Investigation of Container Network Function Deployment Costs in the Edge Cloud

Kien Nguyen*, Filip Simonovski‡, Frank Loh*, Tobias Hoßfeld*, Nguyen Huu Thanh†
*University of Würzburg, Institute of Computer Science, Würzburg, Germany
†Hanoi University of Science and Technology, Hanoi, Vietnam
‡University of Augsburg, Augsburg, Germany
Email:{firstname.lastname}@uni-wuerzburg.de, {firstname.lastname}@uni-a.de, thanh.nguyenhuu@hust.edu.vn

***Abstract*—Service Function Chains (SFCs) can be used with containerization to efficiently enable the ever-increasing Internet of Things (IoT) deployments in heterogeneous edge–cloud environments. In this context, SFC schedulers often employ mechanisms such as migration and consolidation of Containerized Network Functions (CNF) to adapt to dynamic environments. However, the costs of the CNF lifecycle are often not investigated, especially deep into its states and state transitions, have created a gap between theoretical research and practical implementation. To this end, this paper comprehensively examines the capability of the edge–cloud in accommodating CNF-based IoT deployments with a key focus on the CNF lifecycle invocation cost, including latency, resource, and energy consumption. Our findings show the actual cost of using and reconfiguring CNFs, which is a common SFC optimization technique. We then provide a comprehensive profiling that could be used to develop realistic SFC placement strategies and algorithms.**

***Index Terms*—Cloud Computing, Network Function Virtualization, Internet of Things, Service Migration**

I. Introduction

The emergence of massive Internet of Things (IoT) deployments has given birth to a hybrid computing model named edge–cloud, which combines the scalability and power of cloud computing with the low latency and privacy of edge computing. To overcome the heterogeneity of edge devices, flexibility and fault tolerance in resource allocation are essential. This can be achieved through Network Function Virtualization (NFV), which enables the design of IoT applications as loosely coupled Virtual Network Functions (VNF) that reside in Virtual Machines (VM). Virtualization enables the flexible placement, configuration, and migration of VNFs and Service Function Chains (SFC), which can lead to efficient resource allocation, even under dynamic conditions such as unknown arrivals and departures of SFC requests, user mobility, and device failures.

Recently, containerization is expected to further empower NFV in resource-constrained environments such as edge–cloud. In this context, virtual networks are hosted within containers [1], which are significantly more lightweight, agile, and fault-tolerant than traditional VMs. To exploit these benefits, research in the domain of edge-related computing utilizes Containerized Network Functions (CNFs) to enhance various objectives, such as energy-resource efficiency and latency [2]. In the face of rapid environmental changes caused by users and network conditions, state-of-the-art approaches rely on migration to sustain the desired objective. However, unlike conventional VM-based VNFs, which have been extensively studied, CNF implementation costs in dynamic edge–cloud environments have been overlooked, especially those parameters related to sustainability, such as energy and resource consumption. This is partly due to the assumption of container lightweights. To address this gap, this article aims to answer the following two research questions:

- RQ1: What is the real cost of accommodating CNF in edge–cloud regarding latency, resource demands, and power consumption of computing devices?
- RQ2: How can specific edge–cloud computing accommodate heterogeneous CNF implementations considering the cost of the reconfiguration process?

We answer these questions by conducting experiments on a real testbed, where realistic measurement are applied for an IoT system. The contributions of our work are two-fold.

First, to understand the relevant costs of CNF implementation on edge–cloud, we formulate and investigate a complete lifecycle of CNFs. Our results show that CNF states yield different resource consumption, and state transitions incur significant costs on latency, energy, and resource utilization. These effects are even worsened by current cutting-edge resource optimization approaches, which often require massive CNF relocation. Second, based on our measurement results, we profile all targeted metrics as a function of the number of CNFs and their costs. These models can be used to measure the implementation costs of CNFs over edge-cloud environments and serve as the basis for optimized allocation strategies.

The remainder of this article is organized as follows. Section II reviews the background information necessary to understand our work. Section III surveys related works and shows that the impacts of CNF reconfiguration on edge clouds has not been studied in the literature. Section IV presents a use case and a testbed to evaluate the impact of CNF reconfiguration on edge clouds. We provide an in-depth profiling of the CNF lifecycle and its impact on different performance aspects of the edge cloud system, based on real testbed measurements. Section V concludes the article.

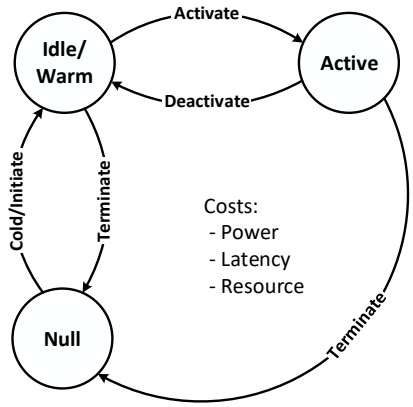


Fig. 1: A typical lifecycle of a CNF.

II. Background Information

As the demands for Big Data and IoT services have been increased recently, the realization of VNF/CNF in edge–cloud and Multi-access Edge Computing (MEC) attracts much attention thanks to its flexibility and scalability. To enhance the understanding of this article, we introduce basic knowledge about containerization in NFV and the SFC embedding problem.

A. Containerized Network Function

Compared to VMs, containers are more agile, lightweight, and resilient. As a result, containers have been standardized as the replacement of VMs to accommodate network services in NFV Release 4 by the European Telecommunications Standards Institute (ETSI) [1]. In this context, VNF terminology is replaced by CNF. Most notably, the CNF relocation, which corresponds to VM migration in VNF terminology, is performed by *stateless configuration*, in which the CNF instance at an unwanted position is replaced by a new one spawned up at the desired location. As this process is often called *reconfiguration* in literature, we will also use this phrase in this work. The CNF reconfiguration essentially invokes the container's lifecycle, which typically consists of three *states* and four *transient actions* as proposed in Figure 1. The states are defined as follows.

Null indicates the non-existent state of a CNF. When a request arrives, the system creates a new CNF instance through the *Initiate* action.

Idle/Warm denotes the non-working state of an instance that has been spawned in the system. The program inside a CNF is already loaded and waiting for traffic. When a job is available, the *warm* container actively runs via the *Activate* action. On the contrary, a *warm* container can be removed through a *Terminate* action.

Active is the working state of a container instance that performs the assigned job. An active container can either be deactivated back to the *warm* state when the task is finished or can be wiped out of the system via a *Terminate* action.

While the figure represents a theoretical view of the CNF's lifecycle, the states and actions can practically be implemented by container orchestrations such as Kubernetes [3]. Normally, *active* state is the main profiling target of any CNF in terms of energy or resource usage. The other states and the transient actions, while also incurring costs (power, resources, latency), have often been overlooked in the literature, to the best of our knowledge. We argue that these costs might be significant in scenarios where millions of events constantly cause containers to jump between states.

B. CNF Reconfiguration in Edge-Cloud

Generally, SFC deployment in NFV-enabled edge–cloud/MEC is challenged by the environment dynamicity. In this context, CNF reconfiguration occurs in the following scenarios to reoptimize the system.

Offloading of CNFs between computing tiers is a key usage of edge–cloud, MEC, or fog computing. For example, from end devices to an IoT gateway/MEC server and a central cloud. In critical cases such as low battery and/or limited computing capacity, offloading helps to guarantee Service Level Agreements (SLAs) [4].

Consolidation of CNFs in the data center also needs to be investigated. Since an *idle* server can consume more than half the power of a fully-loaded one [5], consolidation of low-utilized servers together is often used. In addition, CNFs belonging to SFCs can also be consolidated into one server, or in a group of neighboring servers, to reduce traffic.

Redeployment of SFCs that had been mapped to the system to increase the SFC's acceptance rate. The dynamicity of SFCs requires resource allocation and de-allocation for joining SFCs and leaving SFCs, respectively. Over time, the system will fall into an un-optimized state whose resource needs to be rearranged. Therefore, redeployment of all SFCs inside is performed to re-optimize the system.

Mobility of user's equipment (UE) (smartphone, smartwatch, etc.) from the Radio Access Network (RAN) spectrum of one Base Station (BS), where its functions are processed at an MEC server, to another BS that is far from the original server. To reduce the backbone's bandwidth usage and service latency, CNFs are usually migrated to a server near the new BS. Follow Me Cloud (FMC) [6] is a work that adopts this migration technique to ensure low latency ($1.0\,\mathrm{ms}$ - $10.0\,\mathrm{ms}$) for mobile users.

III. Related Work

In this section, we will survey the state of the art that targets specifically to CNF and CNF-like reconfiguration costs. Then, an overview of SFC deployment that is both aware and unaware of reconfiguration in edge–cloud/MEC is provided.

A. CNF Lifecycle and Reconfiguration

As VNF/CNF is built upon VM/container, its lifecycle is basically the VM/container's lifecycle. From the VM's side, reconfiguration is commonly done by migrating the VM from one machine to another via a network connection. Here, the costs are comprehensively studied in the work of Hines et al. [7]. While VM-alike migration mechanisms are also

applicable for containers [8], profiling results [9] show that a 75 MB RAM-usage container requires approximately 100 – 200 MB data transmission for migration, which is an expensive cost and time-consuming for IoT services. For that reason, along with container usage favoring agility, stateless migration by provoking the lifecycle is preferred.

Costs of the container's lifecycle are concerned more in the context of serverless computing [10]. To reduce latency caused by moving from inactive state to active state of a container, well-known serverless frameworks like *AWS lambda* [11] keep a number of frequently-used containers at *idle/warm* state for a fixed period of time. From the literature, Mohan et al. [12] propose an advanced technique that pre-crafts a warm container to prepare for an upcoming task, which can achieve an 80% reduction in execution time. Kunal et al. [13] exploit the similarities in container content to reduce network requirements for the code-pulling process, hence reducing the time needed for cold starting. Wu et al. [14] propose a container scheduler to keep track of the current status of containers (used, paused, or evicted) to re-use them for upcoming traffic optimally. Overall, the aforementioned works focus only on the *latency* problem of the *cold* process and propose a solution to improve the Quality of Service (QoS) of the incoming requests. While the *warm* state of a container has been used to reduce service downtime significantly, no work analyses the impact of different container states and their transient actions on energy, resource, and other performance aspects of the edge–cloud system, especially under situations where SFCs and their CNFs have high migration rates.

B. Reconfiguration-Aware SFC Deployment

SFC deployment is a process of manipulating CNF locations and networking to utilize edge–cloud resources efficiently. To increase the acceptance ratio, Xu et al. [15] and our previous work [16] consider the redeployment of running SFCs to re-arrange the mapping strategy and eliminate resource fragmentation. VNF offloading between edge and cloud or between edge devices [17] [18] [19] is also deployed to instantaneously save energy and allocate resources for arriving SFCs.

Aware of the complexity and costs of reconfiguration, Padhy et al. [20] introduce *REAP*, an energy-efficient algorithm that chooses the appropriate reconfiguration mechanisms among four options: mapping, redirection, migration, and scaling out the CNF number. Their results show 60% in energy consumption. Wang et al. [4] minimizes the number of CNF reconfigurations in mobility scenarios by applying Markov Decision Process (MDP) to relocate CNF to an optimal place with minimal cost. Similarly, Zhao et al. [21] introduce a two-step algorithm to reduce the downtime and operational cost of the reconfiguration process. VNFs are duplicated to restore users' vital services, such as front-end VNFs, and migrate other VNFs later.

Furthermore, knowing traffic patterns in advance to plan ahead is an effective strategy to reduce repetitive reconfiguration and avoid the incurred costs. To this end, Machine Learning (ML) has been used to predict future demands to alter VNFs/CNFs following environmental changes. Kim et al. [22] predict the future CPU usage of each VNF by using *Long Short-Term Memory* (LSTM). Since their model is general and requires no agent at individual devices, it reduces the pressure on resources for a trade-off of prediction accuracy. Lange et al. [23] forecast load changes using classification algorithms. The algorithm decides if the system needs to add one, remove one, or do nothing for the current number of VNFs. Vahidinia et al. [24] use Reinforcement Learning to learn the lifecycle invocation patterns and determine the best time to keep the containers warm and an LSTM model to predict the future to know how many containers can be kept in the system.

On the other hand, as there are no profiling results of the CNF lifecycle in the literature, the lack of precise quantification of energy, resource demand, and latency during lifecycle invocation of SFC deployment poses a concern about the real cost in realistic implementation.

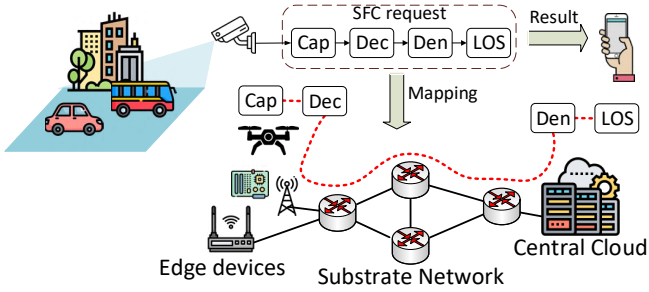


Fig. 2: Smart traffic camera can be massively deployed as SFC in a smart city.

IV. CNF Lifecycle Profiling

In this section, we define and measure an IoT use case over a testbed to investigate the impact of CNF reconfiguration, where lifecycle invocation plays the key role. Targeted metrics are resource consumption, power usage, and latency. Measurement results are then modeled as a set of equations that are crucial for precise energy and QoS-aware SFC deployments and strategies in the domain.

A. Use Case

In this article, we build the testbed and perform the simulation based on an IoT application that is deployed over an edge–cloud system, as shown in Figure 2. The application is about smart traffic monitoring via IP cameras to help authorities in large cities analyze traffic intensity, especially in peak hours, by analyzing the traffic density via captured images. This service is designed in the form of an SFC containing four CNFs: *Capture (Cap), Decode (Dec), Density (Den)* and *Level of Traffic (LOT)*, which perform capturing JPEG images, decoding images sent from the first function, evaluating traffic density, and classifying the level of traffic of the road, respectively. The SFC is deployed over a substrate network containing various edge devices placed at every road

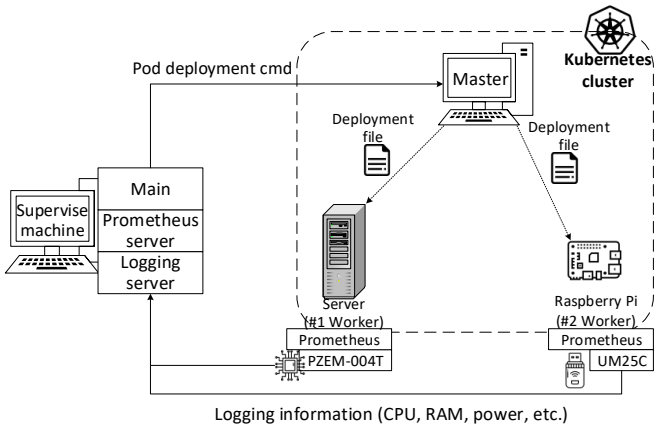


Fig. 3: Testbed implementation.

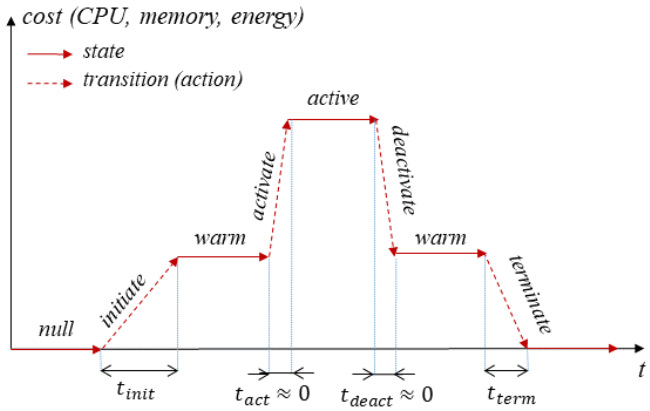


Fig. 4: States and transient actions of a CNF with the corresponding costs and duration.

corner of the city and a central cloud. Depending on the traffic state and time of the day, cameras and applications can be turned on/off following users' demands. Under this scenario, the system can benefit from the agility and flexibility of SFCs in resource allocation. However, as CNFs in this SFC are linearly dependent on each other, interruption caused by one CNF reconfiguration will cause the entire SFC to freeze temporarily, therefore, highlighting the impacts of CNF migration on the edge–cloud system performance.

B. Testbed and Measurement Setups

1) Testbed Instruments: Fig. 3 describes the setup of our testbed. In detail, there is a Supervisor machine, which controls the flow of deploying CNFs and collects measurement results, and an edge–cloud cluster where CNFs are deployed and managed by Kubernetes. The cluster consists of three machines: a Master node and two Worker nodes. The Master node receives deployment commands from the Supervisor machine, converts them into manifest files, and deploys them to one of the Workers. These Workers are IBM server and Raspberry Pi, which represent the cloud machine and edge devices, respectively.

TABLE I: Testbed configurations

	Hardware	Software
Edge	Raspberry Pi 4B: 1.5GHz Quad-core Cortex-A72, 4G RAM, Fast Ethernet	Ubuntu 20.04 server Kubernetes v1.23.5
Cloud	IBM x3650 M4: 2.60GHz Intel Xeon E5-2670, 64G RAM, Gigabit Ethernet	Ubuntu 20.04 server Kubernetes v1.23.5
Supervise	DELL PowerEdge 3640 Fast Ethernet	Ubuntu 20.04 LTS
Measurement instruments	UM25C: 4 - 24V DC, 0 - 5A, Accuracy: 0.5%; Pzem-004T: 80 - 260V AC, 0 - 100A, Accuracy: 0.5%	Prometheus 2.34.0 Node-exporter 1.3.1 UM25C-rdumtool PZEM-004T-v30

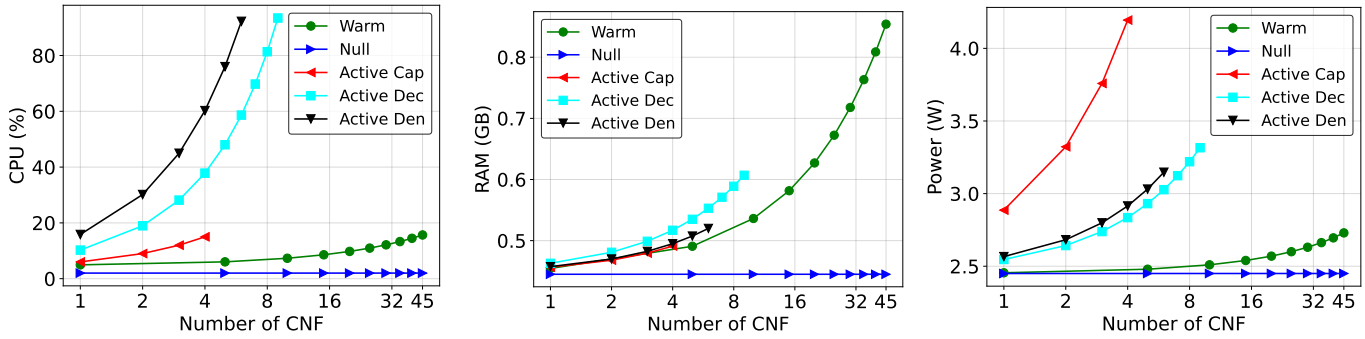
The measurement part is handled by a monitoring software called Prometheus [25] and two power meter devices. Logging results are sent back to the Supervisor machine. More details of the equipment are described in Table I.

CNFs will be deployed as pods, which are virtual units defined by Kubernetes. The maximum number of pods that can be hosted on a device is determined by the ratio of the requested amount of resources (specified in the manifest files) to the device's resource capacity. For example, a 16 GB RAM machine can host a maximum of 16 one-GB-RAM pods. However, CNFs' lifecycles require flexibility in resource usage, so we intentionally ignore the fixed resource provision in manifests to create a higher number of CNFs.

However, we observed that the maximum number of pods that our edge and server machines can run concurrently is around 45 and 100, respectively. Beyond this, the edge starts freezing and some of the server's pods are unable to be turned on. This is restricted by the private Classless Inter-Domain Routing (CIDR) policy of Kubernetes. Therefore, we defined the feasible values of CNFs as 45 for the edge and 100 for the server. This restriction applies to most CNF types, except for the Capture CNF. Because its job is to control the physical camera, it is bound to the limited number of USB ports, which are used to communicate with cameras, at the edge device. Thus, only four pods of this type were able to run in our experiment.

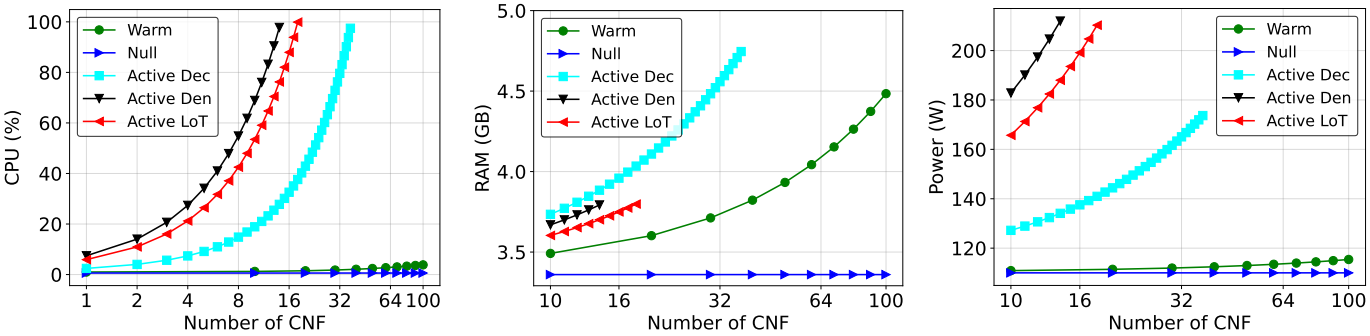
2) Targeted Measurements: Fig. 4 shows three states {*null*, *warm*, *active*} and four transient actions {*initiate*, *activate*, *deactivate*, *terminate*} of a CNF. While the time duration of each CNF state is merely determined by its service time, the time duration of *initiate* and *terminate* actions, denoted as t_{init} and t_{term}, are device-dependent. These durations are the time taken to create or remove a CNF. These actions require efforts from the hosting machine to allocate or de-allocate system resources. Thus, they also consume additional resources as discussed later in the paper. On the other hand, since the difference of a CNF in *warm* and *active* states is whether it receives traffic, the transition time between these two states denoted as t_{act} and t_{deact} in Fig. 4 can be neglected.

Based on the above discussion, we quantify CNF's lifecycle following these states and actions:



(a) Edge's CPU usage (b) Edge's RAM usage (c) Edge's Power usage

Fig. 5: Resource and power usage of different *states* at edge device. Note that the x ticks follow $\log_2$ scale. LoT CNF does not exist at the edge side. Since the warm and null states of all four CNFs exhibit a similar trend, we report one example only.



(a) Cloud's CPU usage (b) Cloud's RAM usage (c) Cloud's Power usage

Fig. 6: Resource and power usage of warm and active *states* at the cloud server. Cap CNF does not exist in Cloud. Other information is similar to Fig. 5's caption.

- *Cold/Initiate* action starts when the deployment order from the Supervisor machine is given to the Master until all of the required CNF instances are spawned up successfully by the Worker machines.
- *Idle/Warm* state begins at the end of the *cold* one. Since CNF instances have not yet received requests, they are essentially staying at the *idle* state.
- *Active* is the state where a warm CNF begins to receive and process incoming traffic.
- *Terminate* action begins when the Master receives the termination command issued from the Supervisor node to delete a CNF. It completes when the CNF instance no longer exists within the cluster.

For each state or action, the following performance metrics are measured:

- *Device resources*, CPU and RAM utilization of an edge or cloud machine in its states and transient actions.
- *Power consumption* of each state and transient action.
- *Latency*, which is the time required to change the CNF from one state to another.

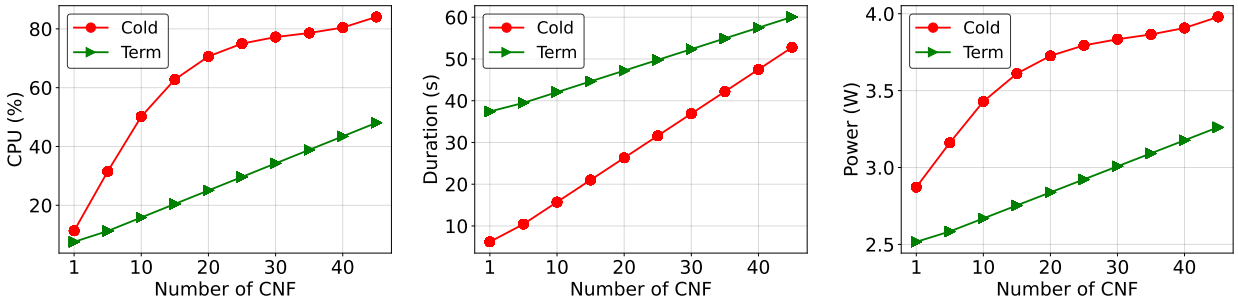
The above measurement process repeats for an ascending number of CNF instances, and the results are recorded accordingly.

C. Result Evaluation

To characterize the measured parameters, we perform 50 repetitive measurements, each representing the average of instantaneous values obtained from monitoring devices. To model these 50 data points, we employ polynomial regression to generate interpolated lines that capture the underlying trend. The accuracy of each polynomial model is assessed using the coefficient of determination (R^2) and the root mean square error (RMSE). The degree of the polynomial is iteratively increased until no further improvement of the R^2 and the RMSE is observed. For clarity, the subsequent result figures depict only the polynomial lines.

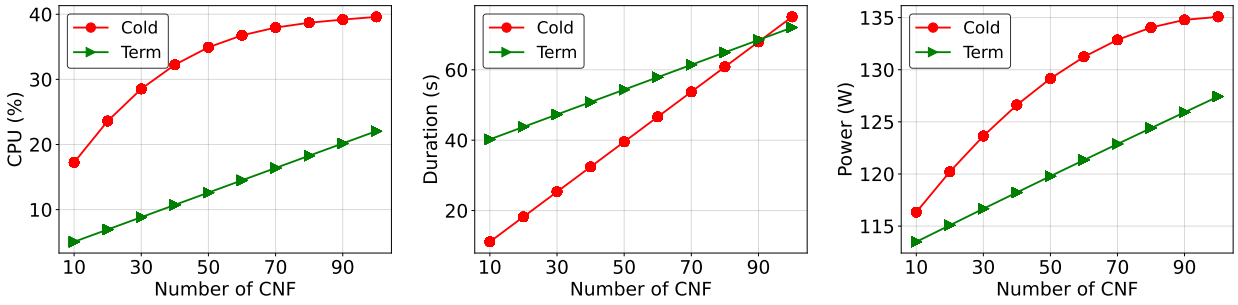
1) State Consumption: The CNF's state {*null, warm, active*} consumption on an edge and a cloud are represented in Fig. 5 and Fig. 6, respectively. Our measurements show that all four CNFs exhibit the same resource usage in the warm state since this state is essentially idle. Therefore, we only report the resource usage of one CNF in the warm state (green line).

The null state is the idle state of the device with no CNF running. In this state, only system tasks are running, resulting in a low CPU usage. In the warm state, CNFs have been



(a) Edge's CPU usage (b) Delay/downtime due to launching and terminating CNFs at edge (c) Edge's Power usage

Fig. 7: Resource, power usage and delay of cold and terminate *actions* at edge device. Note that these CNFs are created/terminated simultaneously



(a) Cloud's CPU usage (b) Delay/downtime due to launching and terminating CNFs in cloud (c) Cloud's Power usage

Fig. 8: Resource, power usage, and delay of cold and terminate actions at the cloud server. Note that these CNFs are created/removed simultaneously

created, and their nested processes are listening for incoming traffic. In this state, the CPU is mainly used by the Kubernetes management component *kubelet*, which monitors pods [26]. This results in low computing usage, even in limited-capability edge devices. However, an increase in CPU usage is observed when the number of CNFs reaches its maximum. This is clear with almost 20% CPU usage for the edge device compared to negligible CPU usage for the cloud server. When modeling these measurements by polynomial regression in Equation (3) and Equation (4) of Table II, the result in about 0.03% and 0.24% CPU for one warm container in the cloud and edge, respectively. Therefore, operators may keep many CNFs warm without any concern about wasted CPU consumption.

The active state consumes the most resources and is linearly proportional to the number of CNFs. Note that the x-scale of Fig. 5 and Fig. 6 is logarithmic, so the linear line appears as a curve. The active CPU usage of each CNF varies depending on the nested program, with CPU-intensive tasks such as calculating density (Den) and estimating Level of Traffic (LoT) using more computing power than less demanding tasks such as Capturing images (Cap) and decoding images (Dec). While the trends are similar on both cloud and edge tiers, CNFs deployed on cloud servers use 3-4 times less CPU than those on edge devices due to the more powerful CPUs in cloud servers. Regarding horizontal scaling, CPU resources are the main limitation for opening more active CNFs.

The memory utilization of both edge device and the cloud machine is shown in Fig. 5–b and Fig. 6–b. As RAM is essentially used for loading process variables, it is linearly proportional to the pod number. For our use case, a warm pod utilizes approximately $9\,\mathrm{MB}$ for the container's libraries and variables of the nested software when hosted on an edge device (Equation (7)) and $11\,\mathrm{MB}$ on cloud server (Equation (8)), which are relatively identical between the two tiers. When CNFs move to Active, more RAM is allocated for storing the input data and processing temporary variables. This result shows that RAM could be the limitation when keeping CNFs in a warm state because they still take up significant space.

As CPU is the main energy-draining source, power consumption is proportional to CPU usage, as shown in Fig. 5–c and Fig. 6–c. The Equation (11) and Equation (12) provide more details on the numbers. Similar to CPU, power consumption is low in the warm state and high in the active state, with different CNFs consuming different amounts of power. Notably, the abnormally high power consumption of the active *Cap* CNF is due to the power usage of the camera that operates alongside this CNF on the edge device. Compared to edge devices, cloud servers consume significantly more base power,

TABLE II: CNFs lifecycle's modeling results in terms of CPU, RAM, power, and latency that is used for plotting Fig. 5 to Fig. 8 and for the proposed algorithm. x stands for the number of CNFs, its range for the edge device is 1 to 45, and for the cloud server is 10 to 100.

Metric	Cloud server		Edge device	
Cold CPU (%)	$cc_s(x) = 6.12 + 1.09x - 1.28 \times 10^{-2}x^2 + 5.28 \times 10^{-5}x^3$	(1)	$cc_e(x) = 5.43 + 6.04x - 1.74 \times 10^{-1}x^2 + 1.74 \times 10^{-3}x^3$	(2)
Warm CPU (%)	$wc_s(x) = 0.93 + 0.03x$	(3)	$wc_e(x) = 4.85 + 0.24x$	(4)
Terminate CPU (%)	$tc_s(x) = 2.58 + 0.20x$	(5)	$tc_e(x) = 3.89 + 1.31x - 8.63 \times 10^{-3}x^2$	(6)
Warm RAM (MB)	$wr_s(x) = 3.36 \times 10^3 + 11.24x$	(7)	$wr_e(x) = 4.45 \times 10^2 + 9.18x$	(8)
Cold Power (W)	$cp_s(x) = 1.12 \times 10^2 + 4.55 \times 10^{-1}x - 2.24 \times 10^{-3}x^2$	(9)	$cp_e(x) = 2.79 + 8.64 \times 10^{-2}x - 2.49 \times 10^{-3}x^2 + 2.58 \times 10^{-5}x^3$	(10)
Warm Power (W)	$wp_s(x) = 1.10 \times 10^2 + 5.01 \times 10^{-2}x$	(11)	$wp_e(x) = 2.45 + 6.20 \times 10^{-3}x$	(12)
Terminate Power (W)	$tp_s(x) = 1.12 \times 10^{-2} + 0.15x$	(13)	$tp_e(x) = 2.50 + 1.69 \times 10^{-2}x$	(14)
Cold Time (s)	$ct_s(x) = 4.24 + 0.70x$	(15)	$ct_e(x) = 5.13 + 1.06x$	(16)
Terminate Time (s)	$tt_s(x) = 36.91 + 0.35x$	(17)	$tt_e(x) = 36.90 + 0.51x$	(18)

as can be seen from the constant terms in the two equations. The power scale for one CNF on a cloud server is also higher than that of an edge device. Therefore, it is a good strategy to keep functions at the edge even when CNFs are in a warm state, as the operating power is low. This is in contrast to cloud servers.

With the above measurement results, we answer the RQ1 as follows: *Lightweight CNFs enable edge-cloud environments to accommodate CNFs even on low-capability devices. Warm state consumes minimal CPU resources and remains consistent across all CNFs when they are not actively handling tasks. Nevertheless, RAM utilization remains substantial in the warm state, whether on edge or cloud devices. Hence, while CNFs at warm do not impact CPU usage, RAM consumption should be considered. Furthermore, if all CNFs on a machine are in the warm state and not serving any actual traffic, then the power used to operate the machine is wasted. While this may not be a concern for edge devices due to their low-power nature, the high idle power consumption of servers is a problem.*

2) Action Consumption: Results for CPU consumption during CNF's cold start (*Cold*) and termination (*Term*) according to edge and cloud are presented in Fig. 7–a and Fig. 8–a, respectively. Overall, the trends are identical for both edge and cloud, with Cold requiring more computing power than termination. Between the two processes, Term CPU usage increases relatively proportional to the number of pods in case of termination. Meanwhile, Cold CPU seems to be saturated when it comes to high pod numbers, especially on the edge device, where the CPU utilization reaches around 80% at 45 pods. The CPU consumption during these transitions is credited mainly to two processes. One is the resource allocation and management handled by Kubelet. The other is the pre-processing of the nested program. For the edge case, the maximum number of CNFs that can be opened is limited by CPU usage, as evidenced by saturated CPU usage at 45 pods. In contrast, the cloud server can still handle 100 pods with less than half of its CPU capacity. Therefore, it is important to consider the resource consumption of CNF lifecycle transition phases when deploying SFCs in a system, as this may interfere with other running CNFs or processes. Equation (1) and Equation (2) provide approximate CPU values for transitions that can be used to calculate the real CPU requirements of SFC deployment in dynamic environments such as edge–cloud.

We also measured RAM usage during CNF actions, but the results were simply a gradual increase from zero to the warm state's value for allocation and vice versa for termination. Therefore, we omit these results and focus on more significant metrics.

Fig. 7–b and Fig. 8–b describe the duration for Cold or Term CNFs at the edge device and in the cloud server, respectively. While both actions show a linear trend, Term starts at a high value, and its orientation is flatter. The reason stems from a policy called *GraceTime* of Kubernetes, which allows a nested container to finish its job in a defined amount of time before being reaped. The default value is $30\,\mathrm{s}$ [27]. Cold process, on the other hand, takes less time at the low number of pods; however, it gradually reaches, or surpasses, Term value with a high pod number. While the cloud server has a better hardware compared to the edge device, it is surprising that the cold starting duration is not much different between them. As shown in Equation (15) and Equation (16), the edge is only 30% slower than the cloud. This might come from the characteristic of the nested application, more sophisticated programs may favor better hardware more clearly. Term process also shows similar results as depicted in Equation (17) and Equation (18). The duration of pod creation and deletion affects not only the latency of the reconfiguration process but also how long a specific amount of CPU resources must be allocated for these actions and the energy that will be consumed when combined with the power results below.

The average power usage is shown in Fig. 7–c and Fig. 8–c. Similar to state results, the power consumption of actions is proportional to CPU usage. Therefore, cold start consumes more power than termination, even though Term power is also

significant. For edge devices, opening a high number of CNFs (e.g., 20 to 45) consumes 150% to 200% more power than idle power. This gap is only 20% for the cloud with more than 70 CNF creations. This difference is due to the higher CPU usage on edge devices than the average CPU usage on cloud servers for cold-starting CNFs. Between the two computing tiers, the cloud consumes more power to create or terminate the same number of CNFs, even though the trend is similar. This is understandable because of the higher performance of typical servers. With this measured power consumption and the duration mentioned above, the exact amount of energy used by transition processes can be derived by multiplying the duration equations and power equations provided in Table II.

Based on the results, we can answer our second research question RQ2 with: *The reconfiguration process, notably the transitions between states, causes a significant delay, resource demand, and power usage, especially on the edge device. This may affect not only QoS but also interfere with other tasks in the system. Therefore, it is strongly recommended to consider the reconfiguration costs before implementing SFC over edge–cloud. Keeping CNFs warm is also a strategy to avoid the expensive cost of lifecycle invocation. However, the trade-off for RAM resources should be considered carefully.*

V. Conclusions

In this paper, we first study the lifecycle of a CNF and its behavior in dynamically changing environments that may have an impact on the general QoS, system resource demands, and energy efficiency. Then, we perform an investigation by measuring an IoT use case over a real testbed over edge–cloud. Our results show that while the edge–cloud is capable of hosting CNFs, the cost of reconfiguration must be considered in this context. These costs are made transparent with our measurement results and fitted model equations. Consequently, these results can be used for SFC placement strategies and algorithms to manipulate the CNF lifecycle towards a specific objective in a more realistic manner.

References

[1] ETSI NFV, "NFV Release 4 Description," 2022, Available online: https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation.

[2] W. Attaoui, E. Sabir, H. Elbiaze, and M. Guizani, "Vnf and cnf placement in 5g: Recent advances and future trends," *IEEE Transactions on Network and Service Management*, 2023.

[3] Kubernetes. Kubernetes: Production-Grade Container Orchestration. [Online]. Available: https://kubernetes.io

[4] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *2015 IFIP Networking Conference (IFIP Networking)*, 2015, pp. 1–9.

[5] C. Jin, X. Bai, C. Yang, W. Mao, and X. Xu, "A review of power consumption models of servers in data centers," *applied energy*, vol. 265, p. 114806, 2020.

[6] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-Me Cloud: When Cloud Services Follow Mobile Users," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 369–382, 2019.

[7] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS operating systems review*, vol. 43, no. 3, pp. 14–26, 2009.

[8] Github. checkpoint-restore/criu. [Online]. Available: https://github.com/checkpoint-restore/criu

[9] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog: A performance evaluation," *Sensors*, vol. 19, no. 7, p. 1488, 2019.

[10] S. Kounev, C. Abad, I. Foster, N. Herbst, A. Iosup, S. Al-Kiswany, A. A.-E. Hassan, B. Balis, A. Bauer, A. Bondi *et al.*, "Toward a definition for serverless computing," 2021.

[11] Amazon. Serverless Computing - AWS Lambda - Amazon Web Services. [Online]. Available: https://aws.amazon.com/lambda/

[12] A. Mohan, H. Sane, K. A. Doshi, S. Edupuganti, N. Nayak, and V. Sukhomlinov, "Agile Cold Starts for Scalable Serverless," in *HotCloud*, 2019.

[13] K. Mahajan, S. Mahajan, V. Misra, and D. Rubenstein, "Exploiting content similarity to address cold start in container deployments," in *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*, 2019, pp. 37–39.

[14] S. Wu, Z. Tao, H. Fan, Z. Huang, X. Zhang, H. Jin, C. Yu, and C. Cao, "Container lifecycle-aware scheduling for serverless computing," *Software: Practice and Experience*, 2021.

[15] S. Xu, B. Liao, B. Hu, C. Han, C. Yang, Z. Wang, and A. Xiong, "A reliability-and-energy-balanced service function chain mapping and migration method for Internet of Things," *IEEE Access*, vol. 8, pp. 168 196–168 209, 2020.

[16] N. H. Thanh, N. T. Kien, N. Van Hoa, T. T. Huong, F. Wamser, and T. Hossfeld, "Energy-Aware Service Function Chain Embedding in Edge-Cloud Environments for IoT Applications," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[17] B. Németh, M. Szalay, J. Dóka, M. Rost, S. Schmid, L. Toka, and B. Sonkoly, "Fast and efficient network service embedding method with adaptive offloading to the edge," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 178–183.

[18] 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.

[19] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vNF placement at the network edge," in *Ieee infocom 2018-ieee conference on computer communications*. IEEE, 2018, pp. 693–701.

[20] S. Padhy and J. Chou, "Reconfiguration Aware Orchestration for Network Function Virtualization With Time-Varied Workload in Virtualized Datacenters," *IEEE Access*, vol. 9, pp. 48 413–48 428, 2021.

[21] D. Zhao, G. Sun, D. Liao, S. Xu, and V. Chang, "Mobile-aware service function chain migration in cloud–fog computing," *Future Generation Computer Systems*, vol. 96, pp. 591–604, 2019.

[22] H.-G. Kim, D.-Y. Lee, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong, "Machine learning-based method for prediction of virtual network function resource demands," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 405–413.

[23] S. Lange, H.-G. Kim, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong, "Machine Learning-based Prediction of VNF Deployment Decisions in Dynamic Networks," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019, pp. 1–6.

[24] P. Vahidinia, B. Farahani, and F. S. Aliee, "Mitigating cold start problem in serverless computing: A reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 10, no. 5, pp. 3917–3927, 2023.

[25] Prometheus. Prometheus - monitoring system & time series database. [Online]. Available: https://prometheus.io/

[26] M. Luksa, *Kubernetes in action*. Simon and Schuster, 2017.

[27] Kubernetes. Pod Lifecycle. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/