# CDN-as-a-Service Provision over a Telecom Operator's Cloud

Pantelis A. Frangoudis, Louiza Yala, and Adlen Ksentini, *Senior Member, IEEE*

***Abstract*—We present the design and implementation of a Content-Delivery-Network-as-a-Service (CDNaaS) architecture, which allows a telecom operator to open up its cloud infrastructure for content providers to deploy virtual CDN instances on demand, at regions where the operator has presence. Using northbound REST APIs, content providers can express performance requirements and demand specifications, which are translated to an appropriate service placement on the underlying cloud substrate. Our architecture is extensible, supporting various different CDN flavors, and, in turn, different schemes for cloud resource allocation and management. In order to decide on the latter in an optimal manner from an infrastructure cost and a service quality perspective, knowledge of the performance capabilities of the underlying technologies and compute resources is critical. Therefore, to gain insight which can be applied to the design of such mechanisms, but also with further implications on service pricing and SLA design, we carry out a measurement campaign to evaluate the capabilities of key enabling technologies for CDNaaS provision. In particular, we focus on virtualization and containerization technologies for implementing virtual CDN functions to deliver a generic HTTP service, as well as an HTTP video streaming one, empirically capturing the relationship between performance and service workload, both from a system operator and a user-centric viewpoint.**

***Index Terms*—Content delivery networks, experimental approaches, cloud computing, network functions virtualization, network planning and service deployment, quality of experience, virtualization.**

## I. Introduction

Internet traffic is dominated by data distributed over Content Delivery Network (CDN) infrastructures, and the current Internet ecosystem is, to a significant extent, shaped by the interactions of different key actors with often conflicting objectives, including Over-the-Top (OTT) content providers, IP transit providers, CDN infrastructure providers and ISPs [1].

Content distributed over the top creates a bidirectional dependence between CDN and network providers: CDN customers, and, in turn, end users, depend on the underlying network infrastructure, and user experience is affected by its conditions. Network operators, on the other hand, are vulnerable to the traffic dynamics caused by time-varying shifts in content demand. At the same time, network providers wish to take advantage of their regional presence and proximity to end users to enter the content delivery market.

P.A. Frangoudis and A. Ksentini are with the Communication Systems Department, EURECOM, Sophia Antipolis, France (e-mail: pantelis.frangoudis@eurecom.fr; adlen.ksentini@eurecom.fr). L. Yala is with IRISA/University of Rennes 1, Rennes, France (e-mail: louiza.yala@irisa.fr).

This work was supported in part by the French FUI-18 DVD2C project. The work of P.A. Frangoudis was carried out while he was with IRISA/University of Rennes 1.

This has led to the emergence of telco CDNs, i.e., content delivery infrastructures owned and operated by network providers. In this case, the network operator installs data centers at its points of presence (or other points strategically located in its network) and offers a CDN service advertising high-performance delivery due to user proximity. From the perspective of a content provider, traditional and telco CDNs are not competitive, but, rather, complementary services: A telco has the advantage of proximity to end users, but typically this is limited to specific geographic locations where it has presence. In contrast, traditional CDN players compete offering a global service.

In this work, with cloud orchestration frameworks and Network Functions Virtualization (NFV) as our enabling technologies, we design a scheme which offers the flexibility to a telecom operator to lease its CDN infrastructure in a dynamic manner, offering a virtual CDN (vCDN) service that can be deployed on demand over the operator's private cloud. This can be considered as an evolution towards opening a telco CDN to potentially (but not exclusively) smaller-scale content providers. Our basic design goals are (i) to offer a well-specified, extensible northbound interface to customers, which will allow them to express service demand specifications and performance constraints, ensuring them sufficient control at the service level, but abstracting internal network and infrastructure details, and (ii) to be able to combine customer-provided demand dimensioning information (e.g., target number of users/video streams per region) with network and compute infrastructure awareness for optimal resource allocation. For the latter, we argue for a measurement driven approach: It is important to know the capabilities of the underlying technologies and be aware of the relationship between service workload and user experience in order to take informed resource allocation, placement, and real-time management decisions, maintaining the level of service agreed with the customer while minimizing operational costs; the experimental results we present in this work serve to this end.

Our work thus aims to provide the technical elements to build mechanisms for cost- and quality-optimized CDNaaS provision. These elements consist in the architecture support to develop algorithms and mechanisms for service delivery (resource allocation, service placement, and elastic resource/service management included) and in the necessary quantitative insight to guide them. We consider these mechanisms, however, as a separate line of research which deserves attention in its own right and do not focus on it specifically in this article. We make the following contributions:

- (Section III) We design and implement a cloud-based

architecture for the provision of a virtual CDN service tailored to telecom providers, and present results of testbed experiments from our prototype implementation on top of OpenStack [2]. Our design features REST APIs which enable content providers to lease virtual CDN resources on demand. We have designed our scheme with extensibility in mind, with the potential of applying a wealth of sophisticated dimensioning, resource allocation and elastic resource management algorithms, and being able to support different types of virtualized CDN services towards a generic Any-as-a-Service provisioning model.

- (Section IV) We carry out an extensive experimental campaign to explore the capabilities and limitations of the virtualization technologies we apply, quantifying the relationship between workload and performance from a system operator and a user perspective.
- (Section V) We propose potential uses of our results (i) to drive resource allocation/service placement algorithms that aim to optimally satisfy customer demand, and (ii) to provide insight to the system operator for SLA design.

## II. Background and related work

### A. ISP-CDN interactions and the emergence of telco-CDNs

In typical CDNs, content delivery functionality is replicated across a number of servers, potentially at a global scale. Content providers delegate the process of delivering their content to CDNs, and the latter select the most appropriate servers to respond to user requests for content items, balancing among maximizing user experience, minimizing delivery cost, and optimally managing CDN server load. CDN performance critically depends on the conditions in the underlying network path between the user and the content location, which however, being managed by network providers, is outside the control of the CDN. Without CDN-ISP collaboration, the CDN relies on estimates which are often not accurate. On the other hand, this network-unaware server selection process can cause unpredictable traffic shifts which can significantly impact the operation of the ISP network. Therefore, both ISPs and CDNs face a fundamental challenge, which is analyzed by Frank et al. [3] along with a thorough description of the technical issues and typical architectures for content delivery over the Internet.

Incentives for collaboration between ISPs and CDNs thus emerge and different strategies can be followed, from a non-cooperative case, where the CDN operates over the top in a network-agnostic manner, to the case where the ISP deploys and controls its own CDN infrastructure, which is the target environment for our work. This can happen either by developing its own content delivery solution, or by acquring a license to CDN software (licensed CDN), potentially sharing revenue with the CDN provider. Since our architecture implies a telecom-operated CDN, many of the technical and managereal challenges stemming from the need for ISP-CDN collaboration are alleviated, since the whole infrastructure is administered by a single entity. Other options involve CDN operators directly installing their servers inside the ISP network and/or setting up specific interconnection agreements.

Poese et al. [4] elaborate on mechanisms for the ISP to offer assistance to the CDN, without it being necessary for each party to reveal sensitive details about their internal workings. They propose enhancements to the ISP's DNS infrastructure in order to direct user requests to the most appropriate server, taking into account optimization criteria set by the ISP and aiming to improve user experience.

In this context, NetPaaS [5] provides the interfaces and the management tools to CDN operators to deploy their service in an ISP-assisted manner, at the same time offering informed user request to content server assignment. It operates for both physical CDN deployments (bare metal servers placed by the CDN in the ISP network) and virtual ones, if they are supported. Our approach assumes a different ownership model, i.e., the ISP is in full charge of the underlying infrastructure and the CDN service. Second, in our case, the customer is the content provider which requests the deployment of a CDN service in a Software-as-a-Service (SaaS) manner, while for NetPaaS the service offering is at the infrastructure or the platform level and the CDN is in full control of the virtual resources. Third, the resource allocation decision in our case is carried out fully by the service operator (ISP) based on measurement-driven empirical models. We should further note that our focus is more on the infrastructure support for CDNaaS provision, including an NFV-MANO-compatible architecture design (Section III-F), than on CDN-service-level details. As such, our system can support various CDN-level functionalities and can further expand towards supporting licensed CDN cases, where the service operator deploys and controls the licensed CDN software as a set of VNFs.

ISP-CDN collaboration models are analyzed by Herbaut et al. [6] using game-theoretic tools, identifying that this collaboration can be implemented on top of an NFV infrastructure and can be mutually beneficial under certain circumstances. In that model, the NVF platform is hosted by the ISP and forms a marketplace (see also the work in the context of the T-NOVA [7], [8] EU-funded FP7 project) where CDN operators can upload VNFs, which can in turn be chained by the ISP into different types of CDN services and under different pricing options. Business-wise, our approach is more on the telco CDN side, and, thus, more ISP-centric: The network provider fully controls both the NFV/cloud platform and the content delivery service. Technology-wise, although both appoaches share some common ground, we focus more on the technical aspects of such a scheme, providing a detailed architecture design and its implementation.

A driving force behind the deployment of telecom-operator-owned content delivery infrastructures is the fact that telcos aim at empowering their role in the content delivery value chain by exploiting their key advantages of network control and end-user proximity, a role that is traditionally challenged by over-the-top content delivery without their strong involvement. The economics of such vertically integrated CDNs are studied by Maillé et al. [9], who make some interesting observations on the incentives of an ISP to operate its own CDN: In the face of competition with other ISPs not operating a CDN, and depending on the price sensitivity of users, while from a user experience and a regulator perspective a vertically

integrated CDN is always beneficial, for their specific model settings there are circumstances under which the ISP has incentives not to run its own CDN. This implies that the decision of a network operator to run a CDN should be carefully studied. Although there is generally a clear business case for ISP-CDN integration, as also demonstrated by network providers rolling out their own CDN infrastructures, a study of such incentives and economic interactions between the involved actors is outside the scope of this article.

Despite some key advantages with respect to nework awareness, which can improve user experience and thus, potentially, the customer base, telco CDNs are limited by their regional coverage. A strategy to overcome this limitation is CDN federation, which is also a subject of the IETF CDN Interconnection (CDNI) Working Group [10]. Given the complementary competitive advantages of telco and traditional CDNs, Lee et al. [11] perform a game-theoretic analysis of the strategic interactions between the two types of players. Importantly, they study the conditions that can lead to alliances among telco CDNs and provide evidence that if a telco CDN properly manages to offer better service quality exploiting its competitive advantages (e.g., joint traffic engineering and content distribution), market benefits are possible. Telco CDN federation can take various forms. As Lee et al. show [12], there are cases under which the potential for full resource pooling and revenue sharing among the federation is beneficial, although in most cases resource pooling on its own brings more benefits to each individual telecom operator.

For an overview of some challenges, design goals and principles for a telco CDN, the reader is referred to the work of Spagna et al. [13].

### B. Data Center Virtualization

Our work is conceptually related with the Virtualized Data Center (VCD) abstraction [14]. In both cases, following a customer request, a set of cloud resources are allocated, operating in isolation on a shared infrastructure, potentially scaling dynamically. However, the VCD abstraction is closer to the Infrastructure-as-a-Service model, where the customer leases resources in the form of VMs with specific compute, storage and bandwidth guarantees, appearing as a dedicated (but virtual) data center for the execution of arbitrary platforms and applications. The vCDN mostly follows the SaaS model: The customer (content provider) requests for the instantiation of a virtual CDN infrastructure on top of a telco cloud for the diffusion of its content. As such, the infrastructure- and platform-level functionality is managed by the service operator. The content provider is agnostic to the infrastructure and potentially even the platform being used. We should note that at the data center network level, the necessary network virtualization functionality needs to be in place in order to offer the appropriate traffic isolation and network resource provisioning in a naturally multi-tenant environment. For a survey on data center network virtualization technologies, the reader is referred to the work of Bari et al. [15].

### C. Network Functions Virtualization

Network Functions Virtualization (NFV) is becoming a key technology for future large-scale service delivery [16]. NFV involves carrying out in software networking tasks that were traditionally performed by costly, special-purpose hardware. It is facilitated by developments in cloud computing and networking technologies, which have helped run such functionality on top of commodity hardware, offering on-demand scaling and automatic reconfiguration capabilities, decoupling the necessary service logic from the underlying hardware, and at the same time allowing (i) network operators to deploy and manage services with more flexibility and reduced capital and operational expenses, including space and energy costs, (ii) third-party application/service providers to innovate the market, lowering the technological barriers stemming from compatibility issues, and (iii) telecom equipment vendors to focus on the functionality of their solutions and to expand their service portfolio.

NFV is being applied to a diverse set of functions. In our case, the basic components of our architecture (service orchestration, virtual infrastructure management, etc.) and the components of the CDN service (caches, load balancers, name servers, etc.) are implemented as VNFs.

It should be noted that intense NFV standardization efforts are currently underway. The European Telecommunications Standards Institute (ETSI) has specified a Management and Orchestration framework for NFV (NFV-MANO [17]), and one of the proposed NFV use cases is the provision of virtualized CDN services [18, Use Case #8]. We put our design in the context of NFV-MANO in Section III-F.

One of the challenges in NFV is appropriate function placement. Clayman et al. [19] propose a VNF management and orchestration framework and experiment with different virtual router placement algorithms. In a similar spirit, Moens and De Turck [20] present a theoretical model for VNF resource allocation, focusing on an environment where a base load is handled by physical hardware, and virtual resources are utilized on demand to deal with load bursts.

As pointed out by Wood et al. [21], the flexibility offered by NFV comes with a performance cost due to virtualization. They therefore identify the need for a carefully designed NFV platform, coupled with a sophisticated SDN control plane. Our work addresses such challenges, by experimentally quantifying the performance capabilities of core enabling virtualization technologies, and by offering expressive management and control interfaces to customers and flexible service representations, which enable to focus on optimal virtual service construction considering customer and operator cost, performance, and quality constraints.

### D. Our prior work

This article significantly extends our prior work [22], where we introduced an early version of our scheme. Here, we present a more advanced modular architecture design and implementation, and the experimental evaluation of our prototype on a small-scale cloud testbed. We further present an extended performance comparison of key virtualization technologies to

support CDNaaS, both from a user-centric and an operator-centric viewpoint, and propose diverse uses of our results.

## III. A framework for CDNaaS provision

### A. Features

We present an architecture which allows a network operator to virtualize its CDN infrastructure and lease it to content providers on demand. Our approach can be viewed as an evolution of the telco CDN model, offering the operator the flexibility to provide lower-cost CDN solutions. These could appear more attractive to small-to-medium content providers, but other options are possible. For instance, the virtual CDN instantiated by a customer can be coupled with other CDN infrastructures; the flexibility offered by our design allows the customer to, e.g., use the leased infrastructure to respond to predicted traffic surges at specific regions, taking advantage of the network operator's regional presence. From the perspective of the CDNaaS provider, our design allows for more efficient use of its infrastructure resources, compared to a less dynamic resource reservation model with static allocation of data center resources to customers. Our approach can also operate transparently over federated clouds, which can be a strategy that telecom operators may follow to address potential limitations due to restricted geographical coverage.

### B. Architecture

Our design involves various functional blocks, communicating via well-specified interfaces. This decouples their operation from any physical location, allowing the CDNaaS provider to execute any of these blocks autonomously as virtual functions over its own (or any) cloud infrastructure. Our layered and modular design also aims to abstract the details of the underlying cloud infrastructure and thus avoid lock-in to a specific cloud management platform. One of our core design principles is to expose open APIs to customers, but also among the components of our architecture, and to design with extensibility in mind, so that our scheme can be extended towards a generic Any-as-a-Service model. In this section we provide the main components of our CDNaaS architecture (Fig. 1), their functionality and their interactions. An early version of our design was presented in our prior work [22].

*1) Customer Interface Manager:* Our system provides a RESTful northbound API, through which customers can request to deploy a vCDN over the telco cloud. This API exposes information on the supported vCDN service types and the regions the operator has presence, abstracting information about the underlying network and cloud infrastructure, and mediating the communication between customers and the Service Orchestrator (SO). Using the northbound API, a customer can specify its service requirements per region that it wishes to cover, and in particular (i) demand specifications, i.e., how many clients it wishes to serve per region, and what is the service lease duration, and (ii) quality specifications, which can be considered as specific Service-Level Objectives (SLOs), such as target Quality of Experience (QoE) ratings, desired response times, target availability levels, etc. After a service has been successfully deployed, this API returns an entry point to the vCDN instance so that the customer can manage its deployment (e.g., infuse content in the vCDN).

*2) Service Instance Descriptor Repository (SIDR):* Each service supported by our scheme has some inherent requirements and constraints. These are encoded in a service instance template, which provides information such as the minimum number of VNF instances of a specific type that need to be deployed for a vCDN service instance, constraints with respect to the processing and memory resources that should be available to each virtual machine (VM), etc. Although we mainly focus on the provision of a virtualized CDN service, our design includes an extensible service template description language which can support further services. The SIDR component stores these service instance templates and provides them to the Service Orchestrator to drive service deployment decisions.

*3) Service Orchestrator (SO):* The Service Orchestrator coordinates vCDN service deployment. After receiving a request from the customer, it derives an appropriate resource allocation and VNF placement, taking into consideration (i) the inherent service requirements, as expressed in the service instance template, (ii) the client demand and quality specifications, included in the service request, and (iii) its operational capacity. Depending on the specific vCDN flavor[1] selected by the customer, the appropriate algorithms are executed, resources are reserved, and the instantiation and termination of the vCDN are scheduled for the desired times, as requested by the customer. The result of the execution of these algorithms is a Service Instance Graph (SIG), which maps VNF instances (VMs) to physical nodes. The SIG is then passed on to the Resource Orchestrator (RO) for deployment.

*4) Resource Orchestrator (RO):* The RO component is responsible for the deployment of a service instance on the underlying telco cloud. The cloud infrastructure is managed by the *Virtualized Infrastructure Manager (VIM)* component, in the ETSI NFV-MANO [17] terminology. In our implementation we are using OpenStack, the de facto VIM platform. The RO receives the SIG that the SO derives following a customer request, and uses the northbound VIM API to launch and configure VNF instances. Note that the RO takes care of specific technical details, such as IP address management for the deployed instances, but also implements the necessary service-level logic (e.g., the boot-time configuration of DNS servers to appropriately handle geolocation or the setup of cache virtual instances to proxy user requests to origin servers); this logic may differ across the various vCDN flavors available from the operator. Since the RO is the only component which directly interacts with the telco cloud, the other components of our design are agnostic to low level platform interfaces and technical details. Changes at the infrastructure level need only be reflected in the RO.

*5) Instance Manager (IM):* There are specific management tasks which are common across any vCDN flavor, such as

[1]We define a vCDN flavor as a specific vCDN type available to customers, which comes with its own internal operation, service-level components (e.g., video streaming servers, caches, DNS resolvers, load balancers, etc.), resource allocation algorithms, and service management and monitoring features.

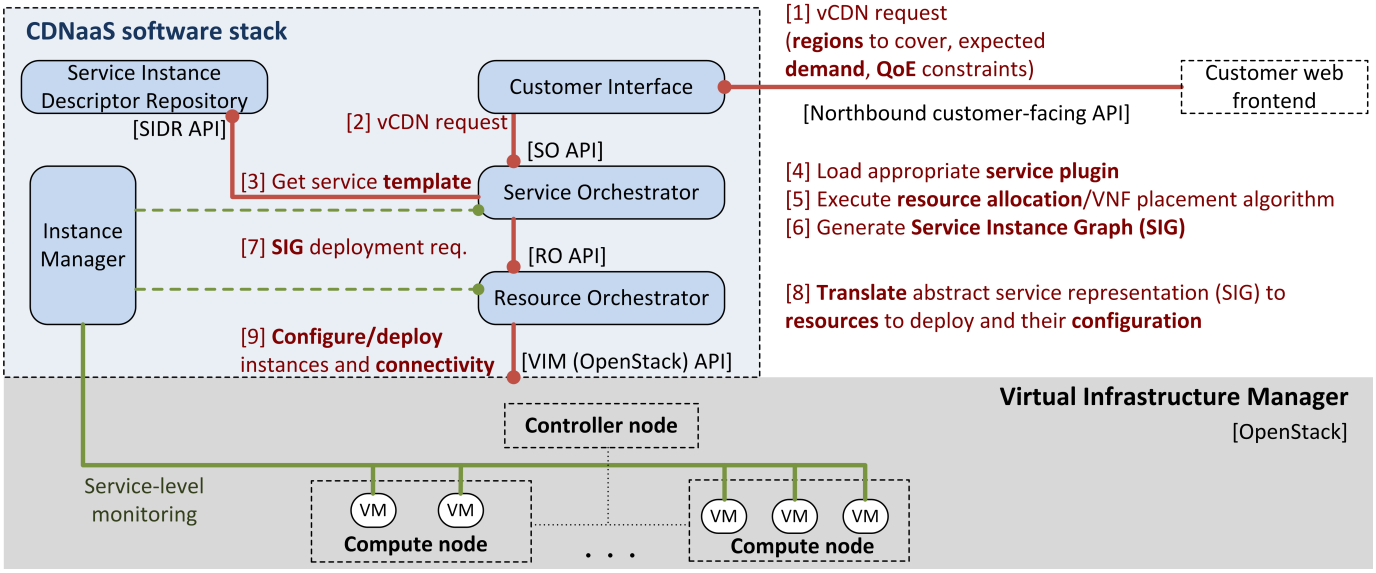



Fig. 1. CDNaaS architectural components and service life cycle.

vCDN instantiation and termination. However, more elaborate critical functions need to be performed in a different way for different vCDN types. This functionality is carried out by the IM and pertains to both the SO and the RO layers. For example, complex dynamic resource management algorithms can be implemented here, which use the SO and RO APIs to scale up or scale down a vCDN service instance when deemed appropriate. Such functionality requires real-time service-level monitoring, which is also performed by the IM.

### C. vCDN service-level components and life cycle

In our reference vCDN deployment scenario, a video content provider operates its origin server(s) in its premises or in external public or private clouds, where it places original video content. Then, it uses the customer-facing API to request the instantiation of a vCDN covering specific regions where the operator has presence, to serve a target maximum number of parallel video streams (demand specification) with the desirable video QoE (quality specification) for a given period.

The vCDN instance that will be created will include a number of caches distributed across the operator's regional data centers. The vCDN service supports a two-level load balancing: A user's DNS request for the URL of a content item is resolved to the appropriate regional data center, based on the user's geographic location, as inferred from the latter's IP address (DNS geo-location). Then, HTTP requests are balancing across all cache VNF instances deployed in a regional data center.

Upon receiving the customer request, the SO runs an algorithm to calculate (and reserve) the resources necessary and a VNF instance placement on the appropriate regional data centers in the form of a SIG, and uses the RO API to request its deployment on the provider's infrastructure. The RO, which is responsible for translating the abstract service representation (SIG) to an actual deployment, automatically configures one or more DNS VNF instances for request geolocation, configures cache VNF instances so that they proxy all user requests towards the content origin server(s), and configures each region's load balancers. Eventually, the customer entry point, with specific information on the vCDN instance (such as the IP address of the DNS server) is included in the API response.

During the operation of the vCDN instance, it is monitored at the service level and reconfigured if necessary (e.g., scaling up the allocated CPU resources to cope with increased demand to maintain the desired QoE level) by the IM. The vCDN service is terminated and the respective resources are released either automatically, when the lease specified in the customer request expires, or when the customer explicitly requests its termination using the northbound API.

### D. Implementation

*1) Technologies:* As a proof of concept, we have implemented our CDNaaS architecture on top of OpenStack. VNF instances (caches, DNS servers, etc.) are executed as Debian Linux virtual machines on kvm [23] hypervisors (compute nodes, in the OpenStack terminology). We are using *nginx* [24] to implement HTTP server/caching functionality, a choice motivated by its wide adoption.

The components of our scheme (customer interface, SIDR, SO, RO) have been implemented in Python and communicate over HTTP by exchanging JSON-encoded information. Service templates and service instance graphs are also representd as JSON objects. Our RESTful northbound API has also made it straightforward to create a web-based customer front end.

*2) Service plugins:* To deal with the different requirements of the various vCDN versions offered by the operator and to ensure the system's extensibility, we introduced the notion of the *service plugin*. Our software architecture provides a plugin interface which exposes hooks for its main components, where the funtionality pertinent to each one if them is to be implemented. In particular, for each vCDN type, a service

plugin needs to be developed, with the following functionality per component:

**Service Orchestrator:** The resource allocation and VNF placement algorithm for the specific vCDN type, the output of which is a SIG, is specified here. The SO plugin functionality is also responsible for reserving the resources that will be needed for the vCDN service instance, if these are available.

**Resource Orchestrator:** Using the RO API, the SIG is deployed over the underlying cloud infrastructure. The plugin is responsible for the service-level functionality, i.e., appropriately configuring VNF instances at boot time.

**Instance Manager:** After successful service deployment, the SO starts an instance management function, which carries out the appropriate monitoring tasks and executes dynamic resource management algorithms. Elasticity functions are implemented here. For example, for a video streaming service, the Instance Manager method of the service plugin may periodically monitor the number of the active HTTP connections of each deployed cache instance to estimate QoE; if it detects that there is a risk of not being able to maintain customer quality constraints, it can invoke the appropriate SO and RO API calls to scale up the service by allocating more compute resources.

Therefore, to offer a new type of service, the following steps are necessary for the system operator:

1) Create the appropriate virtual machine images and register them with the OpenStack image service.
2) Implement and install the service plugin (in our case, plugins are automatically loaded by the SO component).
3) Create a service template and register it with the SIDR component using the SIDR API.

### *E. vCDN flavors*

A content provider can retrieve a list of the supported flavors by accessing the service catalogue via the CDNaaS customer-facing API. Each flavor is implemented by a service plugin. We note that our prototype already supports various vCDN types, with different operational characteristics (e.g., hosting content origin servers in the vCDN vs. outside it, in other public/private clouds or in the content provider's premises) and resource allocation schemes. These schemes range from the simple unelastic case where a fixed number of resources (specified in the service template) is deployed, only aiming to have presence at all regions requested by the customer and without considering service quality, to a more sophisticated QoE-aware elastic service. In the latter, the compute resource allocation and VNF placement algorithm implemented in the plugin aims to guarantee customer QoE constraints [25], constantly monitoring service workload and scaling up/down resources to match current end-user demand and minimize operator cost. The details of such mechanisms are outside the scope of this article. We should note that the possibilities for further CDN flavors are numerous, and could also even include licensed CDN software, deployed on demand by the operator over the telco cloud/NFV infrastructure with the appropriate amount of resources allocated by the CDNaaS service orchestrator given the expressed end user demand.

### *F. Our framework in the context of the ETSI NFV-MANO specification*

Our design is in line with the ETSI Network Functions Virtualization-Management and Orchestration (NFV-MANO) [17] spirit and best practices, and the components of our architecture map to functional blocks of the MANO specification: The functionality provided by the SO and the RO is part of the Virtual Network Function Orchestrator (VNFO) component in NFV-MANO, while we use OpenStack as the de facto Virtualized Infrastructure Management (VIM) suite. IM, in turn, maps to the VNF Manager (VNFM) MANO component, and SIDR implements functionality that would be the responsibility of the VNF Catalogue component.

It should be noted that NFV-MANO allows for various different architectural and functional options [26, §6.3.2, §6.4.1]. We opted for a split NFVO functionality, where the RO is a separate architectural component which operates as the sole proxy between the VIM and the SO or IM instances, "hiding" the details of the VIM northbound interface. This approach has two advantages in our case. First, by decoupling the NVFO and RO operations, it facilitates future extensions of our scheme towards multi-administrative-domain cases, where the CDNaaS operator can also lease resources on cloud infrastructures managed by other providers (each with their own ROs, potentially managing heterogeneous VIMs), still itself controlling service orchestration. Second, by mediating the communication between SO/IMs and the VIM (*indirect mode*), our design avoids being locked to a specific VIM framework; the underlying technology can change, but all software components except for the RO will be unaffected.

### *G. Service deployment time*

We present results on the time it takes for a vCDN service instance to be deployed and be operational. Our results are useful for having an estimate of how long it would take for the customer's content (e.g., video steams) to be initially available to its end users, as a function of the size of the vCDN deployment. We also attempt to identify the main factors influencing service instantiation time, which can drive future efforts on fine tuning and optimizing the performance of our CDNaaS scheme (e.g., by developing algorithms for VM image placement across an operator's data centers).

We set up a cloud testbed with two compute nodes; the cloud controller was collocated with one of the two. We hosted our CDNaaS orchestration services (customer API, SO, SIDR, RO) on a separate computer, from which the necessary calls to the OpenStack API following a vCDN deployment request were performed. To evaluate the effects of the network conditions on service deployment times, we configured the mean round-trip times in the controller-compute and API client-controller paths to $d$, using the Linux `tc/netem` utility. Our testbed setup is shown in Figure 2. In our example setup, following a customer request, a vCDN is deployed with the following service-level components:

- A number (two, in our example) of web proxy/cache VNF instances per compute node (each such node represents a regional Data Center). One load balancer per region is

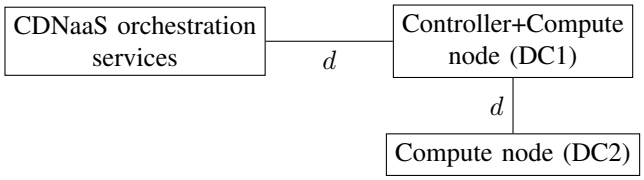



Fig. 2. Testbed configuration. An OpenStack setup with two compute nodes was created. The OpenStack controller was collocated with one of them. A separate machine was hosting our CDNaaS orchestration services (customer API, SO, SIDR, RO), over which we carried out vCDN service deployment requests. The average RTT in each path was configured to $d$.

responsible for evenly distributing HTTP requests among region-local VNF instances.
- One name server VNF.

The *kvm* [23] hypervisor is used to host guest VMs. Each proxy/cache VNF instance is created from a 1.6 GB Debian VM image running *nginx* [24], configured at boot time to act as a reverse proxy to an origin server hosted at the customer premises (its IP address and web server port are provided by the customer over the northbound API).

The time to deploy this service is determined by the following factors:

- The time taken to boot and configure the necessary VNF instances, as well as the other service components, such as load balancers.
- The signaling overhead due to the message exchanges between the controller and the compute nodes and between the OpenStack API client (in our case, the RO component) and the controller.
- The time to transfer the VNF images from the image store to compute nodes.

Therefore, significant parameters that influence the service deployment time are the delay in the paths between the entities of our architecture, the size of the VM images to transfer, and the number and amount of traffic of API calls towards the underlying cloud infrastructure that need to be carried out.

OpenStack allows VM images to be cached locally at compute nodes, which can significantly speed up deployment time. However, there could be cases where an image is not available locally at boot time (e.g., if this functionality is for some reason unavailable, or if it is the first time a specific VM is to be launched on a specific host). Our experiments demonstrate that, in such cases, service deployment time is dominated by the time to transfer the image.

Fig. 3 presents the time it takes for our example vCDN deployment to become fully operational in a single data center. In this case, for each DC, two proxy VNFs are launched and the load balancer is configured. We compare deployment times for the case where the compute node is collocated with the controller and the image store (local DC) vs. the case for a remote compute node deployment (remote DC), for various link RTT values and when image caching is enabled or not. When image caching is disabled, the deployment time is dominated by the time to transfer the 1.6 GB image to the remote host. We also observe that even when launching a VM at a host collocated with the image store, deployment times without image caching are noticeably increased. Finally, increased RTT values in the two links also contribute significantly to the perceived delay, since they affect both signalling (e.g., OpenStack API calls) and data (image transfer) traffic. As expected, the effect of the RTT is more important for a remote DC deployment, more than doubling total delay when per link RTT increases from 100 ms to 500 ms.

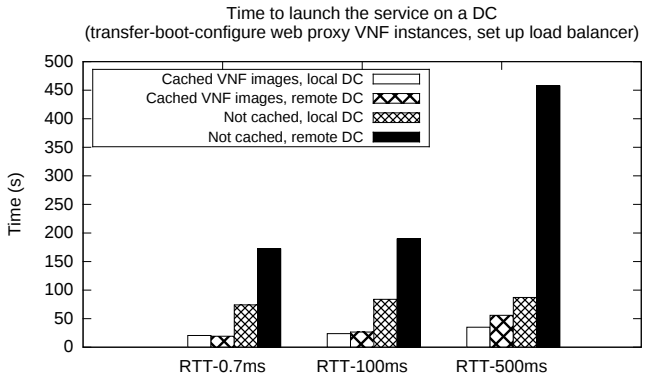



Fig. 3. Time until a vCDN deployment on a single data center becomes fully operational, under different experimental settings.

In an ideal CDNaaS implementation, the VNF instances per region would be launched in parallel and the SO/RO would monitor their status until they become active. Then, each instance would be registered with the local load balancer. Since OpenStack does not allow such registration requests to be carried out in parallel, we estimate the time it takes for the VNF instances in a specific host/region to become operational as the maximum time it takes for a VM to boot (since they are instantiated in parallel), plus the time it takes for all VMs to be sequentially registered with the load balancer. To evaluate the effect of the number of VNF instances per DC, we first measured the time it takes for a single load balancer member registration request to complete under different RTT conditions, and then estimated the time it takes to deploy the service on a local or a remote DC as the number of such VNF instances increases. Since the time to register a proxy/cache VNF instance with the load balancer ranges from 1.5 s (when RTT = 0.7 ms) to 2.4 s (for RTT = 500 ms), we see a small but noticeable increase in deployment times, as the number of instances grows. Fig. 4 and Fig. 5 present these results when image caching is enabled and disabled, respectively.

## IV. Enabling technologies and their performance

A critical aspect of CDNaaS provision is appropriately balancing between service quality and operational cost. Our proposed design, which offers customers the option to target specific levels of user demand and define end-user QoE thresholds, forces the system to operate under customer (quality) and operator (capacity) constraints. Therefore, resource management (and, in turn, pricing-related) decisions need to be taken with awareness of the capabilities and capacity of the underlying cloud and virtualization technologies.

In the direction of offering the necessary insight to the CDNaaS operator for such decisions, we carry out extensive

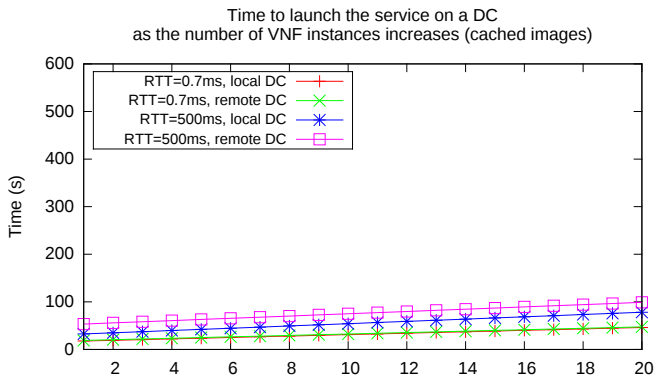



Fig. 4. Time until a vCDN deployment in a single data center becomes fully operational as a function of the number of VNF instances to be launched, when VM images are already cached.

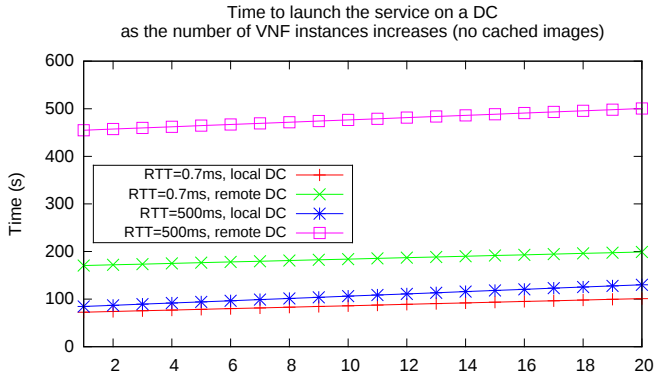



Fig. 5. Time until a vCDN deployment in a single data center becomes fully operational as a function of the number of VNF instances to be launched, when VM images are not cached. For the case of a remote data center, image transfer times dominate the overall service instantiation time.

measurements to quantify the performance of key enabling technologies and its relationship with service workload.

Service performance has many factors affecting it: Network capacity and conditions, resources (CPU, memory, storage) allocated to the service, current demand (concurrent number of users accessing the service), the specific virtualization technologies and their configuration, the software used to implement a service, the cloud platform, etc. Our study is *compute*-oriented: Of our particular interest is to derive the relationship of virtualized service performance vs. workload on a single virtual CPU (vCPU), our unit of processing (or, otherwise put, our unit of scaling), which in our experiments corresponds to *a single CPU core* of a physical host.

Managing network resources (guaranteeing their availability per vCDN instance, ensuring traffic isolation, dealing with transient traffic variations, etc.) is critical and challenging, however outside the scope of this article. We assume that the network operator who is in charge of the CDNaaS architecture has full control of the network infrastructure and full awareness of its conditions.[2] As such, it can appropriately provision the network paths between the virtual instances and end users and perform the necessary traffic engineering tasks, tackling network-related issues independently of compute resource allocation. It should also be noted that given a specific level of demand (e.g., in terms of the maximum number of parallel video streams), deciding on the amount of network resources necessary is more straightforward. However, a measurement study like the one we present here is needed for optimizing the allocation of compute resources in a QoE-aware manner.

Our results can be interpreted under two perspectives: the operator perspective, where the focus is on the performance bounds of the technologies used (e.g., server request throughput), and the user perspective, where our metrics of interest are QoE-oriented (e.g., response time, video viewing experience). For this purpose, we experiment with a generic HTTP service, but also with an HTTP video streaming application.

### A. Virtualization technologies and testbed configuration

We perform a comparative study of two candidate technologies to implement VNFs, each with different characteristics, namely virtualization and containerization. In the first case, we use virtual machines on top of the *kvm* hypervisor, and, in the second, *docker* containers [27]. Note that both technologies are supported by OpenStack, our cloud computing software of choice. For a comparison of the features and the internal workings of the two technologies the reader is referred to the work of Dua et al. [28]. Felter et al. [29], on the other hand, present a thorough experimental evaluation of their associated overhead for specific applications.

We execute our tests on an HP Z800 workstation with a 16-core Intel Xeon processor and 32 GB of RAM, running Ubuntu 14.10. We are benchmarking the performance of the popular *nginx* HTTP server, which is the technology we are using in our proof-of-concept CDNaaS implementation. We have carefully tuned nginx and the operating system (both host and guest) for high performance, and in order to alleviate network and I/O bottlenecks. Nginx works by spawning a number of worker processes, which handle user requests. The optimal strategy is to launch one worker per CPU core available. To deal with large numbers of concurrent users, we increased the maximum number of allowed concurrent connections per worker (and the respective operating system limits on the number of open file descriptors), and set the `tcp_tw_reuse` operating system option, to allow for reusing sockets that are in the `TIME_WAIT` state.

One of the aspects that we wish to quantify is the scalability of the service as a function of the CPU resources available. Intuitively, this should scale linearly with the number of available vCPUs (cores); our intuition is experimentally verified. To achieve the appropriate level of isolation, for each of our experiments, we pin the server VM/container and the load generation tool to separate core sets using the `taskset` utility, and give the highest priority to the respective processes using the `nice` command.

To benchmark our server, we use the *weighttp* tool [30]. Being multi-threaded, it takes advantage of the availability of multiple CPU cores, ensuring that the HTTP traffic generator does not become the performance bottleneck.

[2]This is a realistic assumption made in all other works that deal with the role of the ISP in CDN provision, even if the ISP is not in charge of the CDN itself; for more details, see [3].

TABLE I
STARTUP TIMES FOR A KVM DEBIAN IMAGE AND A DOCKER CONTAINER

| | Mean (s) | 95% confidence interval |
|---|---|---|
| kvm | 11.489 | (11.178, 11.8) |
| docker | 1.129 | (1.124, 1.135) |

Before the experiments, we also verified that network I/O is not the bottleneck. For the experiments using kvm, we activated the *vhost-net* module, which enables in-kernel handling of packets between the guest and the host, reducing context switching and packet copying overheads. By using two virtual Ethernet interfaces on the guest and connecting each one of them to a tap device on the host, we measured (using *iperf*) a 30 Gbps aggregate host-to-guest TCP throughput, enough to saturate our HTTP server. For docker, to avoid the overhead associated with Network Address Translation (NAT) in host-to-guest communication, we launched containers using *host*-mode networking. Thus, containers have direct access to the host networking stack. We also experimentally verified that disk I/O was not a bottleneck, since the file downloaded by the load generator sessions was served from the disk cache. We also repeated our experiments with the web server configured to serve files from a memory-mapped tmpfs file system, and our results were unaffected.

### B. Startup time comparison

One of the advantages of container technologies is that they are more lightweight compared to VMs. In contrast, to start an HTTP server/cache VNF instance hosted in a VM typically requires booting a full operating system. By startup time we define the time interval from launching a VM/container until it is capable of responding to HTTP. To measure it, we start the VM/container and simultaneously scan the TCP port the web server listens to using *nmap* [31] from the host, until the port is detected open (i.e., the HTTP server is ready). Table I presents mean startup times for a kvm Debian image and a docker container (average of 50 experiments, 95% confidence intervals). We notice that the startup time for kvm was approximately 11.5 s, while for docker it was an order of magnitude smaller.

It should be noted that in a cloud environment, booting a VNF instance also involves transferring the VM image or container from the image store to the actual host(s) where it will be executed. As we have shown in Section III-G, if image caching is not enabled, this can have a significant effect on the time to launch a vCDN instance.

### C. Performance of a generic HTTP service

We then experimentally study HTTP performance under different settings using the following metrics: (i) the HTTP/caching server's request throughput, i.e., the number of requests per second that it can process, and (ii) its response times.

*1) Request throughput:* An important aspect of HTTP server performance is the rate at which it can serve user requests. Being able to estimate the request throughput of a VNF instance with specific processing characteristics allows the CDNaaS provider to calculate the number of instances to deploy to cater for the demand of a specific customer, and appropriately respond to demand dynamics by up/down-scaling the deployment. Request throughput is a function of the processing capabilities of the server, its software architecture, the number of parallel users accessing the server, the size of the files to serve, and its disk and network I/O capacity.

We carried out a set of experiments to measure how our nginx-based HTTP server/cache VNF scales with the CPU processors available and the number of concurrent users. In our first test, to study these two properties in isolation, we minimized the effect of disk and network I/O by having HTTP clients request a special URI which corresponds to a single-pixel transparent GIF (the response body amounts to 43 bytes); this is a resource embedded in the HTTP server and causes zero disk I/O. We then emulated parallel users by varying the number of concurrent HTTP sessions on the client side (weighttp). Each emulated user constantly performed HTTP requests and we measured the HTTP request throughput when 1 or 2 CPU cores were allocated to the server. The results of this test are shown in Fig. 6. We notice that request throughput scales roughly linearly with the number of CPU cores.[3] Interestingly, we notice that docker achieves approximately 10K requests/s more than kvm for large numbers of parallel users. Each point in the figure represents a request throughput value calculated from the execution of 100M requests.

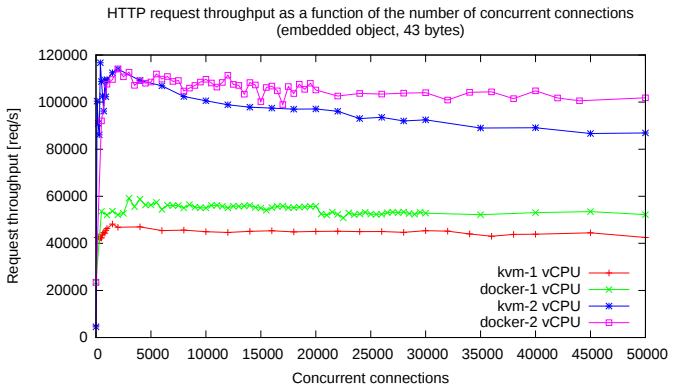



Fig. 6. HTTP request throughput for increasing numbers of parallel connections. We compare the use of docker and kvm and the impact of utilizing more CPU resources.

We further experimented with HTTP object sizes which correspond to a video service. To select realistic such sizes, we encoded a 720p HD video[4] using H.264/AVC [33], and prepared it for Dynamic Adaptive Streaming over HTTP (DASH) [34] delivery using the DASHEncoder [35] tool. Under DASH technologies, the client receives a Media Presentation Description (MPD) file, which includes information on the available representations (different qualities) of the same video, which is segmented in chunks. It then proceeds

[3]Although the figures show only the case for 1 vs. 2 vCPUs, our experiments with higher numbers of vCPUs demonstrate the same linear scaling.

[4]We used the Blender Foundation's "Big Buck Bunny" [32] open-source movie. The same video sequence is used in our video QoE tests presented in section IV-D.

TABLE II
CHUNK SIZES (BYTES) FOR OUR TEST H.264/AVC DASH VIDEO SEQUENCES.

| | Min | **Average** | Max |
|---|---|---|---|
| Low quality (0.29 Mbps) | 69852 | **73262** | 250785 |
| High quality (1.7 Mbps) | 80755 | **434830** | 843744 |

to download the video chunk-by-chunk, potentially switching among the available video representations (and thus bitrates) to better adapt to network conditions. Table II shows average, minimum, and maximum chunk sizes for our test video, for the low (0.29 Mbps) and high (1.7 Mbps) quality representations.

Table III presents the achieved HTTP request throughput for different numbers of simultaneous connections (1000 vs. 10000) when all clients are requesting (i) 73 KB video chunks (low quality), and (ii) 435 KB video chunks (high quality). We notice the expected response rate decrease as the size of the requested objects increases. One vCPU can sustain a high-quality video request throughput of ∼8600 requests/s when the HTTP server is containerized, compared to ∼3800 requests/s when run in a kvm VM for 10000 parallel HTTP connections.

TABLE III
HTTP REQUEST THROUGHPUT FOR DIFFERENT OBJECT SIZES AND NUMBERS OF CONCURRENT CONNECTIONS.

| | 73 KB | | 435 KB | |
|---|---|---|---|---|
| # connections | docker | kvm | docker | kvm |
| 1000 | 15928 | 12578 | 8723 | 6165 |
| 10000 | 15270 | 9897 | 8649 | 3822 |

*2) Response times:* Our other performance metric is related with user experience. We measured HTTP response times as a function of the virtualization technology used, the number of parallel connections, and CPU resource availability. We launched 90% of the concurrent HTTP connections using weighttp and we recorded response times for the remaining 10% using *ab* [36], due to its latency measurement capabilities. All figures of this section present empirical Cumulative Distribution Functions (CDF) of response times; each point represents the percentage of requests which were served in a time less than or equal to a specific value on the $x$-axis.

Fig. 7 compares a virtualized (kvm) to a containerized (docker) HTTP server, to which one CPU core is allocated. As shown, an increase in the number of parallel connections results in higher latencies, due to the connection management overhead on the server. Docker achieves lower response times, since it incurs less overhead for interacting with the operating system and for network I/O (especially given our host-mode configuration, which gives it native access to the host's networking stack). Scaling up processing capacity by adding more vCPUs can significantly reduce latency in both cases.

When clients request larger objects, response times increase. This increase is noticeable in both cases. For kvm (Fig. 8a) it reaches more than 200 ms for 35% of the requests for a 73 KB file when the server VM runs on a single CPU core and there are 1000 concurrent connections (star points). Docker (Fig. 8b) performs better: In the same settings, 98% of the requests are completed in less than 35 ms (blue curve, star points).

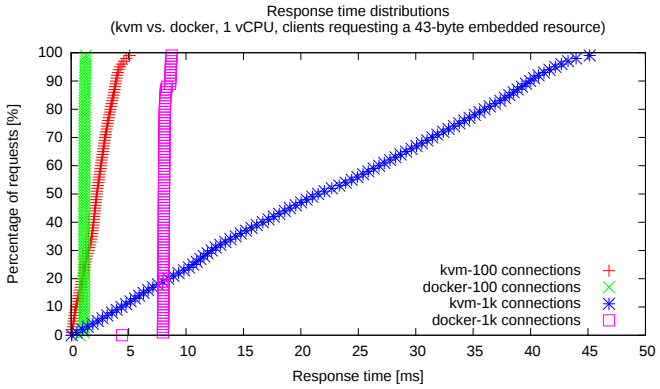
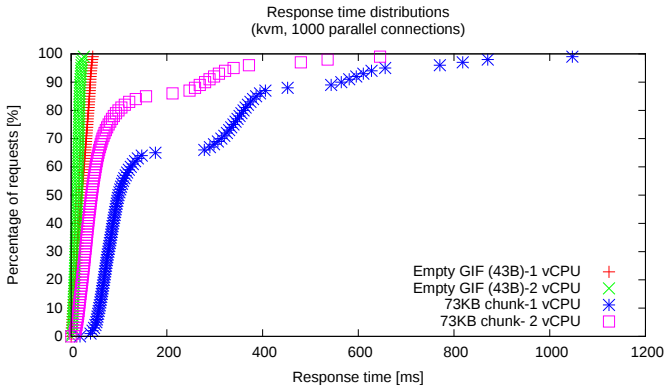
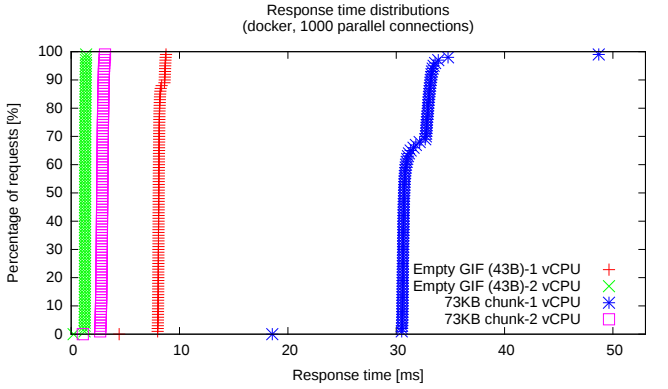



Fig. 7. Comparison of the response time distributions of a virtualized vs. a containerized HTTP server, for different numbers of concurrent connections. Parallel HTTP connections request a minimal embedded object (43-byte empty GIF file).




(a) kvm




(b) docker

Fig. 8. Response time distributions for different sizes of the requested objects.

Adding a second vCPU improves performance significantly in both cases (square points). This latency improvement when we scale up CPU resources is more evident as server load (number of parallel connections) grows, as shown in Fig. 9.

### D. Performance of an HTTP video streaming service

In a similar spirit as in our previous experiments, we measure the performance capabilities of an HTTP video server on a single vCPU in the presence of multiple parallel video

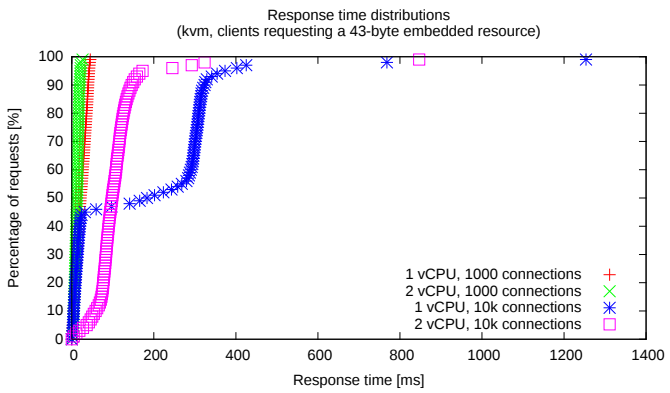



Fig. 9. Response time distributions for a server hosted in a kvm VM as the number of available vCPUs scales, for different numbers of parallel HTTP connections. Clients request a minimal embedded object (43-byte empty GIF file).

sessions. Following a user-centric approach, we empirically quantify the relationship between service workload and QoE in video-service-related terms. We use a vanilla nginx server, which serves user requests for video content prepared for DASH delivery, while on the client side we measure QoE on a DASH video player in the presence of multiple parallel emulated video streams.

*1) Measurement methodology:* Our QoE metric is the Mean Opinion Score (MOS), i.e., the expected rating that a panel of users would give to the quality of the transmitted video in a 1-5 (poor-excellent) scale. To estimate it we apply the Pseudo-Subjective Quality Assessment (PSQA) [37] approach. PSQA consists in training a Random Neural Network (RNN) based on experiments with physical subjects under controlled conditions, where a set of parameters affecting quality is monitored and the ratings of users are recorded. The trained RNN classifier can then be applied in real time and output the expected MOS for specific values of the input parameters. Singh et al. [38] have applied PSQA to estimate QoE for H.264/AVC-encoded video delivered over HTTP, and we are using their tool in our experiments.

The PSQA tool operates on 16 s video segments. As our test video sequences are larger (475 s), we calculate a MOS value for each 16 s window. Since we cannot argue that user experience is independent for consecutive time instances, we maintain a moving average of the calculated MOS in order to capture recency effects according to the following equation:

$$MOS_{i+1} = 0.3MOS_i + 0.7s_{i+1},$$

where $MOS_i$ is the moving average up to window $i$ and $s_{i+1}$ is the MOS sample calculated for the $(i+1)$-th window.

The input parameters for the QoE estimation tool (appropriately normalized [38]) are the following:

- The number of interruptions in the 16 s window.
- The average and the maximum interruption duration.
- The average value of the Quantization Parameter (QP) across all picture macroblocks in the measurement window. QP is an indication of picture quality (the higher the QP, the lower the video bitrate and quality).

To acquire the required input, we extended the *vlc* open-source media player with specific monitoring functionality, i.e., with the ability to record QP values, and playout interruptions and their duration. Furthermore, we modified the vlc DASH plugin to disable video rate adaptation, thus only serving the video representation with the selected bitrate.

We further make the following assumptions with respect to QoE calculation:

- If at a specific 16 s window there is an interruption longer than 8 s (half of the measurement window), we consider that the MOS is 1.0 (minimum value possible). This is because the RNN used shows saturation when quality is very bad and its training focused on being more accurate when quality is good, at the expense of inaccuracies when interruptions are very large or very frequent [38].
- If the video playout is terminated prematurely or if the video fails to start, e.g., due to connection reset problems (something we observed frequently for large server loads), we consider that MOS = 1.0 for all video windows which follow playout termination.

We use the same DASH test video sequence which we prepared for our previous experiments, with each chunk carrying two seconds of video. In all the experiments that follow, the video client is configured to request only the high-quality video representation (no rate adaptation; mean chunk size of 435 KB; video bitrate of 1.7 Mbps). To emulate multiple simultaneous viewers, we generated parallel HTTP sessions using a modified version of the *wrk* HTTP load generator[5] which allowed for specifying the exact request rate. Each of the parallel wrk connections was downloading a 435 KB file at the video rate (i.e., 1 chunk/2 s).

At each experiment, repeated a number of times to acquire the necessary number of QoE samples for statistical confidence, we varied the number of parallel emulated video sessions, at the same time recording QoE-related information on a vlc instance accessing a DASH video from the server.

We used the same host/guest configurations as in our previous experiments. Depending on the experiment, we varied the server workload appropriately to find the load points after which interruptions, and thus quality degradation, start to take place. (In other words, we do not report QoE results for all "low" load values, since QoE was unaffected.) We are interested in the performance of the video service when using one vCPU; the results we present when using two vCPUs serve just to demonstrate that the service indeed scales and to verify that performance degradation for high loads is not due to network bottlenecks.

*2) Results:* Fig. 10 presents the average MOS value observed across all 16 s video samples for each case. We observe that a vanilla nginx server can sustain up to more than 5000 parallel HD video sessions with an excellent quality when kvm is in use and the guest uses one vCPU. For loads of more than 6000 parallel users, video interruptions start to take place, which reduce QoE, especially as load grows. The frequency (Fig. 11) of such playout interruptions follows an increasing

[5]https://github.com/giltene/wrk2

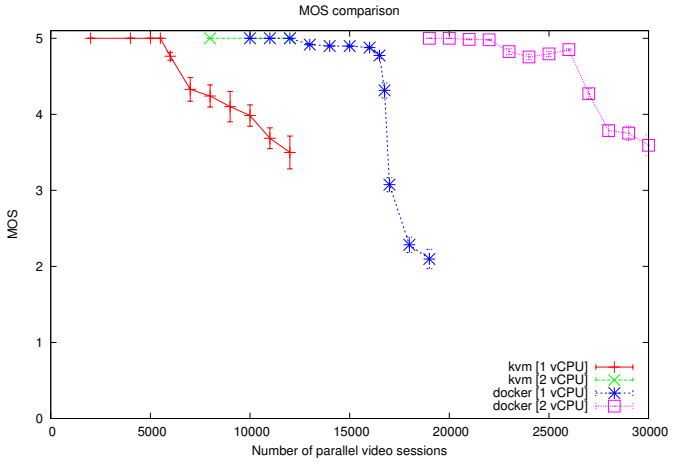



Fig. 10. Average QoE as a function of the number of parallel users accessing a HD video from a web server. Each point is the mean of a few hundreds of QoE samples (MOS values), presented with 95% confidence intervals.

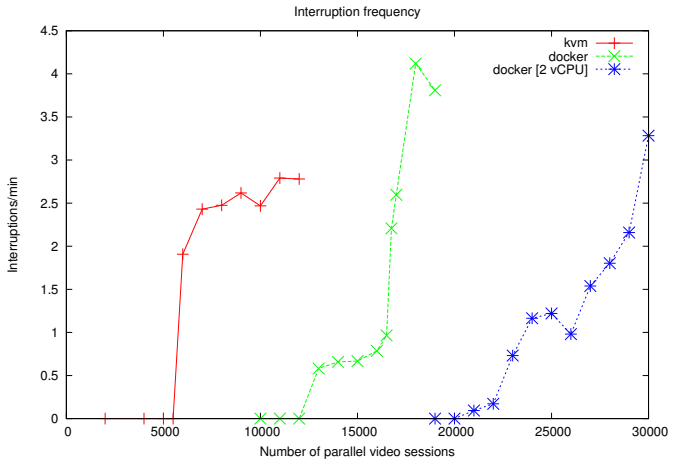



Fig. 11. Interruption frequency (in interruptions/minute) as server load grows.

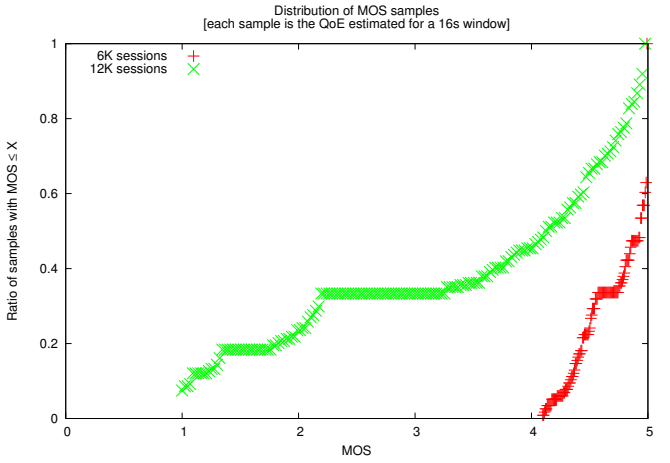



Fig. 12. Empirical CDF of the QoE of video samples for two different server loads when the service is hosted in a kvm virtual machine.

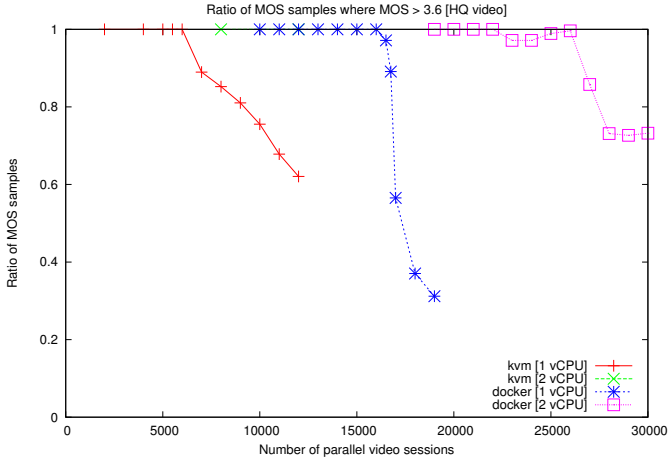



Fig. 13. Probability that quality in a sample is acceptable (i.e., MOS $> 3.6$).

trend with server load, reaching, on average, more than 2.5 per minute when there are 12000 parallel video streams.

A similar trend is observed when docker is used. However, the respective figures for docker reveal that for the same compute resources allocated performance is much better. While for kvm video interruptions start to happen when load exceeds 5500 users, docker on a single vCPU maintains an uninterrupted video service even for double the number of concurrent streams (12000). Performance scales approximately in a linear fashion when adding one vCPU. In this case, a docker container can serve more than 23000 users without interruptions, and even when 25000 users are served in parallel from a 2-vCPU container, MOS reaches 4.8 in the 1-5 scale.

These results are in line with our request throughput experiments. For example, in Section IV-C1 we found that on a single vCPU docker can sustain 8.6K requests/s for the chunk size that corresponds to the HQ video (see Table III). The fact that, for the same configuration, with more than 17K parallel video streams (i.e., 8.5K requests/s, given that each chunk carries 2 s of video) MOS starts to more drastically degrade is a user-centric expression of the same phenomenon.

From a user experience perspective, another interesting metric would be the ratio of viewing time that QoE is above a specific threshold (or, in other words, the ratio of samples in which QoE is above this threshold). This could also be encoded in an SLA of a different format, where the customer is interested in ensuring that end users will enjoy an acceptable QoE level for more than a specific percentage of the video viewing time. To evaluate this metric we resort to the empirical cumulative distribution function of the QoE samples we collected in our experiments, examples of which for kvm (1 vCPU) we present in Fig. 12. This interpretation is depicted in Fig. 13, where we present the probability that QoE for a 16 s sample exceeds a MOS threshold of 3.6, which, in various contexts (e.g., VoIP), is considered acceptable quality.

## V. Uses of our results

Our experiments provide insight on the capabilities of the enabling technologies for our CDNaaS vision. This can be applied to devise appropriate resource allocation and management strategies. Our findings indicate that virtualization comes with a larger overhead compared to using containers. This

observation is in line with other works in the literature [29]. However, it should be noted that, despite their performance and flexibility benefits, containerization technologies for cloud computing have only recently gained popularity, while virtualization can be considered more mature in this context.

We have identified two main uses of such quantitative results: (i) Optimally deciding on the amount of resources to deploy and on their configuration, in order to match customer demand, and (ii) deriving pricing strategies for the offered virtual CDN service. Part of our ongoing work focuses particularly on these two aspects.

### A. Service dimensioning

Via our northbound API, a customer can express specific service demand characteristics, such as a target maximum number of concurrent users that will be accessing its service per region. Also, the request can include a quality specification, which, for a video streaming application, takes the form of a minimum MOS for the end users and for specific video characteristics (e.g., high definition video). With this information in place, and with the empirical model of QoE as a function of workload that we derived, the CDNaaS provider can optimize the amount of the resources to allocate.

We have shown that such informed resource allocation decisions save on operator costs while improving on user experience in our prior work [25]. In particular, we designed an algorithm which calculates the number of vCPUs necessary (minimum cost solution with respect to compute resources) to serve a specific customer-specified demand under capacity (operator) and quality (customer) constraints for a video streaming service over vCDN.

The amount of CPU resources allocated needs to be distributed to a number of virtual instances across the physical hosts in a region's data centers. How exactly these resources are allocated and how the respective instances are placed is a matter of addressing the tradeoff between reliability and cost: Splitting resources across instances and distributing the latter across multiple physical hosts can result in higher service availability, but also increased operational cost (e.g., power consumption) for the operator, which may be reflected in the service price. Our current focus is on studying multi-objective optimization formulations for the joint CPU resource allocation and virtual instance placement problem, where customer preferences are translated to specific weightings for the two conflicting objectives. An extensive review of models and algoritms for the relevant problem of VM allocation in cloud data centers is provided by Mann [39].

### B. Pricing models and SLA design, monitoring and enforcement

From another perspective, our measurement results can be used as input by the system operator to design SLAs and apply different pricing models for the vCDN service. In the following discussion, we assume the case where the service-level objective for the customer is a response time guarantee. As shown in Fig. 14, the operator can select specific points representing response times for a specific request percentile:

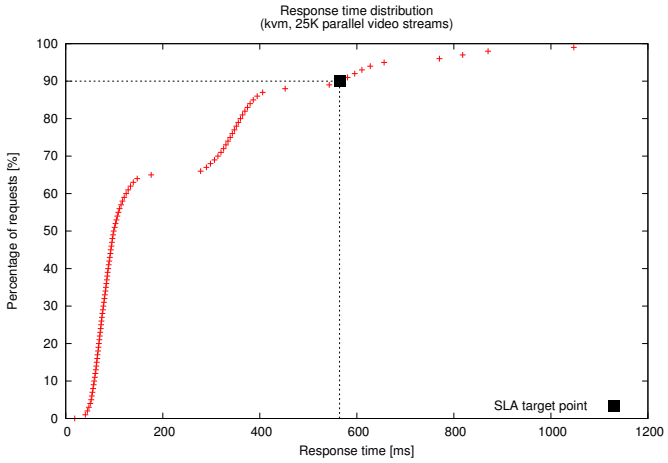



Fig. 14. Response time distribution for low-quality video chunks. Kvm is used as the hypervisor and 1 vCPU is allocated to the HTTP server instance. The square point indicates the $90^{th}$ percentile of the time to complete an HTTP request for a 70 KB file; this objective could be encoded in an SLA.

For example, using kvm (1 vCPU), 90% of the requests for *low-quality* video chunks can be completed in less than 564.5 ms when the traffic volume handled by the VNF instance corresponds to 25000 parallel video streams (this value for docker is 33.185 ms; the respective figure is omitted due to lack of space). Sustaining larger workloads while meeting this specific response time requirement, which can be encoded in an SLA, necessitates allocating more resources (e.g., launching a new instance or scaling up an existing one with more vCPUs).

Taking such information into account, the operator can offer different pricing schemes. Two examples follow:

*a) Flat-rate, non-scalable deployment:* This is a basic service which provides no elasticity. The price is set based on an initial demand estimate that is provided by the customer (content provider): On service instantiation, the customer can express a target number of end-user HTTP sessions per region, with a specific objective with respect to response times. The service provider then decides on the amount of resources (HTTP server VNF instances and the respective vCPUs) that have to be placed to cover this demand, and, in turn, the price. In case of a surge in traffic demand, no guarantees are provided and no service re-dimensioning takes place.

*b) On-demand elastic service:* Here the operator advertises guaranteed response time SLAs. The customer expresses an initial demand and the operator translates it to an initial number of VNF instances (and vCPUs). The number of ongoing HTTP connections per instance is monitored via the service-level monitoring/management interface and, when necessary, the service provider decides to up- or down-scale the service in order to match the target response time distribution. For example, if the total number of parallel user sessions increases beyond a threshold such that the SLA objective cannot be met, a new instance is launched and user requests for content are appropriately redistributed.

Under the elastic model, the operator can apply on-demand pricing: A base price is derived from the initial content provider's service request, and the price is scaled following

the additional resources that need to be deployed to meet the target response times encoded in the agreed-upon SLA. Note that video quality can be appropriately combined with the above models as another pricing dimension.

Finally, we remark that the reported response time distributions are mainly bounded by the CPU capabilities of the VNF platform and the host-to-guest communication overhead, and do not include the additional delay imposed in the user-VNF instance path. However, this path is monitored and controlled by the network operator, which can exploit internal information to derive a more accurate response time distribution (e.g., by shifting the curve appropriately to account for fixed network-level overheads), apply specific "safety margins" in the advertised response time guarantees, or accordingly provision this path with network resources to avoid SLA violations.

## VI. Conclusion

We presented the design and implementation of an architecture allowing a telecom operator to offer a cloud-based vCDN service to content providers on demand. Our approach, which is in line with current standardization efforts on Network Functions Virtualization, improves on resource management and service provision flexibility compared to traditional telco CDNs, thus having the potential of empowering the role of telecom operators in the content delivery value chain. Our CDNaaS design is extensible. It supports the deployment of a multitude of different CDN flavors, each with their own features and resource management schemes. For the latter, a better understanding of the performance capabilities of the underlying virtualization technologies as to vCDN provision is necessary. We therefore carried out an extensive experimental evaluation of state-of-the-art containerization and virtualization schemes on top of which a vCDN service can be built, and derived empirically the relationship between workload and service performance under specific technology configurations. With these results as a basis, our ongoing and future research concentrate on their two main uses that we have identified in this article: cloud resource allocation and elastic management, and designing SLAs and CDNaaS pricing structures.

## References


[1] R. T. Ma, J. C. Lui, and V. Misra, "On the evolution of the Internet economic ecosystem," in *Proc. WWW*, 2013.

[2] OpenStack. [Online]. Available: https://www.openstack.org/

[3] B. Frank, I. Poese, G. Smaragdakis, A. Feldmann, B. Maggs, S. Uhlig, V. Aggarwal, and F. Schneider, "Collaboration Opportunities for Content Delivery and Network Infrastructures," *ACM SIGCOMM ebook on Recent Advances in Networking*, vol. 1, August 2013.

[4] I. Poese, B. Frank, B. Ager, G. Smaragdakis, S. Uhlig, and A. Feldmann, "Improving content delivery with PaDIS," *IEEE Internet Comput.*, vol. 16, no. 3, pp. 46–52, 2012.

[5] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber, "Pushing CDN-ISP Collaboration to the Limit," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, pp. 34–44, Jul. 2013.

[6] N. Herbaut, D. Négru, Y. Chen, P. A. Frangoudis, and A. Ksentini, "Content delivery networks as a virtual network function: A win-win ISP-CDN collaboration," in *Proc. IEEE Globecom*, 2016.

[7] FP7 T-NOVA. [Online]. Available: http://www.t-nova.eu

[8] G. Xilouris, E. Trouva, F. Lobillo, J. Soares, J. Carapinha, M. McGrath, G. Gardikis, P. Paglierani, E. Pallis, L. Zuccaro, Y. Rebahi, and A. Kourtis, "T-NOVA: A marketplace for virtualized network functions," in *Proc. EuCNC*, 2014.

[9] P. Maillé, G. Simon, and B. Tuffin, "Vertical integration of CDN and network operator: Model and analysis," in *Proc. IEEE MASCOTS*, 2016.

[10] Content Delivery Network Interconnection (CDNI) Working Group. IETF. [Online]. Available: https://datatracker.ietf.org/wg/cdni/

[11] H. Lee, D. Lee, and Y. Yi, "On the economic impact of telco cdns and their alliance on the cdn market," in *Proc. IEEE ICC*, 2014.

[12] H. Lee, L. Duan, and Y. Yi, "On the competition of CDN companies: Impact of new telco-CDNs' federation," in *Proc. WiOpt*, 2016.

[13] S. Spagna, M. Liebsch, R. Baldessari, S. Niccolini, S. Schmid, R. Garroppo, K. Ozawa, and J. Awano, "Design principles of an operator-owned highly distributed content delivery network," *IEEE Commun. Mag.*, vol. 51, no. 4, pp. 132–140, April 2013.

[14] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. ACM CoNEXT*, 2010.

[15] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, Feb. 2013.

[16] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 24–31, November 2013.

[17] *Network Functions Virtualisation (NFV); Management and Orchestration*, ETSI Group Specification NFV-MAN 001, Dec. 2014.

[18] *Network Functions Virtualisation (NFV); Use Cases*, ETSI Group Specification NFV 001, Oct. 2013.

[19] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Proc. IEEE NOMS*, 2014.

[20] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. IFIP CNSM*, 2014.

[21] T. Wood, K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, "Toward a software-based network: integrating software defined networking and network function virtualization," *IEEE Network*, vol. 29, no. 3, pp. 36–41, May 2015.

[22] P. A. Frangoudis, L. Yala, A. Ksentini, and T. Taleb, "An architecture for on-demand service deployment over a telco CDN," in *Proc. IEEE ICC*, 2016.

[23] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Proc. Linux Symposium*, 2007.

[24] nginx. [Online]. Available: http://nginx.org

[25] L. Yala, P. A. Frangoudis, and A. Ksentini, "QoE-Aware Computing Resource Allocation for CDN-as-a-Service Provision," in *Proc. IEEE Globecom*, 2016.

[26] *Network Functions Virtualisation (NFV); Management and Orchestration; Report on Architectural Options*, ETSI Group Specification NFV-IFA 009, Jul. 2016.

[27] Docker. [Online]. Available: https://www.docker.com/

[28] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," in *Proc. IEEE International Conference on Cloud Engineering (IC2E '14)*, 2014.

[29] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," IBM Research, Tech. Rep. RC25482, July 2014. [Online]. Available: http://goo.gl/ytEvt9

[30] weighttp. [Online]. Available: http://redmine.lighttpd.net/projects/weighttp/wiki

[31] Nmap - the network mapper. [Online]. Available: https://nmap.org/

[32] Big Buck Bunny. [Online]. Available: https://peach.blender.org/

[33] *Series H: Audiovisual and Multimedia Systems – Infrastructure of audiovisual services – Coding of moving video – Advanced video coding for generic audiovisual services*, ITU-T Recommendation H.264, Feb. 2016.

[34] *Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*, ISO/IEC Standard 23 009-1:2014, May 2014.

[35] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over HTTP dataset," in *Proc. 3rd ACM MMSys*, 2012.

[36] ab - Apache HTTP server benchmarking tool. [Online]. Available: https://httpd.apache.org/docs/2.2/programs/ab.html

[37] G. Rubino, "Quantifying the Quality of Audio and Video Transmissions over the Internet: The PSQA Approach," in *Communication Networks & Computer Systems*, J. A. Barria, Ed. Imperial College Press, 2005.

[38] K. D. Singh, Y. Hadjadj-Aoul, and G. Rubino, "Quality of experience estimation for adaptive HTTP/TCP video streaming using H.264/AVC," in *Proc. IEEE CCNC*, 2012.

[39] Z. A. Mann, "Allocation of virtual machines in cloud data centers–a survey of problem models and optimization algorithms," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 11:1–11:34, Aug. 2015.

**Pantelis A. Frangoudis** received the B.Sc. (2003), M.Sc. (2005), and Ph.D. (2012) degrees in Computer Science from the Department of Informatics, Athens University of Economics and Business, Greece. From October 2012 to January 2017 he was with team Dionysos, IRISA/INRIA/University of Rennes 1, Rennes, France, which he joined under an ERCIM post-doctoral fellowship (2012-2013). He is currently a post-doctoral researcher at the Communication Systems Department, EURECOM, Sophia Antipolis, France. His research interests include mobile and wireless networking, Internet multimedia, network security, future Internet architectures, cloud computing, and QoE monitoring and management.

**Louiza Yala** is currently a Ph.D. student at University of Rennes 1, France, and a member of the IRISA/INRIA team Dionysos. She received a M.Sc. degree (2014) in networking from the University of Bejaia, Algeria, and a M.Sc. degree (2015) in Technologies for Information Processing and Systems Analysis from the University of Technology of Compiègne, France. Her research interests include network virtualization, Internet multimedia, Content Delivery Networks, and QoE.

**Adlen Ksentini** received his M.Sc. degree in telecommunication and multimedia networking from the University of Versailles Saint-Quentin-en-Yvelines in 2002, and his Ph.D. degree in computer science from the University of Cergy-Pontoise in 2005, with a dissertation on QoS provisioning in IEEE 802.11-based networks. From 2006 to 2016, he worked at the University of Rennes 1 as an assistant professor. During this period, he was a member of the Dionysos Team with INRIA, Rennes. Since March 2016, he has been working as an assistant professor in the Communication Systems Department of EURECOM. He has been involved in several national and European projects on QoS and QoE support in future wireless, network virtualization, cloud networking, and mobile networks. He has co-authored over 100 technical journal and international conference papers. He received the best paper award from IEEE IWCMC 2016, IEEE ICC 2012, and ACM MSWiM 2005. He has been awarded the 2017 IEEE Comsoc Fred W. Ellersick (best IEEE Communications Magazine's paper). Adlen Ksentini has been acting as TPC Symposium Chair for IEEE ICC 2016/2017, IEEE GLOBECOM 2017, IEEE Cloudnet 2017 and IEEE 5G Forum 2018. He was a Guest Editor of IEEE Wireless Communications, IEEE Communications Magazine, and two issues of ComSoc MMTC Letters. He has been on the Technical Program Committees of major IEEE ComSoc, ICC/GLOBECOM, ICME, WCNC, and PIMRC conferences. He is currently the Vice-Chair of the IEEE COMSOC Technical Committee on Software (TCS).