

Service management in virtualization-based architectures for 5G systems with network slicing

Borja Bordel^a, Ramón Alcarria^{b,*}, Tomás Robles^a and Diego Sánchez-de-Rivera^b

^a *Department of Information Systems. Universidad Politécnica de Madrid. borja.bordel@upm.es, tomas.robles@upm.es*

^b *Department of Topographic Engineering and Cartography. Universidad Politécnica de Madrid. ramon.alcarria@upm.es, diego.sanchezderivera@upm.es*

Abstract. Future 5G networks are envisioned to provide communication services in a very heterogeneous catalogue of applications. The concept of network slicing, a technology where network resources are packed and assigned to users in an isolated manner, contributes to overcome the relevant differences among all the applications scenarios. This paper proposes a new approach for the implementation and management of services and slices in future 5G systems with network slicing, based on lightweight algorithms and virtualization technologies, such as Docker or Kubernetes. The proposed solution consists of a microservice-oriented virtualization-based architecture, where Docker containers encapsulate virtual network functions configured as microservices. Kubernetes mechanisms take care of most management functions, so (in the proposed approach) only two functional components must be designed. In this paper, a possible implementation for one of these components (the specific service manager) is proposed, based on statistics and metrics generated by Kubernetes cAdvisor and simple mathematical operations. Results show an improvement in relevant network indicators such as availability and configuration time. Besides, the scalability of the proposed technology is evaluated by means of the calculation time it requires.

Keywords: 5G; network slicing; future technological systems; virtualization techniques, management, Docker, Kubernetes

© 2020. This manuscript version is made available under the CC-BY-NC-ND 4.0 license

<https://creativecommons.org/licenses/by-nc-nd/4.0/>

The Version of Record is available online at: <http://dx.doi.org/10.3233/ICA-190609>

*Corresponding author. E-mail: ramon.alcarria@upm.es

1. Introduction

Future 5G networks are evolving from a scientific and theoretical paradigm to a real and commercial technology [1]. Both, the definition of new and innovative applications based on broadband mobile communications (such as Wireless Sensor Networks [2]); and the incredible number of devices and different wireless technologies operating within the same public frequency spectrum bands, are pushing to develop 5G systems as fast as possible.

In fact, during the last ten years, many different wireless technologies and devices have been deployed: from traditional mobile solutions such as Long-Term Evolution Advanced (LTE-A, or 4G+), to Low-Power Wide Area (LPWA) systems for Internet-of-Things (IoT) networks [3], Cyber-Physical Systems [4] and Industry 4.0 [5] (like LoRa or SigFox). Besides, users and companies are exhaustively using some wireless technologies (such as Bluetooth or ZigBee), even in scenarios where they cannot operate efficiently. Furthermore, the same application may employ different wireless technologies depending on the manufacturer, and sometimes complex infrastructures including several Radio Access Technologies (RATs) must be designed to compensate inefficiencies due to the misuse of these wireless solutions.

This chaotic scenario, where different scientific, commercial and politic interests meet, has reduced the global efficiency of wireless communications. In a context where the device density is continuously growing up, and applications are becoming fully pervasive, this is a critical issue to be addressed.

Although different discussions about the most adequate technology (among all the existing ones) to support future systems have been reported, relevant problems such as the spectrum saturation, interferences, scalability and global connectivity suggest the need of a new and totally innovative communication technologies to replace all the currently existing wireless solutions.

Next generation of mobile communications (i.e. 5G communications) have been envisioned to play that role. 5G technologies must fulfil three basic characteristics to be considered real next generation solutions [54]:

- Massive Machine-Type Communications (mMTC): Thousands of billions of devices should be interconnected through the mobile network, at low cost and without creating congestion in the infrastructure.

- Enhanced Mobile BroadBand (eMBB): Data transmission rates between end-users must be equivalent to wired connections.
- UltraReliable Low Latency Communications (URLLC): Sub-millisecond latency communications must be supported, with error rates that are lower than 1 packet loss in 10^5 packets.

Most of these requirements must be addressed at very low-level. However, end-to-end communications must consider other high-level challenges. In particular, the technological challenge for 5G communications is to reorganize wireless solutions with a common vision, but respecting and tolerating the radical differences among the different applications and services they should support. For example, traditional mobile Internet access services may be sized considering the standard traffic profiled for a community of users accessing the web 2.0. However, this approach is not valid for a ubiquitous IoT service where a pervasive sensor network is periodically transferring information.

This situation, where homogenous network architectures cannot meet required characteristics from 5G networks in every application scenario, has naturally led to the notion of network slicing [6]. Being networks organized as a collection of slices, network resources and infrastructures are divided into different vertical dedicated networks (or slices) each one optimized to support a particular service. Resources are, then, packet and assigned in an isolated manner to a set of users, according to their requirements.

Most modern network architectures already consider virtualization technologies, such as Software Defined Networks (SDN) and Virtual Network Functions (VNF) [7], which enable a more efficient network management and, even, establishing programmable management solutions. These techniques, together with traditional technologies to create private networks over public infrastructures (such as circuit switching or virtual private networks) could be employed to support network slicing. However, various problems are associated with this approach.

First, these techniques require a high integration among different agents: the infrastructure owner, the communication service provider and, eventually, the final service provider. In the current economic situation, where specialization is a critical factor, this integration in the value chain is unlikely.

Second, slices are imagined to have a highly dynamic lifecycle (as can be deduced from the current mobile application or cloud computing markets).

Actually, slices should be able to be created automatically and dynamically, according to user demands. Then, original resources could be required to be modified (sometimes exponentially) after only some days or even hours. And, finally, users could cancel the slice as fast as they desired to create it. This approach, similar to the one that nowadays is employed by cloud computing platforms, is incompatible with heavy configurations and traditional contracts usually employed with circuit switching and other similar solutions.

Third, all the referred solutions are focused on the creation of horizontal networks. No solution to relate the circuit switching with resources in servers (for example) is available. Nevertheless, as said, slices must be vertical, to guarantee an isolated use of resources (including from access networks to service management).

Finally, these traditional solutions are not compatible with the isolation characteristics required by slices in future 5G systems. In fact, all the cited technologies (circuit switching, virtual private network, etc.) create divisions which share common functional resources (for example routers), what is unacceptable in network slicing. For example, in future systems, some slices could be considered unsecure, or present a non-guaranteed availability level. To preserve the conditions of other services, these slices must be fully isolated one from each other.

In this context, we argue that all these problems could be addressed by creating architectures for future 5G systems with an exhaustive use of virtualization technologies; over which lightweight slicing and service implementation and management solutions are deployed.

Therefore, the objective of this paper is to describe a new approach for the implementation and management of future 5G systems with network slicing based on virtualization technologies, such as Docker [35] or Kubernetes (K8s) [36]. New and innovative service management solutions are described. In this proposal, physical infrastructures are managed by independent agents, which rent hardware capabilities and resources to communication service providers. These providers design and deploy (using virtualization technologies) functional components supporting different communication services. Slices are implemented using K8s [8], and functional components are deployed in different containers (e.g. Dockers [9]) which guarantee the availability, isolation and security of the different communication services. Moreover, agile, lightweight, programmable and autonomous management solutions associated to K8s fit perfectly

the characteristics of the intra-slice and inter-slice management functions described in 5G standards.

The rest of the paper is organized as follows. Section 2 presents the relevant technologies for our solution (virtualization, 5G networks and network slicing). Section 3 describes the proposed virtualization-based architecture for future 5G networks. Section 4 presents the service management algorithms. Section 5 describes the experimental evaluation; Section 6 concludes the paper and Section 7 describes some future works.

2. Relevant technologies: network slicing, 5G networks and virtualization

In this section, the most relevant technologies associated to the proposed virtualization-based architecture and management solutions for services and slices in future 5G systems are described in a comprehensive manner. The first subsection is focused on network slicing, second subsection describes the state-of-the-art on virtualization solutions (with a special interest on Docker and K8s), and third subsection presents the most modern ideas about 5G systems.

2.1. Network slicing

This section presents an overview of the related research in the most relevant fields associated to network slicing. As systems implementing network slicing present a vertical design, research areas related to this topic cover all network levels: infrastructure, network functions and services.

In general, existing works about network slicing are introductory, so most common works on this topic are about pending challenges (in the three previously referred areas) and existing working groups in standardization institutions [14]. Initial descriptions and conceptual maps about the idea of network slicing [15][16] may also be found.

Most advanced works on network slicing propose the new idea of Network Slicing as a Service (NSaaS), where a new business based on renting slices supported by a hardware platform is described [17]. The proposed solution in this paper also considers and develops this idea. On the other hand, first algorithms to manage and prioritize actions about a certain set of slices supported by the same infrastructure have been reported [18].

Next subsections analyse the state of the art on the three main topics related to network slicing identified before.

2.1.1. Infrastructure management

In order to maximize the flexible definition of future networks, 5G systems are usually described to be supported by general-purpose hardware infrastructures, commercialized following the Infrastructure-as-a-Service (IaaS) paradigm [10].

In fact, most authors agree IaaS paradigm (sometimes also known as Hardware-as-a-Service) is the preferred solution to implement core networks in future mobile systems [11][12]. Works on this topic address seven basic issues [13]: (i) multi-tenancy, (ii) resource management, (iii) network management, (iv) security, (v) data management, (vi) application programming interfaces and (vii) development tools.

However, only the three first topics have been deeply investigated. With respect to multi-tenancy, typical works propose new approaches to apply virtualization using this concept [19]. Energy saving and an efficient use of power resources are the most common topic in relation to resource management [20]. Network management must deal with the increasing number of instances to be controlled, so different instruments have been reported [21][22].

Although most proposals about infrastructure management are about the core network, some authors also describe solutions to implement Access Networks with a very low latency (as required from 5G solutions) [23]. In particular, various mechanisms to create virtual Radio Access Network (RAN) instances on-the-fly [24][25] have been reported.

As in all IaaS solutions, in 5G networks general-purpose hardware infrastructure is virtualized using different techniques, although most common approach is based on Linux containers and kernel-based virtual machines [26]. With this view, resource pools are defined and assigned to service providers to create their communication networks.

2.1.2. Software Defined Networks

NFV and SDN are two of the most popular and interesting research topics nowadays. Many review articles with specific details about these technologies may be found [27][28].

While NFV is the concept of transferring network functions from dedicated hardware devices to software-based applications; SDN employ NFV to create programmable connections of virtual network functions (VNF).

Apart from generic proposals for network virtualization, and focusing on SDN for future 5G networks, the concept of vEPC (virtualized Evolved Packet Core) is the most developed. The basic idea is to build a virtual core network of interconnected VNF, whose structure, management and architecture is programmable and automatable [29][31]. Moreover, orchestration mechanisms to coordinate virtual instances and support network slicing have been also reported [30]. These solutions, however, are usually only theoretical discussions which are not designed to fit the real 5G architectures, characteristics or requirements.

Solutions to create SDN for RAN are the next technological challenge to be addressed. Although some works about this topic may be found [32] (specifying new hypervisor, for example), techniques to create resource pools which include not only CPU processing resources and storage space but also antenna and frequency spectrum resources have not been reported so far.

2.1.3. Service management

Traditionally, deploying a communication service in mobile networks requires an elaborated and permanent configuration [33]. However, in 5G networks services will be very dynamic components, and will be deployed and cancelled following the users demand, almost at real-time.

The main problem that prevents the development of these solutions is the need of knowing the underlying infrastructure, interfaces and components (to provide valid automatic deployment algorithms). As nowadays this information is still being built, works on service management in future 5G networks are sparse [1].

Existing proposals are mostly focused on techniques to map service requirements into available network functions and components [34], so initial checks before service deployment may be done.

Our proposal is totally coherent with all these previous contributions and research, but expands the knowledge in the area by defining a technique and approach being able to address current problems in the three cited topics through a consistent and integrated technology.

2.2. Relevant virtualization technologies: Docker and K8s

In this section a general description about relevant virtualization technologies to be considered is pro-

vided. Basically, two main and commercial solutions are analysed: Docker [35] and K8s [36].

2.2.1. Containers: Docker

Containers are a virtualization instrument more agile and lightweight than traditional virtual machines. Containers virtualize at operating system level, contrary to hypervisor-based solutions that virtualize at hardware level [37].

Docker is a virtualization technology for Linux systems based on three basic previous solutions: LXC (Linux Containers) [38], Control Groups (cgroups) [39] and copy-on-write filesystems [40]. LXC employs kernel-level namespaces to isolate the containers from the host OS; cgroups establish a resource quota for each container; and copy-on-write filesystems (Docker employs the Advanced Multi-Layered Unification Filesystem or AuFS) transparently overlay the host's filesystems and create a new proprietary copy of each file a container needs to modify.

In respect to the use of Docker for implementing future 5G networks, three particular functionalities (that are only available in Docker technology) must be considered. First, cgroups create a large catalogue of statistics and metrics, which enable managers to monitor the containers' state. Second, the corresponding Docker daemon creates an API to invoke its functionalities locally or remotely (a characteristic very important when managing networks from control centers). And, third, Docker daemon is able to run containers from images stored in a remote repository.

Using these functionalities, applications (images or, in our case, network functions, see Section 3) may be developed in any environment, packed, published in the repository and, later, downloaded and run in the corresponding machine.

Besides, using the API and the monitoring capabilities of Docker, it is possible to create remote control mechanisms to detect malfunctions and reboot those applications that are not working properly. This reboot process fully meets the requirements of 5G networks. Specifically, sub-millisecond delays in communications can be easily supported by Docker. Previous works have reported more than one hundred connections may be managed using virtual networks implemented through Docker with sub-millisecond delays [56].

However, systems as complex as future 5G networks (which may include thousands of containers running in sparse geographical locations) cannot be

easily managed by human managers, so automatic and engineered control and management solutions are required. K8s address this problem.

2.2.2. Managing containers: K8s

Future 5G networks are complex systems, where many different components must operate together to provide next generation communication services. Each one of these components may belong to a different tenant, and be developed following a different path (see Section 2.3). To enable this new manner of building mobile networks, components (i.e. virtual network functions, see Section 3) will be disaggregated and structured as small services, which are consumed among them to globally create the final communication service.

In our proposal, each component is packed in a Docker; which is an instrument very related to computer architecture. To abstract the communication service management from the computer architecture details, and, then, make easier the containers management from remote control centres and service providers, K8s technology is employed.

K8s [36] is a reliable technology to deploy, maintain and scale applications based on the coordination of several containers. An application or network based on K8s has four basic components:

- K8s pods: Set of containers (Docker) that must run in the same machine, or node (it may be physical or virtual), and collaborate to execute a complex application. Dockers within a pod share some resource such as the IP address or the machine name. Pods abstract the service management from details about the underlying computing systems. Pods may be managed manually or automatically using controllers.
- Labels and selectors: K8s technology allows creating key-value pairs named "labels" for any element (pod, nodes, etc.) in the architecture. On the other hand, label selectors are queries to the label catalogue, which returns the objects meeting the defined conditions. This is the basic grouping solution in K8s.
- Controllers: Controllers are specific feedback control loops which are executed to move the entire system to a desired state. Some controllers are pre-defined in K8s technology. For example, "Replication Controller" creates a predefined number of copies of the same pod; or "Job Controller" is employed to run a pod

until the program finishes and then it is destroyed.

- K8s services: A K8s service is a set of pods working together to provide a high-level service or application. Services are identified by means of label selectors. K8s provide each K8s service with a name and a fixed IP addressed which is stored in the DNS. Using a round-robin technique, communication traffic is divided among the pods composing the service. By default K8s service are consumed by applications in the node cluster, but it also may be offered publicly.

Components in K8s are weakly linked, so more containers and/or resources may be associated to a K8s service dynamically. These functionalities, as well as other control and management characteristics are executed in a master node which controls the other nodes in the cluster. Figure 1 presents the K8s architecture.

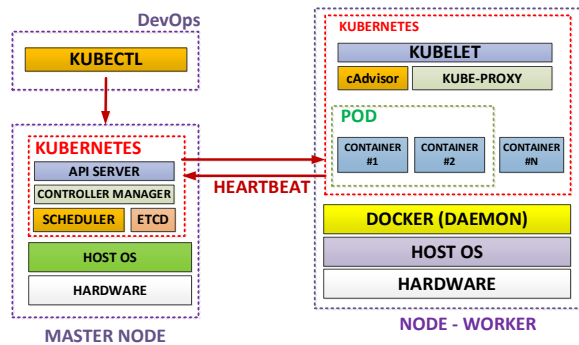


Fig. 1. K8s architecture

K8s follows a master-slave approach, with several K8s nodes or workers and a unique master node. Each worker executes a collection of containers grouped in pods. Besides the pods, each node must execute three control components:

- Kubelet: It is the component controlling the container execution in each node. Kubelet monitor the state of K8s pods in the node and, if they are not in the desired state, regenerates them. Each few seconds, Kubelet communicates the node situation to the master node (a message known as heartbeat). If the node is not working properly, the replication controller executes new pod copies in healthy nodes.
- Kube-proxy: It is a particular implementation of a network proxy and traffic balancer. It

routes datagrams among containers using the same IP address.

- cAdvisor: It is the component in charge of monitoring the resource and CPU usage in each node.

The main node receives orders from managers (DevOps, Development and operations team) through a specific line command interface named as Kubectl, and communicates with workers to dynamically and automatically decide how to implement these orders. With this view, communication services could be deployed, replicated, etc. regardless the computing platform. To reach this objective, the main node must implement four basic components:

- etcd: It is a persistent repository where cluster, K8s service and pod configuration is stored.
- Scheduler: This component selects which node must execute a new non-scheduled pod. To perform this selection, the scheduler considers the current node situation (to ensure no node is congested), the Quality-of-Service limitations, etc. Basically, it matches the resource offer and demand.
- API server: This component provides an internal and external REST API to enable users, containers and other entities to interact with other components and configure new works and services.
- Controller manager: This component executes a process employed to run a particular controller when required. This controller employs the API server to interact with pods, nodes and other components.

Other solutions, such as Docker Swarm, have been reported to coordinate and manage containers. However, they present important limitations making K8s the most employed technology nowadays.

Although deploying K8s may be a costly procedure (more than one hundred seconds could be necessary) [57], this process is only performed before network operation and does not affect the characteristics of slices and services (especially the end-to-end delay).

2.3. 5G networks: standards and relevant proposals

Future 5G networks are being developed nowadays. Different scientific proposals may be found in relation to access networks, handovers, core networks, etc. However, in this paper, we are focusing on realistic network architectures; and, then, standards are our main reference. Future 5G technologies,

as current 4G (Long-Term Evolution) networks, are managed by 3GPP standards. Nevertheless, at the time of writing, no full 5G specifications have yet been released. In December 2018 the first document about future 5G networks (known as 5G Systems – Phase 1) was released [55], but it is mainly focused on RAN and a very low-level description of Core Network (functional components, protocols, messages, etc.). Additional details about service layer will be required to be taken as input in our work. A much more advanced document (with additional, new and relevant details) will be released by the end of 2019; but (at the time of writing) no information about this future standard has been published.

Therefore, we are taking as main reference the white paper “View on 5G architecture” created by the 5G-PPP organization, released in January 2018 [41]. This white paper is considered an official input for future 5G network standardization.

Any case, this paper addresses a challenge that is in a higher level than physical architectures, so even if details of 5G change or new unexpected application domains arise, the basic challenges this work addresses will be still valid.

Future 5G networks are characterized by the unusual, variable and heterogeneous values they present in some relevant indicators: mobility, device density, latency, reliability, etc. The cause of this situation is the large catalogue of applications these networks are envisioned to cover; which are classified into six basic groups: (i) automation and tactile Internet, (ii) smart offices, (iii) dense urban, (iv) connected vehicles, (v) low bandwidth Internet of Things, and (vi) broadband everywhere (50Mbps+).

To provide more homogeneity and make this situation more practical, the idea of network slicing emerges [6]. Network slicing is a technique where network resources are packed and assigned in an isolated manner to a set of users, i.e. to one of the previously cited group of services, according to their particular needs and requirements. Different authors have proved to enable the automatic, efficient and dynamic creation and maintenance of slices, future 5G networks must be softwarized, i.e. they must be based on virtual network functions. In order to achieve this objective, the traditional monolithic architecture of mobile networks (see Figure 2) is divided into three basic layers (physical infrastructure, network layer and service layer) and a set of transversal operations (security and management), where the global network has not a unique tenant, but is the result of the interconnection of several different networks and services belonging to different tenants. In

this proposal, physical infrastructure is recovered by a specific and innovative OS named as “Network Operating System” (sometimes abbreviated as NOS, although some particular proposals may employ specific names for their solutions) [42], which enables the virtualization of infrastructure resources. Slices are, in this context, created by instantiating a generic object and model (see Figure 3).

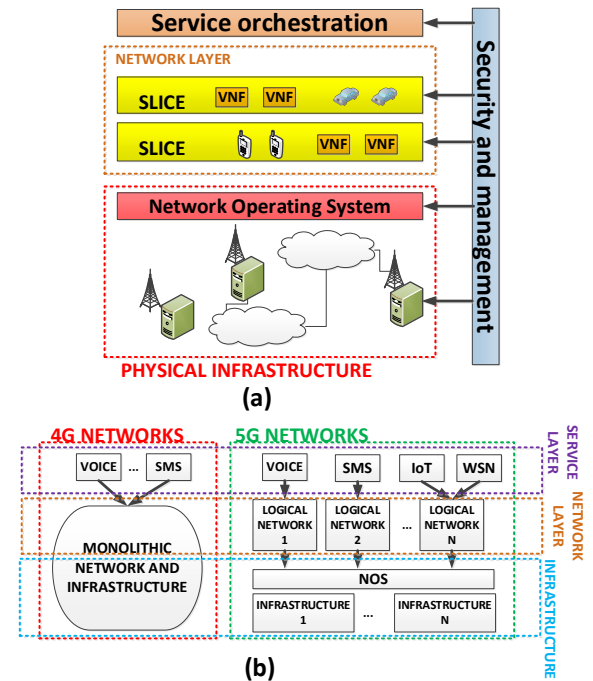


Fig. 2. Overall architecture for future 5G networks (a) reference architecture (b) comparison to 4G networks

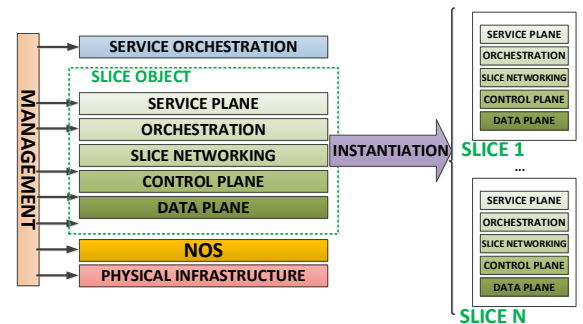


Fig. 3. Slice creation in 5G networks

This object defines all functionalities inside the slice (control, data, service management, etc.). This domain is known as intra-slice domain. The intra-slice domain is in charge of orchestrating and managing all communication services and applications sup-

ported by the slice (see Figure 4), as well as network functions supporting those services. These network functions may be dedicated or shared, depending on if they are consumed by one or several communication services.

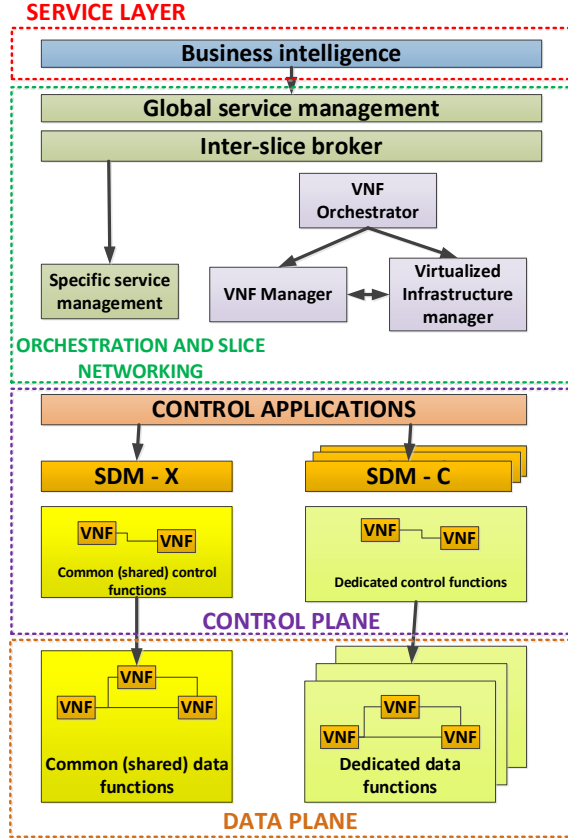


Fig. 4. Intra-slice architecture

In the service plane, the service (business) intelligence is located. These functionalities may be operated by the service tenant if required. The orchestration layer includes a broker to communicate with the global service (slice) orchestrator, and elements to manage the communication services offered by the slice. On the other hand, slice networking functionalities include components to manage the underlying virtual network functions supporting the communication services in the slice, such as capabilities related to VNF Management and Orchestration (MANO) [43].

Finally, control and data planes include all VNF supporting the services in the slice. Control plane includes control applications and data planes data services. For each slice, there is a common group of network functions shared by several services in the

slice, and a set of dedicated functions grouped in different sub-networks (each one associated with a specific communication service). Common functions are controlled through a Software Defined Manager named as SDM-X; and dedicated functions requires a different specific manager (SDM-C) for each sub-network.

Slices are end-to-end entities. The use of this technology, thus, enables the concurrent deployment of several logical, independent and self-contained networks sharing the same infrastructure. These networks are softwarized, abstract and isolated from the environment. Nevertheless, all these entities must be organized and, finally, orchestrated. Therefore, the intra-slice domain is complemented with an inter-slice domain in charge of these global management operations.

Inter-slice domain supports a global broker to coordinate the operations of all slices, and a particular slice (including dedicated virtual network functions) to implement global control applications (no data services are deployed within this domain). In general, the inter-slice domain contains the business logic and control policies configured and managed by the slice provider. Thus, although in the intra-slice domain capabilities to control the offered Quality-of-Service (QoS) and mobility in the specific applications and services supported by the slice are included; in the inter-slice domain global managers for all those characteristics are also considered. These global managers support policies valid for all slices: slice access control as service provider, the (maximum) allowed QoS, service and inter-slice mobility, etc.

Inter-slice and intra-slice management solutions are nowadays very sparse. This paper proposes a solution for 5G networks supported by virtualization techniques. The proposed management solution covers the intra-slice domain, and future works will address the inter-slice domain.

3. A virtualization-based architecture for 5G networks

In order to implement complex architectures as 5G mobile networks, not only one virtualization technique must be employed, but a coordination of several solutions at different levels according to the characteristics and properties (with special interest on security and isolation) of the system to be deployed.

In particular, to implement 5G networks by the coordination of components which are created and

operated in an isolated manner (see Section 2.3), we propose an architecture based on the concept of “microservices” [44].

Microservices is a modern variant of a service-oriented architecture (SOA) where applications (or complex services) are structured as a collection of loosely coupled services. These services may be developed and deployed in independent environments together with all their dependencies. As can be seen, this approach matches both, the requirements of 5G networks and the characteristics of container-based virtualization.

The next subsections describe the agents and businesses involved in 5G networks, and our proposed hybrid virtualization solution to implement 5G networks with network slicing (based on VM, Docker and K8s), following a microservice approach. Also, we discuss how our proposal matches the designed architectures for future 5G systems (see Section 2.3). We pay special attention to those management functionalities that must be present in these next-generation networks, and could be easily covered by the native mechanisms available in Docker and (overall) K8s.

3.1. Involved agents. Proposed virtualization scheme

Considering the information shown in Section 2.3, three different agents must be considered. These agents, also, define three basic business models based on future 5G networks. A fourth hypothetical market could appear, supported by the concept of network slicing. These agents and business models are:

- **Infrastructure owner:** It may be understood as a traditional Platform as a Service (PaaS) or Infrastructure as a Service (IaaS) agent. Using a certain physical infrastructure this agent offers VM and resource packets for which it obtains a fee. This agent must deploy, maintain and manage both, the core and access network (router, antennas, etc.) and physical resources (spectrum).
- **Communication service provider:** Standard telecommunication operators may take this role. Basically, using the VM provided by the infrastructure owners, service providers must deploy the network functions and slices to support communication services (e.g. voice, data connection with a certain throughput and QoS, etc.).
- **Final service provider:** These providers are responsible of Over-The-Top (OTT) services. They rent communication services to telecom-

munication providers in order to deploy and support next-generation user services.

- **Slice provider:** This role is not certain to appear, as it depends on the future economic movements. In order to generate more specialized businesses, telecommunication operators may decide not to deploy and control network slicing technologies. This role, then, will be taken by a different economic agent who would rent VM to infrastructure owners and would create and maintain slices and inter-slice components. The key element in this discussion is the inter-slice domain: agent controlling network slicing must collaborate also in the maintenance of inter-slice functions.

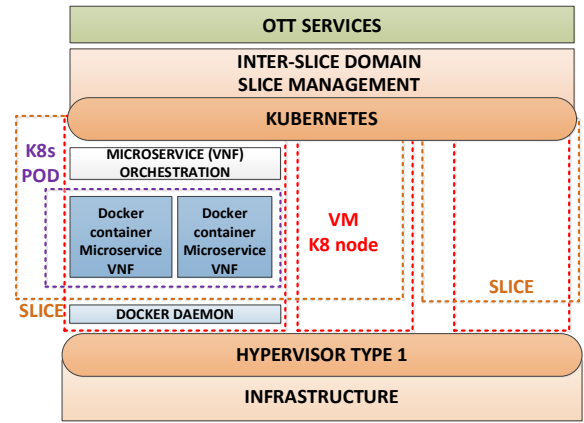


Fig. 5. Proposed virtualization scheme for future 5G networks

However, to enable the real existence of three independent markets, infrastructure must be isolated from network functions, slices and services. To reach this objective, a hybrid virtualization solution must be employed. In fact, this view matches with the design approach of K8s, where also three basic different layers are identified (pods, containers and nodes). Considering this relation, Figure 5 shows the proposed virtualization scheme for future 5G networks.

The proposed scheme virtualizes the physical infrastructure using hypervisor-based type 1 virtualization. These operations are performed by infrastructure owners, whose business is renting VM to communication service providers and/or slice providers. The created VM (the K8s nodes) are assigned to slices. To ensure the maximum isolation level among slices, a VM may only support services of a unique slice. Inside each VM a collection of Docker containers implementing each one a virtual network function is

deployed. Each one of these containers is configured as a microservice. To enable this operation, VM must be provided with Docker daemon.

Communication service providers are in charge of deploying these VNF to create the adequate (common and dedicated) logic network(s) for supporting all communication services for the target application scenario associated to the slice. Telecommunication operators must also collaborate in the maintenance of inter-slice functions if they take care of network slicing. As previously said, if a real slice market appears in the future, slices and inter-slice components would be managed and maintained by the slice providers.

Finally, OTT services are deployed in proprietary servers and hardware infrastructure whose connectivity is guaranteed by telecommunication operators. However, these services do not affect at all the network architecture.

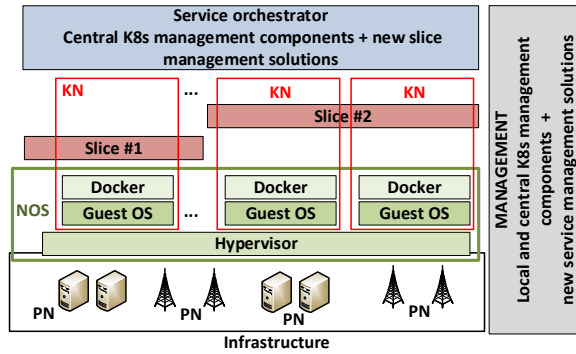


Fig. 6. Virtualization-based overall architecture implementation for future 5G networks

3.2. Proposed virtualization-based architecture

With the proposed virtualization scheme, the global architecture for 5G networks gets strongly simplified (see Figure 6). This proposal, as can be seen, matches the proposed overall architecture for 5G networks in Figure 2, the K8s architecture (Figure 1) and the proposed virtualization scheme (see Figure 5).

The described NOS can be implemented using the Docker daemon, a guest OS and the hypervisor type 1 employed to virtualize the physical nodes (PN). Among these components, only specific hypervisors should be developed. Both the Docker daemon and OS are well known and reliable solutions.

As said in Section 3.1, VM (or K8s nodes) are associated to a unique slice, but all Docker containers inside each node (configured as microservices ex-

cuting VNF) are managed using two instrument types: the K8s local management functions and components and new and specific service management solutions.

K8s local management components include Kubelet, Kube-proxy and cAdvisor. These elements, as described in Section 2.2, can monitor, control, regenerate or report permanent problems about K8s pods executed in the K8s node they manage. These functions match (see Section 2.3) the capabilities associated to controllers SDM-X and SDM-C, to control applications and to VNF management components (orchestrator manager and infrastructure manager) in slices architecture (see Figure 4). Communication services will be configured as K8s services, which enables a more efficient control.

Intra-slice management functions are, in that way, covered by trustworthy, mature, commercial and reliable technologies. Only one component must be developed (see Figure 4): the specific service management (which is the second solution type to support communication service management). A possible first implementation for this element is described in Section 4.

Service orchestration layer groups all inter-domain management functions and components. In an equivalent manner as explained for the intra-slice domain, K8s global management components may take care of an important part of the inter-domain management functions. Specifically, all capabilities related to NFV (microservices or containers) management can be executed by the K8s scheduler, etcd, and controller manager (see Section 2.2.2). On the other hand, global slice management functions (global service manager, global QoS manager, global mobility manager, etc.) must be implemented. Specific solutions for each situation and scenario are required. Some solutions have been already proposed [18]. In future works some proposals will be investigated

4. Proposed algorithm for efficient service management in future 5G networks

In this section, an enhanced service management solution for 5G networks implementing our proposed architecture is described. This technology is designed to operate over the interfaces and functionalities offered by K8s and Docker, covering those functions that these virtualization technologies cannot perform, as explained in Section 3.2.

Service management policies are implemented in the intra-slice domain (in the specific service manager, see Figure 4). In the proposed architecture, communication services are K8s services, composed of, at least, two K8s pods: the common network functions and the dedicated network functions (one sub-network per communication service, as said in Section 2.2.3). Containers in these pods are configured as microservices. While K8s manage these elements from the virtualization point of view (as containers), in this section a management algorithm to control them as containerized microservices (including QoS considerations and other relevant communication indicators) is described. The objective is to automate operations usually performed manually by system managers in the K8s architecture, but that in 5G networks must be performed dynamically by software control elements.

In this section, an algorithm to manage the two main and most basic operations is proposed: the creation of pods (what microservices are put together in the same pod) and the deployment of redundant components (number of copies of each pod to be deployed). In both cases, algorithms try to optimize costs and the use of resources.

4.1. Mathematical model

In the proposed architecture, a slice is modelled as a microservice architecture. A microservice architecture $\mathcal{A}$ is an ordered pair of finite sets $\mathcal{A} = (S_{\mathcal{A}}, I_{\mathcal{A}})$, being $S_{\mathcal{A}}$ and $I_{\mathcal{A}}$ non-empty sets. $S_{\mathcal{A}}$ is named as the collection of microservices (1), whose cardinality is N

$$S_{\mathcal{A}} = \{s_j \mid j = 1, \dots, N\}, \quad (1)$$

and $I_{\mathcal{A}}$ is the set of interconnections (2), being $\text{card}(I_{\mathcal{A}}) = K$.

$$I_{\mathcal{A}} = \{i_j \mid j = 1, \dots, K\}. \quad (2)$$

Both sets $S_{\mathcal{A}}$ and $I_{\mathcal{A}}$ are related by means of three different applications (3)

$$\begin{aligned} \gamma_{\mathcal{A}} : I_{\mathcal{A}} &\rightarrow S_{\mathcal{A}} \times S_{\mathcal{A}}, \\ \sigma_{\mathcal{A}}, \tau_{\mathcal{A}} : I_{\mathcal{A}} &\rightarrow S_{\mathcal{A}}. \end{aligned} \quad (3)$$

$\gamma_{\mathcal{A}}$ is the incidence application of the system, and relates each interconnection i_j with the order pair of microservices $\{s_i, s_j\}$ this interconnection connects. The interconnection is outgoing from s_i and incoming in s_j .

$\sigma_{\mathcal{A}}$ is the source application of the system, and associates each interconnection i_j with the microservice s_i which is the origin of the interconnection i_j .

Finally, $\tau_{\mathcal{A}}$ is the target application of the system, and associates each interconnection i_j with the microservice s_j which is the destination of the interconnection i_j . Although they are different applications, both $\sigma_{\mathcal{A}}$ and $\tau_{\mathcal{A}}$ have the same origin and target sets.

In that way, a microservice architecture may be understood as a directed or oriented graph $G_{\mathcal{A}}$ characterized by any of the two following tuples (4)

$$\begin{aligned} G_{\mathcal{A}} &= (V_G, E_G, \gamma_G), \\ G_{\mathcal{A}} &= (V_G, E_G, \sigma_G, \tau_G). \end{aligned} \quad (4)$$

V_G is the set of vertices of the graph, and E_G is the set of edges. γ_G is the incidence application of the oriented graph, and σ_G and τ_G are the source and target functions, as defined in the mathematical graph theory.

The graph $G_{\mathcal{A}}$ is said to be a reduced representation of a microservice architecture $\mathcal{A}$, if there exists a pair of bijections $(\mathbb{f}_V, \mathbb{f}_E)$ $\mathbb{f}_V : S_{\mathcal{A}} \rightarrow V_G$, ; $\mathbb{f}_E : I_{\mathcal{A}} \rightarrow E_G$ fulfilling that $\forall s_i, s_j \in S_{\mathcal{A}}$ and $\forall i_i \in I_{\mathcal{A}}$ such $\gamma_{\mathcal{A}}(i_i) = \{s_i, s_j\}$ and $\sigma_{\mathcal{A}}(i_i) = s_i$ and $\tau_{\mathcal{A}}(i_i) = s_j$. It is proved that $\exists v_i, v_j \in V_G$ and $e_i \in E_G$ such $\gamma_G(e_i) = \{v_i, v_j\}$ and $\sigma_G(e_i) = v_i$ and $\tau_G(e_i) = v_j$ if, and only if, $v_i = \mathbb{f}_V(s_i)$ and $v_j = \mathbb{f}_V(s_j)$ and $e_i = \mathbb{f}_E(i_i)$.

As can be seen, these functions allow transforming a technological system into a graph and vice versa, so (hereinafter) we are using indistinctly both notions.

Although this representation fully describes any network, in order to perform some advanced calculations and management operations it may not be adequate. Therefore, an expanded representation is proposed.

To create the expanded architecture representation, each microservice s_i is described as a collection of endpoints, being $\mathcal{E}$ the set of all possible endpoints to be implemented in a microservice (5):

$$s_i = \{e_1^{s_i}, \dots, e_{M_{s_i}}^{s_i}\}, \quad e_j^{s_i} \in \mathcal{E} \forall j. \quad (5)$$

Then, interconnections among microservices are replaced by interconnections among endpoints, which allow the interoperation among service providers and service consumers via a certain request type. Between two endpoints, then, as many interconnections as request types they may exchange have to be considered.

In an equivalent manner to which previously described, it is possible to define an oriented and labeled graph to mathematically describe this expanded architecture representation (6):

$$\begin{aligned} G_{\mathcal{A}}^+ &= (V_G^+, E_G^+, \gamma_G^+), \\ G_{\mathcal{A}}^+ &= (V_G^+, E_G^+, \sigma_G^+, \tau_G^+). \end{aligned} \quad (6)$$

As in the previous case, γ_G^+ is the incidence application of the oriented graph, and σ_G^+ and τ_G^+ are the source and target functions for the extended graph.

V_G^+ is the set of vertices of the extended graph (representing each one an endpoint), and E_G^+ is the set of edges (representing each one a request between two endpoints).

Moreover, vertices and edges in the $G_{\mathcal{A}}^+$ graph may inherit some properties from microservices, endpoints and requests. In particular, each edge has associated a label (req_i , res_i , p_i):

- req_i is the request type associated to this edge. This element is taken from the set of possible requests, $req_i \in R$.
- res_i is a real number representing the resource consumptions associated with the request req_i between the endpoints linked by the corresponding interconnection.
- p_i is an integer number representing an statistical estimation of the amount of requests required to obtain a successful result in this interconnection. Packet losses in networks, errors in communications links, and other similar causes increase this number.

On the other hand, vertex are also labeled with four parameters ($Mres_i$, μ_i , $maxRes_i$, $error$, $type$):

- $Mres_i$ is a real number representing the mean resource consumption associated operations triggered by to this endpoint.
- μ_i is an integer number representing an statistical estimation of the amount of requests required to obtain a successful result in this endpoint. This number increases with internal errors only related to the microservice.
- $maxRes_i$ a real number representing the maximum resource consumptions the endpoint may support. Any attempt to consume more resources than described by this parameter causes a reduction in the QoS and degradation in the endpoint performance.
- $type$ is a label indicating if the endpoint may only receive requests from other endpoints in the architecture (backend endpoint) or it may also receive requests from external entities (frontend endpoint).
- Finally, $error$ is the percentage of failed operations associated to the corresponding vertex or endpoint.

Figure 7 shows a graph representing a simple expanded microservice architecture. In this figure, the labels of edges (interconnections) are represented in colour red; and labels of vertices (microservices) are

written in colour black. In order to improve the figure readability, not all labels are shown. In this case, requests associated to interconnections are REST (Representational State Transfer) requests (although other calls could be employed). Proposed values are just examples. The system shown in Figure 7 does not represent any particular slice or coherent scenario; it is just a graphic representation of the data structure employed to work with slices, on which latter calculations will be based.

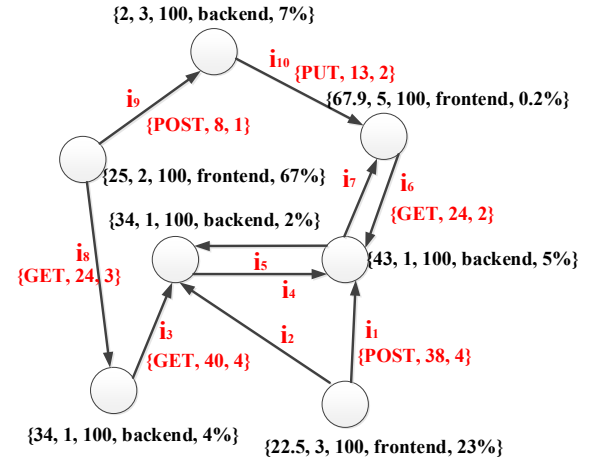


Fig. 7. Example of a graph representing a microservice architecture

From this expanded graph $G_{\mathcal{A}}^+$, it is possible to extract partial graphs considering that in microservice architecture, any request causes a cascade of additional requests. These partial graphs $G_{\mathcal{A}}^{i,j}$, sometimes named as microservice call graphs [45], are specific for each request type in each endpoint. Mathematically, they are directed acyclic graphs. To operate with the call graphs more easily in the next sections, we are employing a simplified notation (7):

$$G_{\mathcal{A}}^{i,j} \equiv \{t_1, \dots, t_M\} = \{(v_1, r_1), \dots, (v_M, r_M)\}. \quad (7)$$

M is the cardinality of the set of vertices V_G^+ , $M = card\{V_G^+\} = \sum_{s_i} M_{s_i}$, t_j are two-tuples representing each one the calls to the v_j vertex the i_j interconnection (request) causes. v_j is a vertex in the expanded graph $G_{\mathcal{A}}^+$, $v_j \in V_G^+$. And r_j is the number of calls made to the vertex (endpoint) v_j caused by i_j interconnection (request).

4.2. Efficient deployment of redundant components

The basic purpose of this algorithm is to calculate the deployment cost of any additional redundant component Q_d^m . This cost is compared to the potential cost (in terms of QoS reduction, more complex managing policies at higher levels, etc.) of not deploying this new and redundant element Q_{n-d}^m . In this context, components are identified as pods (we are assuming its size is optimum, see Section 4.1.3).

Q_d^m is the deployment cost of a new redundant component (pod) when m copies of this same pod are already deployed. Equally, Q_{n-d}^m is the cost of not deploying a new copy pod, after m copies of the same pod are already deployed.

To calculate the cost Q_d^m it is necessary to evaluate the workload $\omega_{s_i}^T$ (8) supported by each microservice s_i during a period of T seconds. The simplest manner to evaluate this workload is to count the number of requests received q_i by each endpoint $e_i^{s_i}$ in the microservice s_i during the corresponding time period (being M_{s_i} the number of endpoints in the microservice s_i):

$$\omega_{s_i}^T = \{(e_1^{s_i}, q_1), \dots, (e_{M_{s_i}}^{s_i}, q_{M_{s_i}})\}. \quad (8)$$

Three basic resources are consumed when a new pod is deployed: (i) computation resources, (ii) storage resources, and (iii) API resources. The consumption of these resources in the m_i -th pod depends linearly on the number ρ_{m_i} of requests the pod is designed to deal with (as microservices in K8s architectures are lambda-backed microservices [45]). K8s solution balances traffic among all the running pod copies, but these copies are not usually designed to manage (each one) the total amount of workload. Thus (9):

$$\rho_{m_i} < \sum_i q_i. \quad (9)$$

This fact (among other factors), as we are seeing, is responsible of the existence of the cost Q_{n-d}^m .

Apart from these variable consumptions c_v , some resources are also consumed independently from the request number to be managed: the IP address, ports, etc.

In conclusion, the total resource consumption C_{m_i} for the m_i -th pod is calculated as (10):

$$C_{m_i} = \rho_{m_i} \cdot c_v + c_c = \left(\sum_{c_s \in RV} c_s \right) + c_c, \quad (10)$$

where c_v is the unitary variable consumptions which depends on the request number, and c_c is the

constant resource consumption. c_s is the specific resource consumption, belonging to a set of resource variables RV (identified before: computing resources, API resources, etc.).

To obtain a real estimation of the resource consumption, this expression is combined with the previously proposed mathematical model. API resource consumption is included in the label of each edge in the expanded graph $G_{\mathcal{A}}^+$, res_i . As this amount is the total resource consumption, where two endpoints (microservices) are involved, we are considering the consumption is equally divided into both vertices (11):

$$res_i^{v_i} = \frac{res_i}{2}. \quad (11)$$

On the other hand, computing and storage resource consumption (a combined and mean value) is included in the labels of vertices: res_i . For each microservice, then, the variable resource consumption is calculated as (12)

$$c_v^{s_i} = \sum_{j=1}^{M_{s_i}} \left(\frac{res_j}{2} + Mres_j \right), \quad (12)$$

and the total consumption for the entire pod (13)

$$c_v = \sum_{j=1}^N c_v^{s_j}. \quad (13)$$

The number of requests to be supported by a pod is obtained from the call graphs and the expanded graph $G_{\mathcal{A}}^+$. System designers would indicate the workload to be supported by the frontend microservices s_i during the hour with the highest peak traffic (according to traffic theory [46]), $\omega_{s_i}^{peak-hour}$. For each one of the endpoints in the frontend microservices the call graph is calculated. Besides, for each request, to generate a successful result, various sequential requests could have to be performed, as indicated by parameters μ_i and p_i .

In consequence, the total number of requests (14) to be managed by the pod is (hereinafter all calculations are referred to the hour with highest peak traffic):

$$\rho_{total} = \sum_{s_i \in S_{front}} \sum_{j=1}^{M_{s_i}} \left(q_j \cdot \sum_{k=1}^M \left(\sum_{i_j} r_k^{i_j} \right) \cdot p_k \cdot \mu_k \right). \quad (14)$$

Each pod, in general, would be designed to manage only a certain percentage of this amount (15). Being α_{m_i} a real number, $\alpha_{m_i} \in [0,1]$,

$$\rho_{m_i} = \alpha_{m_i} \cdot \rho_{total}. \quad (15)$$

As K8s balances traffic using a round robin technique, this percentage must be, at least, enough to accept the corresponding traffic load (16):

$$\alpha_{m_i} > \frac{1}{m_i}. \quad (16)$$

Using these expressions, the resource consumption may be calculated dynamically in a very lightweight manner. It is very important to note that call graphs and other relevant parameters employed in the proposed mechanism, such as res_i, p_i, μ_i or $Mres_i$, are periodically updated and calculated by cAdvisor component (see Section 2.2.2).

System designers must only define parameters $\omega_{s_i}^{peak-hour}$ and α_{m_i} (and both parameters maybe automatically calculated if desired) and the resource consumption is obtained.

To transform this consumption into the cost Q_d^m a cost function $f_{cost}^{Qd}: \mathbb{R} \rightarrow \mathbb{R}$ is employed (17):

$$Q_d^m = f_{cost}^{Qd}(C_{m_i+1}). \quad (17)$$

This function, although it may take different expressions, follows in this paper an exponential law with a vertical asymptote in $C_{m_i+1} = C_T$, being C_T the total amount of resources available in the system. Cost at this point has to grow up indefinitely to indicate no more pods can be deployed (no resources are available). Figure 8 represents this function. Resources which are available in large amounts tend to have a lower cost (it is a basic economic principle). However, this cost tends to go up as resources become sparser.

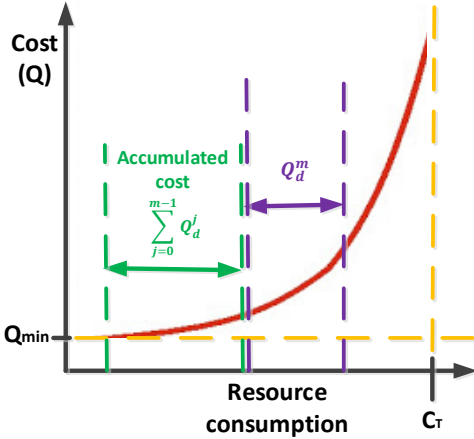


Fig. 8. Proposed cost function

On the other hand, the cost Q_{n-d}^m must be evaluated. In a lower level, a pod (component) is a set $\mathcal{D}$ of Docker containers operating as microservices, whose cardinality is N_d (18):

$$\mathcal{D} = \{cont_i \mid i = 1, \dots, N_d\}. \quad (18)$$

Among them, a maximum of d_f Docker containers (microservices) can fail before the availability of the global communication services in the slice to be in danger. However, the QoS may go down earlier, when only d_q microservices fail. In a traditional network design all these parameters are related (19):

$$d_q < d_f < \frac{N_d}{2}. \quad (19)$$

To these microservices, an arbitrary and finite number of users N_u is connected. As more copies of the same pod are running, parameters N_d, d_q and d_f go up, and user perceives a higher and more stable QoS. However, it always exists a probability of more than d_q or d_f components to fail. In our model we are assuming a pod fails when any of the microservices that make it up fails. Equally, a microservice fails when any of the endpoints that make it up fails. As said in the mathematical model this occurs if the resource consumption is above a certain maximum value (failure because of overload) (20):

$$\exists e_i^{s_i} : \max Res_{s_i} < \left[q_i \cdot p_i \cdot \mu_i \cdot \left(\frac{res_{s_i}}{2} + Mres_{s_i} \right) \right]. \quad (20)$$

But it may also occur because an internal error, a situation which is modelled by probability $error_{e_j}^{s_i}$.

In the same manner, we are considering the QoS is strongly affected when the resource consumption goes above a certain percentage, being β a real number, $\beta \in [0,1]$, usually $\beta = 0.7$. Although other values could be considered, this threshold has been proved to be the inflexion point from which QoS decreases exponentially [52]. Before this point, QoS evolves linearly, so it is an adequate threshold to represent network congestion (21).

$$\exists e_i^{s_i} : \beta \cdot \max Res_{s_i} < \left[q_i \cdot p_i \cdot \mu_i \cdot \left(\frac{res_{s_i}}{2} + Mres_{s_i} \right) \right]. \quad (21)$$

In these previous expressions, p_i and μ_i are real values measured at real-time by cAdvisor. Thus, the increase in the resource consumption may only be caused by an increase in the workload, i.e. an increase in parameter q_i . This parameter follows a certain statistical distribution: a Poisson process. In fact, user connections are independent and the traditional traffic models [46] may be employed. Thus, the probability of this parameter to go above a certain limit q_{limit} is (22):

$$P[q_i > q_{limit}](\lambda) = \int_{q_{limit}}^{\infty} \frac{e^{-\lambda} \lambda^k}{k!} dk. \quad (22)$$

In this expression, λ is the connection rate. As systems are usually designed to work under the worst conditions, the values would be calculated by cAdvisor during the highest peak traffic, i.e. $\lambda = \lambda_{max}$. On the other hand, as improper integrals are difficult to calculate, and using the properties of probability density distributions, a simpler equation may be obtained (23):

$$P_{q_{limit}} = P[q_i > q_{limit}] = 1 - \int_0^{q_{limit}} \frac{e^{-\lambda_{max}} \lambda_{max}^k}{k!} dk. \quad (23)$$

Using the previous expression, and considering that microservices are independent (isolation is guaranteed by Docker technology), two global probabilities (for the entire pod) are obtained. The first one, the probability of failure (24)

$$P_{fail} = \frac{\sum_{s_i} \sum_{e_i^{s_i}} \left(P \left[q_i > \frac{maxRes_{s_i}}{p_i \cdot \mu_i \cdot \left(\frac{res_{s_i}}{2} + Mres_{s_i} \right)} \right] + error_{e_i^{s_i}} \right)}{2M}, \quad (24)$$

and the second one, the probability of QoS degradation (25)

$$P_{QoS} = \frac{\sum_{s_i} \sum_{e_i^{s_i}} \left(P \left[q_i > \frac{\beta \cdot maxRes_{s_i}}{p_i \cdot \mu_i \cdot \left(\frac{res_{s_i}}{2} + Mres_{s_i} \right)} \right] \right)}{M}. \quad (25)$$

Finally, to obtain the probability of a global fail in the system, or a QoS reduction, all deployed pods must be considered. Being deployed m pod copies:

- A global fail is produced only if all pods fail.
- A QoS reduction appears if all pod employs more than β (%) of their resources.
- A QoS reduction also appears if m_{QoS} pod copies fail. Usually: $m_{QoS} = \xi \cdot m$, being ξ a real number, $\xi \in [0,1]$.

Therefore, and considering that pods are independent entities (being m the number of deployed pod copies), the probabilities are calculated as (26):

$$P_{fail}^{sys}(m) = \prod_{k=1}^m P_{fail},$$

$$P_{QoS}^{sys}(m) = \frac{1}{2} \left(\prod_{k=1}^m P_{QoS} + \prod_{k=1}^{\xi \cdot m} P_{fail} \right). \quad (26)$$

Each one of these probabilities, now, is associated with a cost using the corresponding functions, $f_{cost}^{fail}(\cdot)$ and $f_{cost}^{QoS}(\cdot)$. The mathematical expression of these functions must depend on the communication service

under study, the users, the contracts, the corporative strategy, etc. These costs are associated to indemnities, fines, loss of reputation, etc. (27)

$$Q_{n-d-QoS}^m = f_{cost}^{QoS}(P_{QoS}^{sys}(m+1)),$$

$$Q_{n-d-fail}^m = f_{cost}^{fail}(P_{fail}^{sys}(m+1)). \quad (27)$$

To calculate the final cost of not deploying a new pod copy, both partial costs must be combined. A weighted arithmetical average may be employed, employing the weighting factor η_i proposed by system manager or designers (28).

$$Q_{n-d}^m = \eta_1 \cdot Q_{n-d-fail}^m + \eta_2 \cdot Q_{n-d-QoS}^m. \quad (28)$$

The optimum number of pod copies to be deployed is that which makes $Q_{n-d}^m = Q_d^m$ (when both costs are equal).

Algorithm 1 summarizes the entire process. Basically, the proposed algorithm iterates along the list of deployed microservices and analyses all interconnections whose origin is each one of these microservices. Then, probability of a failure and the probability of QoS degradation are obtained for each microservice and, finally, for the entire system (aggregating all previous partial results). The same operation is performed for the resource consumption of each microservice and the whole system. Using these three variables, an optimization problem is defined and solved to obtain the optimum number of pod copies.

Algorithm 1 Calculation of the optimum number of pod copies to be deployed

Input $\omega_{s_i}^{peak-hour} = \{..., (e_i^{s_i}, q_i), ...\} \quad \forall s_i$

Output Number of pod copies to be deployed num

cAdvisor updates values for res_{s_i} , μ_i , p_i , $Mres_{s_i}$, $error_{e_i^{s_i}}$, $maxRes_{s_i}$ and c_c

for each microservice s_i **do**

Obtain $c_v^{s_i}$

for each request i_j in microservice s_i **do**

Obtain the call graph $G_{e_i^{s_i}}^{i_j} = \{..., (v_i, r_i), ...\}$

end for

$$\rho_{s_i} \leftarrow \sum_{j=1}^{M_{s_i}} (q_j \cdot \sum_{k=1}^M (\sum_{i_j} r_k^{i_j}) \cdot p_k \cdot \mu_k)$$

First position in entry is equal to the second position

$$P_{fail}^{s_i} \leftarrow \sum_{e_i^{s_i}} \left(P \left[q_i > \frac{maxRes_{s_i}}{p_i \cdot \mu_i \cdot \left(\frac{res_{s_i}}{2} + Mres_{s_i} \right)} \right] + error_{e_i^{s_i}} \right)$$

$$P_{QoS}^{s_i} \leftarrow \sum_{e_i^{s_i}} \left(P \left[q_i > \frac{\beta \cdot maxRes_{s_i}}{p_i \cdot \mu_i \cdot \left(\frac{res_{s_i}}{2} + Mres_{s_i} \right)} \right] \right)$$

end for

$$\rho_{total} \leftarrow \sum_{s_i \in S_{front}} \rho_{s_i}$$

$$\rho_{m_i} \leftarrow \alpha_{m_i} \cdot \rho_{total}$$

$$c_v \leftarrow \sum_{j=1}^N c_v^{s_j}$$

```

Calculate  $P_{fail}^{sys}(m)$  and  $P_{QoS}^{sys}(m)$ 
 $C_{m_i} \leftarrow \rho_{m_i} \cdot c_v + c_c$ 
Calculate  $Q_d^0$  and  $Q_{n-d}^0$ 
 $cost_d \leftarrow Q_d^0, cost_{n-d} \leftarrow Q_{n-d}^0$ 
 $m \leftarrow 0$ 
while  $cost_d \neq cost_{n-d}$  then
   $m \leftarrow m + 1$ 
   $cost_d \leftarrow Q_d^m, cost_{n-d} \leftarrow Q_{n-d}^m$ 
end while
 $num = m$ 

```

4.3. Optimum pod creation

In comparison to the calculation of the number of pod copies to be deployed, the calculation of the containers to be included in each pod is much simpler.

Small pods are easier to allocate in physical nodes, but more resources are consumed in coordination tasks. The basic idea of this management algorithm is to find out a balance.

The total resource consumption C of a pod was calculated in the previous section, together with the resource calculation for each microservice and endpoint.

On the other hand, a set $\mathcal{N}$ of physical nodes must support all these pods and containers, being L the cardinality of this set (29):

$$\mathcal{N} = \{phn_i \mid i = 1, \dots, L\}. \quad (29)$$

A physical node phn_i is a two-tuple $(ID, phRes)$ where ID is a binary number, unique in the entire system, identifying the node; and $phRes$ is the total amount of resources available in the node.

Then, a partition $\mathcal{P}$ of the set $\mathcal{D}$ of Docker containers must be created. Each element p_i in this partition represents a pod (30).

$$\mathcal{P} = \{p_i\}. \quad (30)$$

The problem to be solved is to find a partition, or partitions, that could be fully allocated into the physical nodes. Mathematically:

$$\exists \mathcal{P} : \forall p_i \in \mathcal{P}, \exists \{q_1, \dots, q_k\} \quad q_1, \dots, q_k \in \mathcal{P} \cup \emptyset \\ \text{that } C_{p_i+q_1+\dots+q_k} < phRes_j.$$

In fact, as smaller pods are created, this task is easier. In general, the total amount B of possible partitions to be evaluated is obtained through the Bell's number (31):

$$B = B_{N_d} = \sum_{k=0}^{N_d-1} \binom{N_d-1}{k} B_k. \quad (31)$$

It is important to note that, it is only necessary to ensure that the previous statement is true, as the final

and real allocation would be performed by K8s scheduler.

However, at least two pods must be created (a requirement of 5G architectures): one for the common network functions $\mathcal{D}_{com}$, and the other for the specific network functions $\mathcal{D}_{spe}$ (one pod per communication service); and these functions should not be mixed. In conclusion, it is necessary to find two independent partitions $\mathcal{P}_{com}, \mathcal{P}_{spe}$ such $\mathcal{P} = \mathcal{P}_{com} \cup \mathcal{P}_{spe}$. And the number of partitions to be evaluated is lower (32):

$$B = \left(\sum_{k=0}^{N_d^{com}-1} \binom{N_d^{com}-1}{k} B_k \right) + \left(\sum_{k=0}^{N_d^{spe}-1} \binom{N_d^{spe}-1}{k} B_k \right), \quad (32)$$

being $N_d^{com} = \text{card}\{\mathcal{D}_{com}\}$ and $N_d^{spe} = \text{card}\{\mathcal{D}_{spe}\}$. Many lightweight and efficient algorithms to automatically calculate all these partitions and solve the allocation problem can be found [47].

With this approach, however, the trivial solution (one container in each pod) would be a valid solution. Small containers, nevertheless, require from K8s components a greater effort and the employment of more resources. Therefore, in this initial proposal, we are considering as the final solution, the partition where a lower number of pods is created. In advanced and future works, a global distance function to evaluate the dispersion of the 5G system may be created and employed to select the optimum partition.

5. Evaluation and discussion

In order to evaluate the behaviour and performance of the proposed algorithms and architecture, in this section we carry out a set of relevant experiments and provide and analyse the obtained results.

5.1. Experimental validation: materials and methods

Simulations were performed using the Network Simulator 3 (NS3) [50] simulator, which implemented a platform based on Terminal Access Point (TAP) [51] bridges and lightweight virtualization. The proposed simulation platform turns independent the physical infrastructure modelling and the definition of algorithms to be executed and evaluated. The infrastructure model is supported by NS3 simulator,

where different libraries describing low-level protocols, mobile communications, wireless links, etc., are available. On the other hand, proposed algorithms are implemented using Linux paravirtualization techniques, as native C programming is the most adequate to implement the proposed solution.

Using the proposed simulation technique, NS3 scenarios recreate very similar conditions to real physical infrastructures, as well as applications, algorithms and functional components in these simulation scenarios may exchange data and operate as real deployed software does.

The simulation scenario is implemented through TAP bridges, which connect virtual Linux machines generated using LXC technologies and the Libvirt library with the NS3 simulator, and ghost nodes, the virtual entity that represents the virtual machine in the network simulator.

This approach, besides, is equivalent to the virtualization-based architecture described in this paper for future 5G networks; where physical infrastructure is virtualized and divided into VM where Docker software and K8s solutions are installed. Using TAP bridges, physical infrastructure is simulated, but VM are, in fact, created, and real containers, VNF and algorithms may be deployed.

All parameters to create the infrastructure model are provided using a configuration script, where hardware devices, interfaces, protocols, communication links, etc., are described. Although this approach is more complex than the configuration of regular NS3 simulations, it represents in a closer manner the system to be evaluated. TAP bridges are configured as "ConfigureLocal". In that way, TAP bridges are generated in a row, inheriting the configuration properties one from the other.

A small network function was simulated. The following elements were considered:

- Twenty-four (24) physical nodes supporting both the core and the access networks. Nodes were considered to be general purpose servers, as they can be found in cloud computing systems.
- Ten cells in the mobile network, to provide communication services to users.
- Sixty (60) virtual machines were created, all of them implementing the Operating System Ubuntu 16.04, and the Docker software. The total amount of physical resources was divided into the VM in this way: two machines using 10% of resources each one (main nodes); eight ma-

chines using 3% of resources each one; fifty (50) machines using 1% of resources each one.

- Virtual Network Functions were implemented in PHP language using the existing algorithms in the NS3 libraries.
- K8s solution was implemented in all VM, acting one of the main nodes as K8s main node.
- Communication services to be provided were considered to be only broadband (50Mbps+) data connections.

The number of user devices in the system, as well as the number of services to be supported, was supposed variable.

Three different scenarios were considered:

- Scenario #1, IoT deployments and Smart Cities: In these scenarios the device density is very high, so 250 user devices were contemplated in this scenario.
- Scenario #2, Residential areas: In this case, traditional users using smartphones and other similar devices have to be considered. The number of users is lower, so only fifty (50) user devices were included in the simulation.
- Scenario #3, Industrial applications: The device density in these scenarios is medium, so 120 user devices were considered to be consuming communication services in this case.

The total simulated time was two days (48 hours), for which an 18 hours simulation was required. Each simulation was repeated five times in order to compensate the possible random effects.

Using these simulation scenarios, two experiments were carried out.

In the first experiment, a simulation scenario is employed to evaluate how the proposed service management solution affects the performance of future 5G networks. Specifically, the most relevant key performance indicators in mobile networks, such as availability and configuration delay, are analysed. Obtained results are, later, compared to values generated by traditional mobile network architectures where virtualization technologies and K8s are not employed.

In the second experiment, the performance of the proposed solution was evaluated, calculating the response time of the proposed technology depending on the number of communication services to be managed.

In respect to the configuration of the proposed service management solution, all weighting parameters were defined to follow a uniform distribution.

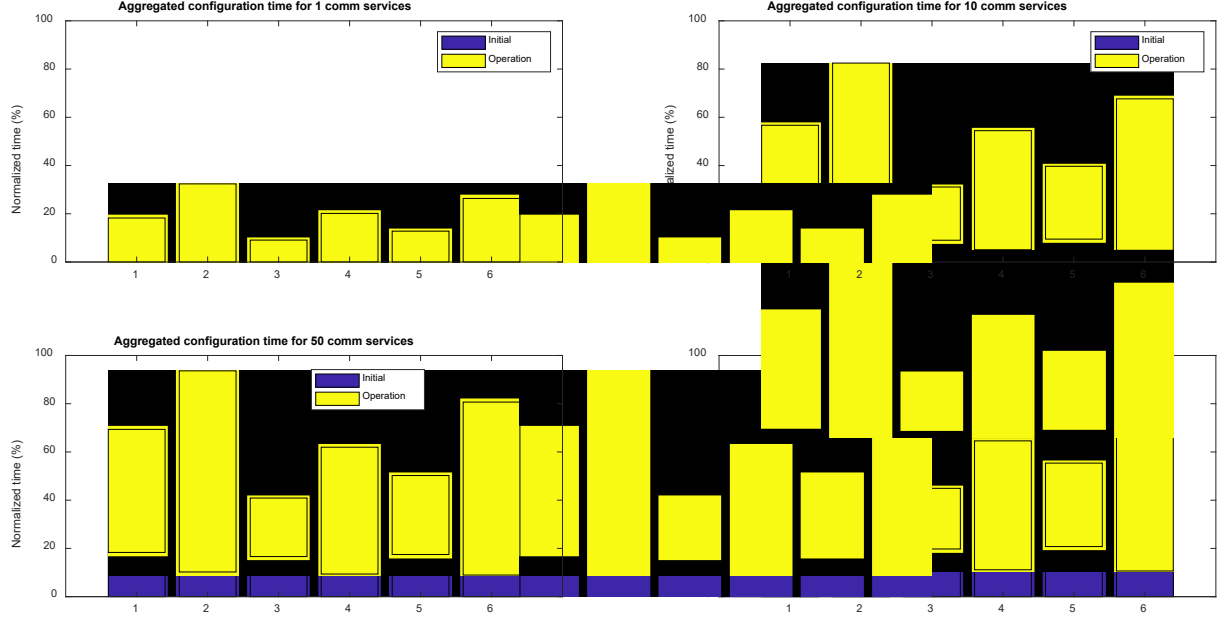


Fig. 9. Configuration time

5.2. Results

Figure 9 shows the accumulated configuration time, depending on the number of communication services being managed, and for the three different scenarios being evaluated. This configuration time is divided into two different amounts: the initial configuration time and the configuration time during operation. Initial configuration time (in blue colour) refers to the delay required to deploy all virtual instances, programs, dependencies, endpoints and, in general, software structures and components before being able to start the slice operations. Configuration time during operation (in yellow colour) refers to the aggregated delay along the whole considered operation period, required to manage all occurring network events, updates, new services to be offered, etc.

To facilitate the results analysis, time is normalized in Figure 9; i.e. all expressed results are relative values (percentage) with respect to the greater time obtained in the whole experiment.

As can be seen, the highest configuration time is always obtained for IoT scenarios, because of the complexity of hardware infrastructures, and the great

number of devices consuming communication services.

Another global important result that must be noted is the relation between configuration time in the traditional approach and in architectures implementing our proposal. As can be seen, the initial configuration time is slightly higher in the proposed solution (as many VM, containers, operating systems, etc. must be started). However, the aggregated configuration time required to maintain the system during the operation time is always much lower for our technologies, which makes that, globally, it presents better results.

As the number of communication services to be supported goes up, the initial configuration time also grows. When one hundred (100) services must be deployed, the initial configuration time represents around 20% of the total aggregated time. Besides, as can be seen, the evolution of this initial configuration time is logarithmic with the number of communication services. As can be seen, almost no variation is obtained when the number of services grows up from fifty to one hundred, and it decreases a 50% when the number of services is reduced in a magnitude order (an evolution pretty similar to a logarithmic function).

In conclusion, the proposed architecture and service management algorithm allows a faster and more

efficient (in time) deployment and management of communication services in 5G networks.

Any case, and although the following analysis does not affect the previous results, additional configurations could be necessary in real scenarios in order to address Docker and K8s vulnerabilities and preserve the entire 5G networks security. Between 2016 and 2018 up to eight different vulnerabilities were discovered in K8s technologies. These vulnerabilities include methods to gain privileges and deny services. Docker technology presents a similar behaviour (nine vulnerabilities were discovered between 2016 and 2018), but in this case some of them were detected after successful attacks (typically Denial of Service attacks). Therefore, in real scenarios, configuration delays could be slightly different as additional mechanisms to preserve network security such as new software patches will be required.

Figure 10 shows the temporal evolution of the accumulated configuration time in the worst case among the reviewed cases in Figure 9 (IoT scenario with one hundred services to be managed). To facilitate the analysis of the results, the ordinate axis has been normalized (results have been proportionally scaled to range between zero and the unit). Besides, collected data have been processed to generate a smooth curve.

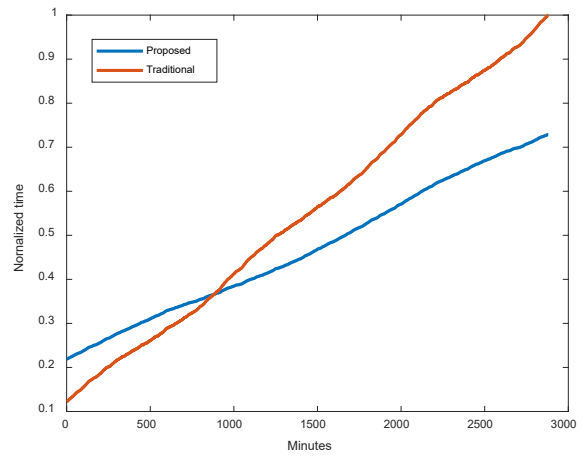


Fig. 10. Temporal evolution of the accumulated configuration time

As can be seen, after only 900 minutes (15 hours approximately), the ordinary management tasks generate an accumulate configuration time equal for both, the proposed and traditional service management solutions. From this point, the proposed technology offers a clear advantage. In other words, for systems whose lifetime is going to be higher than 15 hours (typical situation in standard mobile networks), the proposed approach is (at long-term) much more efficient. However, if the deployed system is going to have a shorter lifetime (a very unusual situation), the

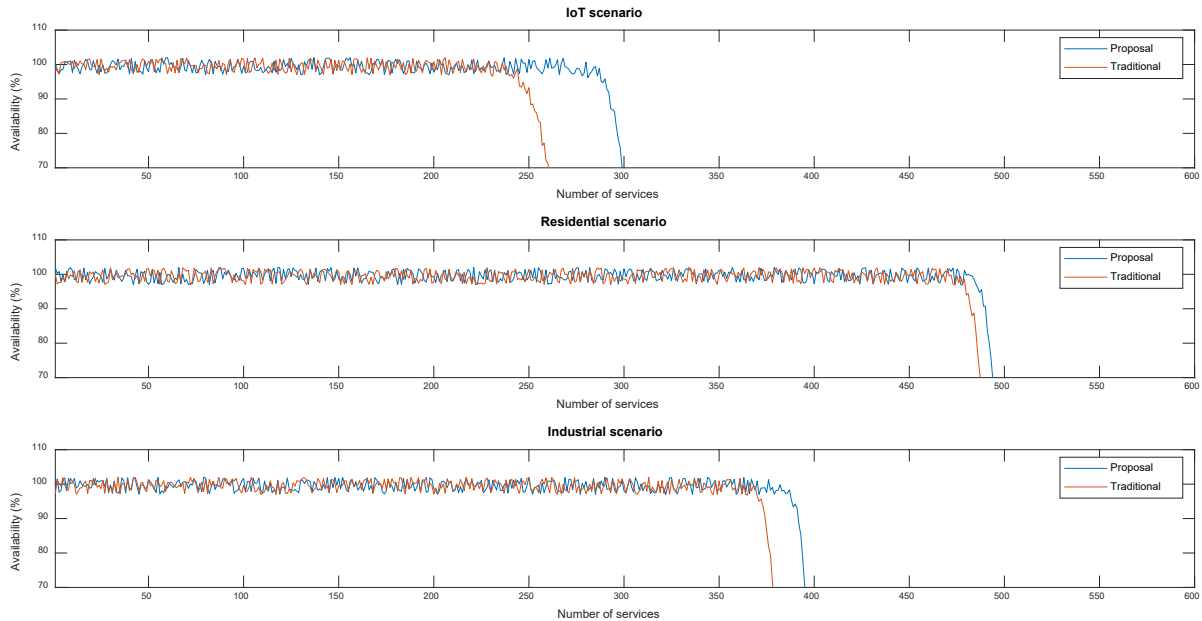


Fig. 11. Availability

required initial configuration time by the proposed approach makes traditional schemes more competitive.

5G service management is characterized by stringent delay requirements. Figure 10 shows the proposed mechanism is effective and met this objective. To reach this requirement the proposed technology employs the monitoring solutions included into K8s, which may deploy new instances and micro-services before the occurrence of fatal failures (thanks to prediction mechanisms and other similar instruments).

Figure 11 presents the results obtained from the availability analysis. In this work we are calculating the availability of the system as the ratio of (a) the total time the communication services (slices) are capable of being used during the considered operation interval to (b) the length of the interval.

As can be seen, for all scenarios, the proposed technology has a higher availability than traditional solutions, although for certain amounts of communication services both techniques offer similar results.

In general, both technologies present availability close to 100%, until the number of communication services goes above a certain limit. This limit is around 300 services for IoT scenarios, where the high device density makes more difficult to manage great amount of services. In this case, the proposed solution increases in 25% the results obtained from traditional techniques.

Both technologies are more similar for residential scenarios, where the limit is around 480 communication services for both solutions: traditional and the proposed technique.

Finally, for industrial scenarios, the proposed solution improves in a 5% the number of services being able to manage for a 100% availability (the limit is around 390 services). These results are coherent with previous discussions, where industrial scenarios have hybrid characteristics between residential and IoT scenarios.

In general, the proposed approach (where services are supported by K8s pods) increases availability when fails affect specific communication services or slices. Thus, regenerating the affected K8s pods is enough to maintain service availability. However, all services will become unavailable if the K8s master node fails and no additional protection policy has been applied. That is why in Figure 11, although availability increases, it does not reach 100% permanently. In order to protect the proposed scheme against fails in the K8s master node, several replicas of this virtual entity should be created.

Finally, Figure 12 shows the calculation time required by both service management algorithms proposed in Section 4.

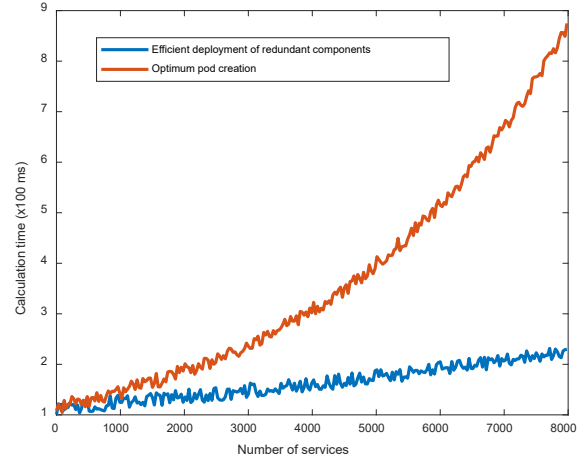


Fig. 12. Configuration time

The algorithm employed to calculate the optimum number of pod copies to be deployed follows a linear evolution, using 200ms to process near 8000 communication services. This is coherent with the use of summations, where scalability is also linear. However, as can be seen, proposed solution to create the optimum pods has an exponential complexity. In fact, Bell's number are also exponential, and the number of possible configurations to be reviewed present this distribution as the number of communication services (microservices) to be allocated goes up. Any case, for eight thousand communication services, less than one second is required to solve the mathematical problem.

It is important to note that this calculation time does not affect the end-to-end delay in communication services supported by slices. The mathematical problems to be solved by the proposed mechanism are addressed before network operation. Once systems are running, Docker and K8s instruments manage all virtual instances (VNF) building slices.

6. Conclusions

Emerging 5G networks are envisioned to provide communication services to a very large catalogue of applications: from traditional mobile phones, to future Cyber-Physical Systems and Industry 4.0 solutions. In general, these heterogeneous application scenarios present different Quality-of-Service requirements and a very dynamic lifecycle. In this con-

text, traditional homogeneous network architectures are very inefficient and new mechanisms to provide communication services have been proposed. In particular, 5G networks will be based on network slicing technologies. Network slicing is a technique where network resources are packet and assigned in an isolated manner to a set of users according to their specific requirements. Standard solutions such as virtual private networks, virtual network functions or circuit switching could support and manage network slicing, and services, in future 5G systems; however, these techniques do not fit with the automatism, isolation and dynamism characteristics required by future applications.

Therefore, this paper proposes a new approach for the implementation and management of services and slices in future 5G systems with network slicing, based on lightweight algorithms and virtualization technologies, such as Docker or K8s. The proposed solution is totally compatible with new 5G standards, proposed by the corresponding international organizations. Using this approach, important indicators such as availability, security and the configuration time are improved. The benefits of the proposed solution are evaluated using a simulation scenario.

Results show an improvement in relevant network indicators such as availability and configuration time. Briefly, systems and networks with a lifetime above fifteen hours would take advantage of the proposed mechanism, obtaining a much more efficient service management. Besides, the scalability of the proposed technology is evaluated by means of the calculation time it requires. Results show the proposed mechanism grows linearly with the number of deployed services, while other solutions evolve exponentially.

7. Future works

Future works will address the design of other pending components such as inter-slice broker. Besides, in order to improve the solution efficiency, advanced mechanisms to represent services and their lifecycle will be considered. For example, visibility graphs [48][49] could be employed to work with time series containing historical data about the services evolution. On the other hand, slices could be modelled as monoids of P systems [53] or spiking neural P systems with communication on request [58], what will enable the use of neural technologies to study the evolution of services.

Acknowledgements

The research leading to these results has received funding from the Ministry of Economy and Competitiveness through SEMOLA project (TEC2015-68284-R).

References

- [1] Bordel, B., de Rivera, D. S., & Alcarria, R. (2018, March). Virtualization-Based Techniques for the Design, Management and Implementation of Future 5G Systems with Network Slicing. In *World Conference on Information Systems and Technologies* (pp. 133-143). Springer, Cham.
- [2] Yang, K. (2014). *Wireless sensor networks. Principles, Design and Applications*.
- [3] Bordel, B., Alcarria, R., & Sánchez-de-Rivera, D. (2017, April). Detecting Malicious Components in Large-Scale Internet-of-Things Systems and Architectures. In *World Conference on Information Systems and Technologies* (pp. 155-165). Springer, Cham.
- [4] Bordel, B., Alcarria, R., Robles, T., & Martín, D. (2017). Cyber-physical systems: Extending pervasive sensing from control theory to the Internet of Things. *Pervasive and mobile computing*, 40, 156-184.
- [5] Sánchez, B. B., Alcarria, R., de Rivera, D. S., & Sánchez-Picot, A. (2016). Enhancing Process Control in Industry 4.0 Scenarios using Cyber-Physical Systems. *JoWUA*, 7(4), 41-64.
- [6] Zhang, H., Liu, N., Chu, X., Long, K., Aghvami, A. H., & Leung, V. C. (2017). Network slicing based 5G and future mobile networks: mobility, resource management, and challenges. *IEEE Communications Magazine*, 55(8), 138-145.
- [7] Hervás, O. R. (2014). *Software Defined Networking*. Universidad Politécnica de Cataluña.
- [8] Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, (3), 81-84.
- [9] Preeth, E. N., Mulerickal, F. J. P., Paul, B., & Sastri, Y. (2015, November). Evaluation of Docker containers based on hardware utilization. In *Control Communication & Computing India (ICCC)*, 2015 International Conference on (pp. 697-700). IEEE.
- [10] Bhardwaj, S., Jain, L., & Jain, S. (2010). Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of engineering and information Technology*, 2(1), 60-63.
- [11] Akyildiz, I. F., Wang, P., & Lin, S. C. (2015). SoftAir: A software defined networking architecture for 5G wireless systems. *Computer Networks*, 85, 1-18.
- [12] Rost, P., Banchs, A., Berberana, I., Breitbach, M., Doll, M., Droste, H., ... & Sayadi, B. (2016). Mobile network architecture evolution toward 5G. *IEEE Communications Magazine*, 54(5), 84-91.
- [13] Manvi, S. S., & Shyam, G. K. (2014). Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. *Journal of Network and Computer Applications*, 41, 424-440.
- [14] Foukas, X., Patounas, G., Elmokashfi, A., & Marina, M. K. (2017). Network slicing in 5G: Survey and chal-

- lenges. *IEEE Communications Magazine*, 55(5), 94-100.
- [15] Alliance, N. G. M. N. (2016). Description of network slicing concept. NGMN 5G P, 1.
 - [16] Alliance, N. G. M. N. (2016). 5G security recommendations package 2: Network slicing.
 - [17] Zhou, X., Li, R., Chen, T., & Zhang, H. (2016). Network slicing as a service: enabling enterprises' own software-defined cellular networks. *IEEE Communications Magazine*, 54(7), 146-153.
 - [18] Jiang, M., Condoluci, M., & Mahmoodi, T. (2016, May). Network slicing management & prioritization in 5G mobile systems. In *European wireless* (pp. 1-6).
 - [19] Siddhisena, B., Warusawithana, L., & Mendis, M. (2011, February). Next generation multi-tenant virtualization cloud computing platform. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on* (pp. 405-410). IEEE.
 - [20] Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., & Doyle, R. P. (2001). Managing energy and server resources in hosting centers. *ACM SIGOPS operating systems review*, 35(5), 103-116.
 - [21] Chiaraviglio, L., & Matta, I. (2010, April). Greencoop: Cooperative green routing with energy-efficient servers. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking* (pp. 191-194). ACM.
 - [22] Gupta, M., & Singh, S. (2003, August). Greening of the Internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (pp. 19-26). ACM.
 - [23] Nikaen, N., Schiller, E., Favraud, R., Katsalis, K., Stavropoulos, D., Alyafawi, I., ... & Korakis, T. (2015, September). Network store: Exploring slicing in future 5g networks. In *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture* (pp. 8-13). ACM.
 - [24] Banchs, A., Breitbach, M., Costa, X., Doetsch, U., Redana, S., Sartori, C., & Schotten, H. (2015, May). A Novel Radio Multiservice adaptive network Architecture for 5G networks. In *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st* (pp. 1-5). IEEE.
 - [25] Foukas, X., Nikaen, N., Kassem, M. M., Marina, M. K., & Kontovasilis, K. (2016, December). FlexRAN: A flexible and programmable platform for software-defined radio access networks. In *Proceedings of the 12th International Conference on emerging Networking EXperiments and Technologies* (pp. 427-441). ACM.
 - [26] Rosen, R. (2014). Linux containers and the future cloud. *Linux J*, 240(4), 86-95.
 - [27] Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turetli, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3), 1617-1634.
 - [28] Nadeau, T. D., & Gray, K. (2013). SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies. "O'Reilly Media, Inc."
 - [29] Hawilo, H., Shami, A., Mirahmadi, M., & Asal, R. (2014). NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). *IEEE Network*, 28(6), 18-26.
 - [30] Ordóñez-Lucena, J., Ameigeiras, P., Lopez, D., Ramos-Munoz, J. J., Lorca, J., & Figueira, J. (2017). Network slicing for 5g with SDN/NFV: concepts, architectures and challenges. *arXiv preprint arXiv:1703.04676*.
 - [31] Yousaf, F. Z., Loureiro, P., Zdarsky, F., Taleb, T., & Liebsch, M. (2015). Cost analysis of initial deployment strategies for virtualized mobile core network functions. *IEEE Communications Magazine*, 53(12), 60-66.
 - [32] Banchs, A., Breitbach, M., Costa, X., Doetsch, U., Redana, S., Sartori, C., & Schotten, H. (2015, May). A NOvel Radio Multiservice adaptive network Architecture for 5G networks. In *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st* (pp. 1-5). IEEE.
 - [33] Mishra, A. R. (2004). Cellular Networks. *Advanced Cellular Network Planning and Optimisation*, 1-13.
 - [34] Samdanis, K., Costa-Perez, X., & Sciancalepore, V. (2016). From network sharing to multi-tenancy: The 5G network slice broker. *IEEE Communications Magazine*, 54(7), 32-39.
 - [35] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239), 2.
 - [36] Hightower, K., Burns, B., & Beda, J. (2017). Kubernetes: Up and Running: Dive Into the Future of Infrastructure. "O'Reilly Media, Inc."
 - [37] Rose, R. (2004). Survey of system virtualization techniques.
 - [38] Helsley, M. (2009). LXC: Linux container tools. IBM developerWorks Technical Library, 11.
 - [39] Rosen, R. (2013). Resource management: Linux kernel namespaces and cgroups. *Haifux*, May, 186.
 - [40] Scheepers, M. J. (2014, June). Virtualization and containerization of application infrastructure: A comparison. In *21st Twente Student Conference on IT* (Vol. 1, No. 1, pp. 1-7).
 - [41] 5G PPP Architecture Working Group. (2018). View on 5G architecture. White Paper, July. <https://5g-ppp.eu/wp-content/uploads/2018/01/5G-PPP-5G-Architecture-White-Paper-Jan-2018-v2.0.pdf>
 - [42] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., & Shenker, S. (2008). NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3), 105-110.
 - [43] Mijumbi, R., Serrat, J., Gorricho, J. L., Latré, S., Charalambides, M., & Lopez, D. (2016). Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, 54(1), 98-105.
 - [44] Newman, S. (2015). Building microservices: designing fine-grained systems. "O'Reilly Media, Inc."
 - [45] Leitner, P., Cito, J., & Stöckli, E. (2016, December). Modelling and managing deployment costs of microservice-based cloud applications. In *Proceedings of the 9th International Conference on Utility and Cloud Computing* (pp. 165-174). ACM.
 - [46] Bingham, J. A., & Bingham. (1988). The theory and practice of modem design (Vol. 157). New York: Wiley.
 - [47] Guerrero, C., Lera, I., & Juiz, C. (2018). Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. *Journal of Grid Computing*, 16(1), 113-135.
 - [48] Ahmadlou, M., & Adeli, H. (2012). Visibility Graph Similarity: A New Measure of Generalized Synchronization in Coupled Dynamic Systems. *Physica D: Non-linear Phenomena*, 241(4), 326-332.
 - [49] Ahmadlou, M., Adeli, H., & Adeli, A. (2012). Improved Visibility Graph Fractality with Application for

- Diagnosis of Autism Spectrum Disorder. *Physica A: Statistical Mechanics and its Applications*, 391(20), 4720-4726
- [50] Carneiro, G. (2010, April). NS-3: Network simulator 3. In UTM Lab Meeting, 20, 4-5.
- [51] Zhang, J., & Qin, Z. (2011, April). Taprouter: an emulating framework to run real applications on simulated mobile ad hoc network. In *Proceedings of the 44th Annual Simulation Symposium*. Society for Computer Simulation International, 39-46.
- [52] Bordel, B., Miguel, C., Alcarria, R., & Robles, T. (2018). A Hardware-Supported Algorithm for Self-Managed and Choreographed Task Execution in Sensor Networks. *Sensors*, 18(3), 812.
- [53] Pan, L., Păun, G., Zhang, G., & Neri, F. (2017). Spiking neural P systems with communication on request. *International journal of neural systems*, 27(08), 1750042.
- [54] Sánchez-de-Rivera, D., Bordel, B., Alcarria, R., & Robles, T. (2019). Enabling Efficient Communications with Resource Constrained Information Endpoints in Smart Homes. *Sensors*, 19(8), 1779.
- [55] Rost, P., Banchs, A., Berberana, I., Breitbach, M., Doll, M., Droste, H., ... & Sayadi, B. (2016). Mobile network architecture evolution toward 5G. *IEEE Communications Magazine*, 54(5), 84-91.
- [56] Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015, March). An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pp. 171-172. IEEE.
- [57] Kristiani, E., Yang, C. T., Wang, Y. T., & Huang, C. Y. (2018, June). Implementation of an edge computing architecture using openstack and kubernetes. In *International Conference on Information Science and Applications*, pp. 675-685. Springer, Singapore.
- [58] Wu, T., Bible, F., Paun, A., Pan, L., & Neri, F. (2018). Simplified and yet Turing universal spiking neural P systems with communication on request. *International Journal of Neural Systems*, 28(8), 1850013.