Allocation and Control of Computing Resources for Real-time Virtual Network Functions

Mauro Marinoni, Tommaso Cucinotta
and Luca Abeni

Scuola Superiore Sant'Anna
Pisa, Italy
Email: {name.surname}@santannapisa.it

Carlo Vitucci

Ericsson
Stockholm, Sweden
Email: carlo.vitucci@ericsson.com

***Abstract*—Upcoming 5G mobile networks strongly rely on Software-Defined Networking and Network Function Virtualization that allow exploiting the flexibility in resource allocation provided by the underlying virtualized infrastructures. These paradigms often employ platform abstractions designed for cloud applications which have not to deal with the stringent timing constraints characterizing virtualized network functions. On the other hand, various techniques exist to provide advanced, predictable scheduling of multiple run-time environments, e.g., containers, within virtualized hosts. In order to let high-level resource management layers take advantage of these techniques, this paper proposes to extend network service descriptors and the Virtualization Infrastructure Manager. This enables Network Function Virtualization orchestrators to deploy components exploiting real-time processor scheduling at the hypervisor or host OS level, for enhanced stability of the provided performance.**

Keywords*–*MANO; TOSCA; NFV descriptors; OpenStack, LXC, Sched_Deadline; Linux kernel; Real-Time scheduling.

I. Introduction

The 5G system architecture has been introduced to provide new services more tailored to specific user needs and Quality of Service (QoS) requirements. These features will allow Telco operators to develop new business cases, able to overcome the current uncertainties that risk compromising their business. In the context of 5G functions, some fundamental requirements have been recognized as of utmost importance for upcoming telecommunication systems [1]:

- the ability to support a wide range of services [2];
- an efficient handling and allocation of resources, through a run-time monitoring and control of resources usage and deployed services;
- the run-time control of the Quality of Service (QoS), including throughput and operational deadlines, so as to comply with a possible Service Level Agreement (SLA).

A feature that is gaining attention in this context is the one of *resource slicing*, the capability of providing strong isolation in resources access, allowing for a precise control of the interferences among different functions/services.

It seems widely recognized that recently proposed Network Function Virtualization (NFV) infrastructures, leveraging on principles of flexible and fully automated infrastructure management typical of cloud environments, enriched with Software-Defined Networking (SDN) techniques employing fully automated and dynamic management and reconfiguration of the network, constitutes the ideal fit for handling the complex requirements of the envisioned upcoming 5G scenarios.

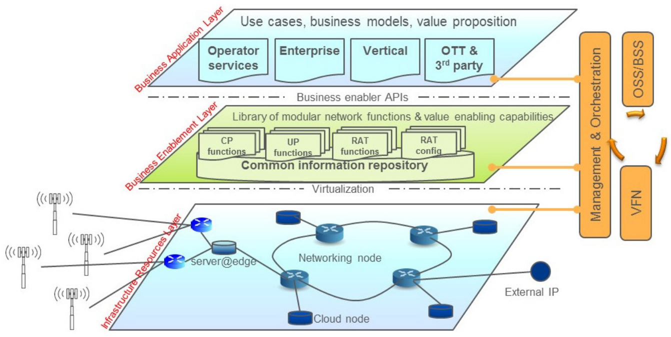
Business Application Layer
Use cases, business models, value proposition
Operator services
Enterprise
Vertical
OTT & 3rd party
Business enabler APIs
Business Enablement Layer
Library of modular network functions & value enabling capabilities
CP functions
UP functions
RAT functions
RAT config
Common information repository
Virtualization
Infrastructure Resources Layer
server@edge
Networking node
Cloud node
External IP
Management & Orchestration
OSS/BSS
VFN

Figure 1. 5G network management proposed by the NGMN Alliance.

To deploy end-to-end SDN-NFV solutions, there is an increasing interest in the use of edge micro-servers. These allow for the provisioning of a virtual, elastic infrastructure that can readily be adapted to time-varying workload patterns. On top of them, network functions can be instantiated and elastically made to grow and shrink, as needed, in a completely automated way, leveraging high-level SDN/NFV function orchestration coupled with appropriate monitoring capabilities. Possible exploitation of these features is the 5G network management approach proposed in [3] by the *Next Generation Mobile Networks* (NGMN) Alliance as shown in Figure 1. It presents an architecture that exploits the structural separation of hardware and software, and the programmability offered by SDN-NFV to support multiple use cases, value creation, and business models.

A. *Problem presentation*

An efficient management of the underlying physical infrastructure, able to achieve high saturation levels and energy efficiency, calls for time-sharing of the available physical resources (e.g., available CPUs, network links, data stores) across a number of deployed functions, exploiting the unlikely occurrence of synchronous workload peaks for a multitude of functions, often from different vendors/operators, that are co-located on the same physical elements. Increasing the level of sharing of the physical infrastructure introduces unpredictable behavior in the hosted virtualized functions, where the virtualized infrastructures carved upon the physical one suffer from one major drawback: *the impossibility to keep a stable processing/networking performance*, due to the several unpredictable interferences among co-located functions.

Therefore, an uprising trend in NFV infrastructure management, is the one to employ, within the physical NFV infrastructure, proper mechanisms for *resource slicing*, preventing applications to interfere with each other applying strong isolation in resources access and use [4]–[6]. In this context, it is noteworthy to mention that traditional ways to control the stability of the performance exposed by a shared infrastructure, and specifically a cloud infrastructure, include:

1) employing *elasticity loops* able to adapt dynamically the provisioned virtual infrastructure size to the dynamically changing workload;
2) *dedicating individual physical resources* to individual virtual resources, e.g., employing a 1-to-1 mapping among virtual cores of the virtualized infrastructure to physical cores of the underlying physical machines;
3) *dedicating individual physical machines* to individual virtual resources and/or virtual functions, like in a bare-metal provisioning model, where a single function or service is deployed onto a whole physical machine, either encapsulated within a virtual machine, or an OS container, or directly deployed on the bare hardware.

The above mentioned point 1) has the drawback that, albeit on average it is possible to control the provisioned virtual infrastructure capability to match the workload requirements, occasional, unpredictable spikes in a function workload, as well as interfering workloads form other co-hosted/co-located functions, make the instantaneous performance of individual virtualized resources (e.g., single VM/container) highly unstable. The resulting effects can be processing queues temporarily filling up with adverse consequences on the overall end-to-end latency of the realized function/service chain. To avoid such problems, it is possible to couple the technique with the use of dedicated physical cores, as mentioned in point 2) above, where interferences due to time-sharing of multiple functions over the same CPUs are avoided. Still, in a multi-core server as usually used in these contexts, temporal interferences can occur among virtual functions deployed onto different physical cores, as due to their sharing of, and temporary saturation of, such resources as the available memory in the Last Level Cache (LLC), the available transfer bandwidth between the CPUs/LLC and the memory banks in the platform, the available bandwidth on shared network adapters, etc. Therefore, it is sometimes necessary to recur to the use of dedicated whole physical machines, as from point 3) above, to ensure predictable performance of the hosted services.

Unfortunately, employment of the techniques in points 2) and 3) above lead to lower the level of sharing of the infrastructure, decreasing the potential economical advantage arising from the adoption of a flexible on-demand provisioning model, corresponding to a use of the underlying infrastructure that becomes more and more inefficient and power hungry.

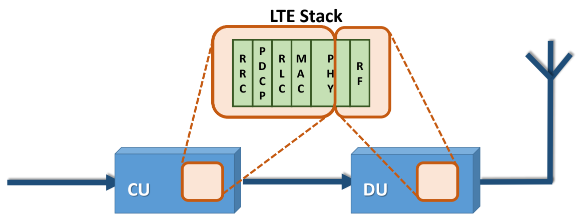

Figure 2. Intra-PHY split (Option 7-1) and placement of the corresponding NFV functions.

In this paper, the point is made that recent advances in scheduling technologies from the realm of soft real-time systems can nowadays be deployed on general-purpose operating systems and hypervisors, allowing for the provisioning of real-time virtualized processing infrastructures whose performance can be controlled at a fine granularity level (see Section III-A). Therefore, it is critical to expose such capabilities at the highest level of the NFV infrastructure management stack, as described by Cucinotta et al. [7]. This approach will improve isolation and predictability of deployed VNFs, with more efficient use of resources and management of the VNF functions more independent of the underlying infrastructure allocation.

B. *Paper organization*

The rest of this paper is organized as follows. Section II describes the Radio Access Networks scenario and outlines how computational resources are handled by NFV orchestrators. Section III describes how to exploit innovative scheduling algorithms to provide CPU reservations across co-hosted Linux containers and presents a proposal regarding the integration of CPU reservations in high-level NFV descriptors. Section IV concludes the paper.

II. BACKGROUND

A. *Virtualized Radio Access Network*

A typical NFV application scenario is the Virtualized Radio Access Network (VRAN), a paradigm that tries to shift computational elements within the networking stack of the access network away from the radio head(s), deploying them in edge micro-server infrastructures. Such a paradigm is also aligned with the increasing need for reducing power consumption in VRAN deployments, which is strongly depending on the radio access network [8], through a wise redistribution of the processing functions. This has the potential to bring tremendous power savings.

This redistribution needs rediscussion of the overall architecture of the network stack, and the optimal split of its components across the hardware elements. The evolved NodeB (eNB) can be divided into two elements: the Central Unit (CU) with a centralized deployed and the Distributed Unit (DU) positioned near the antenna. Regarding the LTE stack, a set of possible functional split solutions are conceivable and still under discussion [9]. Among them, a promising one is the IntraPHY split called Option 7-1 that allows deploying on the DU only the Radio Frequency and part of the Physical layer, while assigning the remaining part of the stack to the CU, as shown in Figure 2. When performing such allocations, it is crucial to consider the timing constraints like the stringent one ($4ms$) imposed on the acknowledgment of packets by the Hybrid ARQ (HARQ) [10] protocol.

The problem of storage and network slicing and isolation has been thoroughly addressed in the literature, primarily driven by data center optimization. For example, Giannone et al. [11] studied the impact of virtualization on the latency in the fronthaul connecting CU and DU for the scenario presented in Figure 2 implemented using OpenAirInterface [12]. Garikipati et al. [13] proposed a new scheduling policy to be applied within a single VNF of a VRAN, but their work deals with dedicated physical resources and the integration with VMs/containers is left as future work. Instead, slicing and temporal isolation at the CPU access level are not yet suitable for RAN purposes and requirements, and the management and orchestration of available SDN-NFV solutions apply simple partitioning strategies only. In this paper, a proposal is presented to bridge the gap between low-level mechanisms for real-time scheduling of VNFs and high-level orchestration layers of a SDN-NFV stack.

Other approaches for dealing with QoS attributes in TOSCA specification can be found, e.g., [14]; however, a comprehensive literature review is out of the scope of this paper.

B. NFV Orchestration

Since NFV involves a consistent number of virtualized resources, its handling demands a significant effort concerning software management, that is named orchestration. Orchestration oversees the required resources from the underlying physical platform for the NFV services. Telco operators apply NFV orchestration to promptly deploy services and Virtual Network Functions (VNFs), exploiting cloud software on COTS hardware platforms.

To address this needs, the European Telecommunications Standards Institute (ETSI) has defined the Network Functions Virtualization MANagement and Orchestration (NFV-MANO) architecture. Inside the MANO architecture is possible to see three main functional blocks: the Virtual Network Function Manager (VNFM), the Virtual Infrastructure Manager (VIM), and the Network Functions Virtualization Orchestrator (NFVO).

The VNFM is in charge of managing the lifecycle of VNFs from creation to termination, dealing with scaling up/down of resources, and handling faults, monitoring, and security.

The VIM manages the NFV Infrastructure (NFVI), including physical resources (e.g., server, storage), virtual resources (e.g., Virtual Machines) and software resources (e.g., hypervisor). In the NFV architecture is possible to have several NFVI, each one handled by a VIM, which is in charge creating/-maintaining/removing VMs, maintaining knowledge on VMs allocation to physical resources, monitoring performance and fault management of all resources in the infrastructure.

NFVO deals with the challenges connected with the management of resources and services in the NFV architecture. It coordinates, authorizes, acquires and discharges NFVI resources within one or more PoPs, interfacing with the VIM instead of directly handling NFVI resources. NFVO interconnects independent VNF functions to provide a coherent end to end service by directly coordinating with the corresponding VNFMs, to avoid talking to every single VNF. Service Orchestration maintains the topology of the service instances.

The TOSCA NFV profile defines a precise NFV data model, allowing to catch in a template all the requirements concerning deployment and operational behaviors. These VNF descriptors are collected in a catalog to make them available for selection, and each one includes three kinds of components (called nodes) that are Virtual Deployment Units (VDU), Connection Points (CP), and Virtual Links (VL). In particular:

- a *Virtual Deployment Unit* describes the features of a virtualized container (e.g., virtual CPUs, memory, disks);
- a *Virtual Link* is a logical connection between VDUs that are deployed dynamically on top of the physical infrastructure. It represents the logical entity to provide connectivity among VNFs;
- *Connection Points* model how Virtual Links connect to Virtual Network Functions and represent the virtual and/or physical interfaces of the VNFs.

Every node can be characterized by type, capabilities, attributes, properties, and requirements, as defined in [15]. When OpenStack [16] is used as Virtual Infrastructure Manager, the descriptor of each VDU node is used to generate a Heat Orchestration Template (HOT) file. The HOT file is supplied to the OpenStack compute service, Nova [17], to express a VNF requirements on the needed virtualized computing platform.

III. Proposed approach

In what follows, we describe our proposal to employ real-time deadline-based scheduling to temporally isolate VNF in a NFV infrastructure. We highlight the advantages of real-time scheduling for VNFs, and describe how we plan to extend high-level MANO descriptors to support the new scheduling capabilities at the hypervisor layer.

A. CPU reservations for NFV

As presented in Section I, the virtualization of network functions has stringent requirements regarding CPU slicing. In particular, some decoding and demodulation activities that are virtualized by executing them in software (possibly in a VM or a container) must complete within a well-specified time (e.g., the Hybrid ARQ timeout for the acknowledgeof MAC packets), otherwise the connection risks to be interrupted.

Moreover, some of those virtualized functions might be computationally intensive, and risk to starve other services and functions if not properly handled. It is therefore important to properly schedule the virtualized functions and services so that each (virtualized) software component is provided with predictable and well-specified QoS (expressed as the probability to respect a max response-time), and cannot consume more than a given fraction of CPU time. While many modern operating systems provide some way to comply with the second requirement (for example, the Linux kernel provides a *throttling* mechanism), satisfying the first requirement is more difficult, and requires a more theoretically-funded approach. For example, CPU reservations [18] can be used, for which stochastic analysis can be applied to ensure that each component respects its timing requirements [19]. A reservation-based scheduler generally associates 2 scheduling parameters Q and P to each task, and guarantees that the task is allowed to execute for Q time units every P. Q is generally known as maximum budget, or runtime, and P is generally known as reservation period. While these techniques were traditionally implemented in research projects [20]–[22] and not supported by commonly used OSs, the mainline Linux kernel scheduler includes `SCHED_DEADLINE` [23] today, a CPU reservation mechanism based on EDF [24] [25].

B. Real-time scheduling of single-threaded NFV components

The `SCHED_DEADLINE` scheduling policy has been shown to be able to provide stable and predictable performance for a number of use-cases [26], including single-threaded NFV functions [19] [27]. For example, whenever submitting a Poissonian traffic to a packet processing server, it is possible to have a precise control on the QoS experienced by the processed packets, when scheduling the processing server under a `SCHED_DEADLINE` policy. Indeed, the latter allows for granting a budget Q every period of P time units to the functions, regardless of other possible workload deployed onto the same system and specific CPU. Indeed, it has been shown [19] that, under said scheduling parameters, and within reasonable ranges for the choice of the period P (which do not impose an excessive rate of context switches within the platform), a M/M/1 processing function with average arrival rate λ and average processing rate μ behaves approximatively like an equivalent M/M/1 queue with a server having a processing rate $\tilde{\mu}$ reduced to a fraction Q/P of the original one: $\tilde{\mu} = \frac{Q}{P}\mu$. This can be shown by observing the response-times distribution and statistics of such an M/M/1 system.

For example, we ran an experiment on a Freescale LS2085A RDB (ARM8) board, with 8 cores running at 1.8GHz and 16GB of RAM, equipped with a Yocto Linux distribution with a Linux kernel 4.1.8. We deployed a synthetic Poisson workload with parameters mimicking the typical strict requirements of a LTE processing function, where an aggregated input traffic with average rate of $15000 pkt/s$ was spread across 8 worker threads, resulting in an input rate at each server queue of $\lambda = 15000/8 = 1875 pkt/s$. We tuned our synthetic processing function for an average rate of $\mu = 5300 pkt/s$ (sequential processing rate obtained by the single server in isolation on a dedicated CPU). In this scenario, we expected a response-time $99th$ percentile below $2ms$. The latter timing requirement maps in a natural way into the period P to be used for the scheduling reservation, while the budget Q can be decided in function of the expected input workload and desired output QoS. Running a set of experiments with varying budget Q above the minimum stability threshold $Q/P \geq \lambda/\mu \simeq 35.4\%$, we get output response-time distributions whose main statistics are summarized in Figure 3, where the statistics of interest for the scenario are the higher-order percentiles, such as the $p99$.

Theoretical results [19] in this case lead to an (approximated) exponential distribution of the response-times with a cumulative distribution function (CDF) $F_R(\cdot)$ of:

$$F_R(t) = 1 - e^{-\left(\frac{Q}{P}\mu - \lambda\right)t}, \tag{1}$$

where, imposing the condition to respect a percentile of ϕ below the R^* threshold, gives us:

$$\frac{Q}{P} \geq \frac{1}{\mu}\left[\lambda - \frac{ln(1-\phi)}{R^*}\right]. \tag{2}$$

For example, for $\phi = 0.99$ and $R^* = 2ms$, we get $Q/P \geq 78.8\%$, coherently with the obtained experimental

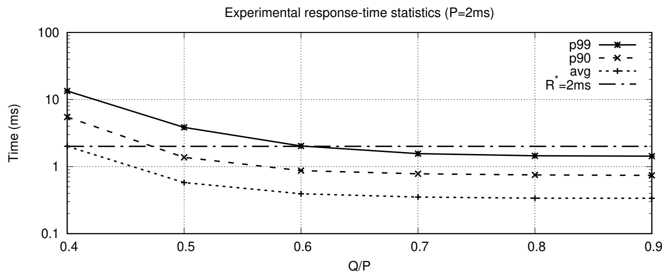

Figure 3. Various statistics on the experimental response-time distribution of a processing M/M/1 system scheduled using `SCHED_DEADLINE`.

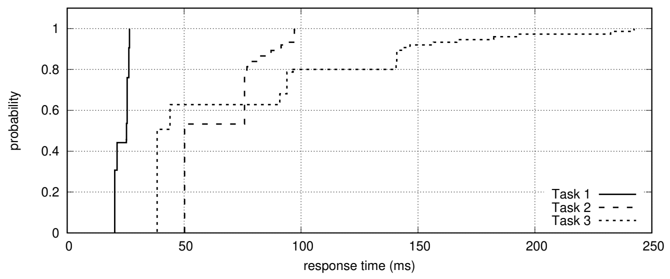

Figure 4. CDF of the response times for three real-time tasks scheduled in a properly dimensioned CPU reservation.

results in Figure 3, where we can see that the $2ms$ response-time percentile requirement is indeed met for $Q/P \geq 0.8$.

C. Real-time scheduling of multi-threaded NFV components

If NFV software components are executed in a KVM-based virtual machine [28], then it is possible to directly schedule the KVM vCPU threads with the `SCHED_DEADLINE` policy; however, to reduce the virtualization overhead people often tend to use container-based solutions (also known as “OS virtualization”), such as lxc [29] on Linux. In this case, the `SCHED_DEADLINE` policy cannot be directly used, because the same CPU reservation must be used to schedule multiple tasks (all the processes and threads in the container). Hence, a *hierarchical extension* for `SCHED_DEADLINE` has been developed, allowing to create two-levels scheduling hierarchies:

- at the root level, a CPU reservation (implemented as a `SCHED_DEADLINE` scheduling entities) schedules the various containers (basically, `lxc` VMs);
- at the second level (inside the container), a fixed priority scheduler (based on `SCHED_FIFO` or `SCHED_RR`) schedules the real-time tasks inside the container.

This solution allows to assign a *runtime* Q and a *period* P to a set of tasks scheduled with fixed priorities, guaranteeing that the tasks will never consume a fraction of the CPU time larger than Q/P, but also allowing to provide performance guarantees to the tasks. Such guarantees can be provided by using hierarchical real-time analysis [30].

As an example, three periodic real-time tasks $(15ms, 70ms)$, $(33ms, 150ms)$ and $(27ms, 250ms)$ (where (C_i, P_i) indicates a task with execution time C_i and period P_i) have been scheduled with fixed priorities (assigned according to Rate Monotonic [24]) inside an lxc container.

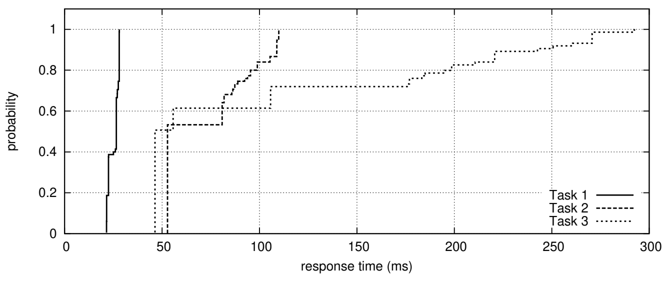

Figure 5. CDF of the response times for three real-time tasks scheduled in a smaller CPU reservation.

When the container is associated to a runtime $Q = 10ms$ and a period $P = 100ms$ and one single CPU core, the container does not consume more than 10% of the core time, showing that the implemented scheduling algorithm provides temporal protection (even if the three tasks require $15/70 + 33/150 + 27/250 = 54.23\%$ of the CPU time). When the runtime and the period are changed according to hierarchical real-time analysis so that the three tasks have response times smaller than their periods, the measured response times are distributed as indicated in Figure 4. As it can be noticed, all the temporal constraints (response time smaller than period) are respected, consistently with the theory; hence the scheduler works correctly.

The implemented algorithm supports multiple CPUs / cores and allows to provide performance guarantee to software modules scheduled with a global fixed priority algorithm [31]. It uses the Linux "control groups" software interface, and is hence usable to serve lxc-based VMs. Basically, the configuration for an lxc VM is extended with two additional attributes indicating the runtime Q and the period P assigned to the real-time tasks running inside the VM. These two low-level parameters are exported to Nova, and OpenStack has to be modified to properly map high-level descriptions into them. A first step in this direction is represented by RT-OpenStack [32], but more work is still needed.

It is also important to properly dimension the two parameters Q and P: while hierarchical schedulability analysis provides the theory needed to exactly control all the response times, this kind of analysis can result in an over-allocation of system resources and a more relaxed approach can be used as shown in Section III-B. For example, returning to the previous example the runtime and period dimensioned according to hierarchical scheduling analysis are $Q = 10ms$ and $P = 15ms$; if the runtime assigned to the lxc container is decreased to $Q = 9ms$, then the distribution of the response times changes as shown in Figure 5. This is consistent with the fact that a runtime $Q = 9ms$ does not guarantee that all the response times of all the tasks are smaller than the period, but shows how changing the scheduling parameters allow to control the response times.

D. MANO descriptors and real-time reservations

Standard MANO descriptors from the TOSCA specification allow one to specify processing requirements of a VDU to be deployed, in the form of properties within the `nfv_compute` descriptor, which is of type `tosca.datatypes.compute_properties`. This type allows for the specification of such properties as `num_cpus`, `mem_size`, `cpu_allocation`. The latter is a map allowing the specification of:

- `socket_count`, `core_count`, `thread_count`: additional details on the desired topology of the needed computational elements;
- `cpu_affinity`: differentiates between virtual cores pinned down onto dedicated physical cores, or left free to migrate among shared physical cores;
- `thread_allocation`: specifies how to map virtual cores onto hyper-threads of the underlying physical host.

```
topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            disk_size: 10 GB
            mem_size: 2048 MB
            num_cpus: 2
            cpu_allocation:
              cpu_policy: reservation
              cpu_runtime: 60 ms
              cpu_period: 100 ms
```

Figure 6. Proposed VDU template

We are working on extending the `cpu_allocation` map of type `tosca...CPUAllocation`, adding additional properties that allow for the specification of our scheduling parameters, namely a `cpu_runtime` and a `cpu_period`, used to instantiate a resource reservation within the underlying hypervisor (or host OS, in the case of Linux+KVM) scheduler.

A sample VDU template showing our proposed syntax is visible in Figure 6, where we are requiring the use of an underlying container with 2 cores, scheduled under a real-time reservation of $60ms$ every $100ms$.

In case heterogeneous processing physical machines are available within the infrastructure, we plan to use a processing requirement specification (the runtime value) in terms of some generic architecture-independent unit, for example expressed in terms of *CPU capacity* [33], a metric used in the Linux kernel scheduler in the context of heterogeneous platforms, e.g., big.LITTLE^{TM} ones.

An implementation of the proposed mechanism, based on Tacker with an OpenStack binding as VIM, exploiting our hierarchical variant of the SCHED_DEADLINE real-time scheduler for Linux, is under way at the moment.

IV. CONCLUSIONS

The current standard for NFV orchestrators manifests limitations in the context of the challenging scenarios posed by VRAN. This paper has shown that the level of detail in describing computational resources is not sufficient to efficiently allocate them while providing strong real-time processing guarantees. To address this issue, an extension to the NFV descriptor has been proposed to exploit experimental reservation capabilities available in a hypervisor CPU scheduler. This solution enables the seamless integration of current infrastructures with dedicated nodes able to deal with timing requirements of specific NFV components.

ACKNOWLEDGMENT

This work was partially funded by Ericsson and has been partially supported by the RETINA Eurostars Project E10171.

REFERENCES

[1] C. Vitucci and A. Larsson, "Flexible 5G Edge Server for Multi Industry Service Network," International Journal on Advances in Networks and Services, vol. 10, no. 3-4, 2017, pp. 55–65, ISSN: 1942-2644.

[2] ITU-R, "IMT vision - framework and overall objectives of the future of IMT for 2020 and beyond," International Telecommunication Union, Recommendation I.2083-0, September 2015.

[3] NGMN Alliance, "NGMN 5G White Paper," Tech. Rep., February 2015.

[4] R. Ravindran, A. Chakraborti, S. O. Amin, A. Azgin, and G. Wang, "5G-ICN: Delivering ICN Services over 5G Using Network Slicing," Comm. Mag., vol. 55, no. 5, May 2017, pp. 101–107.

[5] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," IEEE Communications Magazine, vol. 55, no. 5, May 2017, pp. 94–100.

[6] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," IEEE Communications Magazine, vol. 55, no. 5, May 2017, pp. 80–87.

[7] T. Cucinotta, L. Abeni, M. Marinoni, and C. Vitucci, "The importance of being OS-aware in performance aspects of Cloud Computing research," in Proceedings of the 8th International Conference on Cloud Computing and Services Science, March 2018, pp. 626–633.

[8] SCTE, "SCTE analysis of available Energy 2020 participating MSO data," Brochure, 2016. [Online]. Available: http://www.telespazio.it/docs/brodoc/GCC_eng.pdf

[9] "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Study on New Radio Access Technology; Radio Access Architecture and Interfaces (Release 14) – 3GPP TR 38.801 V1.0.0," 3GPP Organizational Partners, Tech. Rep., December 2016.

[10] "3rd Generation Partnership Project; Transport requirement for CU-DU functional splits options; R3-161813 (document for discussion)," in 3GPP TSG RAN WG3 Meeting 93, August 2016.

[11] F. Giannone, H. Gupta, D. Manicone, K. Kondepu, A. Franklin, P. Castoldi, and L. Valcarenghi, "Impact of RAN Virtualization on Fronthaul Latency Budget: An Experimental Evaluation," in Proceedings of the Workshop on 5G Test-Beds and Trials Learnings from implementing 5G (5G-Testbed), December 2017, pp. 1–5.

[12] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," SIGCOMM Comput. Commun. Rev., vol. 44, no. 5, October 2014, pp. 33–38.

[13] K. C. Garikipati, K. Fawaz, and K. G. Shin, "Rt-opex: Flexible scheduling for cloud-ran processing," in Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies, ser. CoNEXT '16. New York, NY, USA: ACM, 2016, pp. 267–280.

[14] A. Brogi and J. Soldani, Matching Cloud Services with TOSCA. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 218–232.

[15] OASIS, "TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0," Tech. Rep., May 2017. [Online]. Available: http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/csd04/tosca-nfv-v1.0-csd04.pdf

[16] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, vol. 55, no. 3, October 2012, pp. 38–42.

[17] "Nova," visited on March 12, 2018. [Online]. Available: https://www.openstack.org/software/releases/ocata/components/nova

[18] C. W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves: Operating Systems Support for Multimedia Applications," in Proceedings of the IEEE International Conference on Multimedia Computing and Systems, May 1994, pp. 90–99.

[19] T. Cucinotta, M. Marinoni, A. Melani, A. Parri, and C. Vitucci, "Temporal Isolation Among LTE/5G Network Functions by Real-time Scheduling," in Proceedings of the 7th International Conference on Cloud Computing and Services Science, April 2017, pp. 368–375.

[20] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards real-time hypervisor scheduling in Xen," in 2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT), October 2011, pp. 39–48.

[21] T. Cucinotta, G. Anastasi, and L. Abeni, "Respecting Temporal Constraints in Virtualised Services," in 33rd Annual IEEE International Computer Software and Applications Conference, vol. 2, July 2009, pp. 73–78.

[22] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari, "Hierarchical Multiprocessor CPU Reservations for the Linux Kernel," in Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009), June 2009, pp. 1–8.

[23] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," Software: Practice and Experience, vol. 46, no. 6, 2016, pp. 821–839.

[24] C. L. Liu and J. Layland, "Scheduling alghorithms for multiprogramming in a hard real-time environment," Journal of the ACM, vol. 20, no. 1, January 1973.

[25] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in Proceedings of the IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998, pp. 4–13.

[26] J. Lelli, D. Faggioli, T. Cucinotta, and G. Lipari, "An experimental comparison of different real-time schedulers on multicore systems," Journal of System Software, vol. 85, no. 10, Oct. 2012, pp. 2405–2416.

[27] C. Vitucci, J. Lelli, A. Parri, and M. Marinoni, "A Linux-based Virtualized Solution Providing Computing Quality of Service to SDN-NFV Telecommunication Applications," in Proceedings of the 16th Real Time Linux Workshop (RTLWS 2014), Dusseldorf, Germany, October 2014, pp. 1–9.

[28] "Kernel-based Virtual Machine (KVM)," visited on March 12, 2018. [Online]. Available: http://www.linux-kvm.org

[29] "Linux Containers (lxc)," visited on March 12, 2018. [Online]. Available: http://www.linuxcontainers.org/lxc

[30] R. I. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in 26th IEEE International Real-Time Systems Symposium (RTSS'05), December 2005, pp. 10 pp.–398.

[31] E. Bini, M. Bertogna, and S. Baruah, "Virtual multiprocessor platforms: Specification and use," in 2009 30th IEEE Real-Time Systems Symposium, December 2009, pp. 437–446.

[32] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. X. Phan, I. Lee, and O. Sokolsky, "RT-Open Stack: CPU Resource Management for Real-Time Cloud Computing," in 2015 IEEE 8th International Conference on Cloud Computing, June 2015, pp. 179–186.

[33] M. Rasmussen, "Energy cost model for energy-aware scheduling." [Online]. Available: https://lkml.org/lkml/2015/7/7/754