

# Policy-Driven vCPE Through Dynamic Network Service Function Chaining

Vitor A. Cunha  
Instituto de Telecomunicações  
3800-193 Aveiro, Portugal  
Email: vitorcunha@av.it.pt

Igor D. Cardoso  
Instituto de Telecomunicações  
3800-193 Aveiro, Portugal  
Email: icardoso@av.it.pt

João P. Barraca  
University of Aveiro  
3800-193 Aveiro, Portugal  
Email: jpbarraca@ua.pt

Rui L. Aguiar  
University of Aveiro  
3800-193 Aveiro, Portugal  
Email: ruilaa@ua.pt

**Abstract**—Virtual CPEs address the low agility, slow time-to-market and high costs (of development, deployment and support) that are inheritable to traditional CPEs. They take advantage of the economics of Cloud Computing and the ability to run Virtual Network Functions in a general computing platform to build richer services, with faster development and deployment times, all whilst reducing costs. When coupled together with Service Function Chaining and the ability to build/enter chains dynamically, an opportunity arises to enhance this system by making the whole vCPE’s functionality policy-driven. We will start by constructing the vCPE within an OpenSource cloud platform (OpenStack), which will have dynamic SFC capabilities added through own extensions. Then we will make the traffic classification through a DPI (ntop’s nDPI), which will mark flows according to a configurable policy (REST interface) that will determine which service chain that flow must follow. Some virtual functions within a chain may trigger events (such as reaching the traffic quota for a certain service), being the policy immediately enforced at the VNFs and an “on-the-fly” policy change possible through the REST interface (for instance, change the throttle amount to a given application traffic). A live Proof-of-Concept was made which allowed real devices to connect to the vCPE.

## I. INTRODUCTION

With a growing demand for data, the rise of Internet-of-Things (IoT), new competition from Over-The-Top (OTT) providers and the shift towards providing all kinds of legacy services over the Internet Protocol, the embedded-systems that are today’s Customer-Premises Equipments (CPEs) became outpaced, cumbersome to manage and just too expensive to operate.

The solution to this problem came in the form of Virtual CPEs (vCPEs), which detach the network functions from the actual hardware of that embedded-system (in varying degrees according to approach), offloading them to a remote virtual environment and having the hardware act (to the most) as the physical interface to that environment. Leveraging Cloud Computing advantages, vCPEs allow for faster development and deployment of new services that require other network functions – which may also use more computational power to perform more complex tasks (the benefit of a general computing platform vs. an embedded system), this whilst benefiting from the economic principles of a Common Infrastructure, Utility Pricing and ultimately less truck-rolls to perform corrective maintenance tasks at the customer’s home.

The Virtual Home Gateway (vHGW) herein presented off-loads all network functions that do not rely on hardware to the cloud (DHCP, NAT, Firewall and Content Filtering), being the Physical Home Gateway (pHGW) just the tunnel that connects the customer’s devices to that cloud environment.

This allows policies to take greater control over the vCPE, as all functionalities and data-flows are available in the virtual environment. It also eliminates failure-points within the physical equipment, allowing for better cloud-assisted self-diagnostics and ultimately more power to solve issues remotely (therefore less truck-rolls).

Such design also enables access to the private network from different geographical locations, particularly through diverse access mediums (fixed and mobile accesses can communicate with each other, like any other device in the same network). This greatly enriches the role of policies and their value-added to the system, so that fundamentally different access mediums can have a service that is properly adjusted to their constraints and security/privacy preferences are uniformly enforced.

In this document we will start by introducing the related work that gives context to our efforts, followed by a section in which we build the vHGW with policy-driven Service Function Chaining (SFC). Afterwards comes the results section and then the conclusions.

## II. RELATED WORK

In the mobile world, there is Huawei’s Service Based Routing (SBR) [1] architecture which takes the logical blocks of the Policy and Charging Enforcement Function (PCEF)/Traffic Detection Function (TDF) and couples the Traffic Classifier capabilities with new Traffic Steering/SFC. This allows for more granular control over traffic, with more suitable (and powerful) processing chains that best fit that classification and policy. Software-Defined Networking (SDN) and Network Function Virtualizations (NFVs) within a cloud environment are also in the evolution path of that architecture.

The policy-driven vCPE we present draws a parallel with the mobile world, introducing capabilities similar to the SBR in land-line accesses (through a Home Gateway (HGW)). This is just the first step, as the architecture allows for a path which is agnostic to access medium (land-line or mobile).

### A. vCPE Research

Last year Fernando Sánchez and David Brazewell put forward a vCPE proposition [2] that used Linux’s Berkeley Packet Filter (BPF) subsystem as means to load within the Linux Kernel Virtual Network Functions (VNFs) without changing anything else in the equipment (functions could run locally). The Teethered Linux CPE approach held great promise in performance, but ultimately required that virtual functions were written with the skill, limitations and mindset of a BPF program loaded into the Linux Kernel.

CableLabs has active efforts in bringing SDN and NFV to cable operator’s networks [3]. Within those efforts, they have performed a demonstrator of a vCPE [4] which also uses OpenStack as cloud platform and a Raspberry Pi as pHGW (test) replacement. Other than the focus of cable networks, this demonstrator also differs from ours in the fact the pHGW is still the place many network functions will run (being the Cloud mostly for new enhanced functions). Nevertheless, although we are choosing to simplify the pHGW to the most (in a bid to produce savings in truck-rolls), running functions locally also has its own advantages (ie. DHCP still works when the remote environment’s connection is interrupted).

### B. ETSI PoCs

European Telecommunications Standards Institute (ETSI) has its own set of sanctioned Proof of Concepts (PoCs) within the context of “Hot Topics”, which are identified, documented and consolidated by the ISG NFV Working Groups. It is in the first completed “Hot Topic”, HT01 – Use of SDN in an NFV architectural framework<sup>1</sup>, that we can find the Service Chaining for NW Function Selection in Carrier Networks [5] PoC, which addresses many topics of the environment we wish to build to demonstrate our policy-driven vCPE.

In this PoC NTT Corporation used VNFs from providers such as Cisco (CSR1000v for its DPI capabilities), Hewlett-Packard (VSR1000 as the vCPE) and Juniper Networks (Fire-Fly as the Firewall). Together these functions were used to successfully demonstrate the feasibility and value-added of SFC in a set of Use Cases that focused in the residential HGW.

The DPI capabilities were used to perform blocks (URL filter and P2P filter) upon configuration. That is, the DPI function is only introduced when one of the filters is configured for a given user. This departs from our architecture in which the DPI/Classifier is always inspecting traffic so that all chaining is done according to the classification it introduces (conditioned by the configured policies).

Our system is also built only with Open Source components rather than proprietary solutions, even though proprietary solutions could also be used without the need for any special awareness (as long as they can process traffic that comes through their network port and reintroduce the processed result through the same port).

<sup>1</sup>[http://nfvwiki.etsi.org/index.php?title=HT01\\_-\\_Use\\_of\\_SDN\\_in\\_an\\_NFV\\_architectural\\_framework](http://nfvwiki.etsi.org/index.php?title=HT01_-_Use_of_SDN_in_an_NFV_architectural_framework)

### C. Emerging Developments

OpenStack’s Group Based Policy (GBP)<sup>2</sup> is an effort to bring a set of APIs which allows separating the application’s requirements (better known by its developers) from the virtual infrastructure own requirements (better known by the infrastructure teams), through an intent-driven model. Its action points aim to bring three major capabilities to OpenStack: Dependency Mapping, Separation of Concerns and Network SFC. With SFC within GBP at its earlier stages and the focus being on the Policy-Driven vCPE with dynamic SFC, we have resorted to previous work within Instituto de Telecomunicações to achieve that functionality<sup>3</sup> (GBP does remain very relevant and a very keen prospect for future contributions).

Within the vCPE realm, we have OpenStack’s Tacker<sup>4</sup>. Tacker is an effort to bring an NFV Orchestrator with a built-in VNF manager, which is based of ETSI MANO architectural framework, aiming to bring into OpenStack the capability to orchestrate the VNFs end-to-end.

### D. Other Relevant Research

A more detailed study on dynamic SFC of VNFs in Cloud-Based Edge Networks [6] was made available by researchers of the University of Bologna, in which they demonstrated the feasibility of SFC in this formulation through Mininet simulations. Their results give important context for our Traffic Steering solution.

## III. BUILDING THE VIRTUAL HOME GATEWAY

To build the vHGW we are following the steps laid by previous efforts within Instituto de Telecomunicações, from the specification of the environment [7], passing by an extension to OpenStack which brings the capability to establish GRE tunnels that connect the pHGW to the cloud environment, up to a SFC extension made for OpenStack Icehouse (in conjunction with the SDN controller OpenDaylight Hydrogen).

### A. General vHGW design

The simplified pHGW will be at the customer’s house while the virtual environment is at the provider’s infrastructure (a Datacenter/Point-of-Presence (PoP)). The pHGW performs the connection to its PoP (either preconfigured or configured afterwards through TR-069), creating a tunnel to the tenant’s own virtual environment (the vHGW) effectively extending the local switch link-layer all the way to the virtual network. This allows for devices connected to a given pHGW to be within the same broadcast domain as devices connected through a mobile access (3G/4G) or another pHGW (at another location) that also belongs to the same tenant.

The devices that connect to the virtual network are registered (through Neutron ports) in Neutron Database. These are “external ports” and they hold both the MAC address of the device (like a regular switch port) along with additional Neutron configuration (such as the IP address to be given

<sup>2</sup><https://wiki.openstack.org/wiki/GroupBasedPolicy>

<sup>3</sup><https://review.openstack.org/#/c/92477>

<sup>4</sup><https://wiki.openstack.org/wiki/Tacker>

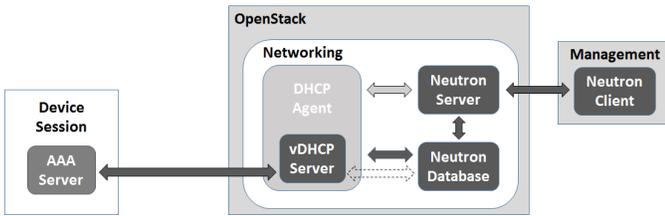


Fig. 1: Modified vDHCP design.

by the vDHCP function). The “external ports” are meant to preserve OpenStack’s existing models and operations. Since any device can connect to the vHGW at any time (so pre-provisioning ports is not feasible), in order to automatically create an “external port” that has the device’s MAC and correspondent lease in that network (allotted by Neutron), we have modified OpenStack’s built-in vDHCP to perform that task (more details will follow). The choice for DHCP instead of a switch-alike port learning process was due to the potential of using DHCP Options (such as Option 82) to relay additional information (ie. the SSID). This does however mean that, for devices with a manually set IP address, proper failsafe policies must be ensured so that the system isn’t compromised.

### B. vDHCP and External-Ports

The vDHCP will, on top of regular DHCP duties, be responsible for: (a) The creation of external ports upon the first connection of a device; (b) Implementing a session control mechanism which, through the different stages of the DHCP protocol, notifies an AAA entity which will in turn notify the Policy Enforcer of device changes;

OpenStack Neutron already has a built-in vDHCP (that uses Dnsmasq<sup>5</sup>), which in order to perform the additional tasks had to be modified as follows (in figure 1). Both the Neutron Server REST API and the command-line Neutron Client were enriched with the functionality to enable automatic port creation, the vDHCP now has direct connection to the Neutron DB (as to create and update the status of external ports) and a new RADIUS connection was created to perform session control with the external AAA.

Port creation is performed upon the first DHCP-DISCOVER, the AAA controls the ability to accept the requests/send the lease to the device through access requests, being the Policy Enforcer notified of that device when the AAA receives the account start. Subsequent connections will follow similar sequences, only skipping the port creation (starts in the second DHCP-DISCOVER) and replacing the account start by an interim. If a device is performing a lease renew then the sequence will start in the DHCP-REQUEST, with the account start being replaced by an interim.

### C. VNFs configuration and policies enforcement

VNFs configuration is made available through an unified VNF-API (fig. 2), which will allow the Policy Enforcer to

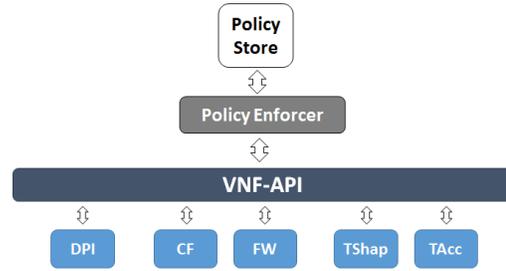


Fig. 2: Policies interaction with the VNFs.

reconfigure the functions according to the actions required to actually fulfill a given policy. It must be noted this API also has the capability to notify the Enforcer (in his own REST interface) of trigger events that may occur in the VNFs (for instance a quota for a given protocol was reached). Additional SFC context may also be carried across VNFs using this API.

All functions are regular Virtual Machines (VMs), running either Ubuntu or Debian, being KVM their hypervisor. The functions available are a DPI (into which all traffic will be steered, implemented with a fork of ntop’s ndpi-netfilter<sup>6</sup>), a Content Filter (CF) which runs DansGuardian and is capable of filtering web page contents, a Firewall (FW) which runs Alpine Firewall, a Traffic Shaper (TShap) which is implemented with Linux Traffic Control and a Traffic Accounter (TAcc) which accrues traffic amounts (per protocol according to DPI classification and policy configuration) and is capable of triggering quota events.

### D. Service Function Chaining

SFCs are built through our OpenStack extension (with northbound developments in OpenDaylight), which in conjunction with a classification criteria (that selects the traffic) and the definition of port chains, enables the use of regular Virtual Machines as VNFs.

All device traffic will always go first through the DPI (both inbound and outbound, fig. 3). Traffic Steering (TS) is then performed hop-by-hop through OpenFlow rules, which will steer matching flows to the next output port (with the dest. MAC address being replaced by the VNFs’s one). The classification criteria can be of any combination of the 5-tuple of a flow (protocol, source IP, source port, destination IP, and destination port), and/or other attributes like Type of Service (ToS). The DPI is able to identify application protocols either per device or per subscription (according to policy configuration) and marks all packets with a ToS that identifies the policy (and subsequently the chain) to be applied.

VNFs can use the ToS to identify a protocol/policy and perform additional actions (for instance the Shaper may have different caps according to ToS). However, given that the ToS is a readily modifiable field to any application, the DPI allows to “reset” the ToS in all packets or even just those that didn’t match a protocol (by default puts them to zero) and the Policy Enforcer must ensure safe fallbacks to account for this caveat.

<sup>5</sup><http://www.thekelleys.org.uk/dnsmasq/doc.html>

<sup>6</sup><https://github.com/betolj/ndpi-netfilter>

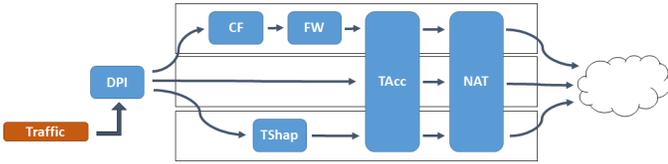


Fig. 3: The DPI will determine which chain each packet belongs to, according to policy configuration.

The port chain allows to define a set of ports through which (a traffic flow which meets a certain classification criteria) must hop through. The classification criteria can be set on a hop-by-hop basis.

This means that new SFCs can be created at any time should new policies or services require it. New VNFs can also be added to SFCs at any time. Likewise, should there be a need to remove or reconfigure any chain such can be done at any time.

#### IV. EVALUATION AND ANALYSIS

First and before most we started with the basics. We ran a series of live PoCs with a slimmed-down version of our vCPE (with dynamic SFC but not policy-driven SFC), featuring actual users (and real devices), to have a preliminary assessment of user experience and feasibility of our implementation.

Then we moved to laboratory tests, with the complete version of the Policy-Driven vCPE, to gather some hard-data regarding its performance. This performance tests include vDHCP evaluation (given the added session control with an external AAA and the need to create an external-port upon first connection), NAT throughput performance (to set a control for the DPI), Classifier throughput evaluation and SFC (time to apply and introduced latency) analysis.

##### A. Early PoCs

The PoCs consisted of a Raspberry Pi with a WiFi adapter acting as a pHGW (establishing the GRE tunnel to the datacenter), then with a single machine nicknamed Alex acting as PoP (in our building’s network). Alex is a 2x Intel® Xeon® CPU E5607 @ 2.27GHz with 16GB of DDR3 RAM @ 1333Mhz.

We had two VNFs pre-instantiated in the PoP (Firewall and Content Filtering), being the SFC configured dynamically (on-the-spot) as needed. Devices could connect freely and started without any chain. Afterwards, the Firewall was used to block traffic from a given device (all traffic or just 5-tuples), while the Content Filter inspected web-pages and blocked all traces configured websites (*4chan.org* was used). The PoC was run more than once with different audiences (and varying skill-sets).

We could observe this vHGW (as presented) was already capable of seamlessly replacing a conventional HGW without giving away the network functions weren’t actually running on the local hardware. SFC was proven to be functioning properly, with chain changes happening in a way that was perceived by the users as being “instant”. The general impressions were very positive.

Hard-data from laboratory tests of the more complete Policy-Driven vCPE follow next.

##### B. Laboratory setup

Logistics determined that Alex was to be replaced by Bica, a 2xIntel® Xeon® X5570@2.93GHz with 48GB of DDR3 RAM@1333 MHz. The virtual networks for each vHGW were pre-provisioned.

In order to prevent transient building network activity from interfering with our results, the pHGWs were simulated within Bica with an additional Open vSwitch bridge that has a tunnel established to Neutron’s managed Open vSwitch. The user devices, connected to the pHGW are Linux namespaces. The virtual networks and their respective KVM VNFs remain as detailed in Section III-A. Each test hereafter presented has been iterated a total of 100 times, unless stated otherwise.

##### C. Connecting to the vHGW

The DHCP client was `dhcpcd5 v6.0.5`. IP configuration times are presented in table I, both for the first time a user device connects to the network (where a Neutron port must be created) and also for subsequent connections. For testing accuracy, the local lease cache was cleared before every request. As comparison, table I also provides the time it takes to acquire a Dynamic Host Configuration Protocol (DHCP) lease with a classic physical HGW, Linksys WRT54GL v1.1 running DD-WRT build 14896 (that also happens to use `dnsmasq` as DHCP server), measured in the same conditions.

These results coupled with the observations from the PoCs allowed us to conclude that the longer time for the first connection (around 4 seconds) is not significant or perceivable in any negative way by the user. Given that subsequent connections are even faster than the legacy HGW, the vDHCP should not pose any performance concerns in this regard.

##### D. NATing and GRE

Although the NAT throughput test is mostly a control to pit against the Classifier results, given the prevalence of IPv4 in the Internet (and its inherent scarcity), NAT is nevertheless a critical function for any HGW.

`iperf` (version 2.0.5 (08 Jul 2010) pthreads) was the benchmark tool of choice, being configured for TCP protocol with 5 parallel connections (per client). This number of parallel connections was determined (beforehand) through crude experimentation, being this a value that yielded both the highest throughput and the most repeatable results.

We are pleased to see that our architecture behaved well in these tests (table I). Although there is a significant drop in performance when using GRE, the net throughput is still within the order of the 10 Gbps links, therefore being a feasible trade-off given the added flexibility it provides.

##### E. Classifier throughput

The NAT throughput tests (IV-D) were repeated adding the Classifier VNF, being the results shown also in table I.

A notable *caveat* is the traffic generated by the benchmark tool (`iperf`) does not fall under any of the protocol filters. This

NAT Throughput	Average	Std. Dev.
Single Device	60.98 Gbps	1.98 Gbps
1 LAN x 10 Devices	78.30 Gbps	5.31 Gbps
10 LANs (1 Device each)	78.45 Gbps	4.93 Gbps
GRE - Single Device	8.10 Gbps	0.22 Gbps
GRE - 1 LAN x 10 Devices	13.83 Gbps	1.10 Gbps
GRE - 10 LANs (1 Device each)	13.64 Gbps	0.57 Gbps
Classifier/DPI Throughput	Average	Std. Dev.
Single Device	4.30 Gbps	0.19 Gbps
1 LAN x 10 Devices	1.60 Gbps	0.35 Gbps
10 LANs (1 Device each)	15.31 Gbps	0.55 Gbps
DHCP Tests	Average	Std. Dev.
Time for connection on legacy HGW	0.525 s	0.289 s
Time for first connection	4.071 s	0.616 s
Time for subsequent connections	0.042 s	0.007 s

TABLE I: Policy-driven vCPE results table.

is the worst case scenario for the performance of this Classifier VNF, as this way every single packet will have to be analyzed (while for known protocols, once the identification is made, packets belonging to that connection are no longer inspected).

Unlike the NAT tests, the 10 devices produce different results according to LAN cardinality. This is likely attributable to the fact the chosen DPI's implementation is not making proper use of multi-threading. Due to the scale of these tests, 10 devices is hardly a representative carrier scenario, therefore some care must be taken as to the claims made. Nevertheless, even in the worst scenario (1 LAN x 10 Devices) a single vHGW is still able to deliver over 1 Gbps of throughput, which is more than enough for a residential HGW and merits continuation of studies at the larger scale.

#### F. Creating and applying a new Service Chain

Should a policy change require the creation of a new chain, it is important to determine how long it will take before existing traffic can start entering that newly created chain (API calls for chain creation included). Creating and applying a new service chain takes  $131.01 \pm 16.78$  ms.

It should be noted that when the SFC is already created, the relevant value follows in the next section (it is equal to a chain length of 1 for entry in the chain).

#### G. Implications of the chain length

SFC can be made virtually as long as desired. However, the impact on user's performance is something that needs assessing. The VNFs used for this test were Debian VMs running under KVM, only forwarding packets (without change).

We can observe the results plotted in figure 4, which follow a linear behavior as the chain length increases. The latency for a service chain with 0 VNFs, or the "control" latency for the normal course of packets towards the default gateway, was 0.069 ms. Furthermore, the increase in latency per increment of the service chain length follows a roughly linear increase. The average latency increase per VNF inserted is 0.245 ms.

### V. CONCLUSION

We have drawn a parallel between solutions that already exist in the mobile world and applied them in a different

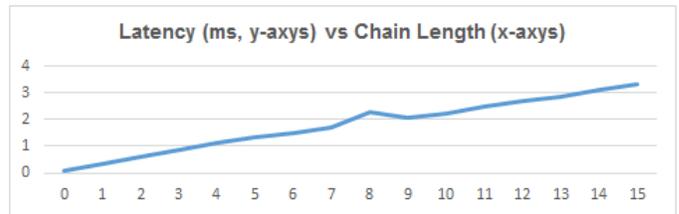


Fig. 4: Latency as the number of steered-into VNFs grows (between the user device and the default gateway).

context, the HGW. This brings new capabilities to this kind of CPEs all whilst maintaining a familiar logic and an evolution path that allows for the convergence of the access mediums (land-line and mobile).

To build the vHGW we have combined previous efforts to bring Traffic Steering (for SFC) to Cloud Computing environments and integration between physical and virtualized network segments. We have evaluated the feasibility of the design, mainly in terms of performance as perceived by an end-customer, where the results from live PoCs and laboratory tests show a worthy solution path within that formulation.

However, further work must be made to address the carrier side of the equation, with proper understanding of how the system behaves at the scale and loads of a carrier deployment, along with the need to address Failure-Detection/High-Availability of VNFs, their APIs and the platforms used to support them. There is also the need for better insight regarding the tunnels that connect the pHGW to the virtual environment, particularly how they behave in realist deployment scenarios.

Shortcomings aside, the policy-driven vCPE concept as presented warrants a good opportunity for further study and development.

### REFERENCES

- [1] Huawei, "Enabling agile service chaining with service based routing." [Online]. Available: [http://www.huawei.com/ilink/en/download/HW\\_308622](http://www.huawei.com/ilink/en/download/HW_308622)
- [2] F. Sánchez and D. Brazewell, "Tethered Linux CPE for IP Service Delivery," in *1st IEEE Conference on Network Softwarization (NetSoft 2015)*, 2015.
- [3] C. T. Laboratories, "Virtualization and network evolution," Cable Television Laboratories, Technical Report VNE-TR-SDN-ARCH, June 2015. [Online]. Available: <http://www.cablelabs.com/specification/sdn-architecture-technical-report/>
- [4] M. Kloberdans, "Virtualizing the Home Network," 2015.
- [5] K. Yogo, "Service chaining for nw function selection in carrier networks," NTT corporation, PoC Interim Report, May 2014. [Online]. Available: [http://nfvwiki.etsi.org/index.php?title=Service\\_Chaining\\_for\\_NW\\_Function\\_Selection\\_in\\_Carrier\\_Networks](http://nfvwiki.etsi.org/index.php?title=Service_Chaining_for_NW_Function_Selection_in_Carrier_Networks)
- [6] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of virtual network functions in cloud-based edge networks," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, April 2015, pp. 1–5.
- [7] J. Soares, C. Goncalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *Communications Magazine, IEEE*, vol. 53, no. 2, pp. 98–106, Feb. 2015.