Toward Superfluid Deployment of Virtual Functions: Exploiting Mobile Edge Computing for Video Streaming

S. Salsano[(1)], L. Chiaraviglio[(1)], N. Blefari-Melazzi[(1)], C. Parada[(2)], F. Fontes[(2)], R. Mekuria[(3)], D. Griffioen[(3)]

(1) CNIT / Univ. of Rome Tor Vergata, Italy - (2) Altice Labs, Portugal - (3) Unified Streaming, Netherlands

***Abstract*— The Network Function Virtualization (NFV) technologies are fundamental enablers to meet the objectives of 5G networks. In this work, we first introduce the architecture for dynamic deployment and composition of virtual functions proposed by the Superfluidity project. Then we consider a case study based on a typical 5G scenario. In particular, we detail the design and implementation of a Video Streaming service exploiting Mobile Edge Computing (MEC) functionalities. The analysis of the case study provide an assessment on what can be achieved with current technologies and gives a first confirmation of the validity of the proposed approach. Finally, we identify future directions of work towards the realization of a superfluid softwarized network.**

Keywords: NFV; Mobile Edge Computing; 5G network architecture; Video Streaming; 5G services.

I. INTRODUCTION

5G networks are characterized by very challenging objectives in terms of system capacity and perceived Quality of Experience (QoE) for users. Specifically, 5G will deliver data rates in the order of tens or hundreds of Mbps to a large number of users with high spatial density, while connecting a huge number of devices from different scenarios, e.g., Internet of Things (IoT) and Machine-to-Machine (M2M) communication. Not surprisingly, current estimates foresee a 1000x increase in the system capacity and the performance parameters w.r.t. currently deployed networks. In order to cope with the increase of capacity and performance, current researches are focusing on the Network Softwarization (NS) paradigm. More in depth, NS allows decoupling the networking and computing functionalities from the hardware equipment physically realizing it. This decoupling is also referred to as *virtualization*. In such a context, it is possible to replace the traditional monolithic hardware equipment with Virtual Functions (VFs). The VFs run on a distributed environment, which is made up of general-purpose computing nodes, much like virtualized servers that can host services and applications in a cloud computing infrastructure.

The trend towards softwarization is already revolutionizing the market of communication equipment and services. In particular, Network Function Virtualization (NFV) [3] and Software Defined Networking (SDN) [4] are two complementary technologies that are part of this trend. There are several advantages for a telecom operator in adopting softwarization. The use of common off-the-shelf (COTS) hardware and the economy of scale related to the purchase and the operation of a large number of one-fits-all computing devices may allow a substantial decrease in the Capital Expenditures (CAPEX) and Operating Expenditures (OPEX) to install and manage 5G networks. Moreover, the softwarization approach tends to increase the openness of the infrastructure, easing the adoption of open source solutions and in general the reduction of the "vendor lock-in". In addition, having a geographically distributed infrastructure which supports the execution of softwarized functions gives the possibility to execute virtual functions close to where it is better for performance reasons and in general to optimize the allocation of computing resources to services, applications, users. Overall, these advantages trigger a greater efficiency and/or reduction of CAPEX and OPEX costs and are key elements to achieve the required increase of capacity and performance for 5G networks.

Although network softwarization is a very promising solution to the deployment of 5G networks, its practical realization in operator networks is still an open issue. In this context, several questions are arising, like: How to design a softwarized 5G network architecture? How to manage the 5G network functionalities in an efficient way? What is the maturity of the current off-the-shelf technology? In this paper, we describe the architecture and the vision proposed by the Superfluidity project [1] [8] and consider the design and implementation of a Video Streaming service as a use case. Superfluidity aims at fully exploiting the features and opportunities of network softwarization, by achieving a dynamic *superfluid* (i.e., flexible, transparent and fast) management of the infrastructure. In this vision, all functions of 5G networks can be programmatically composed using reusable building blocks, called *RFB – Reusable Functional Blocks*. The main architectural concepts of Superfluidity are described in section II. The Mobile Edge Computing (MEC) [6] [7] concept, described in section III, relies on the enhancement of the infrastructure at the network edge, so that it is possible to deploy and execute applications taking advantages of the user proximity and of a direct interaction with capability and services offered by the edge networking equipment. The Superfluidity architecture leverages the MEC concepts. As a case study for the proposed architecture, we report the design and implementation of a Video Streaming service, which can be considered among the main 5G use cases. We have implemented this case study on a prototype of the Superfluidity architecture, exploiting the MEC functionality and the RFB concepts. In particular, section IV describes how the Video Streaming service can be decomposed in RFBs and executed in a virtualized environment. Section V provides the details of the prototype architecture and some results from a running demo. The presented work relies on a previous work [2] on a Video Streaming solution based on late transmuxing in the edge node. The contribution of this work consists in the integration with the Superfluidity architecture and the development of the integrated prototype of the Video Streaming components and of the MEC Traffic Offloading components. The results confirm the feasibility of the proposed approach. In addition, we provide an

assessment on what can be achieved with current technologies in terms of deployment times. Finally, we identify the directions of further work.

II. SUPERFLUIDITY VISION AND ARCHITECTURE

In the Superfluidity vision, the high-level functions of a service provider network are decomposed into components, referred to as *Reusable Functional Blocks* (RFBs). Specifically, a RFB is a generalization of the concept of Virtual Network Function (VNF) [10], which is under standardization within the ETSI NFV group [9]. In the current vision, VNFs represent relatively big components, offering a complex set of functionality and are mapped to Virtual Machines (VMs) or Containers. Although NFV is a promising paradigm, this approach lacks granularity and flexibility. To solve this issue, in our vision, the RFBs can also represent "smaller" components and do not need necessarily to be mapped into a VM or a Container. Moreover, we apply the decomposition of functions into RFBs spanning from radio and access related functions, to packet and network processing functions, up to application level functions.

Hereafter we will highlight the main concepts and elements of the SUPERFLUDITY architecture, which are summarized in Figure 1. We refer the reader to [1] for a more detailed description. The different heterogeneous environments in which the decomposition can be applied are referred to as REEs - *RFB Execution Environments.* A *network-wide* REE corresponds to the current NFV Infrastructure concept, supporting the decomposition of Network Services into Virtual Network Functions and VNFs into VNF Components (VNFCs). A *node-level* REE can be composed of packet processing blocks, like for example in a modular software router, or signal processing blocks, like in a software radio implementation. At the different levels, the RFBs can be implemented with different approaches, from VMs to processes or to software modules that are linked together. In this way, a very granular and flexible decomposition can be performed.

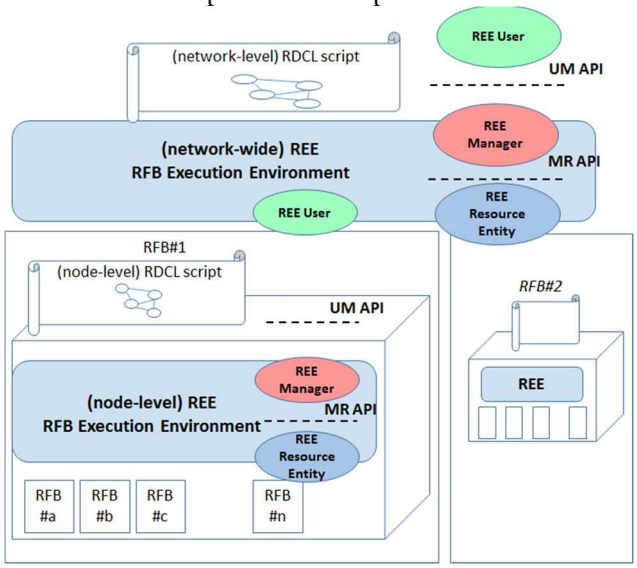

Figure 1 - SUPERFLUIDITY architecture

A *RFB Description and Composition Language* (RDCL), is required to describe the RFBs properties and their interconnections to realize services or other RFBs. However, it is not possible to use a single language to describe the different types of RFBs in all the heterogeneous environments. Therefore, the proposed architecture supports a set of RDCLs, each tailored to a specific environment. For example, the ETSI NSD - Network Service Descriptor and VNFD - VNF Descriptor (see [10]) can be used to describe the VNFs and their combination in Network Services. In addition, the Click language can be used to describe configurations of the Click modular router [12]. In this scenario, Superfluidity provides a holistic vision, by extending and orchestrating the RFBs and the RDCLs of the different environments.

In our vision, the MEC applications can benefit from being realized using the RFB concepts. Moreover, the edge computing infrastructure becomes an extension of the network-wide RFB execution environments.

III. MOBILE EDGE COMPUTING (MEC) ARCHITECTURE

Among the target Key Performance Indicators (KPIs) proposed for 5G networks, one of the most challenging is the '**latency < 1ms**' – actually defying the laws of physics. To achieve an under-millisecond end-to-end latency, the traditional network topology has to be modified, by moving the applications to the network edge, thus closer to end user devices [7]. This new paradigm shift is usually known as edge computing and is becoming very popular. Not surprisingly, the ETSI board created the MEC [6] Industry Specification Group (ISG) in December 2014. The group goal is to standardize a framework for executing applications at the edge, by defining management and orchestration tools for a multi-edge and multi-party environment.

The applications running on MEC (referred as ME Apps in the ETSI MEC terminology) take advantage of a cloud environment (NFV-like), where they can be managed by on-demand, flexible and agile paradigms. ME Apps benefit of local services, which are available at the edge, to leverage the building of complex applications based on information like location, or network information, among others. ME Apps can be managed in their lifetime (i.e., deployment, scaling, disposal, etc.) and centrally orchestrated in order to define where the different instances are running at a certain moment in time.

A MEC environment is characterized by the following properties: **End User Proximity**, delivering services at the end user vicinity; **Ultra-low Latency**, benefiting from end user physical proximity; **Efficient Bandwidth Utilization**, reducing the traffic sent/received to/from the core; **Real-time Access to Local Services**, accessing to a platform of locally provided services.

Figure 2 shows the architecture proposed by the ETSI MEC ISG [11]. Specifically, the following macro blocks are identified: Data Plane Forwarding (DPF), Applications and Services (AS), Management and Orchestration (MO). We now describe each macro block in more detail.

The DPF macro block is located on top of the mobile network data plane, usually between the e-NodeB (eNB) equipment and the core network. This block is responsible for inspecting, identifying and offloading (i.e., redirecting) the user traffic to be exchanged with local ME Apps running on the edge. This action is expected to be transparent both to the end user and to the network. The DPF comprises a

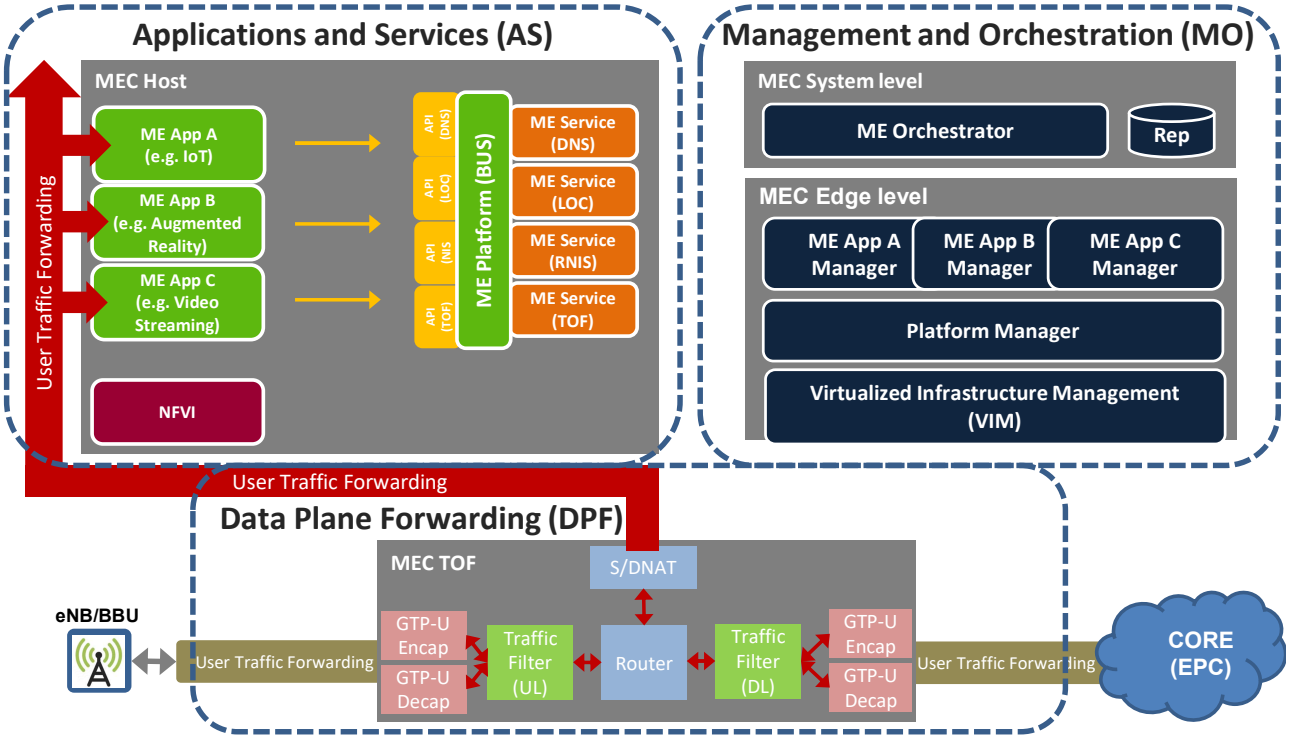

Figure 2 - MEC Software Architecture

component called Traffic Offloading Function (TOF), which can be further decomposed as shown in Figure 2. The motivation for this Traffic Offloading approach is that most services will still be provided in the core and that a smooth transition from current architecture is needed. The TOF can offload traffic to the edge only for services that can benefit of it and only if the edge infrastructure has been deployed by the operator.

In addition, the AS macro block is a NFV infrastructure, which supports the execution of ME Apps. It includes an ME platform that exposes APIs to ME Apps to get context and user information like network conditions, location, etc. Finally, the MO macro block manages ME Apps and the ME Platform, and orchestrates the whole environment.

The MEC technology can take advantage of existing NFV deployments, reusing the IT infrastructure at the edge (e.g. the one devoted for Cloud Radio Access Networks – C-RAN). It is remarkable that the edge computing concept can also be applied to other access networks (DSL, GPON, Wi-Fi, etc.), by sharing the same edge data centres that can be located in the traditional Central Offices (COs). In fact, in phase 2 of ETSI MEC standardization activity, MEC will mean Multi-access Edge Computing, expanding the scope to other networks than mobile ones.

IV. Virtualization and Decomposition of Video Streaming Functions

A key application in mobile networks is Video Streaming. From a technological perspective, Video Streaming is converging to an approach based on HTTP and adaptive bit-rate streaming. Specifically, the video content is divided in small segments (with typical duration of 2-8 seconds) of independently decodable video content that are requested by clients with the HTTP GET method. This approach has the advantage of making Video Streaming transparent to the network, so that the same infrastructure can be shared with any other service transferred over HTTP. The basic operations of Video Streaming in a typical deployment are shown in the top half of Figure 3.

More in detail, the original video content is compressed by an encoder, which can be a live encoder for live video. Alternatively, compressed video can be retrieved from a dedicated storage. In order to support adaptive bit-rate streaming the encoder provides content encoded in different bit-rates or the file is encoded with different bit-rates and then stored. The encoded video is sent to a Just-In-Time (JIT) packager. This entity is in charge of making the video available in small segments that can be independently requested by the clients. More in depth, the JIT packager generates segments for different protocols such as Apple HLS, MPEG DASH, Microsoft HTTP Smooth Streaming HSS, Adobe HTTP Dynamic Streaming HDS. These protocols typically work by making the references (URL's) to these small video segments available in a textual manifest file. A client can then parse and interpret this manifest and download each of the segments from different locations to realize playback. The segments in different protocols use different container formats (such as MPEG-4 ISO Base Media File in MPEG-DASH or MPEG-2 TS in Apple HLS) to encapsulate the encoded media samples. In this context, the operation of generating different segments and container formats for the same video is called trans-multiplexing (*transmuxing*). The Content Delivery Network (CDN) then performs caching of the different multiplexed segments, preferably as close as possible to the end user to reduce delay. Each end user device requests video using one of the protocols and decides which bit-rate segments to request based on locally available information such as reception statistics as well as playback status.

In addition to these basic functions, let us consider a set of advanced functions that can extend the Video Streaming

services. More in detail, content encryption is typically exploited by content owners and distributors. In addition, content customization is of fundamental importance to allow monetization of the streaming service using advertisement content. Specifically, with this approach the videos generated from different sources are stitched together, resulting in new segments or streams. Moreover, transcoding is sometimes needed to generate additional bit-rates that were not produced by the origin encoder. Stream recording is used for storing streams in the network for later playback. Finally, up-sampling of media content is used for example when users have a high resolution Ultra HD device, but the high resolution content is not available. In this case, pixels can be interpolated to create a high resolution version of the video for playback. On the other hand, down-sampling can convert high resolution content to lower resolution content.

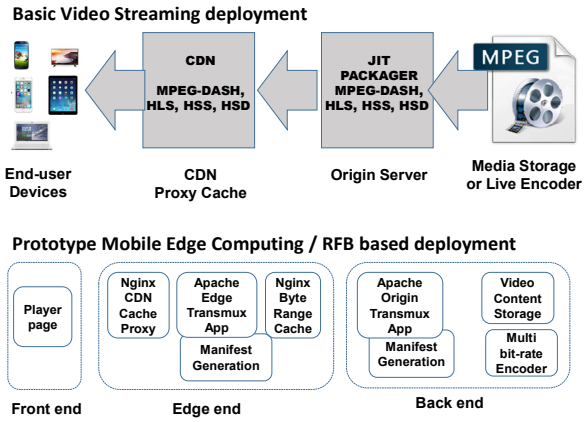

Figure 3 - Video Streaming deployment and decomposition in RFBs

A. *Allocation of functions: Core vs. Edge.*

In current networks, most of Video Streaming functions are performed by origin servers in the core of the network, while only caching is typically performed at the edge. Moreover, the HTTP caching infrastructure is re-used transparently to all these video specific network functions. In our vision instead, the idea of Superfluid and location independent orchestration using RFBs could enable the deployment of functions on the most suitable hardware and on the most suitable location. Specifically, a subset of the Video Streaming network functions benefit from hardware acceleration while others benefit from being close to the end user (location). Let us consider the packaging/transmuxing function, which generates the HTTP segments for the different protocols implemented in the end user devices. If this function is implemented in the origin node, and assuming that four different protocols need to be supported for a given video stream, the packager will send up to four versions of a given video content to the CDN cache function running in an edge node. On the other hand, if the packaging function is implemented in the edge, and the edge node caches the encoded video before the packaging, the amount of traffic from the core to the edge can be divided by four, thus achieving relevant savings. We call this solution late transmuxing, as the packaging/transmuxing is performed late in the delivery chain, i.e., closer to the end user. In the following sections, we describe the design aspects and discuss the results of a prototype implementation of the late transmuxing solution. In principle, the same approach of executing a function in the edge rather than in the core can be applied to the other advanced functions and it can result in large saving of network capacity and in reduction of the latency perceived by the end user. The design considerations and the discussed results also hold for the migration of the other functions to the Edge.

V. PROTOTYPE DEMONSTRATION

Our prototype demonstrates a simple case study in which a common infrastructure is able to concurrently support application level RFBs (for the Video Streaming) and network level RFBs (implementing the TOF). Moreover, following the MEC concept, the application level RFBs can be executed in the core or dynamically deployed in the edge. The demo scenario considers a standard compliant 4G network setup, with User Equipment (UE), e-Node B, Baseband Units (BBU) and Evolved Packet Core (EPC). Figure 4 shows an overview of the considered architecture. In this scenario, the UE starts getting the video streaming from a central server, as there is no edge server available. Based on operator/service rules, the set of Video Streaming RFBs are deployed in the edge. Then the Traffic Offloading Function (TOF) module starts offloading the video traffic requests coming from the UE towards the edge. Again, based on operator/service rules, the video front-end instance at the edge is disposed and the TOF enforces the traffic to follow the standard path, making the UE to be served from the core Video Streaming RFB instances.

The testbed used for this prototype is hosted at Altice Labs in Aveiro (Portugal) and it is composed of one powerful server, with 16 cores Xeon E5 v2 CPUs running at 2.00GHz and 96GB RAM. The host operating system runs an Ubuntu 16.04.1 LTS Linux. In addition, the cloud management is the OpenStack (Kilo) version and KVM is used as hypervisor for the virtualization of computing nodes. Hereafter we first describe some design aspects related to the softwarization of Video Streaming and TOF components, then we report the obtained results.

A. *RFBs for Late Transmuxing and Implementation*

We decompose the functional blocks shown in the top half of Figure 3 into a set of RFBs, as shown in the bottom half of Figure 3 and in Figure 4, mapped into the prototype architecture. More in detail, the multi-bit rate encoder performs the raw encoding of the bit-stream and is used for live video. The video content storage provides storage of the media asset and is used for video on demand services. The manifest generation provides the appropriate manifest for the protocol in use. The Apache transmux app packages the video content in the proper format. The last two RFBs can be executed in the back end or in the edge end (i.e. with *late transmuxing*). When executed in the back end, the video streams to the edge-end are already packaged in a specific protocol format. In the late transmuxing case, the original encoded or stored media information are transmitted between the core and the edge. A "byte range" communication mechanism has been designed, meaning that the HTTP requests indicate specific ranges of the video files to be streamed (details in [2]). The Nginx byte range cache RFB is able to cache the original raw compressed video content before the packaging operation. The Apache Edge Transmux App performs the packaging/transmuxing.

The caching of the packaged video segments in the edge network is performed by the Nginx CDN cache proxy. In addition, we implement a front end player page to test the solution. The platforms used for implementation include a VM running the back end and a VM running the edge end. An alternative implementation was performed where each of the blocks run in a separate Docker container, using the Docker images for Apache and Nginx. Finally, the Unified Origin server was compiled for Alpine Linux, enabling a tiny Docker based transmux container that can run in the Network.

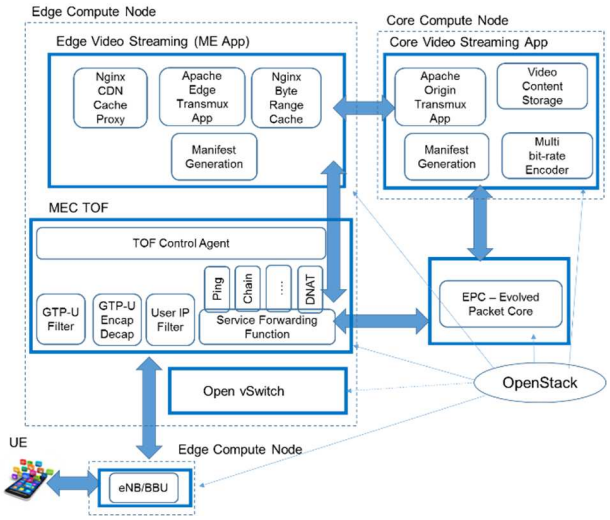

Figure 4 – Prototype architecture

B. *Implementation of the Traffic Offloading as RFB(s)*

The TOF module of the MEC data plane forwarding macroblock is decomposed into a set of RFBs. The decomposition is also shown in Figure 4. More in detail, the TOF control agent RFB is the central entity that controls a chain of data plane RFBs, performing a sequence of actions on the user traffic (orange). The data plane RFBs are the following ones:

- GTP-U Filter filters GTP-U traffic and forwards to the core other traffic (GTP-C or non-GTP).
- GTP-U Encap Decap decapsulates GTP-U packets, extracting the user IP traffic in upstream direction; it encapsulates the user IP traffic into GTP-U packets, forwarding it towards the eNB in downstream direction.
- User IP Filter inspects the user IP traffic and decides whether it is to be forwarded towards the EPC, according to the 3GPP usual standards, or must be offloaded towards the edge datacenter (ME App). The filtering rules are defined by the management layer.
- DNAT is used to translate the user destination IP (global address of the ME App) to the local IP of the Application (edge address of the ME App).
- Ping is used to discover the GTP-U Tunnel Endpoint ID used for traffic coming from the edge (downstream).
- Service Forwarding Function and Chain are intended to support the chaining of ME Apps.

The RFBs are fully independent, self-contained and loosely coupled to each other, making them easily pluggable. The RFBs have been implemented in C language, over the Linux OS. The TOF architecture allows each of this RFB either to run all-in-a-box or to be spread along different nodes (e.g. VMs), depending on the required performance. In the prototype setup show in Figure 4, a single VM hosts all the RFBs and an instance of Open vSwitch (OVS), used to interconnect the data plane RFBs. Each RFB is executed in a different process, reading and writing packets on the virtual interfaces provided by Open vSwitch.

C. *Demo Results*

The analysis of RFB deployment times is reported in Figure 5. In the integrated prototype, using OpenStack and the Ubuntu VM Guest images, the time needed to deploy the guest VM hosting all the edge RFB components is around 1 minute, but the first deployment takes almost 2 minutes. This latency can be decomposed in three parts: i) OpenStack/KVM operations to instantiate the Guest VM, ii) boot up time of the guest VM, iii) startup and initialization of the Video Streaming RFB components. A large part of this time (more than 90%) depends on the boot time of the Ubuntu VMs (and the OpenStack/KVM latency depends on the image size of the guest VM). For this reason, we have implemented the edge Video Streaming RFBs in a Docker container and compiled with Alpine Linux. In Figure 5 the startup time of the edge RFBs in such environment is also reported.

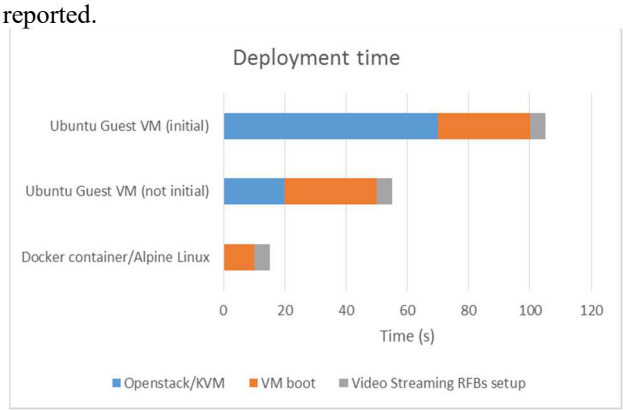

Figure 5 - Deployment time of Edge Video Streaming components

In the following, we focus on two performance metrics when the Video Streaming service is provided in the edge. Specifically, Figure 6 and Figure 7 respectively report the downloading delay for a video chunk and the throughput. The experiments have been performed using wrt, a HTTP benchmarking tool [13]. Wrt can generate the load of multiple connections in parallel (in the specific experiments reported in Figure 6 and Figure 7, 10 parallel connections have been used). The results are averaged by wrt over all chunks transmitted by all connections during an experiment. Wrt also provides the standard deviation of the evaluated metric. The corresponding confidence intervals are very close to the average values, so we have plotted them in Figure 6 and Figure 7. In the experiments, we analyze three different setups, namely the CDN setup, the single LT (Late Transmuxing) setup and the double LT setup. More in depth, in the CDN setup all segment formats are generated at the core and cached in the edge. In addition, the single LT setup only stores byte ranges at the edge and generates the segments on the edge on the fly. Finally, the double LT setup is a combination of the previous ones: byte ranges and segments are stored in the edge and segments are generated at the edge. We then consider the following conditions: i) no video was streamed before (denoted as cold cache), ii)

the same video segment was streamed before (denoted as same form), iii) the same content in a different format was streamed before (denoted as diff form). We can clearly observe from Figure 6 and Figure 7 that the LT approach results in lower latency when video is streamed in a different format before and also gives a better throughput in this case compared to the traditional CDN setup. We refer to [2] for more details and additional results.

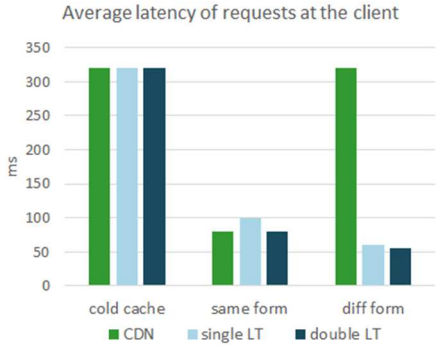

Figure 6 - Average latency of requests at the client

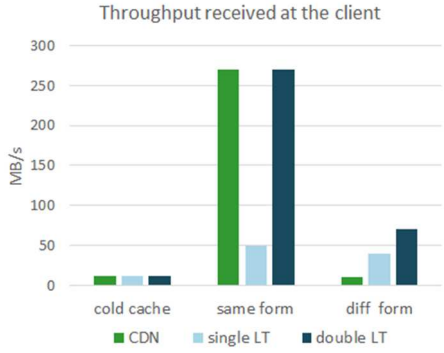

Figure 7 – Throughput received at the client

The implemented prototype allows evaluating the impact of the transitions from core to edge and vice versa on a running Video Streaming application. Note that the Video Streaming applications open a new connection for each video chunk (with intervals in the order of 2-10 [s]), rather than using a single stream. When a transition is performed, the active downloads of video chunks can be blocked. In normal conditions, this does not affect the user, as a relatively long buffer (e.g. 20-60 seconds) is pre-fetched in the UE video player. The download is resumed from the new target (edge or core) before the buffer is depleted, making the transitions seamless for the users.

The TOF prototype has been conceived for functional evaluation rather than as production-ready data plane element. Nevertheless, we have roughly evaluated its performance, in the configuration of Figure 4 (all RFBs implemented as components within a single guest VM). The TOF is able to sustain without any loss flow rates in the order of 3 [Mbps] per each user (i.e., the maximum quality of the video in use) and we tested it up to aggregate rates in the order of 100Mbit/s.

VI. Discussion and Conclusions

We have described the implementation of a Video Streaming service in a softwarized networking environment, representative of a 5G network. The prototype infrastructure offers computing resources both in the core and in the edge, realizing the MEC paradigm. Both the application level functions needed for the Video Streaming and the traffic offloading networking functions needed for MEC have been decomposed in software RFBs, according to the architectural principles designed in the Superfluidity project. The implemented RFBs can be allocated to the core or edge computing resources in a flexible way, offering the possibility to optimize the network efficiency and/or the performances. The results show that, once the required components (RFBs) have been allocated in the edge, the transitions from core to edge and vice versa are seamless for the user. In addition, we have observed that implementing late transmuxing in the edge is beneficial for the perceived user performance and for the utilization of network resources.

On the other hand, the performance in terms of deployment time of RFBs in our prototype are still not optimal for highly dynamic deployment scenarios. Such deployment performance depends on the set of selected technologies (e.g., cloud management system, hypervisor, guest Operating System) utilized for implementing and running the RFBs. In order to improve the performance, we are both tuning the configurations of the selected technologies, and we are considering alternative technologies. Eventually, the availability of a complete prototype of the Video Streaming services and of the MEC infrastructure and RFBs will allow us the comparison of the different solutions based on a realistic reference scenario.

Acknowledgements

This work has been partially funded by the EC under project Superfluidity (H2020 grant 671566).

References

[1] G. Bianchi, et al., “Superfluidity: a flexible functional architecture for 5G networks”, Trans. on Emerging Telecommunication Technologies, Wiley, 2016

[2] R.Mekuria, J. Fennema, D. Griffioen, “Multi-Protocol Video Delivery with Late Trans-Muxing” ACM Multimedia, 15 - 19 October 2016, Amsterdam, The Netherlands

[3] R. Mijumbi, et al. “Network Function Virtualization: State-of-the-Art and Research Challenges”, IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 236-262

[4] D. Kreutz, et al., “Software-Defined Networking: A Comprehensive Survey”, in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015

[5] 5G PPP Architecture Working Group, “View on 5G Architecture”, White Paper, Version 1.0, July 2016

[6] Y. C. Hu, et al. “Mobile Edge Computing—A Key Technology Towards 5G”, ETSI White Paper 11 (2015)

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things”, 2nd Workshop on Mobile Cloud Computing (MCC), ACM, 2012.

[8] SUPERFLUIDITY project home page, http://superfluidity.eu/

[9] ETSI Industry Specification Group (ISG) for NFV Home Page, http://www.etsi.org/technologies-clusters/technologies/nfv

[10] ETSI NFV ISG, “Network Functions Virtualisation (NFV); Architectural Framework”, ETSI GS NFV 002 V1.2.1 (2014-12)

[11] ETSI MEC ISG, “Mobile Edge Computing (MEC); Framework and Reference Architecture”, ETSI GS MEC 003 V1.1.1 (2016 - 03)

[12] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The Click modular router”, ACM Transactions on Computer Systems 18(3), August 2000, pages 263-297

[13] W. Glozer, “wrk - a HTTP benchmarking tool”, https://github.com/wg/wrk