6.1 and 6.2. Section 6.3 tries to identify a convergence approach for NFV, SDN, C-RAN and MEC. Section 6.4 introduces the convergence among Mobile Core of 4G/5G networks and the "central" cloud Data Centres. Sections 6.3 and 6.4 provide the background reasoning for the overall architectural framework introduced in Section 2 (Figure 1).

6.1 Latest 3GPP advancement toward 5G

Towards a more flexible and open network, past 3GPP releases, i.e. Rel-11, Re-12, and Rel-13 (see [45]), introduced a set of new features related to machine type communication and RAN sharing capability where some network entities are shared among virtual operators. These trends will be strengthened in forthcoming releases to be natively supported in future 5G networks.

Towards an API-driven customizable architecture, Enhancements for Service Capability Exposure feature introduced in Rel-13 provide means to securely expose the services and capabilities offered by 3GPP network interfaces to external application providers. This feature is used to support small data services as well as machine type communication (MTC).

Currently, flexibility in RAN is supported by the concept of a capacity broker for RAN sharing [13]. The resource allocation is provided on demand to virtual mobile network operator (VMNO) using signalling.

Beyond the RAN sharing concepts, 3GPP has defined two architectures in 3GPP SA2 [14]: one is based on the Multi-Operator Core Network (MOCN) configuration, where each operator has its own Evolved Packet Core (EPC) and a shared eNBs and Gateway Core Network (GWCN) configuration, where also the MME is shared between operators.

In parallel to this 3GPP enhancements, cloud computing technologies and cloud concepts gained momentum not only from the information technology (IT) perspective, but also within the telecom world. Emerging processing paradigms such as mobile edge computing (MEC) and Cloud-RAN (C-RAN) have to be accommodated to enable flexible deployment and to support programmability to meet different stringent requirements in terms of latency, robustness, and throughput.

Recently, the Network Functions Virtualization (NFV) trend of implementing network services as virtualized software running on commodity hardware has gotten a lot of attention. 3GPP started to study the support of NFV considering the network management perspective in [15]. In Rel-14, some specifications on architecture requirements for virtualized network management have been introduced [16].

It is clear that NFV is an important element for a convergence solution to build a natively customizable and adaptable 5G network changing the current paradigm of product specific application architecture. Superfluidity goes beyond the NFV framework to foster programmability as a main architectural feature of future 5G networks.

6.2 NFV, SDN & MEC Standardization

NFV (*Network Function Virtualization*) is the ETSI group devoted to standardizing the virtualization of *Network Functions* (NFs). It intends to specify the architecture required to accommodate the challenges of the new virtualization paradigm, covering the runtime and management aspects, in order to manage the entire lifecycle of a VNF (*Virtual Network Function*). Furthermore, it also comprises the management of *Network Services* (NSs), which can be built by orchestrating multiple VNFs, according to a *Forwarding Graph* (FG), using a catalog-driven approach.

The ONF (*Open Networking Foundation*) is an organization devoted to promote the utilization of software-defined technologies to program the network. Following a *Software-Defined Networking* (SDN) approach, the network is separated into three different parts: the user-data plane, the controller plane and the control plane. The user-data plane is responsible for forwarding the user data, while the controller plane is composed by SDN controllers, which provide high-level APIs to the control plane above. The control plane is responsible for programming the network, easing the creation of new applications and speeding up the rollout of new services.

MEC (*Mobile Edge Computing*) is the ETSI ISG devoted to standardizing the way application developers and content providers can afford an IT environment to provide their services from the edge. This environment provides ultra-low latency and high bandwidth capabilities, while it has access to operator-provided information, such as location and radio status. ETSI MEC intends to specify an architecture required to accommodate the challenges of computing at the edge, namely providing a dynamic (superfluid) services lifecycle management, allowing applications to cope with the end user mobility, application migration and integration with NFV ecosystems.

The 3GPP (3rd *Generation Partnership Project*) is an organization devoted to producing system specifications regarding cellular networks, including the radio access network, the core network, or services capabilities. It intends to create a detailed description of architectures, network elements, interfaces and protocols, making the interoperation among different vendors and network providers possible. 3GPP usually relies on existing protocols, like IP, SIP and Diameter, specifying only the particular parameters of operation.

The TMForum (*TeleManagement Forum*) is a telecom industry association devoted to provide guidelines to help operators to create and deliver profitable services. One of the biggest TMForum achievements is the definition of a complete business process (eTOM) and application (TAM) maps, including all activities related to an operator, from the services design to the runtime

operation, including monitoring or charging. In order to accommodate the changes brought by SDN/NFV, the TMForum has created the ZOOM (Zero-touch Orchestration, Operations and Management) program, which intends to build more dynamic support systems, fostering the service and business agility.

In the next section, we further detail the architecture of the preceding Standards Definition Organisations and relate the key architectural concepts to those that we promote in Superfluidity.

6.3 Cross Standards architecture

6.3.1 ETSI NFV architecture

The creation of the ETSI NFV ISG intended to bring to the telco sector the appropriate IT tools to take advantage of cloud principles, like on-demand, agility, scalability or pay-as-you-go (PAYG), among others. The hardware and software decoupling and consequent utilization of COTS (*Common Off-The-Shelf*) hardware can also be applied to network functions, leading to a cost reduction, increasing of network agility and vendor independency.

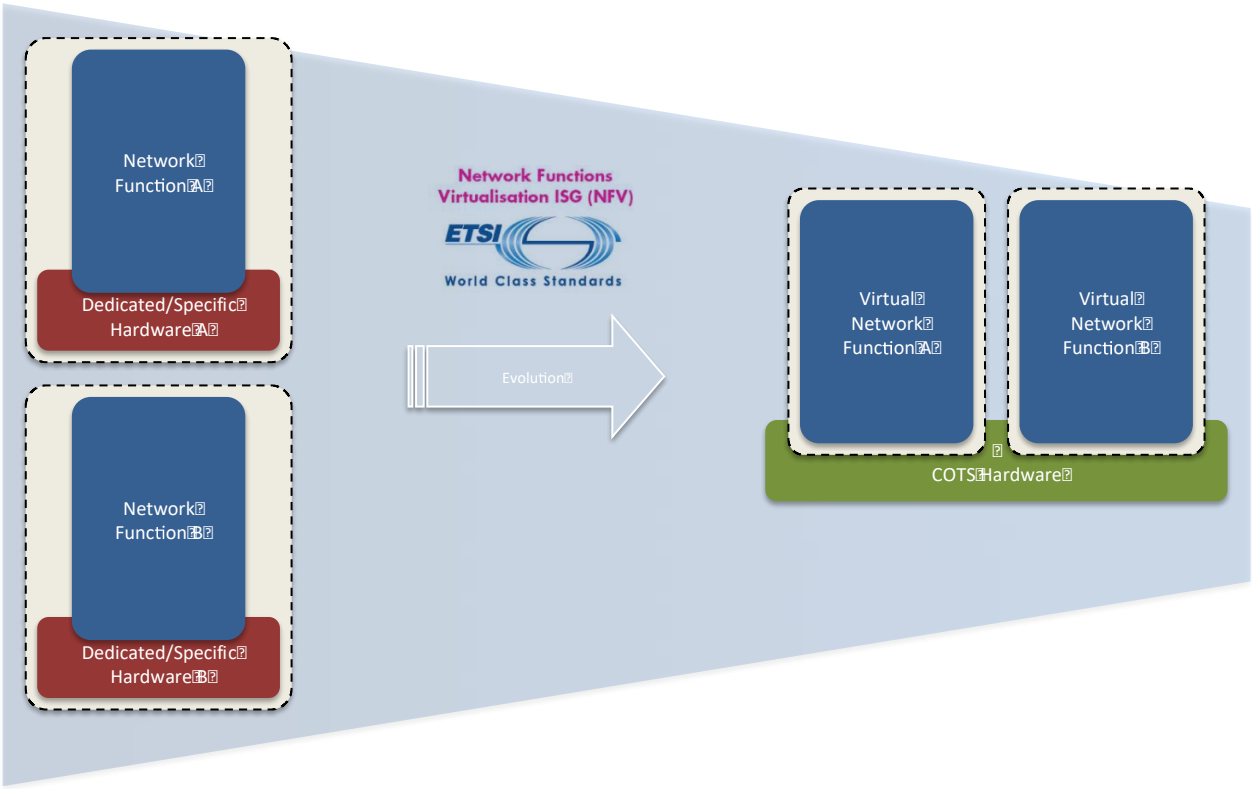


Figure 45: ETSI NFV concept.

NFV is seen as a critical architectural building block to give mobile operators the required tools to address the massive data usage by their customers in the new wave of expected 5G use cases. NFV enables the "cloudification" of network functions (NFs). In this approach, the network function has to be implemented apart from dedicated/ specialized hardware, and has to be able

to run on top of COTS. Typical examples of VNFs are routers or firewalls, but they can also be mobile or fixed components, like P-GWs or eNBs. Figure 45 shows the NFV principle.

The "cloudification" of network functions can be further enhanced by using the management and orchestration environment. In such cases, the environment manages the entire VNFs lifecycle, performing not just the deployment and disposal, but also managing the runtime operations like migration or scaling, according, e.g. to the function load, making a more efficient use of resources. Such a platform is also able to orchestrate combinations of VNFs, creating complex *Network Services* (NS).

Figure 46 depicts a simplified version of the full ETSI NFV architecture. On the left side, the execution and control (runtime) operations can be seen, whilst the right side shows management and orchestration. The bottom left shows the virtual infrastructure, which comprises hardware resources (COTS), the virtualization layer (e.g. hypervisors) and virtual resources (e.g. VMs, Containers, Disk, VLANs). VNFs run on top of one or multiple VMs and use network resources. On the top left, the Management Support Services (OSSs/BSSs) interact with the Management and Orchestration (right side) and with the VNFs. In the right, on the bottom, the VIM (e.g. OpenStack) interacts with the NFVI to manage resources. On the top right, the Orchestrator and Management module manages the complete lifecycle of VNFs and orchestrate NSs.

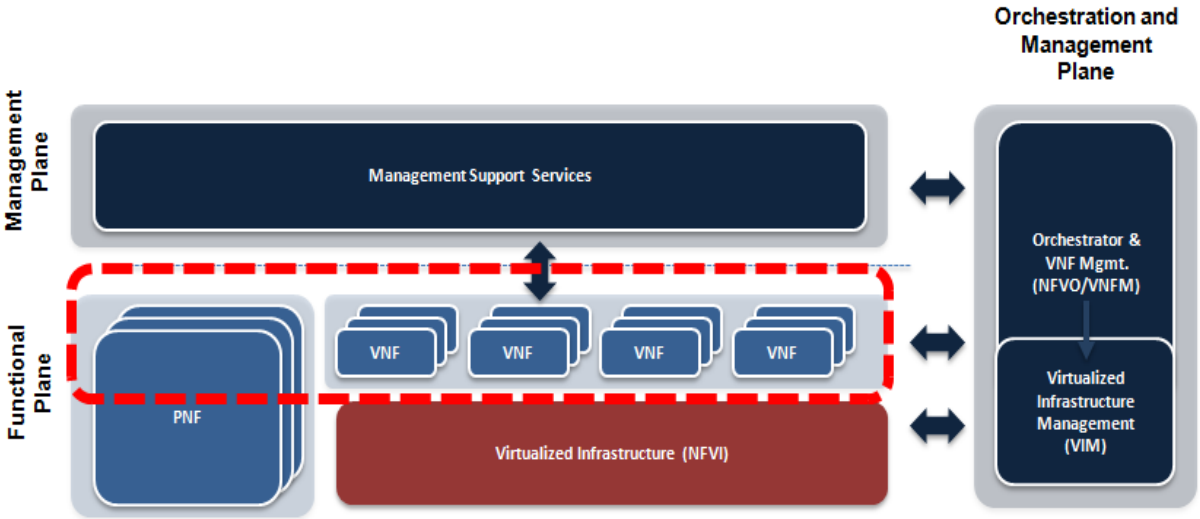


Figure 46: ETSI NFV architecture.

6.3.2 ONF SDN architecture

SDN as core technology in 5G enables operators to simplify service and networking provisioning with visibility across multiple network layers, heterogeneous switch hardware and multi-domain networks. The centralized control of networks SDN allows operators to have greater operational flexibility. For example, SDN enables much higher levels of network automation via programmability, which will make 5G networks more adaptable enabling improved service agility. SDN allows operators to dynamically adapt to changing network conditions to support either

rapid scale up/down of services or the provisioning of new services, which improves operating expenses (OPEX) by shortening the time to new revenue.

SDN splits the network functions into 3 parts: the user-data plane, the controller plane and the control plane. The user-data plane (or Data Plane) is composed of simple switching Network Elements (NEs), responsible to forward the user traffic according to the basic commands received from the north (controller) interface. The Controller Plane is composed by SDN controllers, which provide basic commands to the south (user-data) and high-level APIs to the north (control or Application Plane). Controller APIs are abstractions used to program the network, speeding up the creation of new services. Figure 47 shows the SDN concept, assuming that the starting point is not a traditional NF relying on a dedicated/specific hardware, but an already virtualized NF (VNF), as described in the section above.

Overall, the NEs forwarding process is controlled by the applications, which use high-level APIs provided by the SDN controllers. The SDN controllers interact with the NEs through low-level southbound APIs to enforce basic forwarding rules, using protocols like CLI, Netconf or OpenFlow. SDN controllers provide a northbound interface, abstracting the programmer from the network details and making simpler the network service creation. This is one of the key advantages on the SDN model.

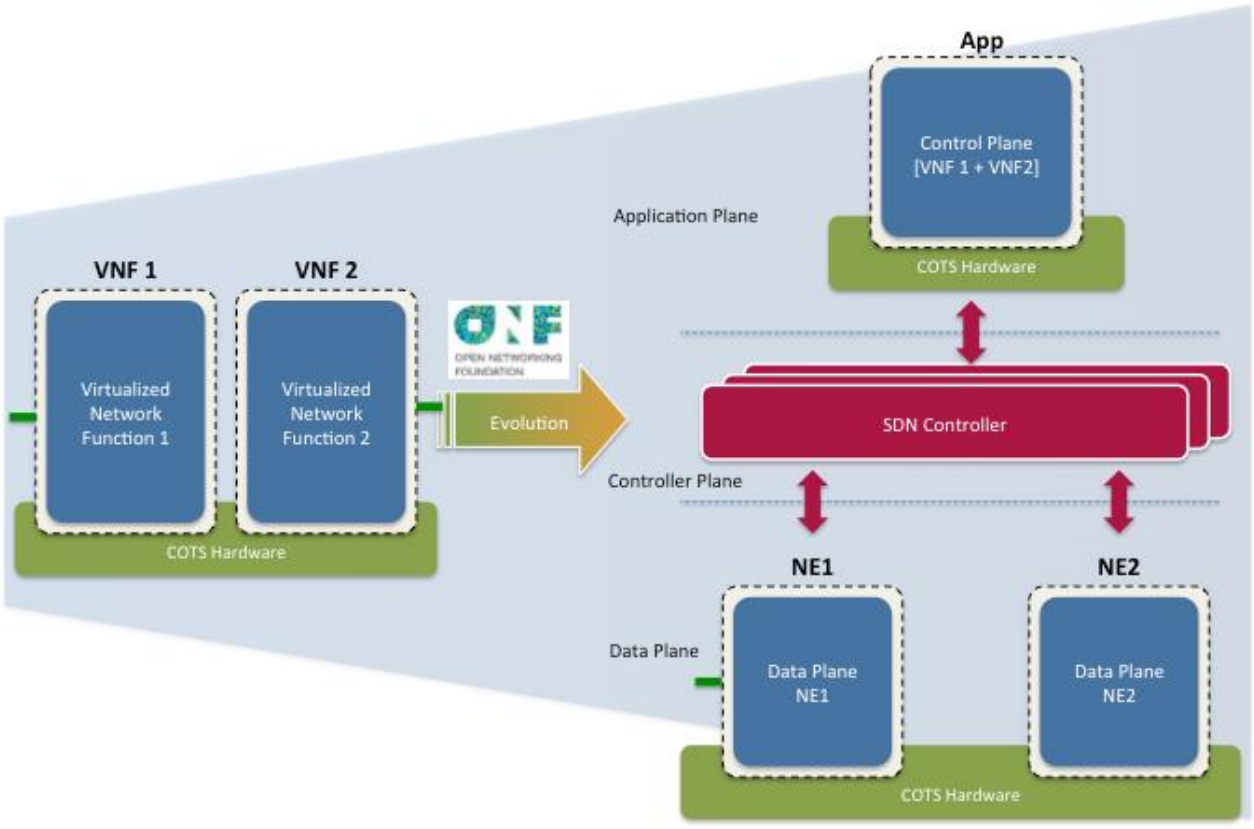


Figure 47: ONF SDN concept.

Figure 47 above shows that, when we move to an SDN world, the transformation to the SDN paradigm has not to be on a 1:1 basis. This means that a VNF may not move directly to the SDN paradigm by splitting itself into 3 parts. In fact, a single VNF can result into multiple applications and/or multiple NEs (N:N). To make this clear, an example of "SDNification" is provided below.

In this example (see Figure 48), the original scenario is an already set of virtualized routers. Each router has an IPv4 and IPv6 forwarding feature as well as the traditional routing protocols, like OSPF, BGP, etc (for simplicity, we can forget other features). On top of that, operators can configure the routers and build services like IPv4 connectivity, IPv6 connectivity or enterprise VPN, among others. Moving to the SDN paradigm, control and forwarding planes are decoupled. On the bottom, NEs are deployed with forwarding-only capabilities, applied according to the policies provided by the control plane. On the top is implemented the control logic, e.g. OSPF, BGP, as well as all the service logic that permits e.g. the provisioning of a new Access, VPN or VPN site.

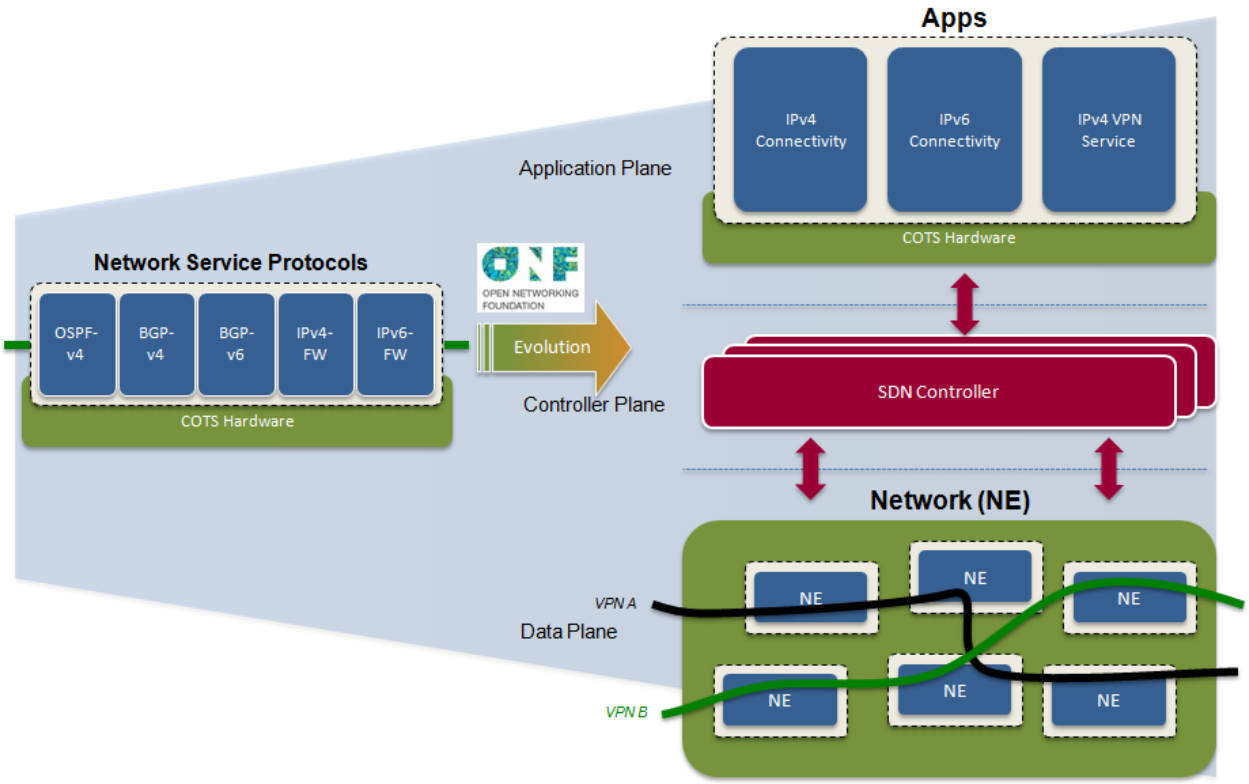


Figure 48: SDN: Example of Transport Network.

In this particular case, there is an N:N mapping between the control plane (SDN Apps) and the user-data plane (NEs); i.e. multiple applications implement different services on top of a common set of NEs. NEs can be virtual software switches (e.g. OVS - Open vSwitch) or more performing hardware specific switches.

6.3.3 Converged NFV+SDN architecture

The advent of SDN and NFV are seen as the critical architectural building blocks to give mobile operators the necessary tools to address the massive data usage of customers in the new of wave of expected 5G use cases. The software focus driving the confluence of NFV and SDN will allow mobile operators to be more responsive to customer demands either dynamically or through on-demand service provisioning via self-service portals.

NFV and SDN were developed by different standardization bodies. However, they are complementary technologies and as a result are often collectively referred as NFV/SDN. Although they have substantial value when exploited separated, when combined they offer significant additional value. The NFV technology brings the possibility of creating new network functions on-demand, placing them on the most suitable location and using the most appropriate amount of resources. However, this requires the SDN to be able to adjust the network accordingly, making network (re)configuration (i.e. programmability) and sequencing functions (chaining).

As NFV and SDN come from different SDOs (Standard Developing Organizations), none of them combines both architectures. Thus, in [18] there is an attempt to propose such combination, taking the ETSI NFV architecture as a starting point and introducing the SDN paradigm.

Starting from the ETSI NFV architecture depicted in Figure 46, we can see the VNFs (in the left part) which represent the virtualized network functions. According to the SDN model, shown in Figure 47, a monolithic VNF is separated into three parts.

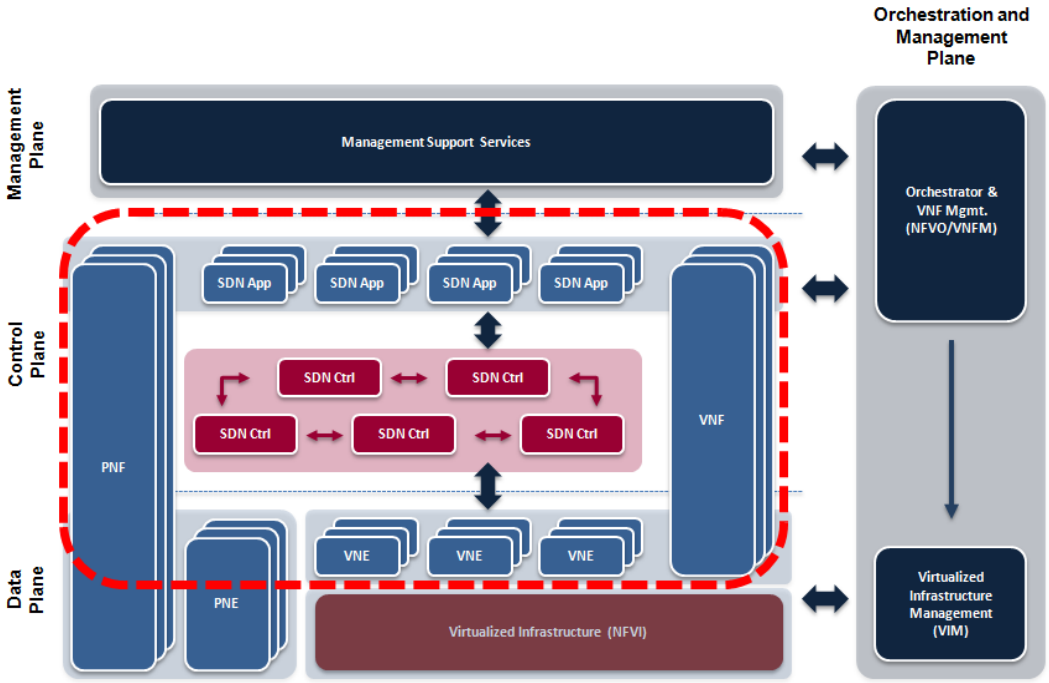


Figure 49: NFV and SDN combined architecture [18].

Note: It should be noted that the red dashed box in Figure 46 and Figure 49 identify the replacement derived from the SDN model.

The combined architecture is not very different from the one defined by the ETSI NFV with the exception of changes regarding the extra additional layers. The naming of the components is another issue that we needed to decide, since similar boxes have different names, depending on whether you take a look at them from the NFV or the SDN perspectives.

In order to consider "legacy" components (non-NFV, non-SDN), we kept the physical NFs in the leftmost side of the dashed red square of the architecture, meaning that we may have *Physical Network Functions* (PNF), which do not apply the NFV and the SDN paradigms. Similarly, we may have virtualized NFs (VNFs), with no SDN capabilities, for which we kept the *Virtual Network Functions* (VNF) naming, as shown in the rightmost side of the dashed red square in Figure 48. In the middle of Figure 48, all components are SDN-aware, meaning that they are split into three layers. In the bottom layer (user-data plane), you may have physical or virtualized *Network Elements* (NE), which can be *Physical Network Elements* (PNEs) or *Virtual Network Elements* (VNEs), respectively. In this case, we opted by names coming from the SDN world, since they describe more clearly the roles they are performing. On the controller layer, we assumed that we may have multiple controllers and at different levels, naming them all *SDN Controllers* (SDN Ctrl). Finally, for the control layer, we used the naming *SDN Application* (SDN App). In this case, we do not specify if it is virtual or not, but it can be both, although we believe that this layer will be mostly populated by virtual applications, considering that hardware specific utilization is declining.

6.3.4 3GPP C-RAN

The *Cloud Radio Access Network* (C-RAN) is a new mobile network architecture that allows operators to meet the needs of increasingly growing users demands and 5G requirements for achieving the performance needs of future applications. C-RAN intends to move part of the eNB to local clouds, taking advantage of the cloud technologies, elastic scale, fault tolerance, and automation. Using this architecture, a minimum set of hardware critical functions is kept on the physical *Remote Radio Head* (RRH) components, while the remaining virtualized (CPU intensive) part, the Base Band Unit (BBU), is deployed and concentrated in small local datacenters for a large number of cells. Figure 50 depicts the basic concept.

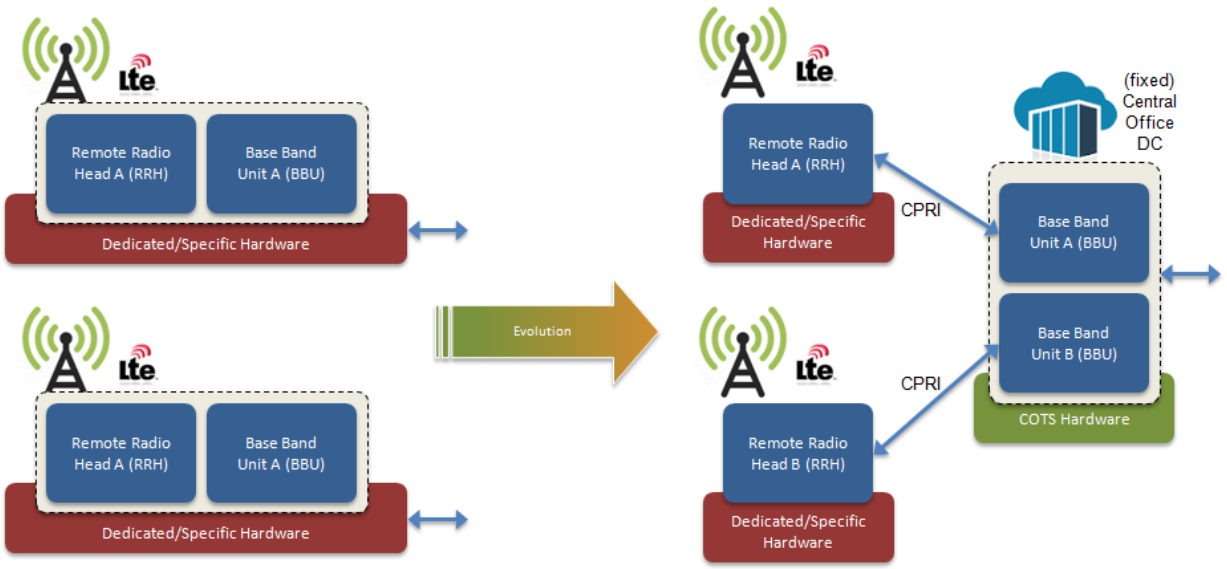


Figure 50: 3GPP C-RAN basic model.

This approach has multiple advantages: 1) It brings resource isolation for C-RAN functions that are now software defined and can be deployed and scale using Commodity hardware and off-the-shelf standard components; 2) it reduces the cost per deployed cell, which is an important aspect, as next mobile generations will reduce the cell coverage (and therefore increase the number of cells) as a way to increase bandwidth; 3) by centralizing BBUs in a local DC, the required amount of resources is lower than for traditional scenarios, as today cell-sites are provided with processing power for peak hours; 4) X2 interfaces, among eNBs, become internal links in top of the rack DC network, reducing latency times on mobility situations; 5) it leveraged C-RAN SDN controllers [16] to improve network capacity and flexibility by providing dynamic control, fine grained resource allocation and better latency control to meet QoS requirements.

There are different C-RAN solutions, which propose different functional splits among the cell-site part (RRH) and the DC part (BBU). This has impacts on the interface between RRH and BBU and on the amount of bandwidth required for that. For this reason, this issue is of extreme importance.

6.3.5 ETSI MEC architecture

Mobile Edge Computing (MEC) is a new technology platform, which is currently under standardization by the ETSI MEC Industry Standard Group (ISG). MEC will offer application developers and content providers cloud computing and an IT service environment to deploy and run applications at the edge of a mobile network.

MEC is recognized by the 5G PPP as one of the key emerging technologies for 5G systems (along with NFV and SDN). The MEC environment will be characterized by:

- Proximity
- Ultra-Low Latency

- High Bandwidth
- Real-time access to radio network information
- Location Awareness

MEC provides a set of services that can be consumed by applications running on the MEC platform. These services can expose information from real time network data (e.g. radio conditions, network statistics) and from the location of connected devices to consuming applications. Exposing these types of information can be used to create context for applications and services with the potential to enable a new ecosystem of use cases, such as dynamic location services, big data, multimedia services and many more, with a significantly improved user experience.

MEC can take advantage of NFV deployments, due to the complimentary use of cloud infrastructure, virtualization layers and management and orchestration components. Operators are increasing looking at virtualization of their mobile networks and, in particular, C-RAN. In this context, operators will have to create small DC infrastructures at the edge, in order to support RAN processing requirements. Having IT infrastructures on the edge, will enable operators to use unused resources to support 3rd parties to deploy applications in an edge computing environment, without having to invest on additional resources especially for that purpose. As a consequence, edge computing becomes cheaper, making it more appealing to network operators, content providers and developers.

The same concept can be applied to fixed networks, where virtualization is being adopted. In fact, for network providers with mobile and fixed operations and traditional central offices can be used to concentrate both mobile and fixed access, making those places good candidates to provide edge computing applications for all customers. Figure 51 outlines this vision.

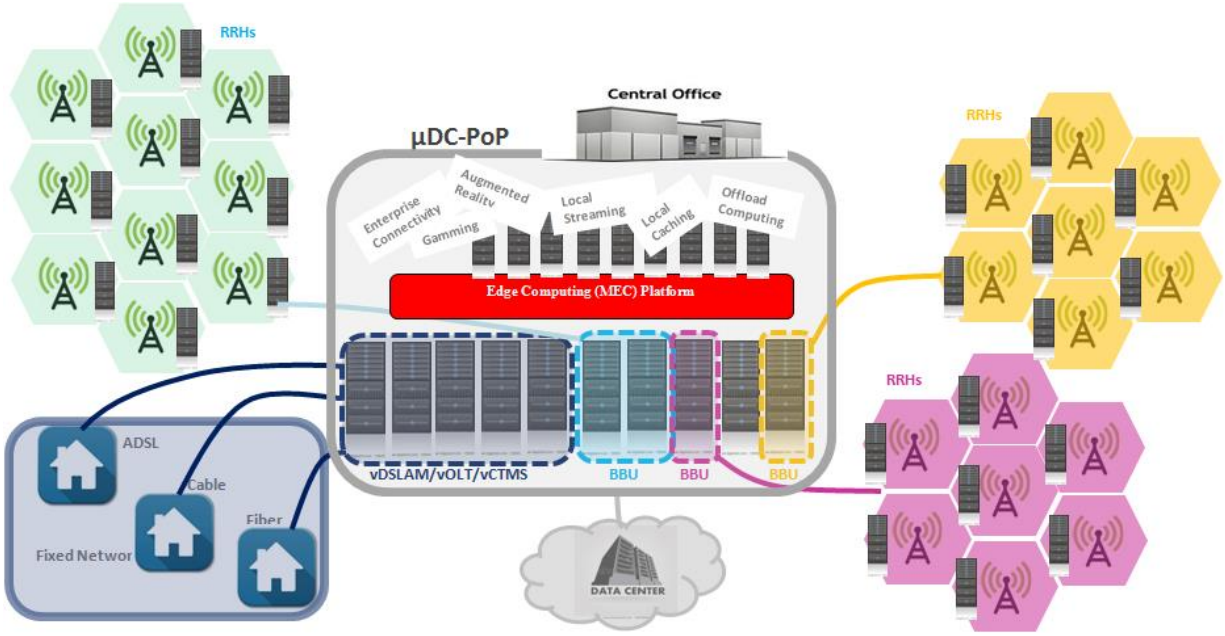


Figure 51: MEC mobile/fixed vision.

The edge computing architecture includes the MEC Platform as the central point. This platform allows 3rd party applications to register and get authenticated/authorized, in order to access a set of services, available via the respective APIs. Using the APIs, applications can request the platform to make traffic offload (TOF) to serve customers from the edge, or access to other information provided by the operator.

The NFVI and VIM defined by the NFV are essentially reused to provide a virtual environment this time for the applications. A similar thing happens to the management and orchestration layer, which can be used to manage the applications lifecycle and orchestrate them according to the best interests of end users and application providers. In the management layer, an additional manager can be found, which is dedicated to manage the MEC platform itself, including services and respective APIs. The Figure 52 shows the proposed architecture for MEC.

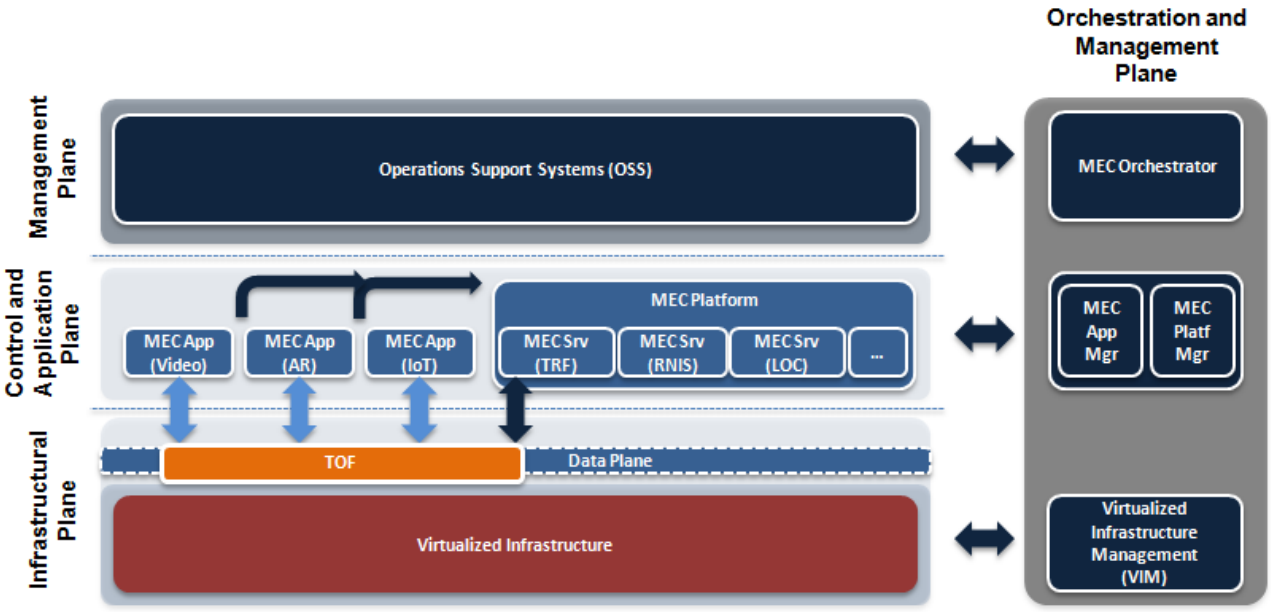


Figure 52: MEC architecture.

It is clear that the MEC will be a key architecture component of 5G in the advent of mobile edge cloud computing. It is also complementary to the key 5G technology building blocks of SDN and NFV through a programmable paradigm. The MEC paradigm will be an important element of 5G as it plays a pivotal role in offering increased monetization scope for service providers helping to transform mobile networks.

6.4 Cross domain architecture

6.4.1 Mobile Core

The Mobile Core is the part of a mobile network, which aggregates the traffic and signalling coming from the eNBs. It is composed of functional blocks dealing with the control and data plane functions. In LTE networks, the Mobile Core is called *Evolved Packet Core* (EPC) and has three

main functional entities: MME, S-GW and P-GW; additionally, there is a repository: HSS. The MME is a control plane function, while the S-GW and P-GW, although essentially executing data plane functions, they also play some signalling roles. The expected evolution of the Mobile Core is to follow the SDN concept of having a clear split between control and data planes.

Figure 53: depicts this model, which is under study by 3GPP in [32].

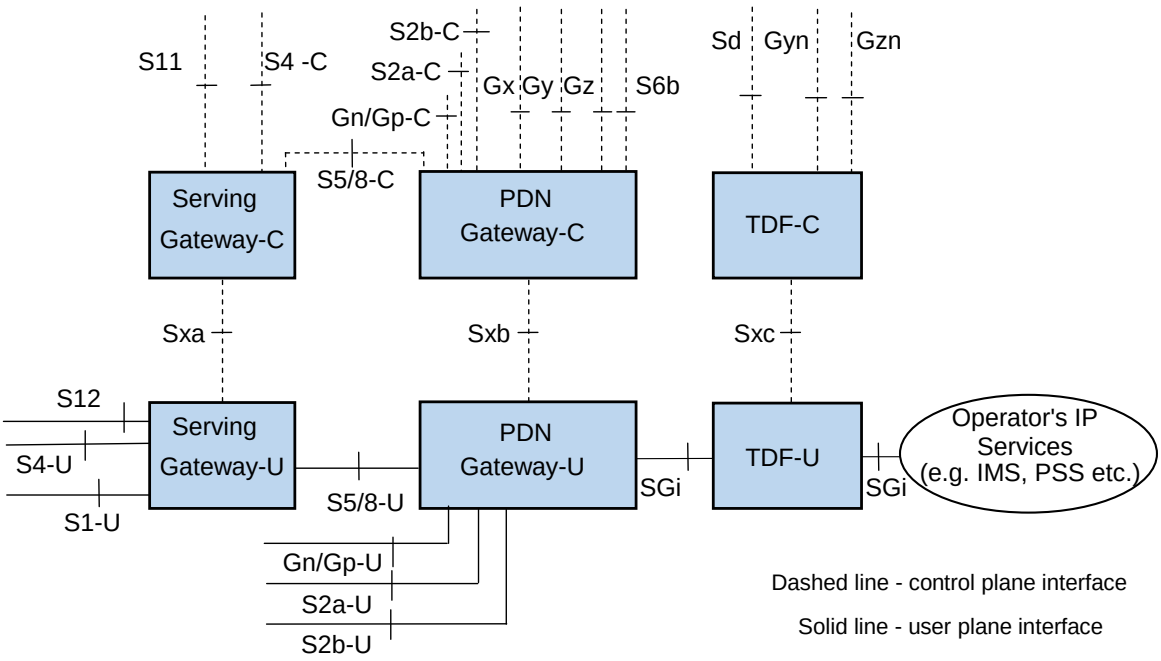


Figure 53: 3GPP Mobile Core [3GPP-23.714].

Although having this Control and Data plan separation, all the currently existing interfaces remain untouched. New interfaces will be created between the separated control plane and data plane functions. Traditional protocols like Diameter can be used for that, or other more SDN-like may be introduced like OpenFlow. Another trend could also lead 3GPP to evolve the architecture by merging some of the control functions into a single functional entity; the same may happen for the data plane entities.

6.4.2 Central-DC

Central DCs are big central infrastructures where most public cloud services are located. Comprising a huge amount of resources (CPU, Storage, Networking, etc.), they are extremely efficient since they benefit from scale factors, reducing costs in aspects such as space, cooling, operations, etc.

Central DCs hold today the majority of cloud applications. With the NFV and SDN advent, many DC networking components, such as switches, routers, firewalls or load balancers, can be already virtualized and controlled by SDN Controllers.

The caveats for large central DCs are the distance to the end users, which increases the latency, having significant impacts on some applications, and waste of bandwidth along the path. On the other hand, the deployment of bandwidth-intensive applications on such environments may cause bottlenecks on DCs itself and in general on the network providers.

6.4.3 Virtualization and Cloud Computing

The key principle that has enabled the resource and service abstraction from the underlying physical resources is virtualization. Cloud computing has allowed for data center owners to virtualize workloads in virtual machines and then benefit from the consolidation of workloads onto fewer physical machines. The manageability of workloads that are abstracted from the physical infrastructure gave rise to many beneficial features such as high-availability, scale-up and scale-out workloads. Once abstracted sufficiently from the hardware, the workloads could be moved between physical machines to ensure high-availability services. Virtual machines have enabled the first stage of consolidation but there are now many approaches looking into making the Virtual Machine overhead smaller, thereby making it possible to have more workloads on a single machine and to remove the inefficiencies of having Operating Systems that are over-provisioned for the types of workloads that are being run. Containers and chroot jails are not new but there is a lot of interest and work going into minimizing the footprint of virtual workloads. Unikernels are light-weight processes that are designed and compiled for specific purposes, thereby further removing memory management sub-systems and other features that a multi-tasking operating system would normally need to handle.

These light-weight virtual platforms allow for more efficient processing of the workloads. ClickOS [21] is a specialized MiniOS (lightweight Xen VM) instance that is optimized for high packet throughput.

PART B – SECURITY FRAMEWORK

7 Security Framework Design

Any architecture which relies on real-time Cloud infrastructure is susceptible to security hazards. All the more so when the architecture relies on virtualization of network functions. Simplistically, two security aspects intervene when virtualization in a cloud architecture is concerned. The provider must ensure security and privacy on the provided infrastructure, while the user must take measures to secure their application(s) by strong passwords, encryption and authentication. The four phases of control which are normally applied to the cloud are logically depicted in Figure 54. Deterrence can be obtained by bringing to the users' attention the potential risk of not securing their data and punishment in case of attempted violations. Prevention control reduces the vulnerabilities e.g., by reducing the number of possible targets and/or hiding some of them. Detection takes action on incidents already in progress. Finally, the correction phase supplies a toolbox for fast recovery from the attacks. Clearly, the provider's objective is to try to stop the potential attacker at the earliest possible phase. All these phases should be adopted by NFV control, by taking into account the new vulnerabilities.

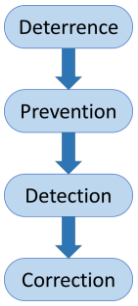


Figure 54: Cloud security control phases

Measures on the provider's side further include having authenticated users and tracking and monitoring actions, especially any unusual activities. Moreover, the remotely stored data should be isolated from any potential undesired access, even if multiple copies are stored and/or cached in CDNs. In the Appendix, we provide detailed security challenges and hazards that the Superfluidity architecture addresses.

A key to ensuring security is understanding what a certain processing block does before it is instantiated. By relying on such information, operators could automatically determine at install time whether it is safe to run certain processing blocks. Accordingly, the Superfluidity security framework will rely on symbolic execution techniques (Task 6.3). However, for symbolic execution to be applicable, we need to convey to the symbolic execution engine what all the basic processing blocks, defined in Superfluidity, actually do. Specifically, instead of adopting a traditional approach which performs verification triggered by data plane changes, our strategy

relies on applying the formal verification process before the actual update of the data plane. This choice is motivated by the fact that performing verification upon changes in the data plane is becoming more and more challenging due to (i) the time scale in which data planes are changing (i.e., data planes are changing much more rapidly) and (ii) to the complexity of detecting abnormal behaviour. Therefore, according to our approach, an updated network model is compiled to the data plane only if it passes an a-priori verification phase, i.e., it is guaranteed to respect the operator's *policy*. In section 8 we will elaborate on the formal verification approach that Superfluidity adopts.

In addition to the security issues related to software processing that we discussed above, other security hazards exist in various layers of the 5G architecture. In sections 9 and 10 we explore the vulnerability of the C-RAN architecture to eavesdroppers. In particular, we analyse the secrecy rate and outage probability in the presence of a large number of legitimate users and eavesdroppers. Specifically, we analyse the asymptotic achievable secrecy rates and outage, when only statistical knowledge on the channels is available to the transmitter. In addition, we examine novel physical layer schemes to secure the communication with minimal rate loss and complexity.

8 Formal Correctness Verification

It goes without saying that modern network data planes have reached a level of complexity that renders the usual process of "manual" administration very inefficient. Humans make errors. With this in mind, we should strive to automate most processes that form the evolution of a certain network deployment. Secondly, human intervention is slow compared to the timescales at which data planes change, meaning that human intervention is possible only in the case of a small subset of the events occurring inside a large network. All this is not new and there is a strong drive in the research community to come up with tools that help in this respect.

Dataplane verification tools such as Header Space Analysis [110], Network Optimized Datalog [111] have recently been proposed and are already used in production. The Superfluidity project has also proposed the SEFL modelling language and the Symnet symbolic execution tool for networks described in SEFL [114].

Dataplane verification is different from traditional network testing: it does not require concrete traffic to be injected through the network. Instead, an abstract model of the network dataplane is used and the desired network policy (e.g. reachability, isolation, loop freedom) is then verified against this model and then every violation of the policy is traced back to the actual network configuration. This approach has proven successful in detecting data plane problems, but has a number of limitations. Firstly, correlation data plane policy violations with the configuration parameters that enabled those violations is very difficult. Secondly, dataplane verification is reactive: problems can be detected only after the dataplane configuration is applied and produces effects: there is a window of time after the update has been deployed and before the dataplane has been formally verified when the bugs are live and can produce costly damage.

In Superfluiduity we propose an approach to generate provably correct networks as follows. Instead of performing formal verification after the data plane has changed, we use formal verification before the data plane is deployed. In other words, we intercept the output of the control plane (i.e. the FIBs computed by the routing protocols or the output of SDN controllers), and build a network model with them. If this *network model* (1) is valid, according to the operator's *policy* (2), it is then compiled to a *concrete data plane* (3) update which is now guaranteed to respect the desired operator's policy.

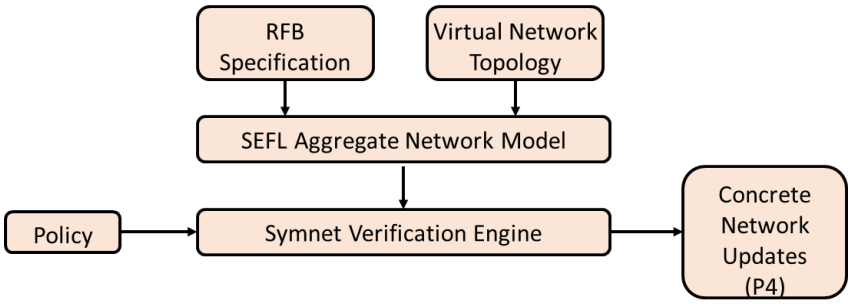


Figure 55: Approach to generate provably correct 5G network data planes

(1) To express the abstract network model we chose SEFL models (as described in section 4.5). SEFL is a high-level modelling language for network processing that bears resemblance to the more popular imperative programming languages (e.g., C). Network processing is modelled as a sequence of SEFL instructions performed on a symbolic representation of packets. RFBs are translated to their corresponding SEFL models. This, together with the virtual network topology (virtual ports and links), completes the SEFL model of the network.

(2) Once the SEFL network model is determined, it is verified against the policy that will be deployed by the time the concrete network update takes place. The verification process takes as input the policy specification and the abstract SEFL network model.

A network policy language is expected to be able to express properties related to: (i) reachability, i.e. the existence of a path between two or more nodes in the network which satisfies given constraints on the packet header); (ii) maintenance, i.e. on each path between two or more nodes, certain packet header constraints are never violated; (iii) more complex combinations of the former two.

With this in mind, we consider Temporal Logic as a formal framework for specifying policies. Temporal Logic introduces the path quantifier operators: (i) “on all paths”, (ii) “there exists a path” as well as the temporal operators: (iii) “sometimes”, (iv) “always”.

A policy in a Temporal Logic combines quantifiers with path operators as well as the standard boolean operators. For instance:

forall G (dest-ip = 69 -> forall F (port = IDS))

expresses that, on all paths, it is always the case that IP packets with destination port 69 will eventually reach the IDS (Intrusion Detection System). Moreover, the second “forall” quantifier ensures that there are no alternative paths for such packets which do not lead to the IDS.

Model checking algorithms are generally deployed for the verification of (concurrent) systems. However, the output of a symbolic execution engine can be naturally viewed as an unfolding of such a concurrent system (under verification). Symbolic execution algorithms can naturally be used for network policy verification.

Model checking temporal logics in the general case is known to be computationally hard. There are two main reasons for this:

(i) The “state-explosion problem”. In our particular case this problem is naturally addressed by SEFL: we do not perform symbolic execution on actual code, and we require that each execution path corresponds to one packet path from the network.

(ii) The hardness of model-checking. By carefully adding restrictions to how path quantifiers and temporal operators can be used, we can obtain verification procedures which are linear w.r.t. the symbolic execution output.

For instance, the CTL language (Computation Tree Logic) is obtained by requiring that each temporal operator is directly preceded by a path quantifier. The policy from the previous example is a CTL formula. CTL is known to have linear-time model checking complexity.

(1) As a first attempt to compile concrete, executable network processing code from the SEFL abstract model we chose to support P4 [112], an up and coming dataplane programming language supported by switches in hardware.

P4 is the successor of OpenFlow. The P4 network programming language offers constructs that enable deployment of much more sophisticated algorithms (arbitrary protocol parsing, stateful processing, …) on boxes acting at the data plane level.

The Symnet tool [114] verifies that a model correctly implements the CTL-based policy provided by the network operator by performing symbolic execution. Symbolic execution works by injecting symbolic packets (which are essentially compact representations of packet sets) at various locations (network ports) and then tracing their evolution throughout the network. Such a trace is referred to as an *execution path*. Whenever a branch condition is encountered, exploration of the current execution path forks and both alternatives are further explored: on one path the branching condition is applied, whereas on the other path its negation is applied. The symbolic execution process halts when there are no SEFL instructions left to be executed on any path. The result is the set of all possible execution paths through a network model, this essentially covering all the cases in which an arbitrary packet can be processed and forwarded by a data plane.

If the set of conditions specified by the policy holds on all paths, it means that no packet can trigger unwanted behaviour inside the network. So far we have a working implementation of the symbolic execution algorithm. This is going to be the foundation on top of which the policy validation algorithm will be implemented.

If paths that violate the policy were explored, we are left with two options:

- Provide the execution path history, together with the symbolic values of the fields and with their corresponding path constraints. This will inform what network path breaks the policy and help the network operator to understand exactly what part of the model needs to be amended.

- Automatically amend the model such that packets belonging to the path breaking the policy will be dropped or forwarded to a different processing pipeline, such that the danger of triggering unwanted behaviour is mitigated.

Once the policy is validated by the model, we are only left with the task of building a SEFL-to-P4 instruction translator. Arguably this translation process can also infiltrate errors in the concrete data plane code (P4) that it generates if the equivalence between the SEFL and P4 code is not correctly enforced. Nevertheless, this is a problem that is limited to the translator itself which rarely, if ever, changes – as opposed to the ever changing network configuration.

8.1 SEFL- P4 translation overview

The P4 packet processing language defines the following constructs:

Headers

Header definitions describe packet formats and provide names for the fields within the packet. Headers can be of any length and they can be referred to by using symbolic labels such as: “source”, “destination”, etc. These language features map directly to the SEFL’s notion of values, which are addressable using concrete packet offsets (via memory tags see D 4.5 for more details) or symbolic names. Neither P4 nor SEFL support indirection in the way pointers do, thus greatly reducing the complexity of the code.

Parsers

The P4 parser is a finite state machine that traverses an incoming byte-stream and extracts headers based on the programmed parse graph. In correspondence, SEFL allows the definition of the packet structure using the following instructions: **CreateTag** – which gives symbolic names to concrete header offsets, **Allocate** – which allocates space for a header fields; this instruction requires the offset and the width of the fields to be specified. If two Allocate instructions refer to overlapping memory spaces, the structure of the packet is considered invalid. These instructions allow for the definition of arbitrary header spaces, thus fully supporting P4’s notion of parsers.

Tables and control flow

P4 tables contain the state used to forward packets. Tables are composed of lookup keys and a corresponding set of actions. To perform more complex network processing, tables can be chained together to form processing pipelines of different sizes. Tables are generated from SEFL’s **Fork – Constrain** construct. The **Fork** instruction replicates the current execution path to a variable number of instruction blocks. If every such instruction block begins with a **Constrain** instruction (which imposes a path constrain), the table lookup logic is performed, execution continuing only in the case of those paths for which the constraint was valid. In order to form a processing pipeline in SEFL one can chain together SEFL instruction blocks using the **Forward** instruction.

Actions

Actions in P4 describe packet field and metadata manipulations. In P4, metadata are fields that do not directly pertain to the packet itself, but hold auxiliary information required by the processing pipeline. In SEFL, values referenced using symbolic keys can hold arbitrary values that can then be modified using **Assign**, or that can influence the processing behaviour either by using the conditional instruction **If** or by applying path constrains that refer to these metadata fields using **Constrain.**

8.2 Runtime checking

Our approach towards generating provably correct data planes relies on the existence of exact models of the desired dataplane functionality. Unfortunately, there can be situations when a complete model of the network is not known - because of proprietary, closed-source network functions being deployed, for instance - or it is simply too difficult to build. In such cases, a static verification tool is of limited value since those policy rules that target this unknown behaviour cannot be verified.

Superfluidity tackles this indeterminate behaviour by combining two approaches (i) the more common approach taken by network operators which detects policy breaches by deploying dedicated boxes that perform traffic analysis and filtering, and (ii) a different approach which relies on the novel static analysis previously presented and verifies that the expected output of incoming traffic to any "black-box" complies with the legitimate behaviour of the black-box.

In the first approach we are proposing a novel universal anomaly detection algorithm which is able to learn the normal behaviour of systems and alert for abnormalities, without any prior knowledge of the system model, or of the characteristics of the attack. The suggested method utilizes the Lempel-Ziv universal compression algorithm in order to optimally give probability assignments for normal behaviour (during learning), then estimate the likelihood of new arriving data (during operation) and classify its behaviour (normal or anomalous) accordingly.

However, anomaly detection has two limitations: (i) no strong guarantees are provided, and attacks are often detected after they have taken place and caused damage (ii) on the other hand, an overzealous anomaly detector towards reducing these security risks might lead to deploying expensive packet processing and filtering in unnecessary cases.

We propose a provably-correct solution to achieving safety in networks containing black boxes. We start with the observation that dynamically deploying network functions that process packets at line rate speed is a reality enabled by the recent advancements in SDN technologies (e.g P4, OpenFlow, Open Packet Processor). With these in mind, our approach consists of the following steps:

- The policy verification algorithm using SymNet (see above) is performed with the following addition: whenever a "black-box" is reachable by traffic, it's output can be anything (a packet with all the headers symbolic).
- Whenever a policy breach is encountered during static analysis that is traceable back to a "black-box" a match rule for this traffic is determined and instantiated using the SDN controller.
- Finally, **guards** will be automatically generated and inserted *around* the black box element when possible to guarantee that desired properties hold.

The benefits of this solution are twofold:

- The process of determining potential policy violations is automated and is triggered whenever new hardware is deployed or the policy is changed.
- We make sure the additional cost of packet processing due to runtime policy enforcement is kept at a minimum by computing the minimal set of filtering rules needed to be deployed to mitigate the potential risks.

To make the discussion more concrete, consider the example network in Figure 56 where there is a black box on the bottom path between A and C. Assume the user wants reachability for HTTP traffic between A and C, and all other traffic should be dropped. To generate the guards, we will use symbolic execution:

1. *Ensuring isolation*: we use a black-box model that allows all traffic, see what packets arrive at C and then derive the constraints that should be added at A to ensure proper isolation.
2. *Ensuring reachability*: here we start with the augmented model from step 1, and use a model for the black box that drops all traffic (i.e. assume the worst case when the black box is configured to drop HTTP traffic). To ensure reachability we can add a forwarding rule at B to forward HTTP traffic from A to C. If the alternative A-B-C path does not exist, we can only deploy the configuration and monitor packets at C to see if any HTTP traffic from A arrives.
3. *Ensuring header fields are not modified:* we can add checksums to the packets at A (changing the payload) and verify the checksums hold at C.
4. *Ensuring the black box cannot read packet contents*. An encrypted tunnel can be used between A and C for this purpose.

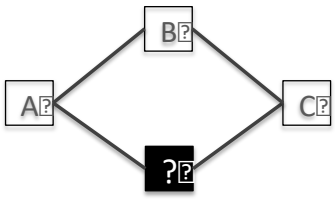


Figure 56: Runtime guards

To enable these algorithms we must add or remove constraints to execution paths and the associated models such that the desired properties hold. While solving simple cases like the one above is trivial, solving this problem in general is very difficult and we plan to use SAT-MAX solvers for this purpose (when given an unsatisfiable set of conditions, such solvers return a maximal subset that is satisfiable).

9 Secure RAN

Physical layer security promises very strong security guarantees at the price of transmission rate. Moreover, unlike traditional secure communications which use cryptographic primitives to secure point to point links, relying on physical later security can ensure baseline secure communication throughout the system, as long as certain guarantees on the signal-to-noise ratios are met (see for example [51]).

In this part of the project, we employ physical layer security to secure the RAN. Specifically, we consider a system with a base station serving multiple users, in the presence of multiple eavesdroppers as well (Figure 57). We study the secrecy rate and outage probability in the special yet highly practical case of Multiple-Input-Single-Output (MISO) Gaussian channels. We further focus on networks comprising a large number of legitimate users and eavesdroppers. Clearly, in such a system, one cannot expect to know the exact channels the eavesdroppers have. Hence, we analyse the asymptotic achievable secrecy rates and outage when only statistical knowledge on the wiretap channels is available to the transmitter. The analysis provides exact expressions for the reduction in the secrecy rate as the number of eavesdroppers grows as compared to the boost in the secrecy rate obtained when the number of legitimate users grows.

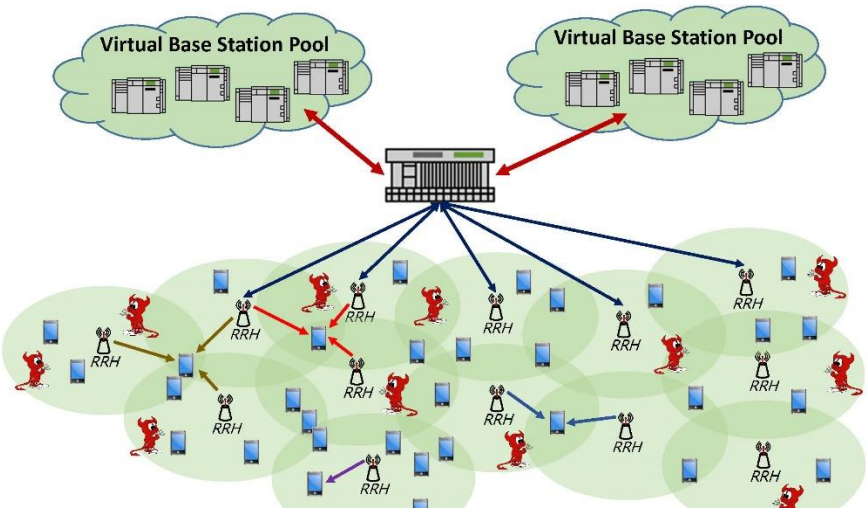


Figure 57: Multiple RRH serving multiple users, in the presence of multiple eavesdroppers

Rigorously speaking, we show that the secrecy rate in such a transmission scheme behaves like the ratio of Gumbel distributions (e.g. [52]). Tight upper and lower bounds on the limiting secrecy rate distribution are given. These bounds are tractable and provide insight on the scaling law and the effect of the system parameters on the secrecy capacity. In particular, the reduction in the secrecy rate as the number of eavesdroppers grows is quantified as compared to the boost in the secrecy rate as the number of legitimate users grows. Additionally, we prove that in the presence of n eavesdroppers, to attain asymptotically small secrecy outage probability with t transmit

antennas, $\Omega(n(\log n)^{t-1})$ legitimate users are required. To support our claims, we conduct rigorous simulations that show that our bounds are tight.

The explosive expansion of wireless communication and wireless based services is leading to a growing necessity to provide privacy in such systems. Due to the broadcast nature of the transmission, wireless networks are inherently susceptible to eavesdropping. One of the most promising techniques to overcome this drawback is to utilise Physical-Layer security. Physical-layer security leverages the random nature of communication channels to enable encoding techniques such that eavesdroppers with inferior channel quality are unable to extract any information about the transmitted information from their received signal [98][64][84].

Many recent studies have explored the potential gains in exploiting multiple antenna technology to attain secrecy in various setups. For example, in the case of a single eavesdropper, when the transmitter has a full Channel State Information (CSI) on the wiretap channel, it can ensure inferior wiretap channel by nulling the reception on the eavesdropper's end, thus, achieve higher secrecy rates [72][92][93]. When the user and eavesdropper are also equipped with multiple antennas, the optimal strategy is to utilize linear precoding in order to focus energy only in few directions, thus achieving the optimal secrecy rate [90]. In case that *only statistical information* on the wiretap channel is available, the optimal scheme is beamforming in the user's direction [86][93]. However, in this case, a *secrecy outage*, the event that at the eavesdropper is able to extract all or part of the message, is unavoidable. To mitigate this risk, one should consider transmitting Artificial Noise (AN) to further degrade the wiretap channel [70][79][80][83][96].

The secrecy capacity *at the limit of large number of antennas* was considered in [79][80]. Particularly, [79][80] studied the asymptotic (in the number of cooperating antennas) secrecy capacity, for a single receiver. [82] used Extreme Value Theory (EVT) to study the scaling law of the secrecy sum-rate under a random beamforming scheme, when the users and eavesdroppers *are paired* (i.e., each user was susceptible to eavesdropping only by its paired (single) eavesdropper). [71] considered the asymptotic secrecy rate, where both the number of users and number of antennas grow to infinity, while all users are potentially malicious and few external eavesdroppers are wiretapping to the transmissions.

In the presence of a single user and multiple eavesdroppers, where the transmitter has no CSI on the wiretap channels, a secrecy outage will definitely occur as the number of eavesdroppers goes to infinite [97]. On the other hand, when there are many legitimate users, and the transmitter can select users opportunistically, the secrecy outage probability is open in general. In particular, the exact asymptotically expression of the number of users required in order to attain sufficiently small secrecy outage probability, is yet to be solved. This study analyses this subtle relation between the number of users, eavesdroppers and the resulting secrecy outage probability. Specifically, we consider the secrecy rate and outage probability for the Gaussian MISO wiretap channel model, where a transmitter is serving K legitimate users in the presence of M

eavesdroppers. We analyze the secrecy outage probability as a function of K and M, and more importantly, the relation between these two numbers.

We assume that CSI is available from all legitimate users, yet *only channel statistics* are available on the eavesdroppers. As previously mentioned, when the transmitter has only statistical information on the wiretap channel, transmitting in the direction of the attending user is optimal when AN is not allowed. Moreover, in large scale systems, using AN may interfere with other cells, and probably would not be a method of choice even at the price of reduced secrecy rate. Beamforming to the attending user, on the other hand, is the de-facto transmission scheme in many MISO systems today. Therefore, we adopt the scheme in which at each transmission opportunity the transmitter beamforms in the direction of a user with favourable channel. We analyse the asymptotic behaviours of the secrecy rate and secrecy outage under the aforementioned scheme. In particular, our contributions are as follows: (i) we first analyse the secrecy rate distribution when transmitting to the strongest user while many eavesdroppers are wiretapping. These results are utilized to attain the secrecy outage probability in the absence of the wiretap channels' CSI. (ii) We provide both upper and lower bounds on the limiting secrecy rate distribution. The bounds are tractable and give insight on the scaling law and the effect of the system parameters on the secrecy capacity. We show via simulations that our bounds are tight. (iii) We quantify the reduction in the secrecy rate as the *number of eavesdroppers grows*, compared to the boost in the secrecy rate as *the number of legitimate users grows*. We show that in order to attain asymptotically small secrecy outage probability with t transmit antennas, $\Omega(n(\log n)^{t-1})$ users are required in order to compensate for n eavesdroppers in the system.

9.1 System Model

In this section, we use bold lower case letters to denote random variables and random vectors, unless stated otherwise. $V^{\dagger}$ denotes the Hermitian transpose of matrix V. Further, $|\cdot|$, $\langle\cdot\rangle$ and $P\cdot P$ denote the absolute value of a scalar, the inner product and the Euclidean norm of vectors, respectively.

Consider a MISO downlink channel with one transmitter with t transmit antennas, K legitimate users with a single antenna and M uncooperative eavesdroppers, again, with one antenna each. The transmitter adopts the scheme in which at each transmission opportunity the transmitter beamforms in the direction of the selected user without AN. We assume a block fading channel where the transmitter can query for fine channel reports from the users before each transmission, while having *only statistical knowledge on the wiretap channels*. Let $\boldsymbol{y_i}$ and $\boldsymbol{z_i}$ denote the received signals at user i and at eavesdropper j, respectively. Then, the received signals can be described as $\boldsymbol{y}_i = \boldsymbol{h}_i\boldsymbol{x} + \boldsymbol{n}_{b(i)}$ and $\boldsymbol{z}_i = \boldsymbol{g}_i\boldsymbol{x} + \boldsymbol{n}_{e(j)}$ where $\boldsymbol{h}_i \in \mathbb{C}^{t\times 1}$ and $\boldsymbol{g}_j \in \mathbb{C}^{t\times 1}$ are the channel vectors between the transmitter and user i, and between the transmitter and eavesdropper j, respectively. $\boldsymbol{h}_i$ and $\boldsymbol{g}_j$ are random complex Gaussian channel vectors,

where the entries have zero mean and unit variance in the real and imaginary parts. $\boldsymbol{x} \in \mathbb{C}^t$ is the transmitted vector, with a power constraint $E[\boldsymbol{x}^\dagger \boldsymbol{x}] \leq P$, while $\boldsymbol{n}_{b(i)}, \boldsymbol{n}_{e(j)} \in \mathbb{C}$ are unit variance Gaussian noises seen at user i and eavesdropper j, respectively.

The secrecy capacity for the Gaussian MIMO wiretap channel, where the main and wiretap channels, $\boldsymbol{H}$ and $\boldsymbol{G}$, respectively, are known at the transmitter, as given in [80][90]

$$C_s = \max_{\Sigma_x} \log\det(I + H\Sigma_x H^\dagger) - \log\det(I + G\Sigma_x G^\dagger) \quad (1)$$

with $tr(\Sigma_x) \leq P$. For the special case of Gaussian MISO wiretap channel, (1) reduces to

$$C_s = \max_{\Sigma_x} \log\det(I + \boldsymbol{h}\Sigma_x \boldsymbol{h}^\dagger) - \log\det(I + \boldsymbol{g}\Sigma_x \boldsymbol{g}^\dagger)$$

In both Gaussian MIMO and MISO, the optimal Σ_x is *low rank*, which means that to achieve the secrecy capacity, the optimal strategy is *transmitting in few directions*. Specifically, for the Gaussian MISO wiretap channel, the capacity achieving strategy is beamforming to a single direction, hence, letting $\boldsymbol{w}$ denote a beam vector, then $\Sigma_x = \boldsymbol{w}\boldsymbol{w}^\dagger$, [93]. Moreover, when the wiretap channel is unknown at the transmitter, it is optimal to beamform in the direction of the main channel, i.e., $\boldsymbol{w} = \widehat{\boldsymbol{h}} = \frac{\boldsymbol{h}}{\|\boldsymbol{h}\|}$, [86][93].

Accordingly, when beamforming in the direction of the user while the eavesdropper is wiretapping, assuming only the main channel is known to transmitter, the secrecy capacity is [49][97]:

$$\boldsymbol{R_s}(\boldsymbol{h}, \boldsymbol{g}) = \log\left(\frac{1 + P\|\boldsymbol{h}\|^2}{1 + P\left|\langle \widehat{\boldsymbol{h}}, \boldsymbol{g}\rangle\right|^2}\right) \quad (2)$$

Recall that in a block fading environment, $\boldsymbol{h}$ and $\boldsymbol{g}$ are random variables and are drawn from the Gaussian distribution independently after each block (slot). Hence, the distribution of the ratio in (2) and its support are critical to obtain important performance metrics. In particular, the *ergodic secrecy rate* (i.e., the secrecy rate when considering coding over a large number of time-slots) can be obtained by computing an expectation with respect to the fading of both $\boldsymbol{g}$ and $\boldsymbol{h}$. Similarly, a certain target secrecy rate R_s is achievable if the instantaneous ratio in (2) is greater than the matching value. On the other hand, a *secrecy outage* occurs if R_s is greater than the instantly achievable secrecy rate $\boldsymbol{R_s}(\boldsymbol{h}, \boldsymbol{g})$, and thus, the message cannot be delivered securely [49]. The probability of such event is $\Pr(\boldsymbol{R_s}(\boldsymbol{h}, \boldsymbol{g}) < R_s)$.

For clarity, let us point out a few statistical properties of the ratio in (2). In the denominator, the squared inner product $\left|\langle \widehat{\boldsymbol{h}}, \boldsymbol{g}\rangle\right|^2$ follows the Chi-squared distribution with 2 degrees of freedom, $\chi^2(2)$ (which is equivalent to the Exponential distribution with rate parameter 1/2), since $\widehat{\boldsymbol{h}}$ is normalized, rotating both $\widehat{\boldsymbol{h}}$ and $\boldsymbol{g}$ such that $\widehat{\boldsymbol{h}}$ aligns with the unit vector does not change the

inner product. Thus, the inner product results in a complex Gaussian random variable [76][94]. Similarly, in the numerator, the squared norm $\|\boldsymbol{h}\|^2$ follows the Chi-squared distribution with $2t$ degrees of freedom, $\chi^2(2t)$, since it is a sum of t squared complex Gaussian random variables. Thus, for any user i and eavesdropper j, the secrecy SNR when beamforming to user i is equivalent to the ratio of $1+\chi^2(2t)$ and $1+\chi^2(2)$ random variables.

9.1.1 Main Tool

To assess the ratio in the presence of a large number of users and eavesdroppers, let us recall that for sufficiently large n, the maximum of a sequence of n i.i.d. $\chi^2(v)$ variables, $\boldsymbol{M_n} = \mathbf{max}(\xi_1, \dots, \xi_n)$ follows the Gumbel distribution [69], p.156. Specifically, $\lim_{n\to\infty} \Pr(\boldsymbol{M}_n \le a_n\xi + b_n) = \exp\{-e^{-\xi}\}$, where a_n and b_n are normalizing constants. In this case,

$$a_n = 2 \tag{3}$$

$$b_n = 2\left(\log n + \left(\frac{v}{2}-1\right)\log\log n - \log\Gamma\left[\frac{v}{2}\right]\right) \tag{4}$$

and $\Gamma[\cdot]$ is the Gamma function. Herein, we study the asymptotic (in the number of users and eavesdroppers) distribution of the ratio in (2), and thus derive the secrecy outage probability, *when the transmitter schedules a user with favourable CSI* and beamforms in its direction.

9.2 Asymptotic Secrecy Outage

In this section, we analyse the secrecy outage limiting distribution. That is, for a given target secrecy rate R_s, we analyze the probability that *at least one eavesdropper among M eavesdroppers* will attain information from the transmission. Obviously, when transmitting to a *single user*, and when only statistical knowledge is available on the wiretap channels, beamforming to the user whose channel gain is the greatest among K users is optimal.

Accordingly, let $i^* = \arg\max_i \|\boldsymbol{h}_i\|^2$ be the index of the channel with the largest gain, and let $j^* = \arg\max_j \left|\langle \widehat{\boldsymbol{h}}_{i^*}, \boldsymbol{g}_j\rangle\right|^2$ be the index of the wiretap channel whose projection in the direction $\widehat{\boldsymbol{h}}_{i^*}$ is the largest. Note that when the transmitter beamforms to user i^* in a multiple eavesdroppers environment, it should tailor a code with secrecy rate R_s to protect the message even from the strongest eavesdropper with respect to i^*, which is j^*. Of course, with only statistical information on the eavesdroppers, j^* is unknown to the transmitter. Accordingly, the probability of a secrecy outage when transmitting to user i^* at secrecy rate R_s is [49]:

$$\Pr\left(\log_2\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle \widehat{\boldsymbol{h}}_{i^*}, \boldsymbol{g}_{j^*}\rangle\right|^2}\right) \le R_s\right) = \Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle \widehat{\boldsymbol{h}}_{i^*}, \boldsymbol{g}_{j^*}\rangle\right|^2} \le 2^{R_s}\right) \tag{5}$$

To ease notation, we denote $\alpha = 2^{R_s}$.

In the following, we analyse the distribution in (5) when the number of users and eavesdroppers is large (e.g., 30 users or more). In particular, we consider the secrecy rate distribution when the transmitter beamforms to user i^*, while all eavesdroppers are striving to intercept the transmission separately (without cooperation).

When the transmitter is beamforming to a user whose channel gain is the greatest, then the squared norm $\|\boldsymbol{h}_{i^*}\|^2$ in the numerator of (5) scales with the number of users like $O(\log K)$ [69]. Nevertheless, the greatest channel projection in the direction of the attending user, in the denominator of (5), also scales with the number of eavesdroppers in the order of $O(\log M)$. Moreover, asymptotically, both the greatest gain and greatest channel projection follow the Gumbel distribution (with different normalizing constants). Thus, in order to determine the secrecy rate behaviour, as K and M grow, one needs to address the ratio of Gumbel random variables. However, the ratio distribution of Gumbel random variables is not known to have a closed-form [89]. Thus, we first express it as an infinite sum of Gamma functions, then provide tight bounds on the obtained distribution, from which we can infer the outage probability. Accordingly, we have the following.

Theorem 1 *For large enough K and M, the distribution of the secrecy rate in (5) is the following.*

$$\Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle\widehat{\boldsymbol{h}}_{i^*},\boldsymbol{g}_{j^*}\rangle\right|^2}\leq\alpha\right)=\sum_{k=0}^{\infty}\frac{(-1)^k e^{-(k+1)\frac{1+b_K-\alpha(1+b_M)}{\alpha a_M}}}{(k+1)!}\Gamma\left[1+\frac{(k+1)a_K}{\alpha a_M}\right]$$

where a_K, a_M and b_K, b_M are normalizing constants given in (3) and (4), respectively.

Note that b_K and b_M grow at different rate. Specifically, although b_K and b_M are both normalizing constant of the χ^2 distribution, b_K has value of $v=2t$ in (4)), while b_M has value of $v=2$ in (4). The proof is given in [100].

To evaluate the result in Theorem 1, one needs to evaluate the infinite sum, which is complex. Thus, the upper and lower bounds described next provide good insight on the value of the infinite sum.

9.2.1 Bounds on the Secrecy Rate Distribution

In the following, we suggest an approach that models EVT according to its tail distribution, which enables us to provide tight bounds to the distribution in (5). This approach has very clear and intuitive *communication interpretation*. Specifically, for an upper bound, we put a threshold on eavesdropper j^*'s wiretap channel projection, and analyze the result under the assumption that its projection exceeded the threshold. For a lower bound, we put a threshold on user i^*'s channel gain, and analyze the result under the assumption that its gain has exceeded this threshold.

When only a single user (eavesdropper), among many, exceeds a threshold on average (average on multiple time slots), then the above-threshold tail distribution corresponds to the tail of extreme value distribution (see chapter 4.2 of [63]). Moreover, the tail limiting distribution has a

mean value that is higher than the mean value of the extreme value distribution, since the tail limiting distribution takes into account only events in which user i^* (eavesdropper j^*) is sufficiently strong, namely, above threshold. Thus, replacing the extreme value distribution of user i^* (eavesdropper j^*) with its corresponding tail distribution will increase the numerator (denominator) in (5) on average. Thus, the resulting secrecy rate is higher (lower), hence, corresponds to a lower (upper) bound on the ratio CDF.

Let u_m denote a threshold on the wiretap channel projection in the direction $\widehat{\boldsymbol{h}}_{i^*}$, such that a single (the strongest) eavesdropper exceeds it on average. Note that such a threshold can be obtained by inversing the complement CDF of the Exponential distribution. Further, note that this inverse is exactly (4) with $v = 2$ degrees of freedom. Accordingly, we have the following lower bound.

Lemma 1 *For sufficiently large K and M, the CDF of the secrecy rate in (5) satisfies the following upper bound.*

$$\Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle\widehat{\boldsymbol{h}}_{i^*},\boldsymbol{g}_{j^*}\rangle\right|^2}\leq\alpha\right)\leq\frac{a_K}{\alpha a_M}e^{-\frac{1+b_K-\alpha(1+b_M)}{\alpha a_M}}\Gamma\left[\frac{a_K}{\alpha a_M},0,e^{\frac{1+b_K-\alpha(1+u_M)}{a_K}}\right]$$

where $\Gamma[s,0,z]=\int_0^z\tau_s e^{-\tau}d\tau$ is the lower incomplete Gamma function.

The proof is given in the [100]. The following corollary helps gaining insights from Lemma 1.

Corollary 1 *For $\alpha\geq 1$, the outage probability satisfies the following bound.*

$$\Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle\widehat{\boldsymbol{h}}_{i^*},\boldsymbol{g}_{j^*}\rangle\right|^2}\leq\alpha\right)<[\Lambda(\alpha)(1-\exp\{-\Lambda(\alpha)^{-1}2^{\alpha-1}\})]^{1/\alpha}$$

Where $\Lambda(\alpha)=\left(\sqrt{e}M\right)^{\alpha}\frac{\Gamma(t)}{\sqrt{e}K(\log K)^{t-1}}$.

Note that the value of $\Lambda(\alpha)$ determines the outage probability. In particular, at the limit $\Lambda(\alpha)\to\infty$, the resulting outage probability is 1 (i.e., when $M\to\infty$ and K is fixed). Similarly, when $\Lambda(\alpha)\to 0$, the resulting outage probability 0. Moreover, we point out that $\Lambda(\alpha)$ decreases with the number of users as $K(\log K)^{t-1}$, while increases with the number of eavesdroppers as M^{α}. Thus, roughly speaking, as long as the number of eavesdroppers $M=o(K(\log K)^{t-1})^{1/\alpha}$, we obtain $\Lambda(\alpha)=o(1)$, hence, secrecy outage in the order of $o(1)$. To prove Corollary 1, the following Claim is useful.

Claim 1 ([101][102][103]**)** *The incomplete Gamma function satisfies the following bounds.*

i. $\Gamma[s](1-e^{-z})^s<\Gamma[s,o,z]<\Gamma[s]\left(1-e^{-z\Gamma[1+s]^{-1/s}}\right)^s$, $\forall\ 0<s<1$. This inequality takes the opposite direction for values of $s>1$.

ii. $2^{s-1}\leq\Gamma[1+s]\leq 1$, $\forall\ 0<s<1$.

iii. $\Gamma[s]\Gamma\left[\frac{1}{s}\right]1$, $\forall\ s>0$.

Proof. (Corollary 1). Applying the normalizing constants in (3)-(4) to Lemma 1 result in

$$\Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle\widehat{\boldsymbol{h}}_{i^*},\boldsymbol{g}_{j^*}\rangle\right|^2}\leq\alpha\right)\leq\frac{1}{\alpha}\left(\frac{\left(\sqrt{e}M\right)^{\alpha}\Gamma[t]}{\sqrt{e}K(\log K)^{t-1}}\right)^{\frac{1}{\alpha}}\Gamma\left[\frac{1}{\alpha},0,\frac{\sqrt{e}K(\log K)^{t-1}}{\left(\sqrt{e}M\right)^{\alpha}\Gamma[t]}\right]$$

To ease notation, let us denote $\Lambda(\alpha)=\frac{\left(\sqrt{e}M\right)^{\alpha}\Gamma[t]}{\sqrt{e}K(\log K)^{t-1}}$. Thus, we rewrite Lemma 1 as

$$\Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle\widehat{\boldsymbol{h}}_{i^*},\boldsymbol{g}_{j^*}\rangle\right|^2}\leq\alpha\right)$$

$$\leq\frac{1}{\alpha}\Lambda(\alpha)^{1/\alpha}\Gamma\left[\frac{1}{\alpha},0,\Lambda(\alpha)^{-1}\right]\overset{(a)}{<}\frac{1}{\alpha}\Lambda(\alpha)^{1/\alpha}\Gamma\left[\frac{1}{\alpha}\right]\left(1-e^{-\frac{\Gamma\left[1+\frac{1}{\alpha}\right]^{-\alpha}}{\Lambda(\alpha)}}\right)^{\frac{1}{\alpha}}$$

$$\overset{(b)}{\leq}\left[\Lambda(\alpha)\left(1-e^{-\frac{2^{\alpha-1}}{\Lambda(\alpha)}}\right)\right]^{\frac{1}{\alpha}}$$

Remember that only $\alpha\geq1$ implies a secrecy rate greater than zero. Thus, (a) follows from Claim $1(i)$ and (b) follows from Claim $1(ii)$ and from the Gamma function recurrence property, $\Gamma\left[\frac{1}{\alpha}\right]=\alpha\Gamma\left[1+\frac{1}{\alpha}\right]$.

For the lower bound, we use a similar approach, however, this time, we refer to user i^* as if its channel gain has exceeded a high threshold. Thus, since only sufficiently strong user i^*, whose gain is above threshold, is taken into account, then the numerator in (55) is larger on average, thus, resulting in a higher rate, which corresponds to a lower bound on the ratio CDF.

Let u_k denote a threshold on the user's channel gain, such that a single strongest user exceeds it on average. Note that such a threshold can be obtained from the inverse incomplete Gamma function, which asymptotically, is exactly (4) with $v=2t$. Accordingly, we have the following upper bound.

Lemma 2 *For sufficiently large K and M, the CDF of the secrecy rate in (5) satisfies the following lower bound.*

$$\Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle\widehat{\boldsymbol{h}}_{i^*},\boldsymbol{g}_{j^*}\rangle\right|^2}\leq\alpha\right)\geq1-\frac{\alpha a_M}{a_K}\cdot e^{-\frac{\alpha(1+b_M)-(1+u_k)}{a_K}}\Gamma\left[\frac{\alpha a_M}{a_K},0,e^{-\frac{1+u_k-\alpha(1+b_M)}{\alpha a_M}}\right]$$

The proof is given in the [100].

Again, to gain intuition regarding the bound, we have the following.

Corollary 2 *For $\alpha\geq1$, the outage probability satisfies the following bound.*

$$\Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle\widehat{\boldsymbol{h}}_{i^*},\boldsymbol{g}_{j^*}\rangle\right|^2}\leq\alpha\right)>1-\Gamma[1+\alpha]\Lambda(\alpha)^{-1}\left(1-e^{-\Lambda(\alpha)^{1/\alpha}}\right)^{\alpha}$$

where $\Lambda(\alpha)=\left(\sqrt{e}M\right)^{\alpha}\frac{\Gamma(t)}{\sqrt{e}K(\log K)^{t-1}}$

Proof. Similar to Corollary 1, we apply the normalizing constants in (3)-(4) to Lemma 2, then, set $\Lambda(\alpha) = \frac{(\sqrt{e}M)^{\alpha}\Gamma(t)}{\sqrt{e}K(\log K)^{t-1}}$. Thus, we have

$$\Pr\left(\frac{1+P\|\boldsymbol{h}_{i^*}\|^2}{1+P\left|\langle\widehat{\boldsymbol{h}}_{i^*},\boldsymbol{g}_{j^*}\rangle\right|^2} \leq \alpha\right)$$

$$\geq 1-\alpha\Lambda(\alpha)^{-1}\Gamma\left[\alpha,0,\Lambda(\alpha)^{1/\alpha}\right] \overset{(a)}{>} 1-\alpha\Lambda(\alpha)^{-1}\Gamma[\alpha]\left(1-e^{-\Lambda(\alpha)^{1/\alpha}}\right)^{\alpha}$$

$$\overset{(b)}{=} 1-\Gamma[1+\alpha]\Lambda(\alpha)^{-1}\left(1-e^{-\Lambda(\alpha)^{1/\alpha}}\right)^{\alpha}$$

where (a) follows form Claim $1(i)$ and (b) follows from the Gamma function recurrence property.

9.3 Simulation Results

In this section, we present simulation results for the suggested scheduling scheme and compare them to the analysis above.

Figure 58 shows the distribution of the ratio in (5) for three cases and compare it to the analytical results herein. In particular, we simulate the secrecy rate in (2) for three cases: (i) when beamforming to strongest user i^*, while the strongest, above-threshold, eavesdropper is wiretapping, (ii) when beamforming to strongest user i^*, while strongest eavesdropper j^* is wiretapping (without threshold constraint), (iii) when beamforming to strongest, above-threshold, user, while the strongest eavesdropper j^* is wiretapping. The dark gray, gray and light gray bars represent these results, respectively. Then we evaluate the bounds given in Lemma 1 and Lemma 2 and compare then to the sum of the first 100 terms in Theorem 1, for $t = 2,4,8$ and $M = K = 30$. From this comparison, it is clear that the bounds are tight and provide an excellent approximation of (5). For a comparison between the bounds given in Corollary 1, Lemma 1, Lemma 2 and Corollary 2, for $t = 4$ and $M = K = 1000$, see Figure 59.

Figure 60 depicts the secrecy outage probability. In particular, we set $t = 4$ and $\alpha = 2$, and fix the number of users to $K = 1000$. Then we examine what is the secrecy outage probability as a function of the number of eavesdroppers. The dots represent the critical ratio between M and K such that $\Lambda(\alpha) = 1$, which is exactly $M = \Theta(K(\log K)^{t-1})^{1/\alpha}$. Indeed, for values of M which have smaller order than the critical value, result in small values of $\Lambda(\alpha)$, hence, small outage probability.

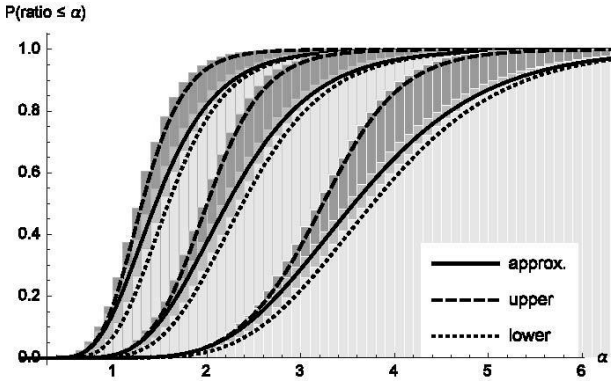


Figure 58: Simulation and analysis of the ratio distribution in (5) for $M = K = 30$ and $t = 2, 4, 8$ antennas, left to right, respectively. The solid line represents the sum of the first 100 terms in Theorem 1. The dashed and dotted lines represent the distribution upper and lower bounds given in Lemma 1 and Lemma 2, respectively.

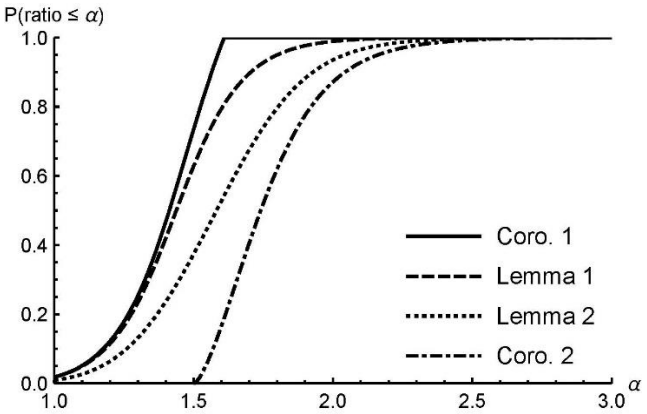


Figure 59: A comparison between the bounds given in Corollary 1, Lemma 1, Lemma 2 and Corollary 2, respectively for t=4 and M=K=1000.

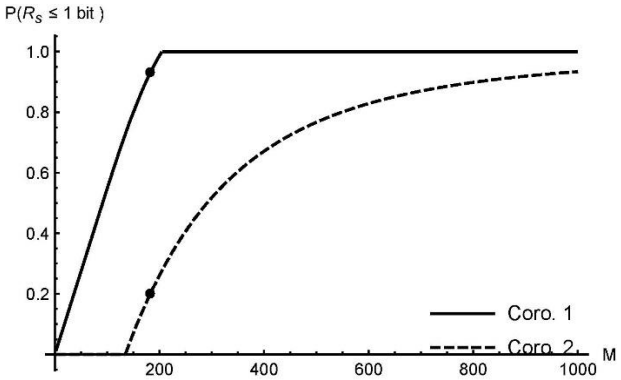


Figure 60: The upper and lower bounds given in Corollary 1 and Corollary 2, for $K = 1000, t = 4$ and $\alpha = 2$, as a function of the number of eavesdroppers M. The marked dot represents the critical ratio where $\Lambda(\alpha) = 1$.

10 Secure Information Dissemination

The increasing demand for network connectivity and high data rates dictates efficient utilization of resources, such as linear network coding [87]. However, in many practical applications, it is important to assure that also users' privacy is not compromised. Gossip protocols that rapidly disseminate messages to all nodes in large networks have gained considerable attention since they were introduced in [66]. In a gossip protocol, nodes exchange messages in rounds. The goal of a gossip protocol is to disseminate all messages to all nodes in the network using as few rounds as possible.

While cryptography could be one solution to secure gossip protocols, it requires high computational complexity [1][73][85]. A gossip protocol, however, should fit networks with simple and limited nodes, e.g., sensor networks, where complex computations at the nodes are not possible. Moreover, cryptographic solutions assume that the computational abilities of the eavesdropper are limited, which is not always the case.

Based on generalizations of *information theoretic security* principles [51], in this work, we design and analyse a novel secure gossip protocol. Specifically, we propose to use information theoretic security to guarantee that all the messages that the source wishes to send to the legitimate nodes will be kept secret from the eavesdroppers. In the considered Secure Network Coding Gossip (SNCG) model, there are legitimate nodes V, source s and an eavesdropper which can obtain w packets across the network. At each round t, the source, as well as any legitimate node which has previously received messages, picks a random node to exchange information with. The algorithm stops when all the legitimate nodes have received all the messages. However, the protocol must assure that the eavesdropper is not able to decode any information from the wiretapped packets. A graphical representation of this model is given in Figure 61.

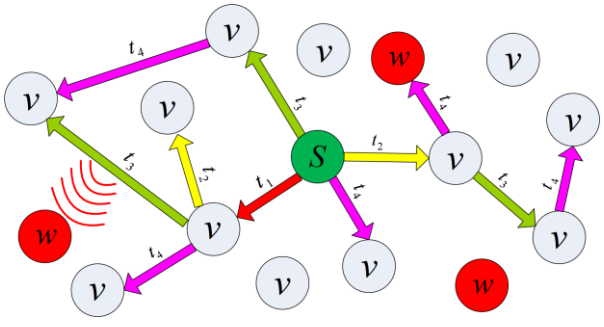


Figure 61: Secure network coding gossip.

10.1 Related Work

10.1.1 Gossip Network

Gossip schemes were first introduced in [66] as a simple and decentralized way to disseminate information in a network. A detailed analysis of a class of these algorithms is given in [77]. In these schemes, nodes communicate in rounds. At each round, a random node in the network chooses a single communication partner according to the gossip algorithm (e.g., selecting a random neighbour). Once a partner is chosen, a limited amount of data is transferred, as defined by the gossip protocol. The main figure of merit is the dissemination time: the average time for all nodes to receive all messages. Such randomized gossip-based protocols are attractive due to their locality, simplicity, and structure-free nature. According to the existing literature, these protocols have been used for various tasks, such as ensuring database consistency and computing aggregate information and other functions of the data [77], [78], [54]. In [65], Deb *et al.* introduced algebraic gossip, a coding-based gossip protocol where nodes exchange linear combinations of their available messages. The advantages in terms of average dissemination time were clear: for example, a node in a complete graph of size n would require $O(n)$ messages to receive all data with high probability. In [74], Haeupler proved a tight bound on the stopping time of algebraic gossip for various graph models.

10.1.2 Secure Networking

The theory and practice of wiretap channel coding [98] has been found useful in a variety of networking scenarios as well. In [67] and [68], the authors considered the canonical wiretap II introduced in [91] as a special case of the wiretap network II. In these works, the authors presented methods to obtain secrecy in a network using new approaches for network wiretap II channels. They proposed to use coset codes to construct a linear secure network code using a Maximum Distance Separable (MDS) matrix. Specifically, for a network encoded by a given linear network code with global encoding kernels, their coding scheme is based on an (n, w)-linear MDS code for $n = k + w$ over a sufficiently large field $\mathbb{F}$. They take a syndrome m of the MDS code, which specifies a coset, as a message. The sender at the source node randomly and uniformly chooses a vector from the coset specified by m and then sends it through the network (which employs network coding) in order to send message m.

Cai and Chan in [55] presented a tutorial paper, where they focus on the theory of linear secure network coding and the limits of confidential communication, when eavesdroppers observe part of the confidential information transmitted in the network. The tutorial is extensively based on previous works, which we will use as well [87][56][57][59][61][99]. In addition, they showed several wiretap models, security conditions for linear network codes and constructions of secure network codes. Specifically, the encoder adds l random bits to the k bits of the message, and by


linear network coding information can be transmitted securely (archiving perfect secrecy) as long as the maximum flow from the source to any sets of wiretap channels W (edges from E) is at most l units, and the maximum flow from the source to any legitimate node is at least $n = k + l$. Harada and Yamomoto in [75] presented the strongly secure linear network coding model. They proposed a new algorithm, which is based on precoding at the nodes using a linear network code. In addition, they defined two levels of security. Specifically, in the first definition, messages $M^k = (M_1, M_2, \dots, M_k)$ can be transmitted securely even if an adversary wiretaps any sets of w edges using a w-secure network coding for $w \leq n - k$, similar to the already presented results of [55]. In the second definition, for a subset of edges $\mathcal{A} \subseteq \mathcal{E}$, let $Z_{\mathcal{A}}$ be the matrix obtained by concatenating all the coding vectors of $\mathcal{A}$. Messages $\left(M_{i_1}, M_{i_2}, \dots, M_{i_{k-j}}\right)$, $\{i_1, i_2, \dots, i_{k-j}\} \subseteq \{1,2, \dots, k\}$ can be strongly w-secure transmitted even if an adversary wiretaps any sets of $w + j$ channels, where $w \leq j = rank(Z_{\mathcal{A}}) - w \leq n$. Cai in [59] showed how the technique of strongly secure linear network coding can be implemented using random linear network coding as well.

In [81], the authors presented secure multiplex coding to attain the channel capacity in the wiretap channel model. To this end, the encoder uses as random bits for the transmitted code word, *other statistically independent messages*. Models where the security allows communication at maximum rate on an acyclic multicast network were presented in [50] as Weakly Secure Network Coding, and later on for random network coding in [95] as Universal Weakly Secure Network Coding, using rank-metric codes and linear network coding. The models presented in [81], [50] and [95] are considered weakly secure since the eavesdropper gets no information about each packet individually, while still potentially obtaining (insignificant) information about mixtures of packets.

In [88], Matsumoto and Hayashi considered the Universal Strongly Secure Network Coding with Dependent and Non-Uniform Messages. In their work, using a random linear precoder at the source node and using network coding and two-universal hash functions [60], they obtain strong security as in [75] and universal security as in [95].

In [53], the authors suggested a client-server setting where the binary erasure wiretap channel approach was found constructive. A multitude of network coding scenarios exist in [58].

10.2 Model and Problem Formulation

SNCG is specified by a directed graph $G = (V, E)$, where V and E are the node set and the edge set of G, respectively. Communication in the network takes place in synchronous round indexed by t. Nodes have the following types of connections: either a node establishes uniformly random connection to one neighbour and either sends (PUSH) or receives (PULL) a message or both (EXCHANGE). The goal of SNCG algorithm is to make all messages known to all legitimate nodes in the network using as little rounds as possible yet keeping the eavesdropper, which is able to observe a subset of the packets transmitted in the network, ignorant.

Throughout this work, we use boldface to denote matrices, capital letters to denote random variables, lower case letters to denote their realizations, and calligraphic letters to denote the alphabet.

In the single source scenario, the node set V contains a source node s with $\vec{M}_1, \dots, \vec{M}_k$ messages of length c bits over the binary field F_q^c, denoted by a messages matrix

$$M = [M_1^c; M_2^c; \dots; M_k^c] \in \{0,1\}^{k \times c}$$

where each row corresponds to a separate message $j \in \{1, \dots, k\}$.

All packets $\vec{Y}$ transmitted in this SNCG algorithm, either from the source or from any other node are a linear combination of the messages available at the node, with random coefficients over the field. However, to maintain secrecy, we allow the source to *encode* the message matrix M. In particular, the source node may *add messages to the matrix*, either using a stochastic encoder or any other method. In fact, it is well known that such a randomization is required to achieve secure communication. Specifically, the source computes a matrix

$$X = [X^n(1), X^n(2), \dots, X^n(c)] \in \{0,1\}^{n \times c}$$

where each row $r \in \{1, \dots, n\}$ denoted by $X_r^c = \{X_r(1), \dots, X_r(c)\}$ can be considered as an encoded message to be transmitted. Note that $n \geq k$, hence, although the new messages are of the same length, more messages are now available at the source. The source may then create a packet to be transmitted using a random coefficient vector $\vec{\mu} = \{\mu_1, \dots, \mu_n\}$, i.e.,

$$\vec{Y} = \sum_{r=1}^{n} \mu_r \vec{X}_r$$

Figure 62 gives a graphical representation of the source encoding.

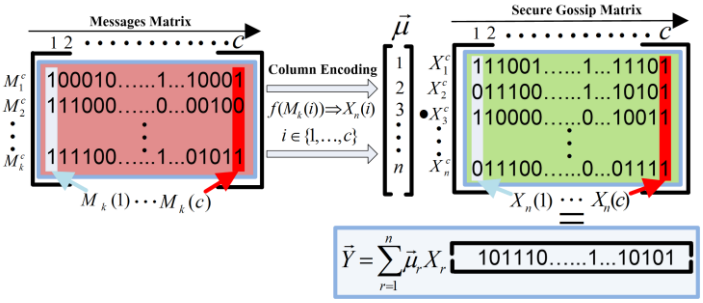


Figure 62: Source encoding by secure gossip matrix.

All the node packets $Out(\vec{Y})$ transmitted in the synchronous rounds are a linear combination of the set packets S_v from $In(\vec{Y})$ maintained by the nodes from the previous rounds, with random coefficients $\vec{\mu}$, i.e.,

$$Out(\vec{Y}) = \sum_{l=1}^{|S_v|} \mu_l In(\vec{Y})_l$$

Figure 63 gives a graphical representation of the node encoding.

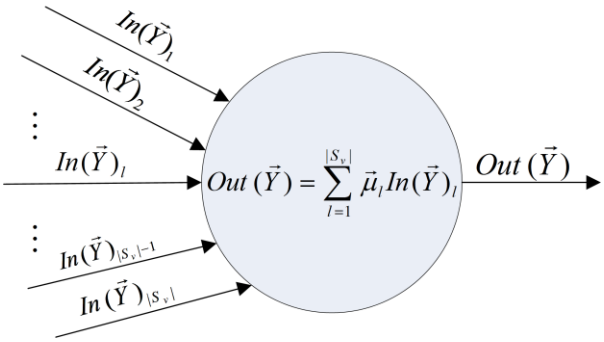


Figure 63: Node encoding.

We assume an eavesdropper which can obtain a subset of the packets traversing the network. Specifically, we define the eavesdropper matrix as:

$$Z_w = [Z_1^c; Z_2^c; \dots; Z_w^c] \in \{0,1\}^{w \times c}$$

assuming the eavesdropper wiretaps some set of w packets, which are linear combinations of the original transmitted messages.

We refer to the secure gossip matrix, together with a suitable decoder, as a secure gossip algorithm. Assume the algorithm runs for n rounds. The following definition lays out the goals of SNCG.

Definition 1 *A gossip algorithm with parameters k, n, w and q is reliable and w-secure if:*

(1) At the legitimate nodes, letting $\vec{Y}_n$ denote the message matrix obtained after n rounds we have

$$P(M(Y_n) \neq M) = 0$$

(2) w-*secure*: at the eavesdropper, observing w, we have

$$H(M|Z_w) = H(M)$$

To conclude, the goal is to design secure network coding gossip where legitimate nodes randomly exchange their available messages in order to disseminate all the messages to all the legitimate nodes in as few rounds n as possible. Additionally, by observing w packets from the communication between legitimate nodes in the synchronous rounds, an eavesdropper remains ignorant regarding the messages.

10.2.1 Information Dissemination in Oblivious Networks

We briefly review the definitions and results from [62], which we will use throughout the rest of this document.

Definition 2 *A network model is oblivious if the topology G_t at time t only depends on t, G'_t for any $t' < t$ and some randomness. We call an oblivious network model furthermore i.i.d. if the topology*

G_t is independent of t and of prior topologies, that is, if there is a distribution over directed hypergraphs from which every G_t is sampled independently.

An important indication on how well a network topology mixes the packets can be obtained from the flooding time, i.e., there is a strong relationship between the speed of information dissemination and the flooding time of a network [62]. We say the flooding process F stops at time t if a single message can be received at all nodes. Let S_F be the random variable denoting the stopping time of F.

Definition 3 *We say an oblivious network with a vertex set V floods in time T with throughput α if there exists a prime power q such that for every $v \in V$ and every $k > 0$ we have:*

$$P\left[S_{F(H;1/q;\{v\})} \geq T + k\right] < q^{-\alpha k}$$

10.3 Main Result

Under the model definition given in the formulation, our main result is the following sufficiency (direct) condition, characterizing the maximal number of rounds t required to guarantee both reliability and security. The proof is deferred to Section 10.4.

Theorem 2 *Assume an oblivious network that floods in time T with throughput α . Then, for any k messages at a single node in the network, algebraic gossip (with random linear coding) spreads the k messages to all nodes with probability $1 - \epsilon$ after*

$$T' = T + \frac{1}{\alpha}(k + w + \log \epsilon^{-1})$$

rounds, while keeping any eavesdropper which observes at most w packets, ignorant.

That is, compared to only a reliability constraint, the number of rounds required for both reliability and secrecy is increased by w rounds, where, again, w is the number of leaked packets at the eavesdropper. The construction of the SNCG algorithm and the proof for both reliability and secrecy are deferred to Section 10.4.

10.4 Code Construction and a Proof

At the source node, we *randomly* map each column of the message matrix M. Specifically, as depicted in Figure 64, in the code construction phase, for each *possible column* of the message matrix we generate a bin, containing several columns. The number of such columns *corresponds* to w, the number of packets that the eavesdropper can wiretap, in a relation that will be made formal next. Then, to encode, for each column of the message matrix, we randomly select a

column from its corresponding bin. This way, a new, $n \times c$ message matrix X is created. This message matrix contains n new messages of size c.[5]

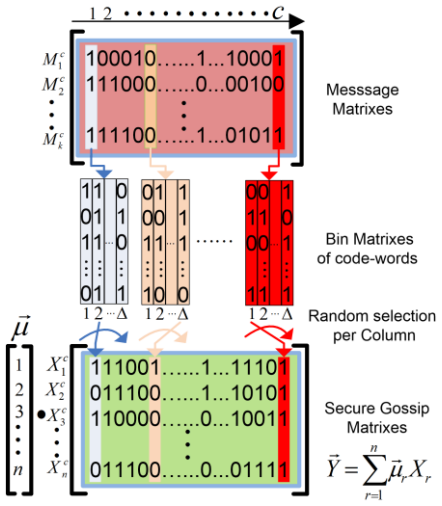


Figure 64: Binning and source encoding process for a SNCG algorithm.

In order to rigorously describe the construction of the matrices and bins, determine the exact values of the parameters (e.g., bin size), and analyse the reliability and secrecy, we first briefly review the representation of algebraic gossip algorithm [74], together with the components required for SNCG.

A SNCG code at the source node consists of the following. A matrix M of $\vec{M}_1, \ldots, \vec{M}_k$ messages of length c bits over the binary field F_q^c of $q \geq k$. We denote the set of matrices by M. A discrete memoryless source of randomness over the alphabet $\mathcal{R}$ and some known statistics $p_{\mathcal{R}}$. An encoder,

$$f: \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{X} \in \{0,1\}^{n \times c}$$

which maps the message matrix $\boldsymbol{M}$ to a matrix $\boldsymbol{X}$ of codewords.

The need for a *stochastic encoder* is similar to most encoders ensuring information theoretic security, as randomness is required to confuse the eavesdropper about the actual information [51]. Hence, we define by $\mathcal{R}_k$ the random variable encompassing the randomness required *for the k messages at the source node*, and by Δ the number of columns in each bin. Clearly, $c \log \Delta = H(\mathcal{R}_k)$.

Then, the source packets $\vec{Y}$ transmitted in this SNCG algorithm are a linear combination of $\{\vec{X}_r\}_{r=1}^{n}$, with random coefficients i.e., $\vec{Y} = \sum_{r=1}^{n} \mu_r \vec{X}_r$ is the linear combination transmitted. Each node maintains a subspace Y_v that is the span of all packets known to the node itself, that is,

[5] As in most linear coding solutions, a header is required as well. The overhead in the header size is negligible as c grows.

received in any of the previous rounds. If node v does not have any messages at the first round, then Y_v is initialized to contain only the zero vector. When node v is required to send a packet, $Out(\vec{Y})$, it chooses uniformly a random packet from Y_v by taking a random linear combination on its basis. At the end of each round, each node updates its subspace. When the subspace spanned by the coefficient vectors is the entire space, the node can decode.

We define

$$M: Y_n \rightarrow M$$

as the decoder mapping, where Y_n denotes the matrix obtained at the node with n packets having linearly independent coefficient vectors. The probability of error is $P(M(Y_n) \neq M)$. The leakage to the eavesdropper is w packets. We may now turn to the detailed construction and analysis.

Codebook Generation

Set $\Delta = 2^{w-\epsilon}$. Using a distribution $P(X^n) = \prod_{j=1}^{n} P(x_j)$, for each possible column in the message matrix generate Δ independent and identically distributed codewords $x^n(e)$, $1 \leq e \leq \Delta$, where ϵ can be chosen arbitrarily small. The codebook is depicted in Figure 64. Reveal the codebook to Alice and Bob. We assume Eve may have the codebook as well.

Source and legitimate Node encodings

For each column i of the message matrix $\boldsymbol{M}$, the source node selects uniformly at random one codeword $x^n(e)$ from the i-th bin. Therefore, the SNCG matrix contains c randomly selected codewords of length n, one for each column of the message matrix. The source transmits linear combinations of the rows, with random coefficients. Nodes transmit random linear combinations of the vectors in S_v, which is maintained by each node according to the messages received at previous rounds.

10.4.1 Reliability

The reliability proof is almost a direct consequence of [62], Theorem 1]. That is, instead of disseminating k messages, we disseminate n messages. Clearly, the number of rounds required is given by Theorem 1. Now, once a legitimate node maintains all the rows of the secure gossip matrix, the indices of the bins in which the columns of the secure matrix reside are declared by the decoder as the original columns in the messages matrix M.

10.4.2 Information Leakage at the Eavesdropper

We now prove that the security constraint is met. In particular, we wish to show that $I(M; Z_w)$ is small. We will do that by showing that given Z_w, Eve's information, all possibilities for M are equally likely, hence Eve has no intelligent estimation of M.

Denote by C_n the random codebook and by $\boldsymbol{X}$ the set of n codewords corresponding to the $\vec{M}_1, \ldots, \vec{M}_k$ messages. To analyze the information leakage at the eavesdropper, note that Eve has

access to at most w linear combinations on the rows of $\boldsymbol{X}$. We will assume these linear combinations are in fact independent, and since Eve has access to the coefficients, she can invert the coefficients matrix and have access to w rows from the original matrix $\boldsymbol{X}$ at the source.

Next, note that the columns of $\boldsymbol{X}$ are independent (by the construction of the codebook). Hence, it suffices to consider the information leakage for each column from $\boldsymbol{X}$ separately. Remember, for each column i of $\boldsymbol{M}$ the encoder had Δ independent and identically distributed codewords, out of which one was selected. Hence, there is an exponential number of codewords, from the eavesdroppers' perspective, that could generate a column in $\boldsymbol{X}$, accordingly, an eavesdropper (Eve) is still confused even given the w bit Eve has from each column.

Let $\boldsymbol{Z}^w(i)$ be the w bits Eve has from column i. Denote $d = n - w$. Define by $Sh(\boldsymbol{Z}^w(i), d)$ the set of all n-tuples consistent with $\boldsymbol{Z}^w(i)$, that is

$$Sh(\boldsymbol{Z}^w(i), d) = \{b^n : b^n(S_Z) = \boldsymbol{Z}^w(i)\}$$

where S_Z denotes indices of the rows $\boldsymbol{Z}$ contains. Clearly, there are 2^d tuples in $Sh(\boldsymbol{Z}^w(i), d)$. See Figure 65 for a graphical illustration.

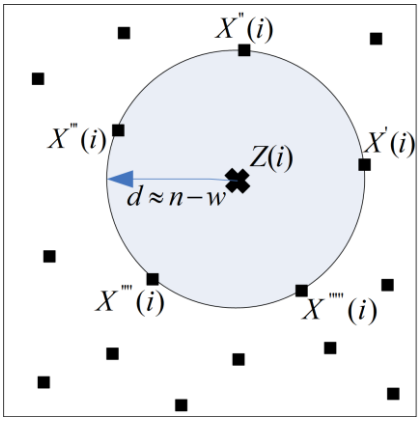


Figure 65: Codewords exactly lie in a circle around $\boldsymbol{Z}$ of radius $\boldsymbol{d \approx \Delta}$.

We assume Eve has the codebook, yet does not know which column from each bin is selected to be the codeword. Hence, we wish to show that given $\boldsymbol{Z}^w(i)$, Eve will have at least one candidate per bin. The probability for a codeword to fall in a given shell is

$$\Pr\left(\boldsymbol{X}^n(i) \in C_n \cap \boldsymbol{X}^n(i) \in Sh(\boldsymbol{Z}^w(i), d)\right) = \frac{\mathcal{V}ol(Sh(\boldsymbol{Z}^w(i), d)}{2^n} = \frac{2^{n-w}}{2^n}$$

In each bin of C_n, we have $\Delta = 2^{w+n\epsilon}$ codewords. Thus, the number of codewords Eve sees on a shell, *per bin* is

$$|\{m(i) : X^n(i) \in Sh(Z(i), d)\}| = \frac{2^{w+n\epsilon} * 2^{n-w}}{2^n} = 2^{n\epsilon}$$

Hence, we can conclude that on average, and if $n\epsilon$ is not too small, for every column in $\boldsymbol{M}$ Eve has a few possibilities *in each bin*, hence she cannot locate the right bin. However, it is still important to show that all bins have (asymptotically) equally likely candidate codewords, hence Eve cannot locate a preferred bin.

To this end, we proved that the average number of codewords per column is $2^{n\epsilon}$. We wish to show that the probability that the actual number of options deviates from the average by more than ϵ is small. Define

$$\mathcal{E}_{C_1}(Z(i), d) \coloneqq \Pr\{(1-\epsilon)2^{n\epsilon} \leq |\{m(i): X^n(i) \in Sh(\boldsymbol{Z}(i), d)\}| \leq (1+\epsilon)2^{n\epsilon}\}$$

By the Chernoff bound, we have:

$$\Pr\left(\mathcal{E}_{C_1}(Z(i), d)\right) \geq 1 - 2^{-\epsilon' 2^{n\epsilon}}$$

Due to the super exponential decay in n, when taking a union bound over all columns, the probability that Eve decodes correctly some column is small. Hence, for Eve, all codewords are equiprobable and $I(M; Z_w) \to 0$.

11 Appendix – Security challenges

NFV applications and services can be seen as combinations of cloud computing and virtualization related objects. Hence, many of the threats are known to the technology industry from these two areas. Nevertheless, due to the prospect of previously naturally secluded application families being deployed into the public environment, and thus being exposed to neighbouring HW and SW structures, the security challenges should be reviewed and new threats should be defined.

In some cases, VNFs should be implemented in symbiosis with accompanying security functions. In other cases, the existing services could be deployed and merely side-guarded by dedicated security functions, possibly by specially tailored separate VNFs. Additionally, some pure security services will need to be deployed as VNFs. In this case, these services should be appropriately revised and enhanced in order not to become a target by their own newly revealed weakness.

In this Appendix we first briefly review security issues related to general cloud computing and virtualization (section 11.1.1). We further review issues with network processing on commodity hardware (i.e., NFV, section 11.1.2). Then, we combine these aspects and formulate the possible security challenges in the Superfluidity architecture (section 11.1.3). In section 11.2, we elaborate on each of the aforementioned challenges and provide some examples encountered throughout the Superfluidity architecture specifications. Finally, in section 11.3 we analyse several Superfluidity use-cases for security issues, mentioning possible security issues which arise when the service is deployed as VNF, and providing possible solution directions.

11.1 Review of security issues

11.1.1 Virtualization security issues

Virtualization brings an additional layer of connectivity and the underlying control, with new security concerns. In particular, local hypervisors can be targeted by potential attackers. These attacks aim at disrupting the virtualization activity and correctness, in order to possibly damage or exploit the providers, for example, by diverting their resources. The orchestrator is a global target (e.g., by breaching the administrator's workstation) which, if successfully attacked, can cause downtime of the entire CDN.

As for security issues, as far as the virtualization technology in general is concerned, virtual platforms are subject to specialised attacks which include compromising VMs, hosting servers, malware use among others. The attacks on hypervisors might be especially severe for the overall functioning. These attacks can be both targeted on hosted and bare-metal hypervisors.

We divide the security issues concerning virtualization into the following categories:

- Snapshot and logging - Enterprises periodically store snapshots of their VMs for backup purposes. These logs include anti-malware software which should constantly be kept

updated. Hence once the system is rolled back to an older version (e.g., during recovery event) some security patches could remain old. This provides a vulnerability which can be exploited.

- Virtual networking - is only one example of virtualization. To demonstrate the possible issues of virtual networking, consider a setup where multiple virtual machines are connected over a virtual switch thus providing a virtual network. Hence, since the traffic does not traverse the actual physical network interface, in the case where one VM attacks another VM, the event will not be detected by legacy intrusion detection system (IDS) or data loss prevention (DLP) agent. This raises a question of additional overhead expressed by supplemental virtual firewalls. These risks are described in more detail in section 11.1.2.
- Compliance - Co-hosting of several VM for different enterprises on the same domain (either physical or virtual) can come in contradiction with SLAs, regulation policies and different security demands. In particular, mixing of confidential and non-confidential information on the same storage can become an issue.
- Dynamical considerations - This issue will be in particular of interest as far as NFV is concerned. The variety of security issues with VM movements and migrations is represented but not limited to, unsecured migrations due to the load balancing, migrations due to physical failures, introduction of migration-related security policies, secured changes of physical address while attaining same virtual address, logging of migrated VMs.
- Deployment configuration and installation - This category considers deployments and erasure of VMs, rebooting of VMs after software installations. In the case where the attacker may know the reboot timings, this can be exploited for timed attack. Concurrent reboot of multiple machines can also create performance issues.
- Management and control - This category contains all issues mentioned above in the sense that a novel security-aware orchestration might be needed in order to make massive network virtualization feasible.

11.1.2 Security challenges in Network Function Virtualization.

Network function virtualisation promises a world where network functions are software running on commodity hardware as virtualised entities. The selling points of NFV are quick instantiation times, ease of scaling and updating of functionality. However, network configuration has traditionally been a very difficult task: ensuring a network complies with the operator policies (e.g. ACL) is difficult even with hardware appliances. Finally, running a large software base from third party vendors opens networks to attacks.

The Superfluidity vision of fast orchestration of a mix of components implemented by different vendors leaves it exposed to two possible security risks:

- **Policy compliance violations –** by wrongly configuring an RFB, or by deploying the wrong RFB, the resulting network configuration violates the policy of the network operators. Examples here include lack of isolation between infrastructure-critical and client traffic, allowing client traffic to reach operator services that it should not reach, etc.
- **Low-level exploits** – software exploits of RFB implementations fall in this category. The results of the exploits may be disabling the function of the RFB and leading to policy compliance violations, or even worse: control of the underlying infrastructure and affecting other tenant's or the operator's own traffic.

11.1.3 Security challenges in Superfluidity architecture

We now combine both aspects reviewed in previous subsections and categorize NFV specifics and Superfluidity architecture in particular, regarding security.

We divide security issues related to NFV according to the following categories.

- **Authentication and identification -** This category covers issues involving password management, identification management, remote identification, virtualization of authentication servers, location of virtual machines and storage units which hold identity data, securing of identification queries and similar topics.
- **Tracing and monitoring** - On-the-fly control of data packets belongs to this category. The objective of these functionalities is two-fold. The first objective is to prevent potential attacks. The second is to create logs in order to facilitate the recovery process once a security incident has taken place. The possible attacks include Denial of service attacks, malware-initiated attacks on virtual resources, attempts to hamper hypervisors, etc. The monitoring tools would be represented but not limited to virtualization of firewalls, deep packet inspection (DPI).
- **Policy compliance violations** – verifying that the deployed network data-plane behaves according to the high level operator policy. There are two approaches in this category: testing and static analysis. Testing implies injecting test packets into the network and observing whether the outputs match the policy. Static analysis implies taking a snapshot of the data-plane configuration and testing it offline to see if it matches the policy for any given packet.
- **Memory safety of deployed code –** ensuring the safety of NFV code is critical to the correctness and appeal of NFV. There are two ways to do this: either write the code in a memory-safe language, such as Java (but give up performance), or use C and apply various techniques to reduce exploitation vectors (e.g. use ASLR, DEP, stack canary values, etc).
- **Availability and feasibility** - This category can be subdivided into two different topics. First one deals with HW targeted attacks, that may cause HW failure and loss of confidential or any other data. The loss of packets also belongs to this subcategory. The second subcategory deals with technical and architectural additions which should be implemented in order to support

the security of virtualized functioning in general. This refers to virtualized network interface (vNIC), new tunnelling solution, etc.

- **Performance scalability** - Once hardware based network services are virtualized, a question of performance cost is raised. In particular, one should assure that the implementation of new security add-ons does not inflict a severe performance penalty as well as does not impair the performance of the corresponding services as compared to before the virtualization. Clearly, firewall processing should not become a traffic bottleneck. Any movement of the traffic congestion function from one point of the network to another as a result of virtualization is undesirable. Hence, even if not inherently security-related issue, performance scalability should be simultaneously addressed and understood.
- **Survivability and damage isolation** -. Certain measures should be ready to be invoked in order to isolate the unit that was infected by malware or compromised by any other means, and assure fast recovery to the full working status and minimization of damage. This involves deployment of special backup and logging packages by cloud providers.
- **Data plane forwarding** - Consider the virtualized network interface (vNIC). The traffic forwarding should be transparent. Practically, the implementation of virtualization comprises virtual switching. The question raised is the performance of encapsulation and opening of the traversing packets, including tunnelling solutions, and deep packet inspection. All of the above should be created on the virtualized level.

11.2 Superfluidity Security challenges

Next we elaborate on each of the topics presented in the previous part and provide some examples and possible directions that should further explored throughout the project.

11.2.1 Authenticity and identification

The subject of identification mostly refers to the interaction with humans. Hence, due to the human intervention the flow intensity in this case is expected to be comparatively sparse unlike in the case of application oriented data flows. However, as the packets contain mostly sensitive information, hence they will always be encrypted.

11.2.1.1 Mobility problem

Superfluidity will support movement of virtual servers, because of the abstraction of the virtual network from physical devices. Hence, it is critical that the dynamics of the VNFs will be followed by the dynamics of the security controls. We describe this security challenge by an illustrative use-case example.

Example (Virtualization of a server which asks for authentication upon access)

A web server, storage server, FTP server and others which are accessed by authorized users, need their authorization functioning to be virtualized concurrently with their main activity. The implication of such a virtualization includes performing authentication by VM. Hence, either storage at remote VMs of the database of user's passwords, or opening specialized secured links which serve the authentication process should be assured. In addition, the movements, i.e., migration of VM, should comply with the security needs. Hence, any kind of migration should be enhanced by specialized security and encryption protocols.

The problem is depicted in Figure 66. The left case shows the problem where the secure database (of users' passwords) should be migrated from one NFV node of presence (NoP) to another. The right case, in Figure 65, shows the problem where the database is not virtualized, which raises the need for specialized secure flow. Note, that this flow may also change its physical characteristics. Hence, the example demonstrates that constant motion and dynamic nature of the virtualized services should be supported by all adjacent security tasks.

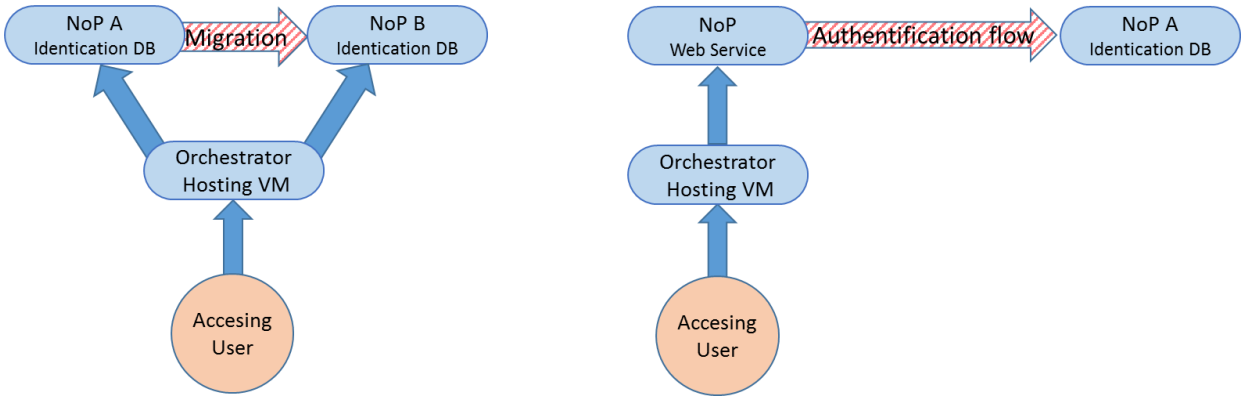


Figure 66: Possible cases of authentication by virtualized server

The encryption is always activated when personal credentials are being accessed, queried or changed.

We next apply two known identification and authentication methods and describe how they will be integrated into NFV-extensive environment. In particular, we describe the solutions that assure these methods conform with NFV security demands. Namely, we propose usage of Biometrics-based identification and Federated Identity paradigm.

We suggest a solution, CloudId, which applies to both methods.

11.2.1.2 Biometrics

Storage of confidential identification data (ID) in the forms of passwords does not answer the security and privacy guarantees which are normally demanded by users. Usage of biometric encrypted identification provides more confidentiality. This is because biometric data is extremely difficult to decipher. The identification process of each query is done by calculating a

distance metric between the stored pattern and the pattern sent in the initiator's query, e.g. see the work in [104].

Since biological data can vary, this metric does not necessarily provide a clear binary answer. As a result, intelligent machine learning algorithms tailored for usage with the biometric information are stored in the system. On the other hand, the content stored in the public domain is much more sensitive. Hence, the following demands of remote storage of biometric data are introduced:

1. The biometric data should be encrypted
2. The query should be encrypted
3. The initiator of the query cannot see the stored encrypted biometric data
4. The storage owner cannot see the result of the test of the query; only the binary answer correct/incorrect query is seen to the (cloud) storage owner.

In order to address all the above requirements, specialized encryption methods should be employed.

The authors in [104] are aware of hill-climb attacks aimed at the core of the biometric content. Note that these threats should be effectively eliminated. Also note that the mere fact of visibility of queries traversing the virtualized network, which is concurrently accessed by various tenants, can provide an opportunity for information leakage.

11.2.1.3 Federated identity management

Due to the virtualization of certain network services, e.g., deployment of home gateways as VNF, the identification processes will be performed on virtualized servers. We now discuss federated identity.

The federation of user identity (FI) grants users which already have secured access to one domain seamless access to another domain. The objective is to minimize the amount of effort put by a user to administration.

Note that in several use-cases the access is performed *on behalf* of a user. The most widespread example is usage of social plug-ins.

Generally, the overall security level is assumed to increase by FI. This is supposed to be fulfilled by one-time, possibly periodically updated, authentication. The corresponding identity information is then exploited across multiple systems and websites, including third-party websites. Hence, the paradigm of FI is naturally generic, and should suit all technologies and standards.

A representative list of standards and technologies being employed by the FI application includes SAML (Security Assertion Markup Language), OAuth, OpenID, Security Tokens, Web Service Specifications, Microsoft Azure Cloud Services, and Windows Identity Foundation.

11.2.1.4 CloudID-based solution

The solution can be based on the CloudID platform [105].

The objective of the solution is to assure that both the cloud provider and any potential third-party attackers are restricted. In both cases, the restriction involves two subjects:

1. The sensitive data which is virtually statically stored in the cloud (i.e., the cloud provider itself)
2. The content of any of the potential individual queries.

The solution in [105] suggests creating encrypted search queries, by means of a k-dimensional tree structure (*k-d tree structure)* of the search process in the encrypted data (details can be found in the cited paper).

11.2.2 Tracing and monitoring

This section deals with monitoring and securing of NFV function on separate VM-level, and on the inter VM level. Most of the VM security was already addressed, however special care should be taken in the NFV context. We mention the following vulnerabilities

- Autostart hooks. Various autostart options implanted by malware, side application, which could be running on the same machine. Such side application can refer to any, seemingly light, network service running in the background. See Example 11.2.2-A.
- Sleeping loops and deliberate delays. Sleeping loops and delays can be imposed on VMs thus deliberately slowing down connections (See Example 11.2.2-B).

Consider the following examples:

Example 11.2.2-A

Consider malware application X being autostarted each time VM a.b.c.d is rebooted, and/or new VNF is deployed. The application takes the shares of the machine cycle. Hence, even if it cannot harm any VNFs concurrently deployed on the same VM, it degrades its performance.

Example 11.2.2-B DoS attack in the context of NFV - *Consider web server deployed as VNF. Now consider a dummy connection which opens TCP port and connection with minimal MSS and minimal CWND. The connection will be particularly slow because of these MSS and CWND constraints. Now, in order to slow it down further, it will delay packets by activating exponential back-off on the other side, and will send a single MSS to avoid closing the connection. Engaging multiple connections of such a type can potentially harm networking functionality of the other side.*

Designated Monitoring Centres (DMC)

Possible directions for the solutions are as follows:

Virtualized and specialized malware protection should be added to each VM. Logging should be intensively practiced. This includes both frequent snapshot and also logging of the movements.

That is, in case VM moves from HW A to HW B and then to HW C, these movements should be logged.

Specific monitoring centres should be deployed on cloud owned HW.

Additional solution: Providing full packages of VNF as a Service (VNFaaS) which handle/incorporate security.

11.2.3 Policy compliance violations

Each network operator has a set of policies that its network configuration must obey to. Example policies include isolation client traffic from operator traffic, isolating tenants renting processing from the operator, ensuring all traffic from clients is NATed and firewalled, etc. In a dynamic 5G network, where network configuration changes constantly due to the instantiation and removal of network functions, how can the operator make sure its policies always hold?

Checking policy compliance is a well-studied field, with many solutions that fall in two categories: active testing or static analysis.

Active testing means injecting packets at many vantage points and verifying that the outcome obeys the policy. This solution is simple to implement, however it cannot guarantee policy compliance: random test packet generation only covers a small part of the possible packets, and correctness must be proven given all possible packets. To increase coverage, tools like ATPG [109] or BUZZ [113] choose test packets after they run static analysis on the network configuration and figure out “classes” of packets.

All static analysis tools (including HSA, SymNet, Veriflow, NOD, etc) use a snapshot of the data plane forwarding state and a model of the processing performed by each box. In this model, a generic packet is injected and different properties are checked such as reachability and loop freeness. The big advantage of static analysis over testing is that it can give much better coverage of the possible set of packets and thus better guarantees of policy compliance. One downside, however, is that the models used by static analysis tools to capture the processing performed by each box are a hard thing to do.

In Superfluidity we will use the Symnet symbolic execution framework [114] to test policy compliance. Symnet has been developed in the previous Trilogy2 FP7 project and Superfluidity, and we will continue developing it and applying it to real networks. Symnet works on models written in SEFL [114], a specialized language that has been optimized to ensure fast symbolic execution. We have created SEFL models for routers, switches and many other boxes (see [114]). We are currently in the process of using Symnet to ensure policy compliance of Openstack tenant configurations ([115]).

11.2.4 Defending against memory exploits

Defending against memory exploits for network functions written in C can be done by using traditional tools including Address Space Randomization (ASLR), Data Execution Protection (DEP) and write integrity protection (stack canary values, WIT [116]). On top of this basic protection, fine grained virtualization of network functions can further limit the damage of faulty NF can incur into the rest of the network.

In Superfluidity, we take a different approach to achieve memory safety. We will develop an approach to automatically generate C implementations from SEFL models that are optimized for symbolic execution. This work has two distinct parts:

- A framework that enables equivalence-preserving optimizations on SEFL models to allow successive optimizations that are faster to run
- A translation from SEFL to C that provably preserves the memory safety properties.

This work is just starting.

11.2.5 Performance and scalability of secured NFV

Virtualization of multiple services will open new security problems, which should be effectively treated. However, the implementation of new secure privacy-related and authentication functionalities should be aware of performance demand of the corresponding VNFs. In particular, deployment of VNFs, jointly with their additional security assuring components, should be still cost effective for the enterprise.

We explain both performance and scalability issues in firewall Example 11.2.5.

Example 11.2.5 (Scalability of virtualization: HW based Firewall)

Consider a legacy firewall, the throughput of the designated device is high; still it is not virtually unlimited. Another advantage of the device is low delay. This is not the case if the firewall is deployed as NFV. The virtualized firewall will be able to support high load amounts and will not be vulnerable to intermittent load peaks.

However, this poses new challenges pertaining to the concurrent secure and effective functioning.

First, the delay will increase. The reason for this can be also attributed to the additional networking which is added to the previous functioning.

Second, the new networking linking the enterprise with the virtualized firewall should be implemented in a secured manner, hence the packets which traverse the (virtual) link should be additionally secured. This may add additional overhead which can affect the performance.

Finally, the newly virtualized firewall should run on isolated and secured VM.

11.2.6 Availability and feasibility

In the case of VNFaaS, availability of a sufficient number of deployable instances should be assured.

Shortage of VNFs, or lack of scalability can be abused by a potential attacker. An additional vulnerability is the inability to effectively access the secured storage, or the storage that contains sensitive data.

Denial of service attacks can also hamper the availability, as explained next.

11.2.6.1 Denial of service attack and tracking attack in NFV environment

Consider the Identification Server (IS) which stores identity details. The potential attacker could deploy an attacking system, which comprises virtualized switches and routers, virtualized DHCP server, that would produce differentiated queries to be sent to the IS. In particular, all queries will be seemingly different and will be hard to see if coming from the single identity by the IS. The virtualized structure of the IS and adjacent network functions could be deliberately misused by potential attacker, see e.g., [107]. The objective of such an attack could be two-fold:

- Sending multiple queries in order to flood the IS, thus causing a denial-of-service (DOS) attack.
- Tracking of queries.

In contrast to the case where the IS resides in the user's static private premises behind a firewall, the virtualized IS in the neighbourhood of the public resources. In order for a DoS attack to succeed, special planning from the side of the attacker should be applied, yet it is theoretically possible. Hence, IS should be employed together with defending envelope, including virtualized firewall, with specially tailored routing rules.

The tracking attack is designated to follow different types of queries in order to extract any kind of side information about both the initiator(s) of the queries and about the confidential identification data stored at the IS. It is not clear how this threat can be alleviated.

11.2.7 Survivability and damage isolation

Once network services are virtualized using the cloud's infrastructure, the responsibility for disaster treatment is taken out from the domain of the enterprise. Typically, the recovery issues should be a part of the SLA. Hence, in case the enterprise has well-defined strict disaster recovery demands, the deployment of the VNF should be performed on the suitable NFVI nodes, which can address these demands. In this case, it is an enterprise's responsibility to assure this availability. Otherwise, in case NFVSaaS is provided, special recovery procedures should be ready and presented to the NFV user.

Normally, the disaster is attributed to one or more than one of the following events (which can be triggered by natural causes, e.g., fire, natural disaster, intended physical attack, etc.) have occurred:

- CPU fallout - Disaster of local dimensions, which lead to stoppage of several deployed services.
- Electricity down - Usually one specific geographical point is affected. Causes all deployed VNF to be stopped. Causes stoppage of all services incoming /outgoing to the points located on the site. Hence VNFs deployed on other sites can no longer access.
- Disk failure - This issue become especially dangerous when it involves the disk serves for authentication of virtually deployed resources. In addition to the damage caused by the stoppage of the service and obscurity of how to handle the existing user's authenticated sessions, the event can be maliciously exploited for the additional attacks, thus providing the vulnerability in the face of the "second-wave" disaster.
- Disrupted link - In the case where a physical line is damaged all associated data transfers with this link are stopped.

Note that we excluded from the list above "disasters" which happen due to extreme system overload. This is because we assume that the load, load balancing and QoS provisioning are handled at other management levels. Hence, load peaks should not lead to the events associated with disasters.

The basic requirements for successful disaster recovery are as follows:

- Understand the dimensions of the disaster- The cloud provider should realize what are the damaged/infected/harmed nodes, what are the locations of the affected sites. It should be made clear what are the geographical coordinates of the CPUs and disks which are involved. In the special case of a networking layer collapse, the location of the damaged links, lines, optical fibres should be understood.
- Isolation of the damaged location - It is crucial to prevent any leakage of the damage to other sites.
- User acknowledgment - The enterprise should immediately understand the magnitude of the disaster and receive the recovery prognosis. The NFV service provider can offer alternative solutions, but this information will allow the customer enterprise to make effective decisions on its own.
- Log and snapshot - This phase should be performed concurrently with the temporal (substitute) deployment. The logged data should be used in order to restore the functioning. Special care and security measures should be taken, because any potential attacker currently *knows* that logs will be taken out of the log servers.
- Temporary deployment/solution - NFVs should be offered backup infrastructure in order to revive the functioning as soon as possible.
- Repair - The actual elimination of the damage.

- **Report and analysis -** Conclusions should be drawn for future prevention.

11.3 Security analysis of Superfluidity use-cases

We now analyse several use-cases for security issues. For each use case we first bring the general description and the methods of the virtualization, next we mention possible security issues which arise when a service is deployed as VNF, and finally we provide possible solution directions.

11.3.1 5G RAN - Network slicing

General description

5G run will support split of micro-services, will be dynamically scalable in both ways.

Slices should provide networking allocations for various services (e.g. a slice for each different service type). The services should be dynamically allocated on the cloud. Hence, this is a heavy networking resources exploiting service. The virtualization would provide the elasticity and scalability of the slicing resolution. The service is large and complex, hence can be logically partitioned into (not necessarily disjoint) sub-services.

Security challenges

As a networking service - the virtualization of such paradigm will involve profound security control which will involve treatment of all risks described above.

Possible solution direction

We mention here general security issues concerning the data flows. The sub-modules of this service block (control, load balancing) are discussed separately below. Secure slicing service should be provided. The challenge is to provide security without affecting the QoS resolution and scalability, to supply secure control plane for the various networking slices and to concurrently provide the basic radio resource management (MAC services).

11.3.1.1 Wireless Software Defined fronthauling (WSDF)

General description-WSDF

Provides the control plan for the 5G - RAN, and actually acts as a main managing unit of the 5G RAN.

It controls the capacity allocation, opening and closing of new resources, reallocation of provided network slices. As such, it operates with resources such as Ethernet ports, SDN agents, Load balancing functionalities, Interface and synchronizations with and among micro services. In addition, once deployed as a complex set of VNFs, this component will be responsible for the effective virtualization.

This will involve virtual resources allocation and further orchestration.

Security challenges - WSDF

All attacks mentioned in this document can be a potential threat for this complex control system. Hence, the design of the control plane should be based on the security needs.

Possible solutions - WSDF

In order to cover multiple security issues associated with the control plane the best solution is conceived in the design of security-aware services. This will include integration of the fronthaul network in a control plane separately deployed (see [108] proposing such an idea) on the private cloud. Namely, this is suggested to be implemented in the following layers.

- Module-based security. The system will be deployed as a hierarchical structure of services controlled by a main orchestrator. This orchestrator will be located in secure cloud and connected to secondary orchestrators by secured links. See Figure 67 for a possible deployment scheme.
- Secondary orchestrators are also advised to be located in the secure cloud. The main reason for this secure structure is to hide the most sensitive parts from all potential threats. Hence, once the entire cloud RAN fronthauling centre is virtualized, it is proposed that secured HW will be used.

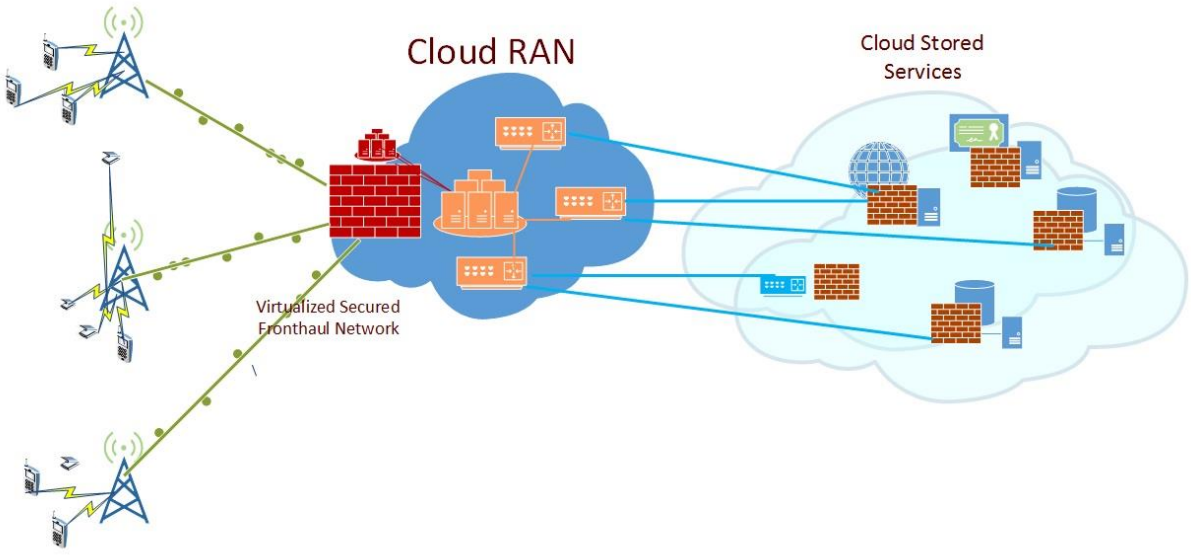


Figure 67: Deployment of virtualized cloud RAN. The main fronthaul layer is mainly attributed to the control interface between the cloud-RAN and users. It is proposed to be deployed at separate secured cloud domain. Hence, the cloud RAN will be deployed in the private cloud which will comprise dedicated security measures specifically applied to this designated domain.

Note that RAN constitutes a set of services with complex structure of various immediately responsive subservices. Hence, in order to be efficiently operated and concurrently secured, it should be *designed* taking into account the aforementioned security demands.

11.3.1.2 Dynamic MAC services allocation in Cloud RAN

General description - Dynamic MAC services

This service will cope with dynamicity of the wireless networks by addressing NFV resource management in the case of load changes, handovers, rapid migration, and by providing virtualized load balancing.

Security challenges - Dynamic MAC services

Load-balancing operations are vulnerable to attacks both on transport and on message level. The transport level, if load balancer (LB) is virtualised, presumes deployment of the *attached firewall*. See Microsoft implementation guidelines [106]

Hence, *virtualised* firewall will be attached to each instance of radio LB.

In the case of multiple instances and in the case of certain users, messaging between the LB and user or among various LBs should be enabled. Hence, these messages should be delivered by means of a secured channel and properly encrypted.

11.3.2 Virtualization of home environment - Virtual home gateways (VHG)

General description - VHG

Aims to move traditional functions (e.g., firewall, parental control, NAT, etc.) residing at the customers' home to a virtual home gateway in the cloud. Allows deploying new services, or upgrading them without changing anything on customers' equipment.

Security challenges - VHG

The main concern in this VNF is privacy and vulnerabilities associated with identity, federated identity and authentication. Additional concern is related to intrusions of all kinds. For example, users which experience malware effect can potentially harm other users by spreading the malware via virtualized centre.

11.3.3 Local breakout - 3GPP optimization

General description - 3GPP optimization

This service's purpose is to apply to situations where communicating sides are attached to the same edge of the network, i.e., geographically close. In this case, it is desirable that the traffic can flow between them directly, not going to the mobile core, which is today the default behaviour.

Security challenges - 3GPP optimization

The main security concern here is disrupting the correct functioning of this service. In this case, the opposite effect can be achieved, once closed users will communicate via remote site.

Therefore, the implementation of this VNF should be performed with security-aware insight.

The servers or orchestrators which will be responsible for the connectivity should be properly firewalled and the messages between them should be provisioned by means of secured links and enhanced by encryption. This service may also suffer from the effect of a malicious user that intentionally reports false location.

11.3.4 Virtualized Internet of things - VIoT

General description - VIoT

Virtualization of Machine 2 Machine communication, control center of "things", IoT Virtual Network.

In this category, we also include Virtualized wireless sensors network (VWSN), and hierarchical WSN which in general is described as a lightweight application service, demanding fast control, with multiple low bandwidth communication links.

Security challenges - VIoT

The effective management of IoT devices will be provided by specialized services deployed on the cloud as dedicated IoT managing VNFs. Hence the actual control and orchestration of IoT devices and units will be implemented on virtualized manner as a dedicated IoT VNF.

The implementation of control of IoT is vulnerable to various potential attacks, on the cloud.

Each IoT instance, whether virtualized or not, incorporates inherently security-related hazards mainly related to the privacy of the corresponding enterprises. Therefore, the elimination of all possible threats should be treated within the framework of secured identification and authentication mechanisms, specifically designed for the NFV demands.

11.3.5 Virtual CDN for TV content distribution

General description - CDN

Assume content cache deployment, close to network edge based on behaviour analysis and forecasts.

Such a deployment is ordinarily associated with virtualised CDN allowing several players to deploy their own CDN, according to their rules and needs.

Note that current Internet contents distribution CDNs are based on traditional content caching algorithms, based on observed content popularity. Hence, these CDNs apply similar distribution rules for all contents.

However, various types of services require different constraints to cache contents. In particular, geographical binding has a crucial impact on latency of content delivery, hence, the content is presumed to be cached in accordance to users' locations, which is especially important for applications like TV.

Usually, the contents of interest are stored in the *edge repository* and become available after their scheduled transmission time. However, after that, they stay available in the edge for only a defined period of time (that is, the interest in certain TV content becomes quite rapidly outdated) after that, they become available only from a central data centre, freeing storage space for other contents.

Security challenges

In the described context, the security will depend on the time-stamp of the content. Hence, we propose to logically divide the content according to one of two possible stages.

1. Is of high interest. This content will be in high demand in the next two days.
2. Archived content. After approximately, e.g., 48 hours, the content loses wide public interest, hence may be moved to the remote, occasionally accessed storage.

Clearly in the first phase, the content will be classified as highly sensitive, thus vulnerable to the various types of attacks, especially DoS attacks. Therefore, CDN for TV content which is deployed as a set of VNFs will need special attention of security enhanced services during the first stage.

After that stage, the security effort can be reduced to the normal standards.

11.3.6 Pure security service and their virtualization

In this subsection we mention virtualized services which will augment other services by providing additional security and trust application.

11.3.6.1 Preventing NDP (Neighbour Discovery Protocol) spoofing

NDP is an IPv6 substitution of the IPv4 address resolution protocol (ARP). It also performs IND (Inverse Neighbour Discovery) which acts similarly to RARP. Unlike ARP, it does not necessarily act on pure link layer, e.g., Ethernet. Namely, NDP acts over ICMPv6, hence it sends IP level packets. This creates the inherent vulnerability and the threat of NDP spoofing.

In order to neutralize the threat all virtualized networks should secure Neighbour Discovery (SEND) Protocol. SEND prevents a potential attacker from accessing the broadcast segment, thus preventing abusing NDPs to trick hosted VMs into sending the attacker any traffic designated to someone else. This technique is also known as ARP poisoning. SEND uses RSA encryption in order to assure security.

This practice should be specified as obligatory for secured and sensitive VNFs.

11.3.6.2 Protection against DDoS (Distributed Denial of Service) attacks

DDoS is a type of DOS attack where multiple infected systems, (e.g., by Trojan), are exploited to attack a single target in order to inflict the Denial of Service (DoS).

DDoS is harder to implement from the side of a potential attacker but likewise is harder to overcome.

A cleverly organized DDoS can be disguised as a common load peak situation and can thus affect hosts, hypervisors, load balancers and other virtualized objects. It can be activated from various locations, by hiding botnets within larger cloud-deployed applications, neighbouring VNFs or even enterprises. Normally, the side(s) which participated in the attack are unaware of this and also considered as victims.

The most effective measure against DDoS is prevention. In particular, as long as multiple logical links would be involved in the attack, with multiple VMs, some of them possibly belonging to the same VNF, deployment should be performed with awareness of possibility of such an attack. Hence, global defending mechanism which will prevent multiple VMs from being infected by such a malware, combined with Firewall(s) protecting from malicious flows from outside should serve as a security prototype. This prototype should be part of the VNF's *initial design*, rather than application of patches in the future. As such, it will be more effective and cost-effective in the future as well.

11.3.6.3 Emergency treatment and recovery VNF

Emergency recovery applications can be implemented as an instantaneously deployable VNF, which will act in the case of global disaster, multiple VM collapse, etc. In order to be successfully activated, it should be hosted on several hypervisors and be active in sleep mode.

For greater security, it is suggested to separately occupy a dedicated HW device. The Emergency VNF (EVNF) will be implemented as self-orchestrated and self-deployable. Its tasks will include the following functionalities:

- EVNF will be automatically triggered once certain conditions will conform to the situation of system collapse.
- Detection of damaged sites by simple probing (e.g., pinging). Once probing results are ready it will send them within a designated packet to a set of manually preconfigured addresses.
- The EVNF will have access to logs and snapshots. Using this data will facilitate deployment of hypervisors and orchestrators which will serve as alternatives to those that are currently out of function.
- EVNF will be immune to all second-wave attacks by rejecting all incoming data which is not expected and from addresses which are not expected to send any packets. This will ensure any further attacking and abuse of the EVNF
- EVNF will send report messages to the enterprises with summary of the infected HW and stopped SW in the environment which is within its responsibility area.

12 REFERENCES (PART A)

[1] Malik Kamal Saadi, 5G: From Vision to Framework, ABI Research, 2015.

[2] ITU-R M.2083-0: IMT Vision - Framework and overall objectives of the future development of IMT for 2020 and beyond. International Telecommunications Union – Radiocommunication Sector. September 2015.

[3] ETSI ISG NFV: Network Functions Virtualisation (NFV); Architectural Framework. ETSI GS NFV 002 V1.2.1. December 2014.

[4] N. McKeown, et al. “OpenFlow: enabling innovation in campus networks”, ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.

[5] ETSI ISG NFV: Network Functions Virtualisation (NFV): Virtual Network Functions Architecture. ETSI GS NFV-SWA 001 V1.1.1. December 2014.

[6] ETSI ISG NFV: Network Functions Virtualisation (NFV): Management and Orchestration. ETSI GS NFV-MAN 001 V1.1.1. December 2014.

[7] ETSI ISG NFV: Network Functions Virtualisation (NFV): NFV Performance & Portability Best Practices. ETSI GS NFV-PER 001 V1.1.2. December 2014.

[8] Cisco ASA 5510 Adaptive Security Appliance http://www.cisco.com/c/en/us/support/security/asa-5510-adaptive-security-appliance/model.html

[9] 5G NORMA D3.1: Functional Network Architecture and Security Requirements v1.0. December 31, 2015.

[10] Mijumbi, R. et al.: Network Function Virtualization: State-of-the-art and Research Challenges. IEEE Communications Surveys & Tutorials. DOI: 10.1109/COMST.2015.2477041. Publication pending.

[11] G. Bianchi, E. Biton, N. Blefari-Melazzi, I. Borges, L. Chiaraviglio, P. de la Cruz Ramos, P. Eardley, F. Fontes, M. J. McGrath, L. Natarianni, D. Niculescu, C. Parada, M. Popovici, V. Riccobene, S. Salsano, B. Sayadi, J. Thomson, C. Tselios, G. Tsolis, “Superfluidity: a flexible functional architecture for 5G networks”, Trans. on Emerging Telecommunication Technologies, Wiley, 2016

[12] 3GPP TR 22.852, Study on Radio Access Network (RAN) Sharing enhancements, Rel.13, Sep. 2014

[13] 3GPP TS 23.251, Network Sharing; Architecture and Functional Description, Rel.12, Jun 2014.

[14] 3GPP TR 32.842, Telecommunication management; Study on network management of virtualized networks, Rel. 13, Dec 2015

[15] 3GPP TS 28.500, Telecommunication management; Concept, architecture and requirements for mobile networks that include virtualized network functions, Rel. 14, Dec 2015

[16] A. Gopalasingham, L. Roullet, Nessrine Trabelsi, C. Shue Chen, A. Hebbar, et al.. Generalized Software Defined Network Platform for Radio Access Networks. IEEE Consumer Communications and Networking Conference (CCNC), Jan 2016, Las Vegas, United States. 2016.

[17] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek. “The Click modular router”, ACM Transactions on Computer Systems (TOCS) 18, no. 3 (2000): 263-297

[18] P. Neves, et al., “The SELFNET Approach for Autonomic Management in an NFV/SDN Networking Paradigm”, International Journal of Distributed Sensor Networks, Vol. 2016, 2016.

[19] N. Fernando, S.W. Loke, and W. Rahayu, "Mobile cloud computing: A survey." Future Generation Computer Systems vol. 29, no.1 pp. 84-106, 2013.

[20] G. Bianchi, et al., “OpenState: programming platform-independent stateful OpenFlow applications inside the switch,” ACM SIGCOMM Computer Communication Review, Vol. 44, N. 2, pp. 44-5, 2014
[21] J. Martins et al, “ClickOS and the Art of Network Function Virtualization”, USENIX NSDI ’14, Seattle, USA, 2014.
[22] https://www.ietf.org/proceedings/95/slides/slides-95-nfvrg-2.pdf
[23] https://www.ericsson.com/research-blog/sdn/unikernels-meet-nfv/
[24] P. Aranda, “High-level VNF Descriptors using NEMO”, draft-aranda-nfvrg-recursive-vnf-01, IETF draft, work in progress
[25] P. Eardley (ed.), “Use cases, technical and business requirements”, Deliverable D2.1 of Superfluidity project
[26] OpenMANO project home page, https://github.com/nfvlabs/openmano/wiki
[27] The Official YAML web page, http://yaml.org/
[28] Y. Xia, Ed. et al, “NEMO (NEtwork MOdeling) Language”, Internet Draft, draft-xia-sdnrg-nemo-language-04, https://datatracker.ietf.org/doc/draft-xia-sdnrg-nemo-language/
[29] OpenDaylight NEMO project, https://wiki.opendaylight.org/view/NEMO:Main
[30] T. Berners-Lee, R. Fielding, “Uniform Resource Identifier (URI): Generic Syntax”, IETF RFC 3986
[31] Docker Home Page, https://www.docker.com/
[32] 3GPP TR 23.714, Study on Control and User Plane Separation of EPC nodes, Rel. 14, Jun 2016
[33] https://wiki.openstack.org/wiki/Telemetry#Ceilometer
[34] B. Sayadi (ed.) “Functional Analysis and Decomposition”, Deliverable D2.2 of Superfluidity project
[35] K. T. Cheng and A. S. Krishnakumar, “Automatic Functional Test Generation Using The Extended Finite State Machine Model,” in ACM Int. Design Automation Conference (DAC), 1993, pp. 86–91
[36] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, C. Cascone, “Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing”, 2016. arXiv preprint arXiv:1605.01977.
[37] ETSI ISG NFV: NFV Release 2 Description; https://docbox.etsi.org/ISG/NFV/Open/Other/NFV(16)000274r3_NFV%20Release%202%20Description%20v10.pdf
[38] ETSI ISG NFV: Network Functions Virtualisation (NFV); Management and Orchestration; Functional requirements specification. ETSI GS NFV-IFA 010 V2.2.1 (2016-09).
[39] ETSI ISG NFV: Network Function Virtualization (NFV); Management and Orchestration; Report on NFV Information Model. GR NFV-IFA 015 V0.7.0 (2016-09).
[40] ETSI ISG NFV: Network Functions Virtualisation (NFV); Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification. ETSI GS NFV-IFA 013 V2.1.1 (2016-10).
[41] ETSI ISG NFV: Network Functions Virtualisation (NFV); Management and Orchestration; Network Service Templates Specification. ETSI GS NFV-IFA 014 V2.1.1 (2016-10).
[42] ETSI ISG NFV: Network Functions Virtualisation (NFV); Management and Orchestration; VNF Packaging Specification. ETSI GS NFV-IFA 011 V2.1.1 (2016-10).
[43] ETSI ISG NFV: Network Functions Virtualisation (NFV); Protocols and Data Models; NFV descriptors based on TOSCA Specification. ETSI GS NFV-SOL 001 V0.0.2 (2016-07).
[44] OASIS: TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0 (17 March 2016).

[45] 3GPP releases, http://www.3gpp.org/specifications/releases
[46] S. Stuijk, M. Geilen, B. Theelen AND T. Basten, “Scenario-Aware Dataflow: Modeling, Analysis and Implementation of Dynamic Applications”, International Conference on Embedded Computer Systems (SAMOS), 2011
[47] iperf page on Wikipedia: https://en.wikipedia.org/wiki/Iperf

13 REFERENCES (PART B)

[48] G. Badishi, I. Keidar and A. Sasson, “Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast,” IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 1, pp. 45-61, 2006

[49] J. Barros and M. R. Rodrigues, “Secrecy capacity of wireless channels,” in IEEE International Symposium on Information Theory (ISIT), 2006

[50] K. Bhattad and K. R. Narayanan, “Weakly secure network coding,” in NetCod, 2005.

[51] M. Bloch and J. Barros, “Physical-Layer Security: From Information Theory to Security Engineering”, Cambridge University Press, 2011.

[52] M. R. Leadbetter, Extreme and related properties of random sequences and processes. Springer-verlag, 1983

[53] M. Bloch, R. Narasimha and S. W. McLaughlin, “Network security for client-server architecture using wiretap codes,” IEEE Transactions on Information Forensics and Security, vol. 3, no. 3, pp. 404--413, 2008.

[54] S. Boyd, A. Ghosh, B. Prabhakar and D. Shah, “Randomized gossip algorithms,” EEE/ACM Transactions on Networking (TON), vol. 14, no. SI, pp. 2508--2530, 2006.

[55] N. Cai and T. Chan, “Theory of secure network coding,” Proceedings of the IEEE, vol. 99, no. 3, pp. 421--437, 2011.

[56] N. Cai and R. W. Yeung, “Secure network coding,” in IEEE International Symposium on Information Theory (ISIT), 2002

[57] N. Cai and R. W. Yeung, “A security condition for multi-source linear network coding,” in IEEE International Symposium on Information Theory (ISIT), 2007.

[58] N. Cai and R. W. Yeung, “Secure network coding on a wiretap network,” IEEE Transactions on Information Theory, vol. 57, no. 1, pp. 424--435, 2011.

[59] N. Cai, “Valuable messages and random outputs of channels in linear network coding,” in IEEE International Symposium on Information Theory (ISIT), 2009.

[60] J. L. Carter and M. N. Wegman, “Universal classes of hash functions,” in Proceedings of the ninth annual ACM symposium on Theory of computing, 1977.

[61] T. Chan and A. Grant, “Capacity bounds for secure network coding,” in Australian Communications Theory Workshop, 2008.

[62] A. Cohen, B. Haeupler, C. Avin and M. Médard, “Network coding based information spreading in dynamic networks with,” IEEE Journal on Selected Areas in Communications, vol. 33, no. 2, pp. 213--224, 2015.

[63] S. Coles, “An introduction to statistical modeling of extreme values,” Springer Verlag, 2001.

[64] I. Csiszár and J. Korner, “Broadcast channels with confidential messages,” IEEE Transactions on Information Theory, vol. 24, no. 3, p. 339–348, 1978.

[65] D. Supratim, M. Médard and C. Choute, “Algebraic gossip: a network coding approach to optimal multiple rumor mongering,” IEEE/ACM Transactions on Networking (ToN), vol. 14, pp. 2486--2507, 2006.

[66] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart and D. Terry, “Epidemic algorithms for replicated database maintenance,” in Sixth annual ACM Symposium on Principles of distributed, 1987.

[67] S. Y. El Rouayheb and E. Soljanin, “On wiretap networks II,” in IEEE International Symposium on Information Theory (ISIT), 2007.

[68] S. El Rouayheb, E. Soljanin and A. Sprintson, “Secure network coding for wiretap networks of type II,” IEEE Transactions on Information Theory, vol. 58, no. 3, pp. 1361--1371, 2012.
[69] P. Embrechts, C. Klüppelberg and T. Mikosch, “Modelling extremal events: for insurance and finance,” Springer, vol. 33, 2011.
[70] S. Ali, A. Fakoorian and A. L. Swindlehurst, “On the optimality of linear precoding for secrecy in the MIMO broadcast channel,” IEEE Journal on Selected Areas in Communications, vol. 31, no. 9, p. 1701–1713, 2013.
[71] G. Geraci, S. Singh, J. G. Andrews, J. Yuan and C. I. B., “Secrecy rates in broadcast channels with confidential messages and external eavesdroppers,” IEEE Transactions on Wireless Communications,, vol. 13, no. 5, p. 2931–2943, 2014.
[72] G. Geraci, J. Yuan, A. Razi and I. B. Collings, “Secrecy sum-rates for multi-user MIMO linear precoding,” in IEEE 8th International Symposium on Wireless Communication Systems (ISWCS),, 2011.
[73] R. Guerraoui, K. Huguenin, A. M. Kermarrec, M. Monod and S. Prusty, “On tracking freeriders in gossip protocols,” in IEEE Ninth International Conference on Peer-to-Peer Computing, 2009.
[74] B. Haeupler, “Analyzing network coding gossip made easy,” in 43rd annual ACM symposium on Theory of computing, 2011.
[75] K. Harada and H. Yamamoto, “Strongly secure linear network coding,” IEICE transactions on fundamentals of electronics, communications and computer sciences, vol. 91, no. 10, pp. 2720--2728, 2008.
[76] K. P. Jagannathan, S. Borst, P. Whiting and E. Modiano, “Efficient scheduling of multi-user multi-antenna systems,” in 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006.
[77] R. Karp, C. Schindelhauer, S. Shenker and B. Vocking, “Randomized rumor spreading,” in Annual Symposium On Foundations Of Computer Science, 2000.
[78] D. Kempe, A. Dobra and J. Gehrke, “Gossip-based computation of aggregate information,” in FOCS, 2003.
[79] A. Khisti and G. W. Wornell, “Secure transmission with multiple antennas I: The MISOME wiretap channel,” IEEE Transactions on Information Theory, vol. 56, no. 7, p. 3088–3104, 2010.
[80] A. Khisti and G. W. Wornell, “Secure transmission with multiple antennas-part II: The MIMOME wiretap channel,” IEEE Transactions on Information Theory,, vol. 56, no. 11, p. 5515–5532, 2010.
[81] D. Kobayashi, H. Yamamoto and T. Ogawa, “Secure multiplex coding to attain the channel capacity in wiretap channels,” arXiv preprint cs/0509047, 2005.
[82] I. Krikidis and B. Ottersten, “Secrecy sum-rate for orthogonal random beamforming with opportunistic scheduling,” IEEE Signal Processing Letters,, vol. 20, no. 2, p. 141–144, 2013.
[83] J. H. Lee and W. Choi, “Multiuser diversity for secrecy communications using opportunistic jammer selection: Secure dof and jammer scaling law,” IEEE Transactions on Signal Processing,, vol. 62, no. 4, p. 828–839, 2014.
[84] S. Leung-Yan-Cheong and M. E. Hellman, “The Gaussian wire-tap channel,” IEEE Transactions on Information Theory, vol. 24, no. 4, pp. 451-456, 1978.
[85] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi and M. Dahlin, “BAR gossip,” in 7th symposium on Operating systems design and implementation, 2006.
[86] J. Li and A. P. Petropulu, “On ergodic secrecy rate for Gaussian {MISO} wiretap channels,” IEEE Transactions on Wireless Communications, vol. 10, no. 4, pp. 1176-1187, 2011.

[87] S.-Y. R. Li, R. W. Yeung and N. Cai, “Linear network coding,” IEEE Transactions on Information Theory, vol. 49, no. 2, pp. 371-381, 2003.
[88] R. Matsumoto and M. Hayashi, “Universal strongly secure network coding with dependent and non-uniform messages,” in arXiv preprint arXiv:1111.4174, 2011.
[89] S. Nadarajah and A. H. El-Shaarawi, “On the ratios for extreme value distributions with application to rainfall modeling,” Environmetrics, vol. 17, no. 2, pp. 147-156, 2006.
[90] F. Oggier and B. Hassibi, “The secrecy capacity of the MIMO wiretap channel,” IEEE Transactions Information Theory, vol. 57, no. 8, p. 4961–4972, 2011.
[91] L. H. Ozarow and A. D. Wyner, “Wire-tap channel II,” in Advances in Cryptology, 1985.
[92] S. Shafiee, N. Liu and S. Ulukus, “Towards the secrecy capacity of the gaussian MIMO wire-tap channel: The 2-2-1,” IEEE Transactions on Information Theory, vol. 55, no. 9, p. 4033–4039, 2009.
[93] S. Shafiee and S. Ulukus, “Achievable rates in gaussian MISO channels with secrecy constraints,” in IEEE International Symposium on Information Theory (ISIT), 2007.
[94] M. Sharif and B. Hassibi, “On the capacity of MIMO broadcast channels with partial side information,” IEEE Transactions on Information Theory,, vol. 51, no. 2, p. 506–522, 2005.
[95] D. Silva and F. R. Kschischang, “Universal weakly secure network coding,” in IEEE Information Theory Workshop (ITW), 2009.
[96] C. Wang, H. Wang, X. Xia and C. Liu, “Uncoordinated jammer selection for securing SIMOME wiretap channels: A stochastic geometry approach,” IEEE Transactions on Wireless Communications,, vol. 14, no. 5, pp. 2596-2612, 2015.
[97] P. Wang, G. Yu and Z. Zhang, “On the secrecy capacity of fading wireless channel with multiple eavesdroppers,” in IEEE International Symposium on Information Theory (ISIT), 2007.
[98] A. D. Wyner, “The wire-tap channel,” The Bell System Technical Journal, vol. 54 , no. 7, p. 1355–1387, 1975.
[99] R. W. Yeung, Information theory and network coding, Springer, 2008.
[100] J. Kampeas, A. Cohen and O. Gurewitz, “On Secrecy Rates and Outage in Multi-User Multi-Eavesdroppers MISO Systems,” arXiv preprint arXiv:1605.02349, 2016.
[101] W. Gautschi, “The incomplete gamma functions since Tricomi,” in In Tricomi's Ideas and Contemporary Applied Mathematics, Atti dei Convegni Lincei, n. 147, Accademia Nazionale dei Lincei, 1998.
[102] Laforgia and P. Natalini, “On some inequalities for the Gamma function,” Advances in Dynamical Systems and Applications, vol. 8, no. 2, pp. 261-267, 2013.
[103] C. Mortici, “New sharp bounds for gamma and digamma functions,” An. Stiint. Univ. AI Cuza Iasi Ser. N. Mat, vol. 56, no. 2, 2010.
[104] E. Pagnin, C. Dimitrakakis, A. Abidin, and A. Mitrokotsa. 2014. "On the leakage of information in biometric authentication." Progress in Cryptology - INDOCRYPT. Springer. 265-280
[105] M. Haghighat, S. Zonouz, and M. Abdel-Mottaleb. 2015. "Cloudid: Trustworthy cloud-based and cross-enterprise biometric identification." Expert Systems with Applications vol. 42, no. 21 pp. 7905-7916
[106] Things to consider when implementing a load balancer with wcf. https://msdn.microsoft.com/en-us/library/hh273122%28v=vs.100%29.aspx
[107] Wueest, C. 2014. "Threats to virtual environments." Symantec Research
[108] A. de la Oliva, "Xhaul: The 5g integrated fronthaul/backhaul."

[109] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, “Automatic test packet generation”, Proceedings of the 8th international conference on Emerging networking experiments and technologies (CoNEXT '12), 2012.

[110] P. Kazemian, G. Varghese, N. McKeown, “Header space analysis: Static checking for networks”, 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2012)

[111] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, G. Varghese, “Checking Beliefs in Dynamic Networks”, ”, 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2015)

[112] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, “P4: Programming Protocol-Independent Packet Processors”, CCR July 2014

[113] S. K. Fayaz, Tianlong Yu, Y. Tobioka, S. Chaki, V. Sekar, “BUZZ: Testing Context-Dependent Policies in Stateful Networks”, Usenix NSDI 2016

[114] R. Stoenescu, M. Popovici, L. Negreanu, C. Raiciu “Symnet: scalable symbolic execution for modern networks”, ACM SIGCOMM 2016

[115] R. Stoenescu, D. Dumitrescu and C. Raiciu, “OpenStack networking for humans: symbolic execution to the rescue”, IEEE LANMAN 2016

[116] C. Cadar, D. Dunbar, D. Engler, “KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs”, USENIX OSDI 2008