Abstract—Network functions virtualization is one of the most promising networking technology trends for mobile operators. Such functions often are considered as virtualized version of physical devices. They follow the same implementation, deployment, and operation and management model as such physical entities. However, some key characteristics of the underlying cloud technologies, such as the elastic provisioning of resources, the different hardware failure rates and the increasing programmability of the underlying network enable new design paradigms. Thus, virtual network functions should leverage on IT technologies like Software-as-a-Service (SaaS) to reduce operational costs and enable new business models. We introduce a classification of different design paradigms, detail how a SaaS implementation can be performed for mobile network gateways, and provide a comparison of the operation cost implications of employing such paradigms to create network nodes in a mobile network.

I. INTRODUCTION

Network Functions Virtualization (NFV) is an approach to deliver communication services that applies virtualization and automation techniques from the IT world to move the current network functions (e.g. Firewall, DPI, Serving Gateway, ...) in an operator’s network from dedicated hardware to general purpose IT infrastructure [16]. These transformed network functions are known as Virtual Network Functions (VNF).

Currently, discussions on VNFs (also the recently published phase one documents from ETSI NFV [7]) implicitly assume an operational model that is very much comparable to the one of today’s physical devices where hardware and software are coupled: each virtual network function is considered and provisioned as a stand-alone entity. It has its own (virtualized) hardware resources assigned to it. With this approach, if multiple VNF instances are needed, multiple software installations are running in parallel, increasing the operational costs, as each of the instances needs its own maintenance (e.g. software updates). Furthermore, each instance is itself responsible of reporting faults on its allocated resources (e.g. a virtual or physical machine failure) to a management entity (e.g. OSS/BSS). While this approach might be well suited to quickly derive virtualized network functions from an existing software code base by porting this software to a cloud computing environment, it does not leverage on the specific features and capabilities of such an environment.

To overcome these issues, this paper makes three contributions. First, it provides a classification of different operational models for virtualized network functions and explains their motivation as well as their advantages and disadvantages. Second, it outlines an operational model for VNFs based on the Software-as-a-Service (SaaS) paradigm. Third, it compares the administrative costs of the different operation models.

To discuss these virtualization aspects, we focus on mobile network nodes and use the Packet Data Network Gateway (PGW) of an Evolved Packet Core [1], [11] as example: this is an important function in a mobile network, and the discussion is also applicable to other VNFs in such a network.

This paper is structured as follows. Section II introduces different implementation models for virtualized PGWs on a cloud infrastructure, followed by a detailed discussion of a Software-as-a-Service approach in Section III. Section IV compares the cost of the different models, followed by Section V that discusses related work. Section VI concludes this paper.

II. IMPLEMENTATION MODEL CLASSIFICATION

An overview of the different implementation models for virtual PGWs (vPGWs) according to the classification proposed in this paper is given in Table I and Fig. 1. The details will be explained in the following sections.

Note that the boundaries between the different implementation models are not impermeable, but introduced for classification purposes: software developed according to one of the models (e.g. the device model) can be evolved to another model when adding the respective features, thus crossing such a boundary. While this requires substantial effort, it is a clear trend in the marketplace.
### TABLE I. FEATURES OF PGW VIRTUALIZATION MODELS.

<table>
<thead>
<tr>
<th>Model</th>
<th>Creation</th>
<th>Key features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>existing software for a physical device ported to the cloud</td>
<td>Limited scalability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Redundancy model similar to physical device (2N)</td>
</tr>
<tr>
<td>Cloud-aware</td>
<td>software adapted for cloud features</td>
<td>Unlimited scalability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Novel redundancy models</td>
</tr>
<tr>
<td>SaaS</td>
<td>addition of multi-tenancy to cloud model</td>
<td>Unlimited scalability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Novel redundancy models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multiple logical PGWs on a single software layer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separates admin task into operation and management</td>
</tr>
</tbody>
</table>

A. Device model

In the device model, it is assumed that existing software–initially created for advanced telecommunications computing architecture (ATCA) based hardware platforms–is ported to an IaaS cloud. Such a vPGW is most often composed of a set of “applications” with different tasks, which are instantiated on different virtual machines (VMs). The VMs correspond to the ATCA blades, and in order to work properly, the different VMs need to be started in the right order. This set of entities together appears as a virtual gateway. The device model thus realizes 3GPP network functions on an IaaS cloud according to the existing, physical device based thinking.

Although this approach allows the PGW to be virtualized in a cloud environment, its operational model is the one of a physical device. Each virtual PGW instance is independent and stand-alone from other such instances, thus maintenance operations need to be performed for each instance. Furthermore, as the software was initially created for a system with physical limits (e.g. the maximum number of blades in such a system), it inherits these limitations: the scalability is limited to a certain amount of physical servers. Furthermore, for resiliency, such systems often use hot standby techniques where for each working node (physical blade, virtual machine, …) a standby exists that can take over in case of a failure. This results in a fully redundant configuration with respective resource requirements, but also a high availability. The reason to realize a virtualized PGW according to such a model is first and foremost time to market: porting an existing software to a new platform tends to be simpler than completely redesigning and rewriting the software for the new platform from scratch.

B. Cloud-aware model

The cloud-aware model is the next logical step and consist of designing the PGW software with the capabilities but also limitations of the underlying virtualization technology and hardware in mind [19]. The ones we consider here are the impression of unlimited availability of resource in the cloud, as well as the different failure characteristics of the used hardware. As the cloud provides resources (e.g. in the form of virtual machines) on demand, a PGW according to the cloud-aware model should allow for an easy scale-out of its capacity by adding virtual machines to its resource pool. This also means that some artificial scaling limit is not well suited here, and the vPGW according to this paradigm should scale to ”infinity”. As outlined above, cloud hardware has different failure characteristics. Furthermore, the declined profitability of mobile services motivates designs with a higher resource usage efficiency. Thus, instead of the (resource consuming) 1+1 hot standby mechanism used for the device model that requires for each working blade a failover blade, new resiliency models can be applied. Examples of such designs for mobile core networks are stateless nodes [19] that store session states on an external database. Other approaches might apply concepts from high availability computing where M backup nodes serve as standby for N working nodes (which saves costs if $M < N$).

C. Software-as-a-Service model

The Software-as-a-Service (SaaS [12]) model is widely known and applied in the IT world. Here we adapt it to the case of virtualizing PGWs, the result is an as-a-service version of the virtualised PGW: the PGWaaS. This model can be built with a comparable functional architecture as the cloud-aware model, the key differentiator to enable it is the addition of multi-tenancy [6]: this allows to create several virtual PGWs on the same software platform, i.e. out of one installation. This platform will be called PGW service layer in this document, it has the capability to provision isolated views of its processing capabilities to assign them to different logical PGWs.

The SaaS principle separates the tasks of management and operation into different logical roles, SaaS provider and SaaS consumer [12]. A consumer is the (economic) entity that uses the PGW to provide services for end-customers, e.g. a mobile network operator or a certain department of such an operator. The provider is the (economic) entity that provides the PGW and controls the necessary hardware and software for this. This can be a cloud provider, a vendor that hosts PGWs for other companies, or another department of the mobile network operator.

This role separation becomes clear when comparing the device and the SaaS model as illustrated in Fig. 2 and Fig. 3. In the device (and also the cloud-aware) model, the consumer also has the role of the provider and controls the complete (virtual) hardware and software stack. He must perform all management and operation tasks. In the SaaS model, these roles are separated and the consumer only can control application-specific settings of the virtual PGW and thus just operates a virtual instance of a PGW. The consumer does not possess direct access to the operating system or the underlying hardware.

In contrast, the provider controls the rest of the software stack like application, middleware, OS as well as the underlying hardware like storage, servers, or networking. He is responsible for tasks like deploying, configuring, scaling, updating, and managing the operation of service layer and might also need to manage the hardware. Handling specific configurations for single PGW instances, however, is out of scope for the SaaS provider.

The biggest benefit of this task separation is the promise of cost savings in terms of operational expenditures (OPEX) and capital expenditures (CAPEX). The consumers’ responsibilities are limited, he only needs operational expertise. In contrast, the provider can offer multiple logical instances of a network function on a single service layer but only needs to manage one such layer. Furthermore, also other cost savings of...
multi-tenancy exist [6]. This can strongly reduce OPEX, which will be studied in Section IV. Other benefits of SaaS are a promise of greater efficiency (e.g. higher resource utilization) and the ability to instantly deploy new PGW instances.

Other than the mentioned cost savings, the SaaS model enables new business models in the telecom operator ecosystem which are already present in the IT world. In fact, as the task separation and multi-tenancy are made possible, a separation of roles is also enabled, allowing service providers to focus on the value added services to end users without the need of operating a full infrastructure. Correspondingly, operators can implement infrastructure sharing to sell for example spare capacity with far higher value offering than with the current models, while keeping a simpler and less costly management of the infrastructure. A detailed analysis of the business models enabled by the SaaS model is out of the scope of this paper. However, it is of key importance to understand its potential and thus a logical next step.

At the same time, applying this approach to mobile network virtualization also needs to overcome some drawbacks, some of which are open research problems. First, there is the additional cost of developing a multi-tenancy capable service layer that also exists for other forms of SaaS [6]. Furthermore, as multiple PGW instances share resources and service quality is of utmost importance for paid mobile network services, performance predictability is needed. Finally, a service failure potentially affects all running PGW instances, thus the service should be designed for failure and thus be able to quickly recover without perceivable impact on end users.

III. SAA S IMPLEMENTATIONS OF PGWs

The PGW service layer as we envision it implements all functionality of a PGW. It has the capability to process both control plane (C-plane) and user plane (U-plane) messages as described in the relevant 3GPP standards [1]. It can create isolated views of its processing capabilities and assign them to different virtual PGWs (vPGW). Each of these vPGWs can have a unique configuration, and the service layer takes this configuration into account when processing user plane packets and signaling messages intended for a certain vPGW. Only a single service layer needs to be installed and managed to run different, completely isolated vPGW instances.

A. Implementation proposal

We envision this service layer to follow a split implementation model in which the control / management plane and the data plane follow different approaches. This split is driven by the different nature of the respective workloads.

Control and management plane requires high amounts of compute and storage resources, but comparatively low throughput. We therefore propose an implementation following current cloud application design guidelines, where the processing of messages is implemented with a set of loosely coupled microservices [14]. These microservices exchange messages via well-defined APIs, e.g. according to the REST paradigm. Together, the microservices implement the functionality of the control plane. This approach has gained a lot of popularity due to trends like cloud computing, continuous delivery, or DevOps and companies like Netflix have applied this design pattern to great effect. The isolation of multiple logical PGWs running in this context can be achieved by appropriately tagging each message and restricting access to information and configuration parameters from other instances.

In contrast, the data plane requires high throughput. A design following the microservice principle is not suited for these workloads, as this might e.g. introduce high latency when a packet traverses several microservices (and their APIs). We therefore propose a processing model where a U-plane packet is handled by only few or even only a single physical server, e.g. with the support of SDN [4]. Thus, instead of having many small, loosely coupled components (the microservices) which implement only a small function set, all U-plane processing is implemented in a single piece of software running on a single server (or virtual machine). The isolation of multiple logical PGWs in this setup can be achieved by assigning different physical servers (or virtual machines) to different logical PGWs, and overall scalability is achieved by running many machines in parallel.

B. Service layer in the context of ETSI NFV

ETSI NFV [16], [8] is one of the most important recent trends in virtualizing mobile network functions and will soon become a commercial reality [17]. Thus, the proposed SaaS model must fit in this context to be of practical use. How this can be achieved is shown in Fig. 4. The proposed service layer is installed in the infrastructure like a normal VNF would be deployed. However, instead of providing VNF functionality...
The key to bring the proposed service layer concept to the ETSI NFV context is thus the appropriate signaling, which is performed in five steps that are also visualized in Fig. 5: 1) the VNFM request the creation of a new logical PGW from the service layer; 2) the service layer checks the available resources, and in case these are not sufficient requests more resources. Note that this is different from a conventional resource allocation according to the device model, as only some subcomponents of the service layer might need to be deployed, depending on which of the components of the service layer suffers from the resource shortage. In particular, this does not mean to create another service layer instance; 3) the service creates a new logical PGW; 4) once the new logical PGW is created, the service layer notifies the VNFM about it and provides the identification information of such logical PGW for future references; 5) the VNFM provides initial configuration to the newly created logical PGW in order to make it operative e.g. IP address before handing over the control to the consumer.

Fig. 5. PGW instantiation according to the SaaS model in the NFV context.

directly, the service layer provides the ability to create logical VNFs, that can then themselves provide VNF functionality.

C. Fault Management

Logical PGWs use resources from several functional blocks implemented by the service layer. They do not have a global view of where and how such functional blocks have been implemented. Thus, the management of faults needs to be designed carefully to take this separation of concerns into consideration. A logical PGW can only report alarms directly related to its configuration, e.g. misconfiguration, a change in a parameter value or decreased packet processing capacity.

In contrast, the service layer has a global view on how the functional blocks are distributed and mapped to (virtual) infrastructure resources, e.g. physical servers or virtual machines. The service layer also has information on which functional blocks serve which virtual PGW. A failure in the service layer like a crash of a physical servers that host some of the service layer building blocks may affect some of the logical PGW instances provisioned by such a layer, but not all of them. Thus, the service layer should be responsible for fault correlation and identifying which subset of logical PGWs has been affected by the failure. The service layer is also responsible for sending a fault alarm to the respective management entity e.g. OSS/BSS informing about such failure and the subset of affected logical PGWs. While logical PGWs may also report on correlated events, such as a performance degradation, they are not aware of the details or the reason why they are experiencing such degradation.

The necessary steps in the context of the ETSI NFV architecture – also outlined in Fig. 6 – are: 1) the service layer detects a malfunction in one or several functional blocks; 2) the service layer identifies which logical PGWs are making use of the functional blocks that failed. 3) the service layer sends a failure report (or alarm) to the respective management entity, e.g. the OSS. This report includes a notification of the fault in the service layer, together with the cause of such fault (e.g. “physical server down”), the list of affected logical PGWs, and the resulting type of fault for each of the affected logical PGWs (e.g. “reduced packet processing capacity”).

IV. EVALUATION

This section studies the cost implications of the presented models with a focus on OPEX. While CAPEX is also important, we assume that the biggest CAPEX driver is the overall throughput that is needed. The throughput per server will be determined by technologies like data plane development kit (DPDK) and single root I/O virtualization (SR-IOV), and each of the proposed implementation models will try to use these capabilities to their maximum extend. Thus, we expect that the amount of servers needed to process a certain load and with this CAPEX is about the same for the different models. In contrast, the three discussed models have different operational approaches and thus a different operational complexity. This mostly impacts OPEX, the cost spent on service operation and administration, which is therefore the focus of this evaluation.

A. Administration Effort Definition

1) Device model: According to the device model, one PGW entity consists of up to $b_d$ IT server blades. One IT server blade has a traffic processing capacity of $c$, which is assumed to be the same for all PGW operational models. The device model uses a one plus one hot-standby backup mechanism at the server blade level. We furthermore assume that an operator will run multiple types of PGWs in parallel which are assigned to process different types of traffic, and thus need different configurations. For instance, one set of PGWs is responsible for machine-2-machine type communication and another set of PGWs processes smartphone traffic. There are $N$ such PGW types and the traffic load that needs to be processed by PGW type $x$ is $L_x$, where $x \in [1, N]$. The total traffic load in the network over all types of traffic can be denoted as $L = \sum_{x=1}^{N} L_x$. Therefore, the total amount of PGW entities is given by $N_d = \sum_{x=1}^{N} \left\lceil \frac{L_x}{c} b_d \right\rceil$.

The device model is used as the baseline scenario for our analysis, hence we set the administration effort to manage one
PGW entity to one unit. To handle traffic with load $L$, the admin work load $A_d$ can be expressed as $A_d = N^d_\alpha$ units of administration.

2) Cloud-aware model: Compared to the device model, the major difference between the cloud-aware model and device model comes from the fact that is has been adapted to the features provided by a cloud environment. For instance, a PGW implemented according to the cloud aware model profits from the “unlimited“ resources a cloud can provide, thus it has a different scaling capability. Furthermore, it also uses another resiliency strategy. A PGW according to the cloud-aware model can handle more server blades, we assume here that one PGW entity can scale up to $b_c$ blades, where $b_c \geq b_d$. Furthermore, by adapting the resiliency strategy, a $n + m$ backup strategy can be applied to provide resiliency: $n$ server blades are used as primary working blades and $m$ blades are used as backup blades, normally $n > m$. To handle traffic with load $L$, the amount of PGW entities is given by $N_c = \sum_{e=1}^{\lfloor \frac{L}{N^c_e} \rfloor \frac{1}{b_c}}^n$. Hence, the admin effort $A_c$ can be expressed as $A_c = N^c_c$ units.

3) Software-as-a-Service model: The SaaS model splits the roles in running a PGW into those of a provider and a consumer [12] as shown in Fig. 3. Thus, the related admin effort in a SaaS model can be separated into operation and management. Operation related activities include only tasks directly related to using the PGW functionality, e.g. performing its configuration. The percentage of this operation effort of the overall effort is denoted as $\alpha$ here. Management related activities include the remaining share of effort to run a PGW, e.g. software installation and updates, maintenance of the infrastructure etc., the percentage of all the admin effort is denoted as $(1-\alpha)$. The major difference between the SaaS model and the cloud-aware model comes from the multi-tenancy feature. The SaaS model is capable to provide multiple PGW instances from the same platform. This comes at a cost in the form of an additional admin effort $\beta$. It is not straightforward to model $\beta$, because it depends on the software implementation of the PGW service layer. However, it is reasonable to assume that the admin effort increases with the amount of PGW entities. Hence we propose to have $\beta = \delta \alpha + f(N^c_s)$. Function $f(.)$ represents the admin effort that is influenced by the total amount of supported PGW entities, which is a monotonically increasing function. For a well-designed software, $f(.)$ should behave as a linear function in order to well tackle auto-scaling issues. Hence, we represent $f(N^c_s) = \theta N^c_s$, where $0 < \theta < 1$. The admin effort to manage one PGW entity in the service layer should be much less than the effort to operate such an entity, thus $\theta \ll \alpha$. In our simulations, we therefore set $\theta$ to 30% of $\alpha$. In the equation, $\delta \alpha$ represents the minimum admin effort from the “PGW service layer” to enable the capability for multi-tenancy and it is a factor that does not change with $N^c_s$. As the SaaS model is also cloud-feature aware, it uses the same virtual resource scaling and a $n + m$ backup strategy as the cloud-aware model. Compared to the cloud-aware model mentioned above, computing resource could be used more efficiently, hence we have $N^c_s = \left[ \frac{n+\alpha m}{N^c_s} \frac{1}{b_c} \right]$. For a short summary, the admin effort $A_s$ for the SaaS model can be expressed as

$$A_s = 1 - \alpha + \alpha N^c_s + \beta.$$ (1)

### Table II. Simulation related parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of subscribers</td>
<td>1 million</td>
</tr>
<tr>
<td>One server blade traffic processing capability $c$</td>
<td>1 Gbps</td>
</tr>
<tr>
<td>$#$ blades per PGW entity for device model $b_d$</td>
<td>16</td>
</tr>
<tr>
<td>Upper $#$ blades limit for cloud-aware model $b_c$</td>
<td>80</td>
</tr>
<tr>
<td>$n + m$ backup strategy: $n$ and $m$</td>
<td>16, 4</td>
</tr>
</tbody>
</table>

#### B. Simulation and Results

Based on the above outlined costs, the administration effort for the different implementation models has been compared with a Matlab-based simulation. The related simulation parameters are chosen as listed in Table II if not otherwise stated in the text.

1) Overall Comparison: We first assess the influence of increasing traffic load on the three models as shown in Fig. 7 (a). Here, $\alpha$ and $\delta$ have been fixed at 0.4 and 1, respectively: this means that 40% of the overall administration effort is assumed to be spent on operation, and the additional admin effort for a service layer is comparable to the operation effort of a single PGW. Moreover, only one type of PGW is considered in this first simulation. As shown in the figure, the device model is more strongly influenced by increasing traffic load. In comparison, the admin effort increase of the cloud-aware model is more moderate due to its better scalability. On top of this, the SaaS model profits from the split of the admin effort, especially for higher load and thus the higher number of PGW instances needed to process this load.

2) Resiliency Strategy: Fig. 7 (b) illustrates the influence of the resiliency provisioning strategy provided from the cloud features. As shown in the figure, the value of $m$ is varied from 4 to 16. Among the three models, the cloud-aware and the SaaS model are influenced by $m$: once $m$ becomes bigger, the admin effort also becomes higher. However, even when the $m$ is increased to be the same as $n$, the admin effort from the two models is still much lower than the device model. This comes from the effects already visualized in Fig. 7 (a).

3) SaaS Model: Fig. 7 (c) further illustrates the admin effort of the SaaS model by varying $\alpha$ and $\delta$ between the minimum and maximum traffic load used in the previous two figures. A bigger $\alpha$ (shown on the $x$-axis of the figure) means an increasing share of operation compared to management of the overall admin effort. The higher $\alpha$, the lower the gains that can be achieved by the multi-tenancy support provided by the SaaS model: for the most extreme example examined ($\alpha = 0.9$, corresponding to an assumed management effort of only 10%), the overall admin effort of the SaaS model is slightly higher than the effort for the cloudified model (which can be seen by comparing with Fig. 7 (a)) because the additional overhead of multi-tenancy is higher than the possible savings. For the cases where $\alpha$ is moderate, the SaaS model has the lowest admin effort. Interestingly enough, the admin effort for the service layer $\delta$ does not have a strong impact: even for a value of two which means additional admin effort comparable to the operation of two PGWs, the overall admin effort increases only moderately for both traffic loads.
V. RELATED WORK

A lot of contributions are pushing the concept of virtualizing EPC gateways. One class of work performs this virtualization with the help of OpenFlow switches [10], [5], thus implementing the data plane of such gateways in hardware. In contrast, our work focuses on approaches that implement all gateway functions in software. Major vendors have made proposals for such implementations [3], [13]; these can be classified to belong to the device model category. Further steps are followed by ETSI NFV [15] that discusses the SaaS use case. However, no further implementation details are presented, nor a cost model is given. The FP7 research project “Mobile Cloud Networking” [9] also deals with the virtualization of the EPC and proposes EPC-as-a-Service [18]. From the consumer perspective, their proposal appears as SaaS as complete EPC instances can be requested on demand. However, the provider still instantiates a complete EPC instance for each such request. Thus, the proposal can be classified as a cloud-aware model that is packaged for the consumer to appear as SaaS.

To summarize, while the idea to apply the SaaS concept to the problem of network function virtualization exists, we are not aware of a concrete proposal how to actually implement this concept for mobile networks in the ETSI NFV framework. If such concepts are examined in more detail, the solutions still lead to a situation where for each new instance, new pieces of software need to be deployed. This avoids the multi-tenancy which is a key concept of SaaS, leading to potentially higher OPEX. Furthermore, we are also not aware of any investigation on the OPEX implications of different models as we have provided it.

VI. CONCLUSIONS

In this paper, we have discussed different approaches how to implement the packet data network gateway (PGW) of an EPC network in software. While the device model recreates a physical device on a cloud platform, the cloud-aware model means that the gateway is enabled to cope with the features a cloud provides, like virtually unlimited resources and different hardware failure characteristics. The Software-as-a-Service (SaaS) paradigm for PGWs extends this to multi-tenancy: a single software platform is able to provide multiple logical, isolated PGWs from the same software layer. Our cost study of these different approaches shows that especially the cloud-aware and SaaS models have cost advantages in terms of OPEX, which is a promising future approach for operators, also in the context of 5G [2].

ACKNOWLEDGEMENTS

The authors thank David Perez-Caparros for his contributions to this paper during his work with DOCOMO Euro-Labs.

REFERENCES