SUPERFLUIDITY

A SUPER-FLUID, CLOUD-NATIVE, CONVERGED EDGE SYSTEM

Research and Innovation Action GA 671566

DELIVERABLE I6.2:

INITIAL DESIGN FOR SLA BASED DEPLOYMENT

Deliverable Type:	Report
Dissemination Level:	Public
Contractual Date of Delivery to the EU:	01/07/2016
Actual Date of Delivery to the EU:	12/07/2016
Workpackage Contributing to the Deliverable:	WP6
Editor(s):	Itai Segall (Nokia IL Erez Biton (Nokia IL)
Author(s):	Itai Segall (Nokia IL, Erez Biton (Nokia IL), George Tsolis (Citrix), Carlos Parada (Altice Labs), Isabel Borges (Altice Labs), Francisco Fontes (Altice Labs), Omer Gurewitz (BGU), Mark Shifrin (BGU), Pedro Andres Aranda Gutierrez (Telefónica, I+D), Haim Daniel (Red Hat)
Internal Reviewer(s)	Omer Gurewitz (BGU)
Abstract:	This internal deliverable carries report on the design of an SLA based descriptors and APIs that are required for the access-agnostic deployment of the network

	service as envisioned in SUPERFLUIDITY. The deliverable describes the initial algorithms for the deployment and SLA provision.
Keyword List:	Orchestration, Management

INDEX

List of Figures

List of Tables

Glossary

SUPERFLUIDITY DICTIONARY	
TERM	DEFINITION
UE	User Equipment
OSS	Operation Support System
VIM	Virtual Infrastructure Management
VM	Virtual Machine
MANO	Management And Orchestration
NFV	Network Function Virtualization
KPI	Key Platform Indicator

Table 1: SUPERFLUIDITY Dictionary.

Introduction

1.1 Deliverable Rationale

This internal deliverable will report on the design of an SLA based descriptors and APIs that are required for the access-agnostic deployment of the network service as envisioned in SUPERFLUIDITY. The deliverable should further describe the initial algorithms for the deployment and SLA provision.

1.2 Quality Review

Review Team member responsible of the deliverable: _____

VERSION CONTROL TABLE			
VERSION N.	PURPOSE/CHANGES	AUTHOR	DATE
1	I6.2 draft		

1.3 Executive summary

1.3.1 Deliverable description

In order to realize the vision developed in the Superfluidity project, there is a need to develop an effective management and orchestration system. Applications and service chains deployed in this system will declare their Quality of Service (QoS) requirements, and the system will be able to gather these requirements, compare them against the existing resources, and their current and planned utilization, and take placement decisions to facilitate these requirements. These decisions consist of initial placement and allocation, as well as ongoing migration as required.

1.3.2 Summary of results

This document describes the initial steps taken towards SLA-aware management and orchestration.

We first present NEMO – a language to be used as SLA descriptor, for an application owner to declare the QoS requirements of her application. NEMO is a human-readable command language used for Network Modelling. It is placed by the authors in the scope of Intent Based Networking (IBN), since it is more descriptive and prescriptive. NEMO provides basic network commands (Node, Link, Flow, Policy) to describe the infrastructure and controller communication commands to interact with the controller. We are proposing to extend the Node definition command to import TOSCA or OSM based descriptors as Node definitions. Since Node models can make use of previously defined node models, the resulting language would be recursive and therefore support our notion of (recursive) reusable function blocks.

Resource allocation and placement is examined in this document both for service chains in NFV deployments, as well as for MEC applications. For service chains, we study two possible placement strategies – one that gathers all components of each chain into the same host, and one that distributes them between different hosts. This is performed both in the presence of hardware acceleration (DPDK), and without it. Following the comprehensive evaluation of the two placement strategies of service chains, we also model the cost of network switching. Given an arbitrary number of service chains, our model accurately predicts the CPU cost of the network switching they require. In the context of MEC applications, we discuss the management and orchestrator and the resource allocation problem. We also examine service migration for MEC applications.

We further formulize the scaling decision and load balancing problems. In particular, we formulize the decision whether to increase the resources allocated to a certain VNF (scale out operation) or to release some of these resources (scale in operation) based on the current and expected demand from the network service and the required QoS as a Markov Decision Process (MDP). Our formulation also incorporates the load balancing challenge steering the traffic flows to the different VMs and balancing the load between them.

2 SLA-Based Descriptors

2.1 NEMO

NEMO [1] is a human-readable command language used for Network Modelling. It is placed by the authors in the scope of Intent Based Networking (IBN), since it is more descriptive and prescriptive. The NEMO project has launched a series of efforts to get the language standardised. As such, the IBNEMO project within the OpenDaylight (ODL) community is classified in the Intent Based northbound interfaces (NBIs) group.

NEMO provides basic network commands (Node, Link, Flow, Policy) to describe the infrastructure and controller communication commands to interact with the controller.

2.1.1 Enhancements proposed in SUPERFLUIDITY

We are proposing to extend the Node definition command to import TOSCA or OSM based descriptors as Node definitions. Since Node models can make use of previously defined node models, the resulting language would be recursive and therefore support our notion of (recursive) reusable function blocks.

This concept has been proposed as an Internet draft [2] at the NFV research group (NFV-RG) of the IRTF. In addition to the import process proper, two additional features are defined in NeMo: 1.- the ConnectionPoint to map the VNF's interfaces that are significant in the function description and the connections between them, and 2.- the CONNECTION as a way to express the relations between the enhanced VNFCs (here NodeModels) in a service graph.

Importing OSM VNF Descriptors is proposed in the draft as a two-step process, where the descriptor is first imported and then used to provide the pointers to the connection points. The proposed syntax to import VNFDs is:

```
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:ge
```

The proposed way to define service graphs in NeMo is:

```
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:wa TYPE p2p
       CONNECTION internal FROM InputVNF:data_outside TO ShaperVNF:lan TYPE p2p
```

3 Hardware Offloading

Another feature to be taken into account when considering SLA-based deployment is that of hardware acceleration. Different applications may have different acceleration requirements, which can or can not be supported by various infrastructure components.

Many hardware acceleration technologies have been developed over the years (both proprietary and open-source). Typically, acceleration technologies implement new hardware interfaces that are tailored to bypass performance weaknesses in the specific domain of acceleration (e.g., domain of packet processing flow). However, these technologies are usually cumbersome to configure and deploy, and they are hard to program for. More importantly, these accelerations usually carry with them security risks. Such vulnerabilities are mostly related to developing applications in error-prone environments that might be abused by malicious parties.

3.1 Data Plane Development Kit (DPDK)

A very successful open source technology (lead by Intel) that draws attention in recent years is DPDK (Data Plane Development Kit) [3] . DPDK is a set of user-space libraries that enables a user space application to manage and own the hardware NIC, thus completely bypassing the kernel networking stack.

Shifting the entire networking stack to the user space and implementing a poll mode driver, overcomes two major performance weaknesses: (i) zero-copy packet forwarding from the NIC to the user space application, and (ii) no need for context switching to handle interrupts. On the other hand, since DPDK requires the user to manage and own the physical hardware, the user must acquire privilege rights and install a complete network stack (in the user space) that processes the entire traffic. For security sensitive service providers (such as telecommunication industry), this weakness might be a show stopper.

Nevertheless, we assume that at some point in the future, some of these security obstacles will be resolved, and DPDK will be part of many NFV deployments.

4 Management and Orchestration

Two crucial aspects of the management and orchestration system are those of resource allocation, and migration. The former refers to the preliminary decision of which resources to allocate for each application component, while the latter discusses post-deployment changes in this decision. Typically, placement is performed based on the application's SLA requirements, as depicted using the various descriptors, as well as the current and expected status of the resources. Service migration then modifies the allocation, as decided in the initial placement, based on ongoing changes in the application behaviour and requirements, as well as changes in its environment.

4.1 Resource Allocation and Placement

4.1.1 Network Function Virtualization (NFV)

The big promise of NFV is that the shift from dedicated hardware into common servers will significantly reduce cost (both OPEX and CAPEX), and at the same time the use of virtual infrastructure will allow more agility both in terms of reducing the time to market of new services and by allowing a more dynamic allocation of resources to the different functions.

One of the main challenges in realizing this promise is the efficient use of resources since clearly when the resources are underutilizes both the equipment cost as well as the operation cost remain high. This problem becomes complex since the term ``resources'' covers different (possibly orthogonal) aspects such as the servers CPU, the storage (or disk) usage, networking capacity, and in many cases also use of common utilities (like file systems, key management, etc.). We focus here on placement strategies for network service chains.

Given a set of network service chains, and available resource (the computing and networking infrastructure) our focus is to address the question of what would be an optimal placement strategy?

To illustrate our challenge, consider the two placement strategies presented in Figure 1. In this example, we are given three identical servers A, B and C, and three identical service chains φ^1, φ^2, φ^3. Each service chain φ^i, (for i = 1, 2, 3) is tagged with a fixed amount of required traffic and is composed of three chained VNFs: $\varphi^i = < \varphi^i_1, \varphi^i_2, \varphi^i_3 >$. Figure 1(a) illustrates the first placement strategy (referred to as ``gather''), where all VNFs composing a single chain are deployed on the same server. Note that in the gather case the majority of traffic steering is done by the server's internal virtual switching. Figure 1(b) illustrates the second placement strategy (referred to as ``distribute''), where each VNF is deployed on a different server, and therefore the majority of traffic steering is done between servers by external switches.

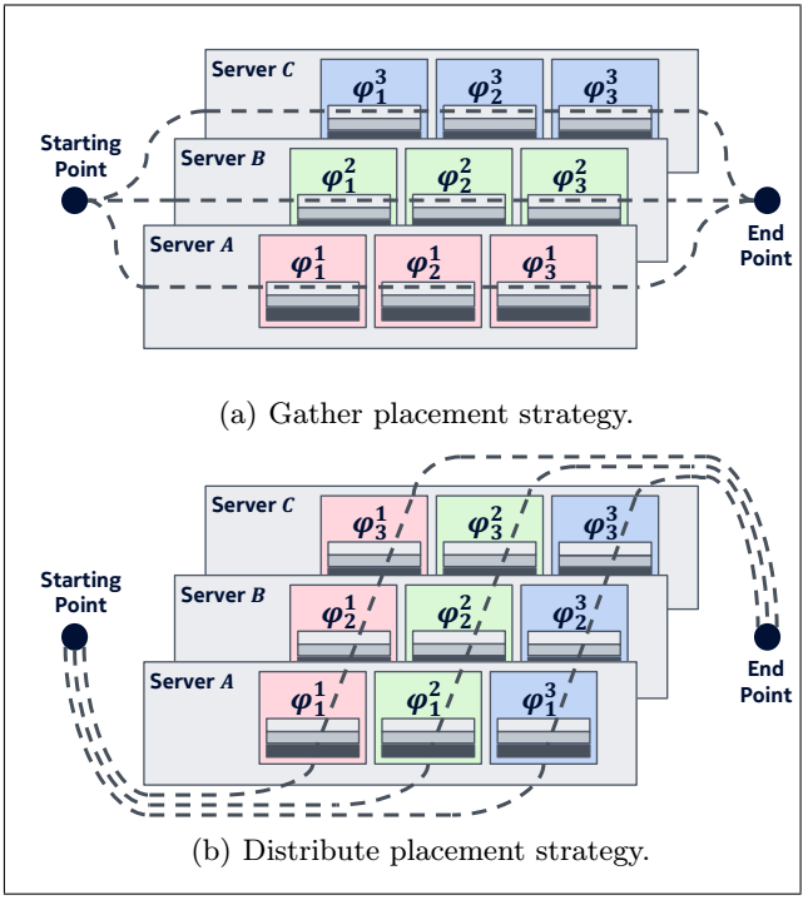

Figure 1 – Example of two placement strategies to deploy 3 given service chains

One can immediately see that these two placement strategies require from the VNF the same amount of resources to operate, however differ in the cost of their software switching (in terms of CPU consumption). Additionally, it is also not completely clear how would the network perform for each of the placement strategies, with respect to metrics such as throughput and/or latency. For instance, Figure 2 depicts the service latency (end-to-end), given a set of servers that are installed with either kernel-OvS or DPDK-OvS. Per each installed environment, we evaluate the two placement strategies discussed above. Obviously the latency increases as the chain size increases, however note that there is a non-negligible difference between the two placement strategies. When the servers are installed with kernel-OvS (resp. DPDK-OvS) the service latency of the distribute placement requires 50% more time (resp. 100% more time) than the service latency time required for the gather placement.

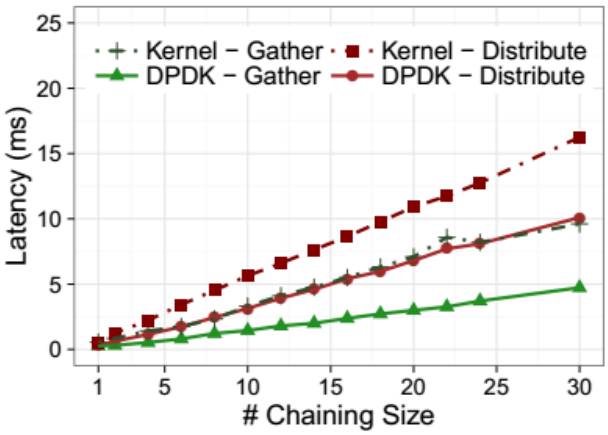

Figure 2 - Average end-to-end latency on different deployment strategies

Going back to Figure 1, in order to decide which of the placement strategy is better, we need to identify the specific optimization criteria of interest. In this paper we focus on reducing the virtual switching cost. For NFV-based infrastructure, virtual switching is an essential building block that on one hand enables flexible communication between VNFs; but on the other hand, it comes with a cost of resource utilization. Therefore, assessing and understanding this cost is a crucial step towards driving cost-efficient service deployments.

Crafting a monolithic CPU cost function for NFV-based environment is an extremely important task and essential in order to:

1 Efficiently guide VNF placement in a NFV-based infrastructure.
2 Understand how software switching impacts deployed services. Namely, analyze and comprehend how latency- and/or throughput-sensitive services behave, is an essential step towards (proper) orchestration of a multitude of services.
3 Whenever possible, reduce the costs to the NFV provider and ultimately, foster the development of cost optimized deployment solutions.

In [4] , we develop a model to estimate the cost of software switching for different placement strategies over NFV -based infrastructure. We conduct an extensive and in-depth evaluation, measuring the performance and analyzing the impact of deploying a service chain on Open vSwitch – the de facto standard software switch for cloud environments ([5] [6] [7]). Based on our evaluation, we then continue to craft a generalized abstract cost function that accurately captures the CPU cost of network switching.

The main contributions of [4] can be summarized as follows:

1 **Comprehensive evaluation of placement strategies of service chains.** Based on a real NFV-based infrastructure, we examine our placement strategies, and assess performance metrics such as throughput, packet processing, and resource consumption.
2 **Model the cost of network switching.** Given an arbitrary number of service chains, our model accurately predicts the CPU cost of the network switching they require.

4.1.2 Mobile Edge Computing (MEC)

4.1.2.1 MEC – Resources Allocation

In a MEC System, MEC Applications will run at ME (Mobile Edge) Hosts in a virtualized environment, as VDUs (*Virtualization Deployment Units*), e.g. VM or Container. For that purpose, and in line with ETSI NFV, management and orchestration components have been added to the MEC system, composed by "Mobile Edge Host Level" and "Mobile Edge System Level". That is, while the latter has a global view of the entire MEC System, including all ME Hosts, the former acts at the ME Hosts level, with the functions: Element Manager for the ME Host, Rules and Requirements management for MEC applications, and MEC Applications LCM operations (similar to VNFM for NFV). At each ME Host, the ME Platform component may or may not be deployed over the virtualization infrastructure, possibly using specific HW and SW. Figure 3 depicts the full ETSI MEC architecture.

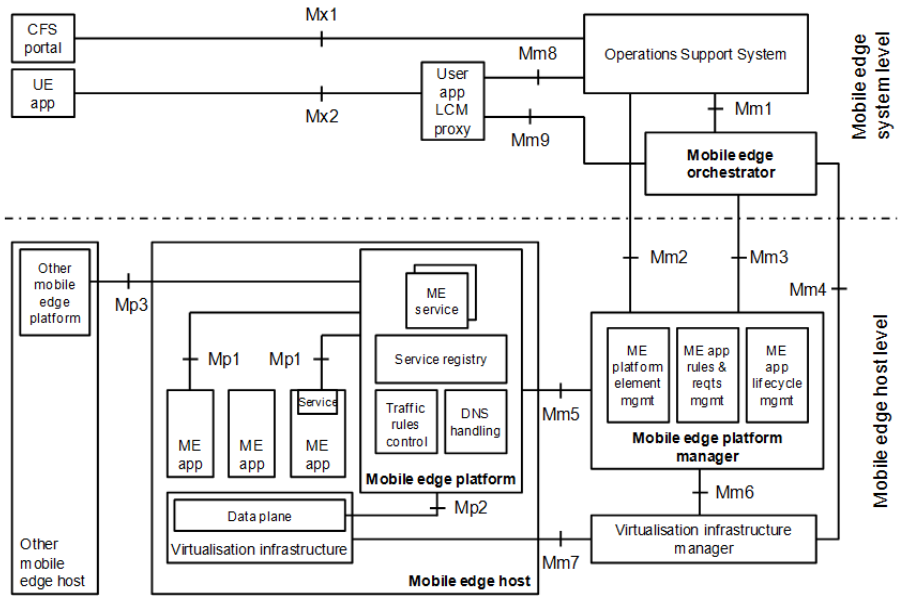

Figure 3 - ETSI MEC Architecture [ETSI-MEC-Arch.]

In this context, resources allocation to MEC Applications will occur on the virtualization infrastructure, existing at each ME Host. This will be controlled by the ME Orchestrator, as part of ME Applications Life-Cycle Management (LCM) operations. Superfluidity Deliverable I6.1 "Initial Design for Control Framework" documents MEC management and orchestration requirements as well as respective flows.

The following entities, hosted at the MEO, are required for ME Applications resources allocation:

- MEC App Catalogue

- Stores the catalogue of deployable MEC Applications, with associated images and MEC Descriptor (where applicable rules and requirements are specified)

- MEC App Descriptor
  - Even if not yet defined by the ETSI MEC ISG, MEC Applications will have a descriptor associated to them. This will be in line with the VNF Descriptor, including specific information like, delay budget, required MEC Services to run and required virtualization resources.
- MEC Infrastructure
  - Keeps track of available, reserved and in use resources at all ME Hosts, for usage by MEC Applications
- MEC Hosts Inventory
  - Lists and describes the ME Hosts under MEO control. This shall include, per ME Host, a mapping with served eNB and available services (LOC, RNIS, DNS, etc.)

MEC Apps instantiation may occur:

1. Triggered by MEO
   - Selection of ME Hosts to deploy MEC Apps shall be automatic and based on existing information at the MEO and associated MEC Applications' Descriptors. Thus, from no need for allocate resources (only on-boarding), to allocate resources at all ME Hosts, all scenarios are possible.
2. Triggered by management (OSS)
   - Selection of ME Hosts for Applications instantiation will be the decided by a third-party entity, eventually including the analysis of the information contained in MEC Application Descriptors.
3. Triggered by UE
   - Via the MEC "User App LCM Proxy" component, entities at the UE may request the instantiation of specific MEC Apps. Application requirements (e.g. latency, compute resources, storage resources, location, network capability, security condition etc.) will be analysed in order to select a host fulfilling all the requirements.

In any of the previous situations, upon identification of the ME Hosts for Application deployment, resources at the corresponding virtualized infrastructures, must be reserved, involving the defined ME Platform Manager and the VIM, in a similar way to what happens with VNFs.

Even if current ETSI ISG MEC work does not address MEC Applications scale in/out or up/down, there is no reason to exclude that as part of LCM operations, in the context of resources allocation. In the same sense, ETSI ISG MEC considers that individual MEC Applications will be made of single VDUs. Similarly to VNF, there is no strong reason for not considering that MEC Applications may be the result of the composition of several VMs.

Resources allocation for MEC Applications will be managed by the interactions between MEO, MEPM and VIM. Flows identified in Superfluidity's Deliverable I6.1 shall be followed.

4.1.2.2 MEC – Placement

Placement of MEC Applications, meaning defining at which ME Hosts they will be deployed and run, must be determined by their requirements, especially by the maximum allowed delay to the UEs to be served. Considering that MEC addresses the need to have services provided as close as possible to users, it is expected that MEC Applications will be deployed at the closest ME Host to UEs to serve. However, the existing delay budget and resources availability may decide differently. Additionally, availability of certain ME Services, like RNIS (radio interface status) or LOC (location), will also constrain the MEC Applications placement.

In addition, depending on Applications' provided services and resources availability, Applications may not be immediately moved to all the ME Hosts, as part of the on-boarding procedure. Business and licensing conditions may also determine when and where to deploy and run MEC Applications.

ME Hosts will serve a number of eNBs. This number may vary from one ME Host per eNB, possibly running at the eNB, to several eNB being served by a single ME Host. Considering C-RAN (Cloud-RAN) deployments, it makes all sense to consider the deployment of ME Hosts, sharing the same cloud infrastructure with RAN centralized components.

ME Orchestration (MEO) must have a topological view of the entire MEC System, in order to determine where to deploy ME Applications to serve specific geographical areas. Thus, a mapping between eNB and ME Hosts is required. This information was referred before as stored in the MEC Hosts Inventory repository.

In addition, ME Application descriptors must include parameters to help MEO to decide where to deploy each ME Application. This must be complemented by Operators' rules and established business relationships, and translated into MEO understandable policies.

MEC Applications' placement will be decided based on Applications and Operators' requirements and constraints, and not UEs, with the exception of instantiations requested by entities running at UEs (e.g. Applications counterparts running at UEs).

Thus, in general, the following placement factors may apply, determining how many MEC Application instances are required and where to deploy those instances:

- Delay

  MEC Application Descriptors shall indicate desirable and maximum admissible delay towards UEs to be served.

- Geographical scope

  Some ME Applications may provide localized services, like Augmented Reality for a specific building or street. The geographical scope will, most likely, be provided in such a way (e.g.


reference to a monument or specifying a geographical area) that will need to be mapped to eNBs, and from that to ME Hosts, by the MEO.

- Required resources

  Besides specifying computational and networking resources to run, MEC Applications may require specific hardware to run, like video processing.

- Available resources

  Considering that all on-boarded ME Applications cannot be deployed and run at all ME Hosts, their instantiation may depend on the availability and establishment of priorities to access ME Hosts' resources.

- Required MEC Services

  MEC Applications may need to access local MEC Services to run, for instance, with RNIS or LOC.

- Licensing and agreements with service provider

  Business agreements with ME Application providers may also determine where and how many instances to deploy. The limitations may be in any of the parties (Operator only allowed or allowing to simultaneously run a certain number of Application instances).

- UEs' location

  This is a specific scenario that will apply to Applications' instantiation requests generated by the UEs (via the "User App LCM Proxy", e.g. for Application computation off-loading). Besides any of the abovementioned factors, this specific one will be determinant in the placement of the requested MEC Application instance.

MEO shall handle the identification and evaluation of the parameters that apply. This will happen after MEC Applications on-boarding and whenever the MEC System topology change, for instance, by the addition or removal of MEC Hosts or the reorganization of the mapping between eNB and ME Hosts. It will also happen whenever a UE or the OSS requests the instantiation of some MEC Application. No algorithm or parameters evaluation process is proposed for the moment.

4.2Service Migration

4.2.1 Mobility in MEC scope

In MEC context, service migration need, or relocation, may happen as a result of UEs' mobility, and applies to MEC Applications. While moving, most likely UEs will attach to different eNB. Depending on the network topology, the same or different MEC Hosts may serve those eNB. While moving between eNB served by the same MEC Host, no relocation shall happen, as UE's mobility will not be noticed by the MEC System. For the other situations, the frequency and the type of relocation to be executed depend on the Application type and mode of operation:

- Generic Applications (not tied to any UE in particular)


- Stateless applications: no need for any relocation action
  - Application instance is already working at the destination ME Hosts: Service is provided at the edge
  - Application instance is not working at the destination ME Hosts: Service is not provided at the edge
- Stateful applications: state may be is need
  - Scenario detailed below

- UE specific Applications
  - Service needs to follow the UE, independently from being stateless or stateful, in order to keep proximity, according to the specified requirements

(Generic) MEC Applications will be deployed at certain ME Hosts, as described above. They will have traffic rules associated to them, instructing the ME Host Data Plane on how to identify and handle traffic of interest, to be delivered to each MEC Application. These rules will be defined, in general, based in destination FQDN or IP/Ports. It means that, in general, Applications will be deployed and make their services available in anticipation and independently of the UE which will request them. Thus, MEC Applications' provided services are accessible at the specific ME Hosts on which the MEC System decided to deploy those MEC Applications. Therefore, UE's mobility shall not trigger, in general, MEC Application relocations, except for the ones instantiated under UE's request. However, application state created and associated to the UEs may need to be relocated.

MEC Applications mobility is mentioned in current ETSI MEC ISG documents and its discussion triggered the creation of an Work Item (WI), to be developed till end of MEC Phase 1 (end of 2016). For instance, [ETSI GS MEC 002, Mobile-Edge Computing (MEC); Technical Requirements] states that:

"Other mobile edge applications, notably in the category 'consumer-oriented services', are specifically related to the user activity. Either the whole application is specific to the user, or at least it needs to maintain some application-specific user-related information that needs to be provided to the instance of that application running on another mobile edge host.

As a consequence of UE mobility, the mobile edge system needs to support the following:

- *continuity of the service,*
- *mobility of applications (VM), and*
- *mobility of application-specific user-related information.*

"

The same document identifies three mobility requirements:

1. *"[Mobility-01] The mobile edge system shall be able to maintain connectivity between a UE and an application instance when the UE performs a handover to another cell associated with the same mobile edge host."*

2. *"[Mobility-02] The mobile edge system shall be able to maintain connectivity between a UE and an application instance when the UE performs a handover to another cell not associated with the same mobile edge host."*
3. *"[Mobility-03] The mobile edge platform may use available radio network information to optimize the mobility procedures required to support service continuity."*

As a summary, the MEC System shall guarantee service continuity, by application or "application-specific user-related information" mobility, or maintaining connectivity to the ME Host where the required MEC Application is running. This last option will have as a limitation the maximum delay allowed by the Application.

Whenever relocation is needed, the following types may apply.

4.2.2 MEC relocation types

Considering the stated requirements, the following application relocations types can be envisioned:

- Application state relocation

  Application state associated to served UEs is transferred between different MEC Applications instances, located at the old and new MEC Hosts, involved in the UE's mobility.

  Even if it is possible to have a MEC System assistance (aspect not yet defined at ETSI ISG MEC), MEC Applications will most likely manage themselves the state transfer via communication between the involved instances.

- Application instance live relocation

  Applications running at a ME Host, as VDU, and providing services to specific UEs, are live migrated from old to the new ME Hosts (between different NFVI), serving the eNBs where the UEs are now connected. Mechanisms for a complete VDU relocation, shall resort to virtualized infrastructure capabilities.

  MEC Management and Orchestration needs to participate in the process, coordinating it.

- Application instance emulated relocation

  Application instances relocation can be emulated by the creation of a new instance and deletion of old instance, at originating and target ME Hosts

  If there is no need for Application migration or state transfer but the provided services are needed at the new MEC Host, relocation can be emulated by starting a new Application instance at the new ME Host, while stopping the previous instance at the originating ME Host. For the UE, this operation shall be transparent and the service works in a continuous way.

  MEC Management and Orchestration needs to deal with this process.

- No relocation, keeping service anchor


In order to keep service continuity and due to required Application state maintenance specificities, the solution may consist in keeping the provided service anchored at the original MEC Application instance, running at the first ME Host the service started being provided.
This implies associated traffic rerouting between ME Hosts.

- No relocation at all

  At target ME Host, either the MEC Application is not running (e.g. because of the defined Application geographical scope) or an already existing instance is able to provide the same service, without need for any communication with previous instance (e.g. stateless MEC Application). There is no state or application relocation.

4.2.3 UE's mobility detection

In all relocation scenarios, and without considering any interaction with EPC control plane (e.g. S1-MME), UE's mobility is only detected and the new serving MEC Hosts is only known once the UE attaches to the new eNB and its traffic is identified. Even if the UE is aware of eNB change, it is not aware of the possible MEC Host change. This way, any needed relocations can only be triggered after UE's arrival at the new MEC Host.

UE's mobility detection can be done at two levels:

1. By the ME Hosts, by observing traffic send to/from a new IP/TEID (Data Plane level)

   As a result, the detecting ME Host may proactively query neighbouring ME Hosts about source IP (which ME Host was previously handling that IP).

   The previous ME Host handling the UE, may query/notify all running Applications or only the ones that stated the existence of state associated to that IP, about relocation needs.

2. By the MEC Applications themselves, when applicable traffic rules deliver traffic to them

   As a result, and if required (stateful Applications), will require the hosting ME Host to provide its mobility services.

In addition, UE's mobility may be detected by the UE itself. If eNB change is notified to local Application counterpart, this one may notify the MEC System, via the "LCM Proxy", about the possible need for relocation actions.

4.2.4 Relocation need detection

In parallel to that, the need for relocation (instance or state) must be identified. This can be known, and also considering previous scenarios:

1. Previously by the ME Host as part of the MEC Application description and communicated to the ME Host at instantiation time
2. At Application execution time (state is created and exists, associated to certain UEs) and communicated to the ME Hosts via an appropriate API:


a. Inform the MEC System, what IP addresses need to be monitored regarding mobility aspects (proactive)
b. Whenever a new IP is detected by an Application instance, it may ask the hosting ME Host to trigger the required relocation actions (reactive)

3. Unknown by the ME Host

4.2.5 Proposed processes

Both aspects, UE's mobility detection and need for relocation, must be considered together. One approach consists in delegating to MEC Applications the identification of UE's mobility and the identification of relocation needs. Based on that, the following proposals are made.

A. Proposed process for Generic MEC Applications:

1. UE handovers to a new eNB, served by a new MEC Host
2. Configured traffic rules will extract and deliver UE's traffic to applicable MEC Applications
3. Upon detecting traffic from a new IP, stateful MEC Applications will query the hosting ME Host if Application's state exists in another ME Host, expressing relocation actions
4. ME Host queries neighbouring ME Hosts about state related to that IP and MEC Application (IP, AppID)
5. If existing, previous ME Host, notifies, based on AppID, local Application instance about relocation needs
6. As a consequence, that MEC Application instance requests the ME Host to:
    a. Establish a tunnel to the new MEC Host for traffic redirection or
    b. Provide new MEC Application IP address, for managing state relocation
7. Depending on the previous:
    a. Traffic rules at the new ME Hosts are changed accordingly
    b. Application instances exchange state and service for that UE continues at the new Application instance

A similar behaviour may be achieved but with Applications communicating with the corresponding Manager, running at the respective ME Platform Manager. The Application Manager may work as a central point for all instances, coordinating with the MEC System entities, relocation needs and actions. No details for this option are provided.

B. Proposed process for UE's specific MEC Applications:

1. UE handovers to a new eNB, served by a new MEC Host
2. ME Host Data Plane detects a new IP/TEID
3. ME Host queries neighbouring ME Hosts about specific MEC Applications running for that UE related to that IP
4. If existing, previous ME Host, notifies, the local Application instance about UE's mobility


5. As a consequence that MEC Application instance requests the ME Host to:
    a. Establish a tunnel to the new MEC Host for traffic redirection or
    b. Proceed with Application instance relocation to the new ME Host

4.2.6 MEC: Mobility API

Besides the proposed relocation mechanisms, other solutions are possible. The existence of an API for Applications to communicate with ME Hosts is needed and additional options may be provided, giving place to the definition of other solutions. It is therefore proposed the following API features:

1. Apps to notify ME Host about UEs (IP addresses) to be monitored
2. Apps to request ME Host about UE's originating ME Hosts
3. Apps to request ME Host about originating App IP address
4. Apps to request ME Host about another hosting ME Host IP address
5. Apps to request ME Host to store information at local persistent storage
6. Apps to request ME Host to move/copy stored information to another ME Host
7. Apps to request ME Hosts LCM operations (Stop, Create)
8. ME Hosts to notify Apps about UE's mobility (IP address)
9. ME Host to notify an App about the arrival of stored information (AppID)
10. ME Host to query other ME Hosts about UE handling

4.3Service Scaling

A key advantage of the Superfluidity technology is the support for a faster scaling of VNF. That is, the amount of virtual resources allocated to a certain VNF dynamically increased (via scale out operation) or decreased (scale in operation) according to the demand from the network service (e.g., number of flows a firewall handles). In a scale out/in operations, we increase/decrease the number of VMs that are allocated to the VNF, respectively. We note that, in this study, we do not consider the scale up and down operations, which are less common in NFV use cases. (In scale up/down operation we can, for example, add/remove CPU cores to a given VM).

The decision whether to scale out or in, typically performed by the VNFM, remains an important problem that needs to be addressed, especially in the dynamic scenarios that are facilitated by the Superfluidity technologies. Here, increasing the number of VMs allocated to a VNF would improve the performance of the VNF, yet would consume more resources. Accordingly, our scaling decision should minimize the consumed resources, while in keeping with the application required SLA.

In conjunction with the scaling decisions, we are also facing a load balancing challenge, steering the traffic flows to the different VMs and balancing the load between them. In this study, we tackle both

the scaling decision and the load balancing strategy as a single problem, formulated as a Markov Decision Process.

4.3.1 System Description

We assume our cloud infrastructure (the NFVI) can host a finite yet flexible number of VMs. For simplicity, we assume that an instance of a VNF is deployed in one VM, two instances deployed in two VMs and so forth. That is, in a scale out operation, adding a VM translates to adding another instance of the VNF. We further assume, that the load balancer is deployed in a single VM and can handle all traffic demands irrespectively of the number of VNF instances (i.e., number of VMs).

Each VM is modeled as a single queue that handles the traffic demands (e.g., flows handled by the firewall) in a FIFO manner. Each queue (VM) can be deployed and destroyed dynamically based on the demand. The total demand of a VNF (e.g., from a firewall) is modeled by Poisson process of arriving tasks. The incoming tasks are scheduled into one of the active queues, according to some load balancing policy π. The tasks are processed in FIFO order, with equal rate at all queues (VMs). For simplicity, we assume the processing times are exponential, yet, general SMDP model formulation allows any known service time distribution. In the case where no VM is currently deployed, or the existing VMs (queues) are full, the incoming task is rejected. Hence, in the case where deployment time is instantaneous, the rejection can be avoided by immediate queue deployment.

Each processed task gives fixed reward, while rejected tasks incur fines. There is fixed cost for VM deployment and destruction and fixed maintenance cost per time unit for each active VM. A system description is given in Figure 4.

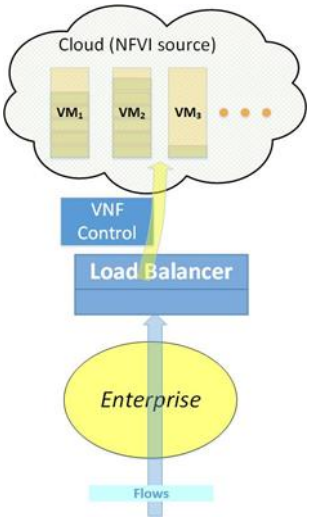

Figure 4: System scheme

Next, MDP problem formulation follows.

4.3.2 MDP formulation

Denote VNF instance arrivals as Poisson arrival processes with average interarrival times given by λ. For simplicity, we assume all queues are identical. The system costs are given by two sets as follows. The reward for each admitted task and the fine for rejected task are denoted by couple $\{r, z\}$. The *queue operation cost set* includes deployment, destruction and maintenance cost per unit of time and is given by the triplet $O = \{b, d, h\}$.

Denote the number of active queues at time t by $q(t)$ and the number of tasks in queues by $n_i(t)$, where i is general indexing of the queues. Each queue serves tasks with equal rate with average μ. Theoretically, the number of queues may infinitely grow for some sets of parameters. For example, in the case where comparatively $\lambda \gg \mu$ and O contains comparatively (to reward) low values. On the other hand, the amount of available queues is expected to be bounded by general system limitations. We will assume henceforth that $q(t) \leq \boldsymbol{q}$, for some given $\boldsymbol{q}$. We will assume that $-1 \leq n(i) \leq B$, $i \in \{1, \dots, \boldsymbol{q}\}$, where B stands for the maximal number of tasks in each queue, and state "-1" means the queue is inactive. Note that this is different from state 0 which means the queue is active but empty. The vector $\{n_i\}$ will form the *state-space* of the MDP.

At each task arrival, the task will be scheduled into one of the queues, according to some policy π. This *scheduling policy* may be predefined, or be part of the optimization objective. Denote the scheduling by vector $\boldsymbol{v}(t) = v(t) \in \{i, 0\}$, where i stands for scheduling to the queue i, while 0 means rejection. The *queue operating policy* is activated on each task arrival and departure event, and is described by vector $\boldsymbol{u}(t) = \{u_i(t)\}, u_i(t) \in \{1, 0, -1\}$, where the three possible values for u_i stand for "deploy","no change","destroy", respectively.

In what follows we will deal with counting processes of the form:

$$A(t) = sup\left\{m; \sum_{i=0}^{m} a(i) \leq t\right\},$$

where $a(i)$ is the time between increment (e.g., arrival time, service time) $i-1$ and i. Denote the arrivals counting process as $V(t)$ and task completion processes as $D_i(t)$. Write the cost functional discounted with discount factor γ:

$$J = \int_0^{\infty} e^{(-\gamma t)} \left[\sum_i^{\boldsymbol{q}} \left(b * \boldsymbol{I}_{(u_i(t)=1)} + d * \boldsymbol{I}_{(u_i(t)=-1)}\right)\left(dV(t) + \sum_i^{\boldsymbol{q}} dD_i(t)\right) + \left(\sum_i^{\boldsymbol{q}} h * \boldsymbol{I}_{(n_i(t)\geq -1)}(n_i + 1)\right) dt + \left(\mathrm{z} * \boldsymbol{I}_{(v_i(t)>0)} - \mathrm{r} * \boldsymbol{I}_{(v_i(t)=0)}\right)\mathrm{dV(t)}\right]$$


where the first part stands for the queue activation cost, the second part stands for the queue holding cost, while the third part stands for the scheduling cost.

More precisely, the build cost b is paid for each queue deployment (denoted by indicator function $\boldsymbol{I}_{(u_i(t)=1)}$. The destroy cost d is paid for each queue destruction, (denoted by indicator function $\boldsymbol{I}_{(u_i(t)=-1)}$. Note that at all times when the queue is neither deployed nor destroyed (i.e., is merely active) we have $(u_i(t) = 0)$, and no special cost is paid at time t. We also assume that both destroy and build operations only take place during the arrival or departure events. In the most general form, both actions are allowed to be taken at arrival events and any of the departure events, that is, once the counting processes V and D_i increase.

The continuous holding cost is accumulated for all non-empty queues, according to the number of residing tasks. Note that even empty queues require holding cost (hence the addition of 1 to $n(i)$). Alternatively, constant cost of active queue maintenance, regardless of its size, equal to given constant c might be added. Finally, the scheduling cost or fine is applied at each arrival.

The next step is writing the Bellman equation. Denote the arrival operator which schedules the arriving task to the queue i by $\mathbb{A}_i$. Operator $\mathbb{H}$ acts on the vector of state-space and calculates the holding cost by $\sum_{i,n_i\neq-1}(n_i h + c)$. Hence, only active queues are taken into account. Finally we define operator $\mathbb{M}$ which is responsible for building and destroying queues. Denote the normalizing factor by δ (its exact value will be calculated in the sequel). The most general Bellman equation solves for the value function V, at each possible state n and is as follows.

$$V(n) = \delta\lambda\left[\left(\max_i(\{\mathbb{A}_i V + r\}, \{V + z\})\right) + \mathbb{M}V\right] + \delta\sum\mu_i[L_i V + \mathbb{M}V] + \delta\mathbb{H}n \qquad (1)$$

$\mathbb{M}$ acts as follows:

$$\begin{cases} V = \max\{B_i V, V\} & n_i = -1 \\ V = \max\{D_i V, V\} & n_i = 0 \\ V = V & n_i > 0 \end{cases} \qquad (2)$$

that is, checks for the empty queues and decides if to destroy them, and checks for the non-active queues and decides if to build (i.e., deploy) them.

4.3.3 Reinforcement Learning (RL)-based solution

Note that the state space is exponential in number of queues. Hence, instead of direct value iteration, which is widely used to solve moderately-sized MDPs, we resort to reinforcement learning (RL). The value function is found in exploit-and-explore method. Each visit to some state n, Equation

(111211111) is solved to update the corresponding $V(n)$. Next, the next state is randomly chosen according to the following probabilities:

$$\begin{cases} \lambda * \delta & \text{arrival} \\ \mu_i * \delta & \text{departure at queue i} \\ \gamma * \delta & \text{no event, hold paid} \end{cases} \quad (3)$$

The learning procedure is repeated using ϵ-greedy choice. The outcome of the long RL run is the value function and nearly optimal scheduling and queue operating policy.

5 C-RAN

TBD

6 Conclusion

TBD

7 References

[1] "NeMo: An Application's Interface to Intent Based Networks" http://nemo-project.net/

[2] "High-level VNF Descriptors using NEMO" https://datatracker.ietf.org/doc/draft-aranda-nfvrg-recursive-vnf/

[3] "Intel DPDK" http://dpdk.org/

[4] M. Caggiani-Luizelli, D. Raz, Y. Sa'ar, and J. Yallouz. The Actual Cost of Software Switching for NFV Chaining. Submitted.

[5] "Open vswitch" http://www.openvswitch.org/

[6] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In Proceedings of HotNets, Oct. 2009.

[7] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The design and implementation of open vswitch. In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pages 117–130, Oakland, CA, May 2015. USENIX Association.