Flyweight Network Functions for Network Slicing in IoT

Clovis Anicet Ouedraogo, Samir Medjiah, Christophe Chassot, José Aguilar

Flyweight Network Functions for Network Slicing in IoT

Clovis Anicet Ouedraogo*, Samir Medjiah*†, Christophe Chassot*‡ and Jose Aguilar§
Univ. Toulouse, * CNRS-LAAS, † UPS, ‡ INSA, F-31400 Toulouse, France
§ Universidad de Los Andes, CEMISID, Mérida, Venezuela
{*ouedraogo, medjiah, chassot*}@laas.fr, *aguilar*@ula.ve

***Abstract*—This paper proposes an approach for the network slicing provisioning for Internet of Things (IoT) platforms based on the concept of flyweight network functions (fNF). The network slicing is a main characteristic of the 5G networks that our proposal extends to IoT platforms, in order to support the most diverse QoS requirements. The paper defines the fNF concept and presents its utilization in the context of network slices provisioning for IoT applications having QoS requirements and sharing the same IoT platform. Additionally, we present a use case and a comparison with previous works.**

***Index Terms*—Network Function, Flyweight Network Function, Network Slicing, Internet of Things (IoT).**

I. Introduction

The potential number of connected devices, the massive amount of data these devices generate, and the growing complexity of the Internet of Things ((IoT) infrastructure, is a high challenge to build future IoT applications. Such applications exploit a set of characteristics, such as heterogeneity, interoperability, dynamicity, and geographical distribution. Additionally, the quality of service (QoS) notion is particularly relevant in this context, because it affects user experience, resource consumption, and energy efficiency, among other things. To cope with the QoS problem, solutions have to be defined not only at the network level but also within the IoT platforms (typically based on intermediate gateways and server between applications and things) on which the applications are distributed (see Fig. 1).

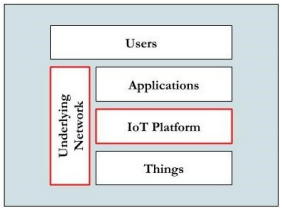

Fig. 1. IoT context.

Parallelly, the fifth generation (5G) networks will enable a fully mobile and connected society, with new applications, many of which being still unknown [1]. To realize the full potential of 5G networks, it is necessary to rethink the way to design architecture of the underlying communication infrastructure to support the variety of requirements for latency, throughput, capacity, and availability of these applications. New concepts related to design and management of such architectures are thus required. Particularly, for the design of IoT platform architecture et more generally for the future 5G-based communication platforms, a promising concept is the network slicing. A network slice is a set of network functions, which logically create a dedicated virtual network that satisfies the specific requirements of an application [2]. Network slicing consists in provisioning and managing network slices on demand. Network slicing enables multiple independent virtual networks over a same communication infrastructure (such as an IoT platform), which is shared by several applications. A IoT slice consists in a set of the composition of network functions - NF (e.g. broker, stream processor ...) which are dynamically deployed in the IoT platform with the aim to tackle a given set of QoS requirements. Most of the time, these NF are mainly virtualized, thanks to the use of new virtualization and network programmability technologies such as Network function virtualization (NFV) and Software-defined networking (SDN). However, this virtualisation-based approach may be inappropriate in some domains, typically the IoT, due to the limited resources of the targeted equipements (i.e. on which the NFs are deployed). In this paper, we propose an approach based on the notion of flyweight network functions (fNF) for the implementation of the network slicing concept to IoT platforms. The rest of this paper describes the fNF concept and its utilization in the context of network slicing for IoT (section IV), a use case (section V), and finally a comparison with other works (section VI). Next sections II and III precise the background and key issues, and detail the limitations of virtualisation-based network slicing for the IoT.

II. Background And Key Issues

A. The IoT applications QoS requirements

The IoT is based on connection of billions of objects, going beyond laptops and smartphones, including connected cars, wearables, smart cities, smart homes, among others. According to [3], connected devices will reach more than 20 billion objects in 2010, with unknown new connections, such as connected products or connected business processes, which are very important in the Industry 4.0. In this context, new IoT-based business applications are going to emerge in diverse domain such as smart health/ cities/ transport/ etc. Those future applications will have specific QoS needs (bounded response time, availability, etc.) that will have to be considered by the IoT platforms such as the one promoted by the OneM2M

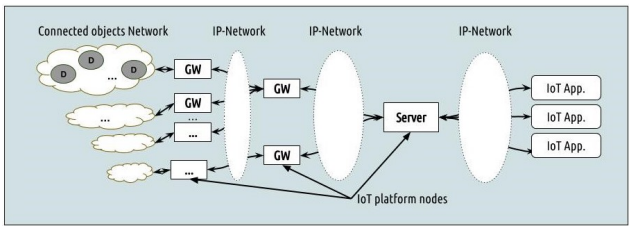

Fig. 2. IoT platform QoS bottleneck.

consortium [4]. Such platforms (see Fig. 2) are formed by heterogeneous infrastructure nodes, typically server and gateways having different resources / capabilities. These nodes represent potential bottlenecks with respect to the QoS, for instance when a too high number of application level requests have to be processed by a given node [5].

B. Existing solutions to tackle QoS in IoT limitations

Several solutions have also been proposed that address the QoS issue at the platform level [6] [7] [8]. Most of them are based on a service differentiation principle that allows processing the requests differently, depending on their priority. Here, the services managed on each node are the ones that have been provided at the initialisation of the platform, such as traffic marker/shaper, message or task scheduler, etc. [9]. This principle is adequate when the available services allows tackling all the application requirements. However, it becomes inadequate when a service is not existing on a node, or when the computing resources are not enough sufficient. The network slicing is a new concept that allows answering this limitation.

C. Network slicing concept

The maturity of technologies such as SDN and NFV, allows to consider new network concepts such as slicing networks that will allow a more flexible management of networks. In this section we present, this concept and the limits of its current implementations especially for IoT.

1) Network Slicing: The ITU-T [10] defines a network slice as a logical network that provides specific network capabilities and network characteristics. The (network) slicing consists in building slices on demand. It has been initially thought to share resources on a communication infrastructure. It is now more and more considered to perform QoS provisioning. The slicing concept is based on the notions of network function (NF) and NF Service chaining. Basically, a NF is a processing function applied on a given data traffic (e.g. a delaying function applied on IP packets). More formally, a NF is defined by 3GPP as a processing function in a network, which has defined functional behavior and interface. NF Service chaining represents a path taken by traffic routed through several NFs to benefit from a network service (NS). For instance, a NS composed of Firewall, IDS and Parental control Nfs.

2) Limits of the existing slicing implementations: In general, the instantiation of network slices is in the form of VNFs via virtualization containers (VM / CNT). The use of virtualization containers induces a virtualization overhead [11] potentially problematic for some IoT deployment targets (e.g. RPi used as IoT gateway) with very limited resources. Moreover for some NF, by their size, utility, to be instantiated in the form of VNF can be counterproductive. This type of NF is very similar to the anti pattern SOA known as Nanoservice [12]. Also, this method of instantiation of NF does not cover heterogeneity of the future 5G networks, i.e. the underlying networks will be both classical and cloud-enabled. The concept of network slicing, as it is conceived currently, is based on cloud-type infrastructure (to allow the deployment of VNFs) and will be hardly usable to achieve end-to-end slices, i.e. connecting data producers and consumers. Our proposal aims to extend the network slicing to environments that do not support virtualization, through the concept of Flyweight Network Functions.

III. Flyweight Network Functions

In this section, we propose to extend NFs instantiation as implemented today to include a new category. Indeed, different studies, like Nandugudi et al. [13], show that the virtualization techniques used nowadays, generally induce a large consumption of resources. For example, deploying a VNF as a standard linux VM requires a minimum[1] of 256 MiB RAM, a 300 MHz x86 processor, and 1.5 GB of disk space. Deploying this VNF as a container requires a minimum[2] of 29 MiB of disk space. This requirement strongly limits the number of VNFs deployable on a host, and also reduces the number of compatible hosts (i.e. which can host VNFs) by their limited resources. For most IoT equipment, it is difficult to host such NFs. To tackle this issue, we propose a new NF instance class that we call Flyweight Network Features (fNF). Before introducing this new concept, we first discuss about the NF isolation issue.

A. About isolation of NFs in IoT

One of the features highlighted in current NFV platform deployments is the isolation so that NFs running on the same (physical) hardware do not interfere with each other from two standpoints [14]: security and performance. To provide this functionality, virtual machines (VMs) and containers (CNTs) are used. However, as presented in the following section, using these techniques induces an overhead. In this paper, we claim that this feature can be discussed and ignored in some areas, such as IoT. Indeed, when it comes to:

- Security: assuming that the slice provider is the only actor capable of building slices, the isolation can be ignored and the burden of protecting the source code and the traffic of the NFs will be guaranteed upstream by integrity verification techniques (before deployment of NFs) and encryption of the traffic.
- Performance: considering a slice as a chain of NFs offering services with well-defined characteristics, the management of the overall performance of the slice will make it possible to balance the expected ”characteristics” by taking into account the workload of hosts hosting NFs.

[1]help.ubuntu.com/community/Installation/SystemRequirements
[2]hub.docker.com/r/_/ubuntu/

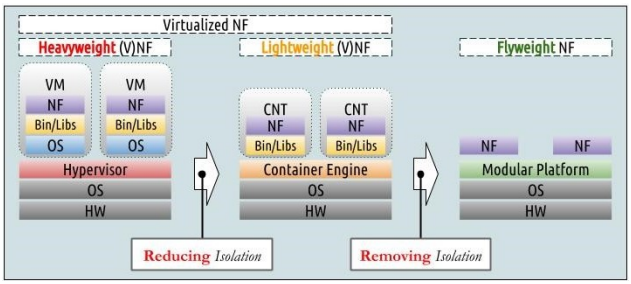

Fig. 3. NFs instantiation frameworks.

Thus, by removing the isolation techniques between NF:

- we lose: level of security, performance guarantee;
- we win: removal of the overhead (resource, deployment time, etc.), reduction of the complexity of the host platform of NFs, increase of the possible number of hosts of NF.

Considering, under certain conditions/domains, the isolation as a non mandatory functionality for the installation of the NFs, we propose the concept of fNF in order to allow network slicing for the field of IoT.

B. Presentation of the fNF concept

In the instantiation of VNFs, we distinguish, according to their resource consumption and the overhead related to virtualization, two subcategories (see Fig. 3): heavy VNFs instantiated by VMs, and light VNFs instantiated by containers. Flyweight NF are deployable network functions in the form of software modules. Their deployment induces almost no virtualization overhead and is adapted to IoT equipments. Note: this is due to the NF size which is around a few hundred KiB. Formally, a fNF is defined as an instantiation of NF having the following properties:

- P1: a fNF is the instantiation of a NF in the form of a software module without virtualization overhead;
- P2: a fNF is an implementation of a NF without isolation in the User space, just like an application;
- P3: a fNF is dynamically deployable / deletable / editable / configurable;
- P4: a fNF is instantiable on a compatible platform for fNFs deployment, typically a modular framework.

What a fNF is not:

- N1: a VNF, because it is not instantiated as a virtualization container (CNT/VM) but as a software module;
- N2: a PNF, because it is not instantiated on hardware built for this unique (dedicated) use.

The fNFs are adapted:

- for network slice deployment;
- for network function chaining;
- for environments with constraints in resources like the IoT gateways (ex: RPi, Odroid, etc.).

The fNFs will not be adapted:

- for highly security-sensitive environments (because the isolation between NF is not assured);

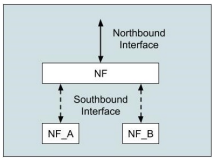

Fig. 4. Model of 3GPP network functions.

- for complex NFs, i.e. whose source code is several hundred KiB and whose failure will cause a general failure of the host.

C. A Framework of Modular Flyweight Network Functions

Several works exist in NF modeling. In this paper, we consider the model proposed by the 3GPP [15] shown in Fig. 4.In this model, a NF has 2 types of interface: a first one offering a functional management to an external entity and another one for data exchange.

According to this model, for the implementation of fNFs, we propose the architecture shown in Fig. 5:

This architecture is composed of several components that communicate through specific interfaces. Components:

- Function Management represents all the functionalities needed to configure fNFs. This is a specific component of the host. Its role is to configure the fNFs and the Service Function Chaining component with the slicing policy it has received from the Slice Controller through the service controller interface (SCi);
- Service Function Chaining (SFC) deals with the interconnection of fNFs between them. It performs this task based on the configuration received from the Function Management;
- Modular Platform is the fNFs execution platform; it implements a complete and dynamic component model for the fNFs deployment. It allows to remotely install, start, stop, update, and uninstall fNFs on demand. CCM, OSGi, Vert.x are examples of modular execution platforms that allow deploying and managing (non virtualized) software components [16].

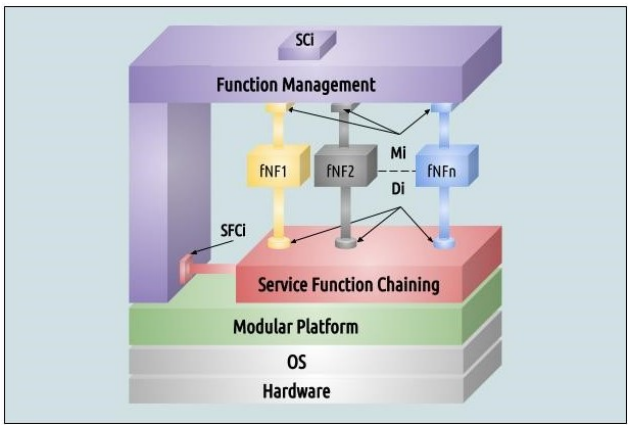

Fig. 5. Architecture of an compatible-host.

Interfaces:

- Mi: Management interface (north interface). This is the interface that allows the Function Management to manage the fNFs;
- Di: Data interface (south interface). This interface allows the fNFs to exchange data through the SCF;
- SCFi: Service Chaining Function interface. It allows the Function Management component to configure routing in the managed host;
- SCi: Slice Controller interface. This is the external interface of the framework; it is used by the Slice Controller for setting up and managing slices.

Several frameworks exist for the software implementation of this architecture [16]. We propose the use of the OSGi framework for building fNFs as OSGi bundles.

D. Network Slices provisioning

With respect to the concept that we presented below, the deployment of a slice is done in 5 steps (Fig. 6).

Fig. 6. Slices provisioning steps.

The slice controller:

0) retrieves the QoS requirements of IoT applications;
1) analyzes these needs and choose a network service consisting of V/fNFs with properly defined characteristics;
2) packages NFs according to hosts that can meet the desired characteristics;
3) instantiates these V/f NFs on the selected hosts;
4) implements the policy associated with the network service, i.e.: (i) configures the deployed fNFs; and (ii) configures the policy associated with the slice on the SFC.

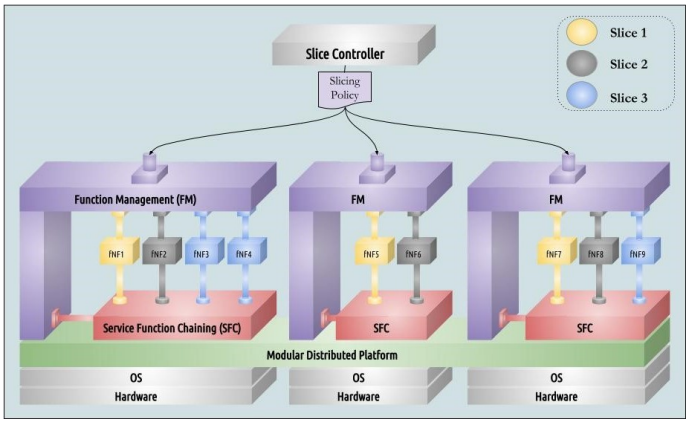

Fig. 7. Multi-hosts (distributed) management architecture.

The consistency of the multi-node slicing is ensured by an external Slice Controller, as is shown in Fig. 7.

IV. Use Case

As presented above, the fNFs can be used at the different protocol layers. In this section, we present a case of utilization of fNFs for creating network slices at the platform level of the IoT domain.

A. Presentation of the use case

The considered application is wildfire monitoring. We also consider an environment composed of a Cloud, a Fog node, and two IoT gateways. Relationships between these elements are shown in Fig. 8.

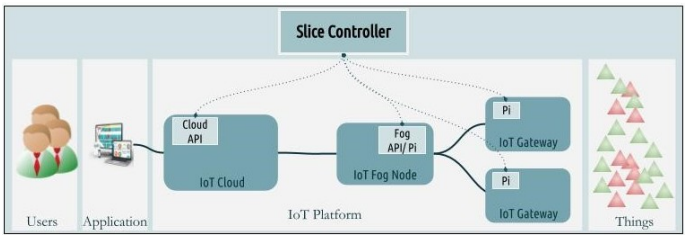

Fig. 8. Architecture of the case study.

A user, typically a fire brigade, request a given platform level slice from a Slice Controller. The controller, through a set of successive tasks, sets up this slice using VNFs, but also fNFs.

B. Characteristics of the requested slice

The requested slice has the following functional and non functional (i.e. QoS oriented) characteristics:

- allowable latency: 10ms
- availability: 90
- services: Data Collection, Stream processing, Data Storage
- service life: 7h.

C. Slice construction

Step 0 & 1: Upon receipt of the user's request, the slice controller selects in a service catalog the network service (NS) to offer for such a request. This NS is composed of nine NFs (Fig. 9): four brokers, four stream processors, and a database.

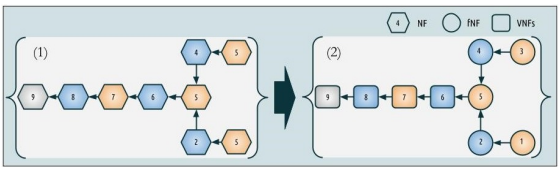

Fig. 9. Mapping Network Services to available hosts.

Step 2: The Slice Controller then packages the selected NFs into VMs or CNTs (for VNF) and Components (for fNF). This selection of packages is done with regard to the resources available in the environment. As introduced in section A, we have two gateways with limited resources that can only host fNFs, a Fog node that can host both fNFs and VNFs, and a cloud able to host VNFs. From this observation, the Slice Controller completes the NS with the associated packages information (Fig. 10):

- each gateway: an fNF broker and an fNF Stream processor,
- the Fog node: an fNF broker and a VNF Stream processor,
- the Cloud: a VNF broker, a VNF Stream processor and VNF Storage.

Step 3: Once the NS is built, the V/f NFs are deployed on the selected hosts as shown in Fig. 10.

Step 4: At the end of the deployment, the V/f NFs are configured with the slicing policy associated with the NS. The slice is then ready to be used, and a positive response is sent to the user having requested the slice.

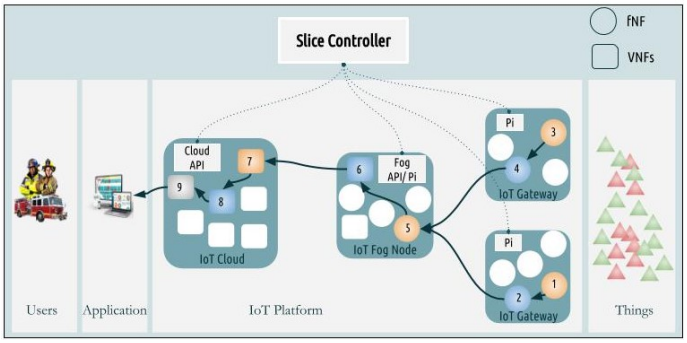

Fig. 10. Slice deployment.

V. Related Works

Our approach to reduce overhead induced by VM/CNT in areas such as IoT, has like related works in the literature classified into two major groups: the first group we call User-space flow-level network function frameworks, which does not use virtualization or containerization as an isolation technique for NFs; and the second group that uses these techniques but trying to reduce their overhead, which we call Light NFV frameworks.

In the first group we find frameworks like [17], but also other frameworks that isolate instances of NF with techniques such as Intel Software Guard Extensions (SGX), Zero Copy Soft Isolation (ZCSI), etc. In the early 2000s, Kohler et al. have proposed a software architecture for the construction of modular and extensible routers [17]: The Click Router. This architecture later inspired CoMb [18], ClickMB [19], ClicNF [20], NetServ [21] and Click-Up [22]. From a different inspiration, Eden [23] is an architecture for implementing network functions at end hosts with minimal network support. Eden comprises three components, a centralized controller, an *enclave* at each end host, and Eden-compliant applications called stages. To implement network functions, the controller configures stages to classify their data into messages and the *enclaves* to apply action functions based on a packets class. Clayman et al. offer VLSP [24], propose a framework that provides a complete environment from the protocol stack up to the service management level, including a tailor-made monitoring facility. Duan et al. have proposed LightBox [25], a system for secure and trustworthy middlebox outsourcing, built on top of SGX. LightBox allows enterprise to outsource middlebox functionality with minimal development and deployment effort. Poddar et al. have proposed SafeBricks, an extension of NetBricks [26], a system that shields generic network functions (NFs) from an untrusted cloud. SafeBricks [27] ensures that only encrypted traffic is exposed to the cloud provider, and preserves the integrity of both traffic and the NFs. At the same time, it enables clients to reduce their trust in NF implementations by enforcing least privilege across NFs deployed in a chain. Boucher et al. [28] proposed, a novel design for providing functions as a service (FaaS): cold launch times in microseconds that enable even finer-grained resource accounting and support latency-critical applications. Their proposal is to eschew much of the traditional serverless infrastructure in favor of language-based isolation.

In the second group, we find NFV frameworks that try to minimize the overhead induced by the isolation techniques they use. Palkar et al. have proposed E2 [29], a framework for NFV packet processing. It provides the operator with a single coherent system for managing NFs, while relieving developers from having to develop per-NF solutions for placement, scaling, fault-tolerance, and other functionalities. Riggio et al [30] have proposed 5G-EmPOWER, a Multi-access Edge Computing Operating System supporting lightweight virtualization and heterogeneous radio access technologies. Cziva et al. [31] have proposed Glasgow Network Functions (GNF), an NFV platform built on top of standard Linux containers. Yasukata et al. have proposed HyperNF [32] a high performance NFV framework aimed at maximizing server performance when concurrently running large numbers of NFs. HyperNF implements hypercall-based virtual I/O, placing packet forwarding logic inside the hypervisor to significantly reduce I/O synchronization overheads. Gallo et al. have proposed CliMBOS [33], a scalable NFV-based solution, as a novel approach that satisfies the stated requirements for user-centric support of IoT devices. The table 1 compares the existing works according to 3 criteria: i) the isolation technique used, ii) support for an orchestration of NFs on different nodes, iii) the virtualization overhead. In this table, we consider that the VM has a very high overhead, the CNT high overhead, the JVM medium overhead, and non isolation or the use of techniques like SGX, language-based produce low overhead. The main differences between all these frameworks and our proposal concern the isolation of NFs and the distribution of the framework. Our approach is the only one among the approaches without isolation between NF in a distributed framework.

VI. Conclusion

Network Slicing is a promising concept in response to QoS and other needs of future communication infrastructures. In particular, future IoT platforms will benefit from the implementation of this concept, which allows addressing the limitations of more traditional QoS management approaches. In this paper, we have shown how to extend the basic network slicing tools that are VNFs (implemented in the form of VM or CTN) to overcome the resource limitations of the IoT environments on which to deploy the NF constituting the slices. We have defined and formalized the concept of flyweight NF (fNF), which allows to build slices not only on VNFs but also on software

TABLE I
COMPARISON OF THE NF FRAMEWORKS.

Framework	Isolation	Distributed	Overhead
Eden [23]	VM	No	Very high
VLSP [24]	JVM	Yes	Medium
LightBox [25]	SGX	No	Low
Click-based: CoMb [18], ClickMB [19], ClicNF [20], Click-UP [22], NetServ [21]	None	No	Low
NetBricks [26]	ZCSI	No	Low
SafeBricks [27]	SGX and Language	No	Low
Putting the Micro Back in Microservice [28]	Language	No	Low
E2 [29]	VM	Yes	Very High
EmPOWER [30]	CNT	Yes	High
GNF [31]	CNT	Yes	High
HyperNF [32]	VM	No	Very High
CliMBOS [33]	VM	No	Very High
fNF	None	Yes	Low

components that can be integrated into modular environments, out of context virtualization. We then defined the steps sequence to be followed in constructing a slice including VNFs and fNFs for a case study of fire monitoring. As a final contribution to this paper, we have made a comparison of the concept of fNF with all these equivalents in the literature. The perspectives of our current work deal: on the one hand, on the design of a testbed for the deployment of slices based on VNfs and fNFs, and on the other hand, on an evaluation of the performances induced by the use of the fNF concept, compared to the VNF one.

REFERENCES

[1] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, pp. 1–125, 2015.

[2] P. Hedman, "Decription of network slicing concept (ngmn 5g p1)," *NGMN (Next Generation Mobile Networks) Alliance*, 2016.

[3] M. Hung, "Leading the iot, gartner insights on how to lead in a connected world," *Gartner Research*, pp. 1–29, 2017.

[4] oneM2M, *oneM2M Requirements*, 9 2016. Release 2.

[5] Y. Banouar, T. Monteil, and C. Chassot, "Analytical model for adaptive qos management at the middleware level in iot," in *Computers and Communications (ISCC), 2017 IEEE Symposium on*, pp. 1201–1208, IEEE, 2017.

[6] A. Agirre, J. Parra, A. Armentia, *et al.*, "Qos aware middleware support for dynamically reconfigurable component based iot applications," *International Journal of Distributed Sensor Networks*, vol. 12, no. 4, p. 2702789, 2016.

[7] W. B. Heinzelman, A. L. Murphy, H. Carvalho, *et al.*, "Middleware to support sensor network applications," *IEEE network*, vol. 18, no. 1, pp. 6–14, 2004.

[8] S. Yu, C. Shih, J. Hsu, *et al.*, "Qos oriented sensor selection in iot system," in *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom/CPSCom), IEEE*, pp. 201–206, IEEE, 2014.

[9] C. A. Ouedraogo, S. Medjiah, and C. Chassot, "A modular framework for dynamic qos management at the middleware level of the iot: Application to a onem2m compliant iot platform," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2018.

[10] ITU, *Terms and definitions for IMT-2020 network*, 2017. Recommendation ITU-T Y.3100.

[11] Z. Li, M. Kihl, Q. Lu, *et al.*, "Performance overhead comparison between hypervisor and container based virtualization," in *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*, pp. 955–962, IEEE, 2017.

[12] A. Rotem-Gal-Oz, "Services, microservices, nanoservicesoh my," 2016.

[13] A. Nandugudi, M. Gallo, D. Perino, *et al.*, "Network function virtualization: through the looking-glass," *Annals of Telecommunications*, vol. 71, no. 11-12, pp. 573–581, 2016.

[14] A. Panda, *A New Approach to Network Function Virtualization*. PhD thesis, UC Berkeley, 2017.

[15] 3GPP, "Study on architecture for next generation system," Tech. Rep. 23799, 3GPP, 2016. V14.0.0.

[16] I. Crnkovic, S. Sentilles, A. Vulgarakis, *et al.*, "A classification framework for software component models," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 593–615, 2011.

[17] E. Kohler, R. Morris, B. Chen, *et al.*, "The click modular router. acm transactions on computer systems," *ACM Transactions on Computer Systems*, vol. 18, no. 3, p. 263, 2000.

[18] V. Sekar, N. Egi, S. Ratnasamy, *et al.*, "Design and implementation of a consolidated middlebox architecture," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 24–24, USENIX Association, 2012.

[19] R. Laufer, M. Gallo, D. Perino, *et al.*, "Climb: Enabling network function composition with click middleboxes," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 4, pp. 17–22, 2016.

[20] M. Gallo and R. Laufer, "Clicknf: a modular stack for custom network functions," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, USENIX Association, 2018.

[21] J. W. Lee, R. Francescangeli, J. Janak, *et al.*, "Netserv: active networking 2.0," in *Communications Workshops (ICC), 2011 IEEE International Conference on*, pp. 1–6, IEEE, 2011.

[22] J. Wang, Y. Huang, H. Qi, *et al.*, "Click-up: Towards software upgrades of click-driven stateful network elements," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, pp. 117–119, ACM, 2018.

[23] H. Ballani, P. Costa, C. Gkantsidis, *et al.*, "Enabling end-host network functions," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 493–507, ACM, 2015.

[24] S. Clayman, L. Mamatas, and A. Galis, "Efficient management solutions for software-defined infrastructures," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pp. 1291–1296, IEEE, 2016.

[25] H. Duan, X. Yuan, and C. Wang, "Lightbox: Sgx-assisted secure network functions at near-native speed," *arXiv preprint arXiv:1706.06261*, 2017.

[26] A. Panda, S. Han, K. Jang, *et al.*, "Netbricks: Taking the v out of nfv.," in *OSDI*, pp. 203–216, 2016.

[27] R. Poddar, C. Lan, R. A. Popa, *et al.*, "Safebricks: Shielding network functions in the cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18), Renton, WA*, 2018.

[28] S. Boucher, A. Kalia, D. Andersen, *et al.*, "Putting the micro back in microservice," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 645–650, USENIX Association, 2018.

[29] S. Palkar, C. Lan, S. Han, *et al.*, "E2: a framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, pp. 121–136, ACM, 2015.

[30] R. Riggio, M. K. Marina, J. Schulz-Zander, *et al.*, "Programming abstractions for software-defined wireless networks.," *IEEE Trans. Network and Service Management*, vol. 12, no. 2, pp. 146–162, 2015.

[31] R. Cziva and D. P. Pezaros, "Container network functions: bringing nfv to the network edge," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 24–31, 2017.

[32] K. Yasukata, F. Huici, V. Maffione, *et al.*, "Hypernf: building a high performance, high utilization and fair nfv platform," in *Proceedings of the 2017 Symposium on Cloud Computing*, pp. 157–169, ACM, 2017.

[33] M. Gallo, S. Ghamri-Doudane, and F. Pianese, "Climbos: A modular nfv cloud backend for the internet of things," in *New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on*, pp. 1–5, IEEE, 2018.