A Coloured Petri Net Model for Hybrid NFV-MEC Systems Supporting URLLC

Caio Souza, Marcos Falcão, Andson Balieiro
Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), Recife, Brazil
{cbbs, mrmf, amb4}@cin.ufpe.br

***Abstract*—Integrating Multi-Access Edge Computing (MEC) and Network Function Virtualization (NFV) favors URLLC, but introduces challenges in resource allocation due to the reduced dimension of NFV-MEC nodes, the overhead caused by virtualization, the use of heterogeneous virtualization technologies, and potential failures during service processing. These factors require careful consideration in resource dimensioning/allocation to ensure that URLLC requirements are met without incurring excessive power consumption due to overprovisioning. This paper proposes a Coloured Petri Net (CPN)-based model for dynamic resource allocation in NFV-MEC systems supporting URLLC services. The proposed model represents a hybrid NFV-MEC system, accounting for virtualization overhead, failure costs, service buffering, and dynamic resource scaling based on demand. It also accommodates system features with diverse behaviors and provides analysis of the system's availability, latency, and power consumption. Results showed that for high arrival rates, improving the container service rate has a greater impact than adding VMs. The benefits of a higher container setup rate on response time become noticeable when VM capacity is exceeded.**

***Index Terms*—Hybrid NFV-MEC, URLLC, Coloured Petri Nets, Dynamic Resource Allocation**

I. Introduction

Integrating Multi-Access Edge Computing (MEC) and Network Function Virtualization (NFV) is essential for supporting Ultra Reliable and Low Latency Communications (URLLC) [1], as it enables hosting virtualized network functions (VNFs) and applications closer to the end users, thereby reducing response times and supporting dynamic resource allocation by instantiating VNFs based on system demand [2].

In the past, NFV has traditionally been implemented over VMs, but containers have gained attention due to their lower computational resource consumption and reduced instantiation overhead [3], which makes them well-suited for dynamic resource allocation. However, by sharing a single kernel, containers may face additional security risks compared to VMs, which offer better isolation [4]. Nonetheless, containers can complement VMs in hybrid NFV-MEC systems, leveraging the strong isolation provided by VMs and the flexibility of containers. For instance, VMs, with their higher setup times that could violate URLLC requirements, can remain continuously active to handle regular demands while containers are dynamically scaled to address variable demands.

Although the NFV-MEC integration favors URLLC, it also presents challenges [5]. For instance, the reduced dimension of NFV-MEC nodes constrains their service capacity, potentially resulting in the forwarding of URLLC service requests to neighboring nodes or the central cloud [6], which introduces uncertainty regarding latency. Moreover, the overhead associated with virtualization, such as NFV setup time and performance degradation, and potential failures during service processing, requires careful consideration in resource dimensioning/allocation to ensure that URLLC requirements are met without incurring excessive power consumption due to overprovisioning.

Resource allocation in NFV-MEC systems has garnered attention in the literature. However, some studies overlook key factors that impact URLLC services, such as the overhead introduced by virtualization, by assuming instantaneous VNF provisioning times [7], presupposing fault-free cloud environments [8], or disregarding repair delays [7], [9]. In addition, many works focus on a single virtualization technology, either VMs [9]–[11] or containers [12]–[15], thereby missing the potential benefits of their integration in hybrid NFV-MEC systems supporting URLLC. Furthermore, several studies are based on formalisms tied to specific probability distributions (e.g., exponential) to model URLLC services or system behaviors [5], [13], [14], [16].

By addressing these gaps, this paper proposes a Coloured Petri Net (CPN)-based model for dynamic resource allocation in NFV-MEC systems supporting URLLC services. The proposed model represents a hybrid NFV-MEC system, accounting for virtualization overhead, failure costs, service buffering, and dynamic resource scaling based on demand while accommodating diverse behaviors for the system features, and providing analysis of the system's availability, latency, and power consumption. Results showed that at high arrival rates, improving the container service rate has a greater impact than adding VMs. The benefits of a higher container setup rate on response time become noticeable when VM capacity is exceeded. Moreover, adding processing resources increases energy consumption, with VMs contributing more at low workloads and containers scaling energy use at higher workloads. Our model may assist in the proper dimensioning and configuration of NFV-MEC systems to support URLLC.

The remainder of this paper is organized as follows. Section II reviews the related works. Section III describes the NFV-MEC system under consideration and presents the proposed CPN-based model. Section IV analyzes the result obtained by extensive discrete-event simulations. Finally, Section V

concludes this paper and highlights potential future directions.

II. Related Work

The integration of MEC and NFV has been essential for reducing latency and improving flexibility in 5G networks, enabling URLLC services. However, resource limitations at the edge may lead to competition among users, causing service quality degradation or even violating the critical requirements of URLLC. In this context, resource allocation in NFV-MEC systems has been widely addressed in the literature. For instance, [17] proposes a quantum approximate optimization algorithm (QAOA) for dynamic resource allocation. The authors apply a generative adversarial network to predict future demands in various regions and use QAOA to pre-allocate edge server resources based on these predictions. Similarly, [18] employs a long short-term memory (LSTM) neural network to predict user demands within a given range. Based on these predictions, the authors dynamically pre-allocate resources from available MEC servers to meet the latency requirements of mobile users by using dynamic programming.

An algorithm to minimize the system's overall energy-time cost by jointly optimizing task offloading and resource allocation in MEC systems is proposed in [19]. The authors formulate a mixed-integer non-linear programming (MINLP) problem, considering interdependencies among tasks within the context of digital twin-aided edge computing for URLLC. Similarly, [7] proposes a subgraph isomorphism-based resource allocation solution for end-to-end slicing targeting URLLC. In [9], the VNF placement problem for URLLC is formulated as an optimization problem with two conflicting objectives: minimizing latency and maximizing service availability. The authors propose a solution based on Genetic Algorithms. Meanwhile, [10] presents a model based on continuous-time Markov chains and stochastic geometry to jointly characterize the communication and computation performance of a dependable MEC-enabled wireless system. The model is used to evaluate the influence of various system parameters on performance and determine the optimal number of virtual resources to maximize task execution capacity. The authors in [20] model and analyze the end-to-end service delay of Service Function Chaining in SDN/NFV architectures by using network calculus. The availability and sensitivity of a containerized IP Multimedia Subsystem for 5G networks are assessed in [21]. The authors use Reliability Block Diagrams for model system interconnections and Stochastic Reward Networks to analyze the system behavior under failure and repair.

Although many studies present important results and provide valuable insights into resource allocation in MEC environments, most overlook key factors that impact URLLC services. Some disregard the overhead introduced by virtualization in NFV-MEC systems, assuming instantaneous VNF provisioning times [7]. Others preuppose fault-free cloud environments [8] [17] [18], or neglect repair delays [7] [9]. In addition, several works focus on a single virtualization technology, either VMs [9] [10] [11] or containers [13] [12] [14] [15], thereby missing the potential benefits of their integration for URLLC support. Moreover, some studies overlook the use of NFV integrated with MEC [17] [18] while others do not specifically target URLLC support [10] [17] [18]. Furthermore, some studies, despite incorporating many of these features, rely on formalisms constrained to specific probability distributions (e.g., exponential) to model URLLC services or system behaviors [5] [16] [13] [14].

III. System Model

We consider an NFV-MEC node that leverages VNFs to process URLLC service requests originating from UEs and routed through the RAN. The VNFs operate equally and independently on single microkernel-based VMs [22] or containers, both sharing a common physical machine (PM). While VMs execute uninterrupted, containers are scaled upon demand by an admission control unit, which only activates containerized VNFs when all VM-hosted ones are busy. The containerized VNF activation includes the kernel image initialization and the specified function instantiation, which is interpreted as setup time, during which power and resources are consumed, but no service processing occurs. Multiple VMs operating in parallel on the same physical node influence each other, leading to a degraded computational power [23]. The node may admit service requests until its maximum capacity is reached and includes a limited buffer to hold requests awaiting service processing.

Active containerized VNFs may experience failures during service processing, necessitating either a service migration to an available VM/container or a repair, which triggers a new setup period. In both cases, service processing is restarted. If no available container/VM exists, the URLLC service is placed back in the buffer, having a higher service priority than new URLLC services. Once an operational VNF completes processing and no additional requests remain, the VNF instance can either be powered down together with the host container or remain active if hosted by a VM. The shutdown delay is considered negligible compared to the significantly longer setup or repair durations [24]. As containers and VMs exhibit different virtualization overheads, mainly when multiple instances coexist on the same physical node, their task processing rates differ even when executing the same VNF.

To analyze the hybrid NFV-MEC system supporting URLLC, we designed a CPN-based model using CPN Tools [25]. This model integrates key system dynamics, including admission control, service queue (buffer), failure occurrences during service processing, repair/setup time, and dynamic resource allocation. Our model admits expressing the service arrival, service processing, failure, and setup/repair events via different probability distributions or functions, not limited to the exponential ones used in Continuous Time Markov Chain (CMTC) and Stochastic Petri Nets approaches, for example.

The CPNs provide a suitable structure for modeling and simulating concurrent and distributed systems with asynchronous and synchronous communication [26]. This structure employs a visual representation comprising places (circles),

transitions (rectangles), and arcs (directed arrows) to model concurrent systems. Places represent the system's state and may contain sets of tokens, with each token assigned a color indicating its data type. All tokens within a place share the same color, which can be simple (e.g., integers) or complex (e.g., composed of other colors). Transitions represent events that can alter the system's state by manipulating tokens based on defined firing rules. Arcs illustrate the relationships between places and transitions, specifying how the state changes when events occur. Arc inscriptions act as functions that define the number and type of tokens transferred between states [26]. The distribution of tokens across places determines whether the conditions for triggering transitions are satisfied.

Fig. 1 illustrates the designed CPN-based model, which comprises 123 arcs, 35 places, 32 transitions, and 17 colors (data types). It is divided into five parts: Service Arrival (Section III-A), VM Service and Management (Section III-B), Container Setup and Service Allocation (Section III-C), Container Service and Failure (Section III-D), and Fast Allocation and Shutdown of Containers (Section III-E). The following description adopts a bold font for tokens, an italic font for functions and variables, and a bold and italic font for transitions and places.

A. Service Arrival

Fig. 1 (highlighted in red) illustrates the model segment for URLLC service arrivals in the NFV-MEC node. Incoming requests are modeled through the ***Customer Arrival*** transition. When this transition fires, it removes a **C** token from the ***Customers*** place and deposits it into the ***Entry Place*** place, indicating the arrival of a new request. Another **C** token is returned to the ***Customers*** after a time interval defined by the function *InterArrivalTime()*, which characterizes the time between successive URLLC arrivals. This interval can follow probability distributions or functions representing the URLLC service being analyzed. Additionally, a **C** token is added to the ***Print*** place to count the number of requests that have arrived. When the number of tokens in ***Print*** reaches the value of *StopCriterion*, the simulation concludes by triggering the ***End of Simulation*** transition.

The admission of new requests is determined by the number of resources (**R** tokens) in the ***Unavailable Resources*** and ***Available Resources*** places, which represent the resources currently in use and those available for allocation, respectively. When ***Unavailable Resources*** contains *systemSize* tokens, indicating that the system is at full capacity, a **C** token in ***Entry Place*** triggers the ***Rejection*** transition, which removes a **C** token from ***Entry Place*** and adds it to ***Blocked Customers***, meaning that the service request was rejected. On the other hand, tokens in ***Available Resources*** and ***Entry Place*** trigger the ***Customer Admission*** transition, representing a service admission. This transition consumes a token from these places and inserts a token (**(U,t)**) into ***Admission Queue***. This token denotes a service request along with its admission time, as obtained by the *ModelTime()* function. In addition, a **R** token is placed in ***Unavailable Resources***. The total number of tokens in ***Unavailable Resources*** and ***Available Resources*** always equals the total system capacity.

B. VM Service and Management

Fig. 1 highlights in blue the segment model for VM Management and Service Provision. Once admitted into the ***Admission Queue*** place, service requests can be handled by containers or VMs. The VMs are provisioned during system initialization and remain active throughout the system's operation. This provisioning is triggered by the ***Create VM*** transition, which consumes tokens from ***countVM*** and ***numVMsSim***. These places are initialized with one and **numVMs** tokens, respectively, where **numVMs** represents the total number of VMs in the NFV-MEC system, and each token assumes the value **1**. When ***Create VM*** fires, it inserts a VM token (**B(C,i)**) into the ***Idle VMs*** place, where **i** represents the VM identifier. This place models the pool of VMs available for service processing. Additionally, the ***Create VM*** transition places a new token with a value of **i+1** in ***countVM***. Together, ***countVM*** and ***Create VM*** function as an identifier generator for the VMs.

Tokens in ***Idle VMs*** and ***Admission Queue*** trigger the ***Service Beginning in VM*** transition. It removes tokens from both places and inserts a **E(C,t,i)** token into the ***Working VMs*** place, meaning that the service request that arrived at time **t** is being processed by VM **i**. A resource token (**R**) is also added to the ***Busy VMs*** place to keep track of the number of VMs currently in use. Finally, a temporized token **E(C,t,i)** is generated in ***Servicing by VM*** after an interval time defined by the *VMServiceTime()* function. This function models the service processing time within VMs and can be described using probability distributions or functions.

When no VMs are available to process requests (i.e., no tokens in ***Idle VMs***), a container can be initialized to handle the request queued in the ***Admission Queue***. During the container initialization process, the request waits in the ***Initialization Buffer*** place. However, if a VM becomes available (a token is added to ***Idle VMs*** after a service conclusion) while the request is still in the initialization buffer, the request is immediately assigned to the VM for processing. This scenario represents the second scenario of service handled by VMs. Processing begins by triggering the ***Service Beginning in VM C2*** transition. This transition consumes a token from ***Idle VMs*** and generates tokens in the ***Working VMs***, ***Busy VMs***, and ***Servicing by VM*** places, while consuming the request token from the ***Initialization Buffer***.

Once the temporized token **E(C,t,i)** is available in ***Servicing by VM***, the ***Service Termination in VM*** transition fires, generating a **D(c,t,i)** token in the ***Ended Services*** place. This token indicates the conclusion of service processing by VMs in both scenarios. Subsequently, the **D(C,t,i)** token is consumed by the ***ResponseTimeCalc*** transition to compute the service response time. Moreover, the ***Service Termination in VM*** transition also adds a token to ***Idle VMs***, indicating that a VM is now available to process requests, and a resource token to ***Available Resources***, enabling the admission of a new request

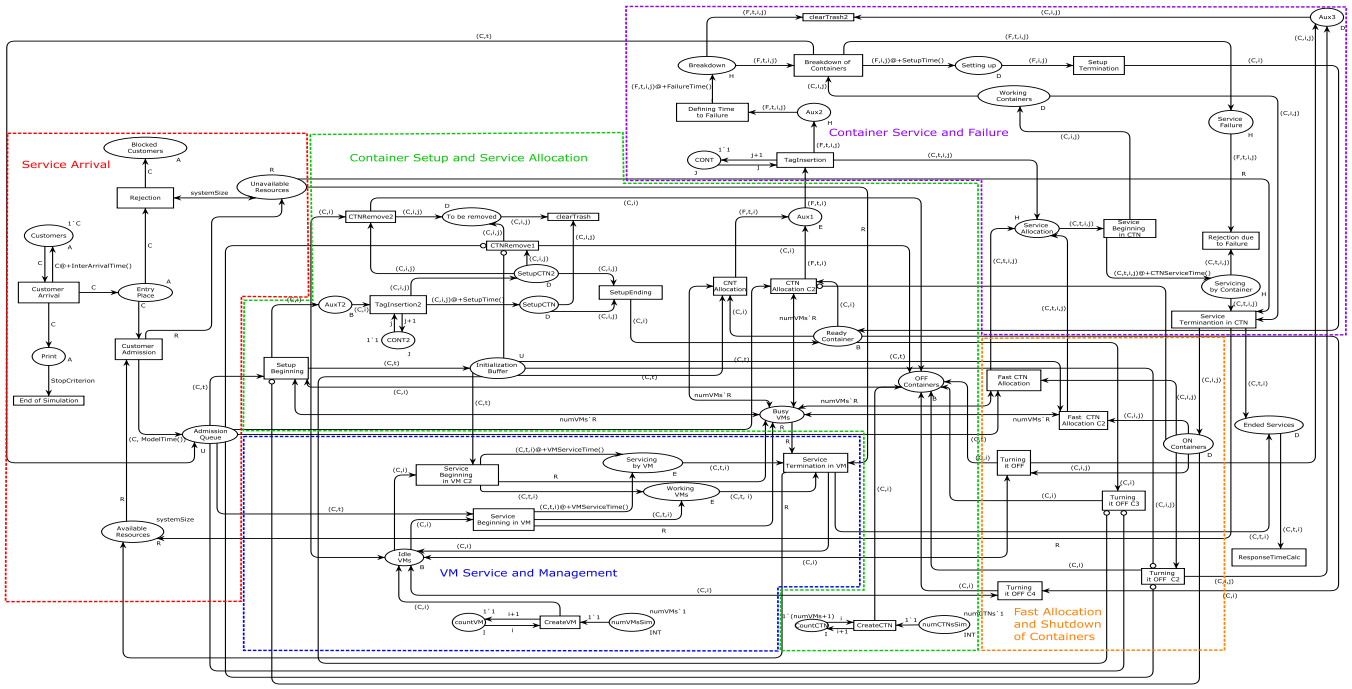


Fig. 1. The CPN-Based Model for Hybrid NFV-MEC Supporting URLLC.

into the system. Additionally, other tokens are consumed by the ***Service Termination in VM*** transition, including tokens from ***Working VMs***, ***Busy VMs***, and ***Unavailable Resources***. This last one prevents new requests from being rejected.

To account for CPU overhead caused by concurrently operating VMs, we use Eq. 1 to define the effective VM service rate (μ^*_{VM}). This rate considers the nominal service rate (μ_{VM} services/unit), the number of parallel VMs, and a computation degradation factor (d). The resulting effective rate is then passed as input to the *VMServiceTime()* function.

$$\mu_{VM}* = \frac{\mu_{VM}}{(d+1)^{(n-1)}} \tag{1}$$

C. Container Setup and Service Allocation

The green segment in Fig. 1 models Container Setup and Service Allocation in the NFV-MEC node. The containers are created in inactive state firing the ***Create CTN*** transition, which consumes a token from ***numCTNsSim*** and another from ***countCTN***. These places are initialized with *numCTNs* tokens with value 1 and one token valuing *numCTNs+1*, respectively, representing the number of containers dimensioned to the NFV-MEC system and the initial identifier for containers. When fired, ***Create CTN*** inserts a **B(C,i)** token into ***OFF Containers*** place, with i being the container identifier and a new token with a value of **i+1** into ***countCTN***.

When no available VMs are present in ***Idle VMs*** to process a service request, a container from the ***OFF Containers*** place can be instantiated to handle the requests. This process is modeled by triggering the ***Setup Beginning*** transition, which requires no tokens in the ***ON Containers*** place. Upon firing, this transition removes a request token from ***Admission Queue*** and a token from ***OFF Containers***, representing the container being initialized to handle the request. Additionally, it checks (consumes and places back) the *numVMs* **R** tokens in the ***Busy VMs*** place, ensuring that all VMs are busy. The ***Setup Beginning*** transition generates a **B(C,i)** token in ***AuxT2***, indicating that the container **i** is being initialized, and a token **U(C,t)** in ***Initialization Buffer***, which denotes that the request was placed in the buffer. During the container setup, the request may be processed by another resource (either an instantiated VM or container) that becomes available in the meantime. In this case, to optimize computational resources, the container under initialization must be turned off, as there are no other requests waiting for processing. To do so, our model defined an identifier for each container initialization by triggering the ***TagInsertion2*** transition, which consumes a token from ***AuxT2*** and another token valuing **j** from ***CONT2***.

The firing of ***TagInsertion2*** generates a **D(c,i,j)** token in ***SetupCTN2***, indicating that the container **i** has initiated the setup process **j**. After a delay defined by *SetupTime()*, a corresponding token is placed in ***SetupCTN***. If, during setup, no requests remain or a VM becomes available, the ***CTNRemove1*** and ***CTNRemove2*** transitions cancel the setup, removing the **D(c,i,j)** token and returning **B(C,i)** to the ***OFF Containers*** place. The completion of the container setup is modeled by the initialization token **D(c,i,j)** and its corresponding pair **i** and **j** in the ***SetupCTN*** and ***SetupCTN2*** places, respectively. These tokens trigger the ***SetupEnding*** transition, which adds a container token **B(C,i)** to the ***ReadyContainer*** place. To prevent the accumulation of tokens from canceled container

setups in ***SetupCTN***, the ***CTNRemove1*** and ***CTNRemove2*** transitions generate a **D(c,i,j)** token in ***To be removed***. This token, along with its corresponding **D(c,i,j)** token in ***SetupCTN***, is subsequently cleared by the ***clearTrash*** transition.

Once the token **B(C,i)** is placed in ***ReadyContainer*** and a request token is present in ***Initialization Buffer***, the ***CNT Allocation*** transition fires, assigning the container to process the request. In addition to consuming tokens from the previous places, this transition verifies the presence of *numVMs* **R** tokens in ***Busy VMs***, ensuring that container allocation occurs only when all VMs are occupied. Furthermore, it places a token **E(F,t,i)** in ***Aux1***, where **t** represents the request's arrival time in the system, and **i** identifies the container.

The initialized container can process another admitted request in ***Admission Queue*** if the request that triggered its initialization has already been attended. This behavior is modeled by the ***CNT AllocationC2*** transition, which functions similarly to ***CNT Allocation***, but consumes the request token from ***Admission Queue*** instead of ***Initialization Buffer***. This transition fires only when there are no tokens in ***ON Containers***, indicating the absence of ready containers for processing.

D. Container Service and Failure

Fig. 1 (in purple) models the service processing in containers and eventual failure occurrences during processing. When a token is placed in ***Aux1***, the ***TagInsertion*** transition is enabled and combines the token with a failure token valued **j** from ***CONT***, generating a token **H(F,t,i,j)** in ***Aux2*** and a service token **H(C,t,i,j)** in ***Service Allocation***, where **j** identifies the container **i** in failure events. Since a container may handle different services throughout the NFV-MEC operation, this failure marking allows for identifying the specific service instance during which the failure occurred. The ***CONT*** is initialized with an integer token equal to 1, which increments by 1 upon firing of ***TagInsertion***.

The ***Defining Time to Failure*** transition models potential failures during service processing. It consumes a token from ***Aux2*** and generates a temporized token **H(F,t,i,j)** in ***Breakdown***, with the failure time determined by the *FailureTime()* function. This function may follow probability distributions that characterize the interval between successive failures. Meanwhile, the service processing is initiated by firing the transition ***Service Beginning in CTN***, which generates a temporized token **H(C,t,i,j)** in ***Servicing by Container*** after a time interval determined by the function **CTNServiceTime()**, characterizing the container service time, and a token **D(c,i,j)** in ***Working Containers***.

If the service time token appears in ***Servicing by Container*** before the failure token in ***Breakdown***, the ***Service Termination in CTN*** transition fires, signaling the conclusion of the service. This transition removes the token **H(C,t,i,j)** from ***Servicing by Container***, the token **D(c,i,j)** from ***Working Containers***, indicating that the container is no longer occupied, and also a token **R** from ***Unavailable Resources***, enabling the admission of another service request into the system. In addition, this transition adds a **R** token in ***Available Resources***, a **D(c,i,j)** token in ***ON Containers***, representing that the container **i** is ready to process another service, and a request token **D(c,t,i)** to ***Ended Services*** for subsequent computation of the total service response time through the firing of ***ResponseTimeCalc***.

On the other hand, if the failure token **H(F,t,i,j)** becomes available in ***Breakdown*** while the service is being processed, the ***Breakdown of Containers*** transition triggers, indicating a failure occurrence. This transition consumes the **H(F,t,i,j)** and **D(c,i,j)** tokens and returns the service request token **U(C,t)** to the ***Admission Queue*** place, preserving the original request's arrival time **t**. Simultaneously, the failed container undergoes reinitialization by generating a temporized token **D(F,i,j)** in ***Setting up***, with a time interval defined by the function *SetupTime()*, simulating the container setup time. Once the token appears in **D(F,i,j)**, the ***Setup Termination*** transition fires, placing a container token in ***Ready Container***, denoting that the container is ready to resume service processing. To ensure the proper handling of failed services and avoid the accumulation of unused tokens in ***Servicing by Container***, the ***Breakdown of Containers*** transition generates tokens **H(F,t,i,j)** in ***Service Failure***, which enable the ***Rejection due to Failure*** transition to remove the correct service tokens **H(C,t,i,j)** from ***Servicing by Container***.

E. Fast Allocation and Shutdown of Containers

The final model segment (orange in Fig. 1) represents the NFV-MEC node's Fast Container Allocation and Shutdown processes. When service processing is completed, the container is made available in ***ON Containers*** to handle other requests from ***Admission Queue*** and ***Initialization Buffer*** through the ***Fast CTN Allocation*** and ***Fast CTN Allocation C2*** transitions, respectively. However, these transitions only fire in the presence of (**numVMs**) **R** tokens in ***Busy VMs***, representing that all VMs are occupied. The failure times of containers allocated by these transitions continue to be counted, as determined by their prior initializations.

If there is at least one VM available for processing (i.e., the ***Idle VMs*** place is not empty), the container in ***ON Containers*** is deactivated via the ***Turning it OFF*** transition, which verifies the existence of **B(C,i)** tokens in ***Idle VMs***, consumes a token **D(c,i,j)** from ***ON Containers***, and generates a container token **B(C,i)** in ***OFF Containers***. This transition also places a token in ***Aux3*** to enable the ***clearTrash2*** transition, thereby preventing the accumulation of unused tokens (deactivated containers) in ***Breakdown***. Similarly, if ***Admission Queue*** and ***Initialization Buffer*** are empty, the containers in ***ON Containers*** are turned off by the ***Turning it OFF C2*** transition, which does not require checking of tokens in ***Idle VMs***. If a token container becomes available in ***Ready Container*** and either a token exists in ***Idle VMs*** or no requests are waiting service processing, the ***Turning it OFF C3*** and ***Turning it OFF C4*** transitions deactivate the container, functioning similar to the ***Turning it OFF C2*** and ***Turning it OFF***, respectively, but not generating tokens in ***Aux3***, as failure times have not yet been assigned to these containers.

F. Performance Metrics

In NFV-MEC networks, service processing at the MEC server is highly dependent on resource availability and response time, while efficient power management is crucial for optimizing network operational costs [27]. Thus, availability analysis is essential in NFV-MEC systems, resource constraints can limit service capacity and introduce uncertainties in latency. Here, availability (A) is defined as the ratio of admitted services (triggering ***Customer Admission***) to the total number of service requests received (triggering ***Customer Arrival***), reflecting the system's capability to accommodate new URLLC service requests. Response time is critical for URLLC services due to their stringent requirements. In our model, it is calculated as the difference between the firing times of ***Service Termination*** and ***Customer Admission***, encompassing all events from service admission to service conclusion (e.g., processing delay, queuing delay, VNF instantiation delay, and VNF recovery delay).

Power consumption (P) is a crucial component of the NFV-MEC node's operational costs. In the proposed model, it is calculated by Eq. 2, which sums the power consumption (W) of containers in idle, setup, and busy states, denoted as P_{idle}^{CNT}, P_{setup}^{CNT}, and P_{busy}^{CNT}, respectively, and the power consumption (W) of VMs in idle (P_{idle}^{VM}) and busy (P_{busy}^{VM}) states. These components are computed using Eqs. 3, 4, 5 from [28] for containers, where x represents the number of containers in the respective state. Similarly, 7 and 6 compute Pv_{idle} and P_{busy}, respectively, where y denotes the number of VMs in the corresponding state.

The number of containers in idle and busy states is represented by the number of tokens in ***Ready Container*** and ***Working Containers***, respectively. The sum of tokens present in ***Setting Up 1*** and ***Setting Up*** indicates the number of containers in the setup state. Similarly, the number of tokens in ***Idle VMs*** and ***Busy VMs*** corresponds to the number of VMs in idle and busy states, respectively.

$$P = P_{idle}^{CNT} + Pc_{busy}^{CNT} + Pc_{setup}^{CNT} + P_{idle}^{VM} + P_{busy}^{VM} \quad (2)$$

$$P_{idle}^{CNT} = 110 + 0.0501x \quad (3)$$

$$P_{setup}^{CNT} = 112 + 0.455x - 0.00224x^2 \quad (4)$$

$$P_{busy}^{CNT} = 110 + 7.23x \quad (5)$$

$$P_{idle}^{VM} = 110 + 0.2505y \quad (6)$$

$$P_{busy}^{VM} = 110 + 7.858y \quad (7)$$

IV. Result Analysis

Results from our CPN-based model (markers) were validated against our previous CTMC-based model (lines) [16]. Thus, the functions *InterArrivalTime(), SetupTime(), FailureTime, VMServiceTime(), and CTNServiceTime()* were set as exponential distributions. Due to page limitations, only exponential distributions are used in this paper; however, our model is flexible and can accommodate other behaviors for the features. We consider three scenarios, each evaluating the influence of a pair of parameters on the system performance: number of containers (c) and number of VMs (n) in Section IV-A, which denote the size of the NFV-MEC system; VM and container service rates, (μ_{VM}) and (μ_{CNT}), in Section IV-B, which may represent enhancements in service request process speed by VMs and containers, respectively; container setup rate (α) and number of VMs in Section IV-C, where the former denotes hardware and software improvements for reducing the time to make network functions ready for processing services. For all scenarios, we considered different URLLC loads, varying the arrival rate (λ) from 5 to 30 requests/ms. Unless stated otherwise, the baseline values for the failure rate (γ) and α were set to 0.001 and 1 unit/ms, respectively, in accordance with [24]. The arrival rate values in our scenarios were based on 3GPP Release 16 (TR 38.824) [29] URLLC use cases and adapted to reflect varying user loads at the node level. Thus, the rates for exponential distributions were set according to Table I. The following results represent the average of each metric, considering 10 simulation instances, with 2700000 steps and a confidence interval of 95%.

A. Varying the Amount of VMs and Containers

This section examines the effect of varying the number of containers and virtual machines (VMs) on system performance. Fig. 2a illustrates that increasing the processing resources leads to enhanced system availability, as expected. At lower request rates, all configurations exhibit overlapping curves with availability near 100%. However, as the request rate approaches the system's capacity, significant differences in availability emerge between configurations.

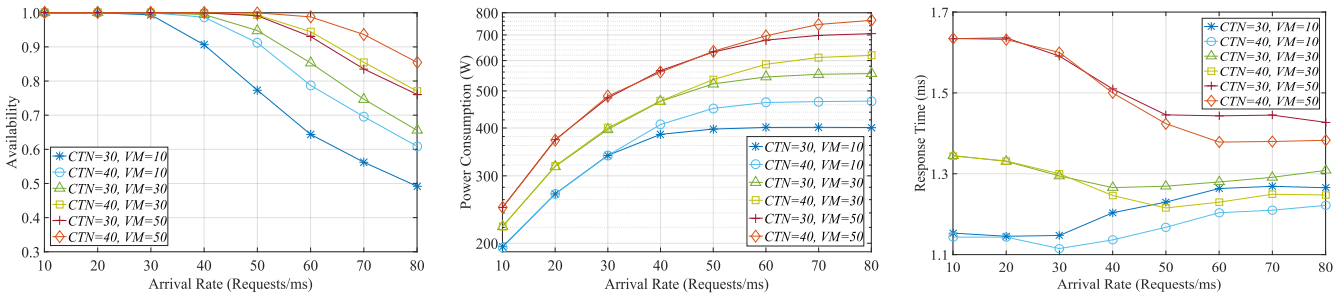
For instance, the configuration with 40 containers and 50 VMs (orange curve) demonstrates a 36% higher availability compared to the configuration with 30 containers and 10 VMs (dark blue curve) when the request rate reaches its peak at 80 requests/ms. Interestingly, the type of resource also influences availability differently. Comparing the light green curve (40 containers and 30 VMs, totaling 70 resources) to the red curve (30 containers and 50 VMs, totaling 80 resources), the system with more containers achieves a 2% higher availability despite having 10 fewer resources when the request rate reaches 70 requests/ms. This underscores the trade-offs between resource types and system performance under different workloads.

In terms of energy consumption, Fig. 2b reveals that while adding processing resources improves performance, it also increases energy consumption. At low workloads, energy consumption primarily correlates with the number of VMs, with an observed increase of 27 W for every 20 additional VMs. Conversely, adding containers under low-load conditions has a negligible impact on energy consumption, as requests are initially handled by pre-activated VMs. Containers are only activated when the workload exceeds VM processing capacity, and their near-zero energy consumption in idle states ensures minimal impact on overall consumption during low demand.

Under high workloads, however, both resource types remain active, driving higher energy consumption. Systems with more VMs exhibit lower consumption variations, as containers are

TABLE I
SCENARIO SETS

Section	Parameters	n	c	q	d	γ	α	μ_{VM}	μ_{CTN}
IV-A	c, n	10, 30, 50	30, 40	10	10^{-2}	10^{-3}	1	1	1
IV-B	μ_{VM}, μ_{CTN}	10	40	10	10^{-2}	10^{-3}	1	1, 2	0.5, 1, 2
IV-C	α, n	10, 30, 50	40	10	10^{-2}	10^{-3}	10^{-1}, 1	1	1



(a) Availability. (b) Energy consumption. (c) Response Time.

Fig. 2. Effects of varying the amount of VMs (n) and Containers (c).

less frequently activated, leaving VMs to handle the majority of requests. In contrast, configurations with fewer VMs require earlier activation of containers, increasing overall energy consumption. For example, configurations with 10 VMs (light and dark blue curves) show a 17% higher energy consumption compared to configurations with 50 VMs (red and orange curves) when 10 additional containers are introduced at a request rate of 80 requests/ms.

Regarding the response time, Fig. 2c shows that at low request rates, service requests are predominantly handled by VMs. Configurations with more VMs experience longer response times due to the overhead of managing operating systems within each VM. For example, increasing from 30 VMs (light and dark green curves) to 50 VMs (orange and red curves) leads to a 22.5% increase in average response time when the arrival rate is 10 requests/ms. As the request rate rises, more containers are activated, leading to faster request processing due to the reduced overhead associated with containers. This dynamic reduces the system's average response time. However, near maximum capacity (80 requests/ms), response times rise slightly, especially in configurations with fewer resources, where higher queue lengths contribute to the observed delay.

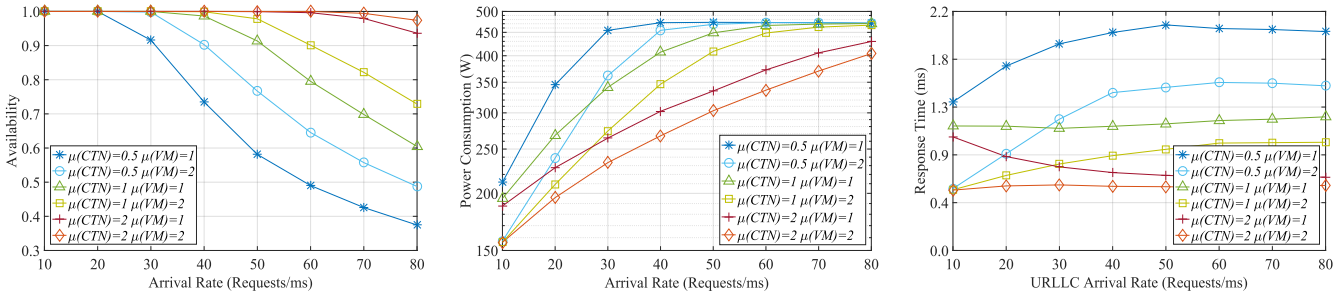
B. Varying the VM and Container Service Rates

This scenario investigates the influence of varying service rates for containers and virtual machines (VMs) on system performance. As illustrated in Fig. 3a, the impact of increasing the container service rate becomes apparent only when the request arrival rate surpasses the processing capacity of the 10 VMs deployed in the environment. Systems utilizing containers with higher service rates demonstrate reduced sensitivity to increased processing demands, resulting in improved availability. For instance, when the request arrival rate reaches 50 requests/ms (approaching the system's total processing capacity), configurations with a VM service rate of 1 and container service rates of 0.5 (blue) and 1 (green) exhibit a 33% difference in availability.

Conversely, enhancing the VM service rate leads to notable improvements in availability as service demand intensifies. Increasing the VM service rate from 1 to 2 yields an 18.5% improvement in availability for configurations with a container service rate of 0.5 (light and dark blue), particularly as the arrival rate approaches 50 requests/ms. Interestingly, boosting the VM service rate proves more effective than merely increasing the number of VMs. For example, raising the VM service rate from 1 to 2 in configurations with a container service rate of 1 (light and dark green) achieves availability gains comparable to increasing the number of VMs from 10 (light blue) to 30 (light green), as observed in IV-A when the arrival rate reaches 80 requests/ms.

Fig. 3b shows that higher VM service rates (represented by the orange, light green, and light blue curves) result in lower energy consumption, particularly at lower request arrival rates (e.g., 10 requests/ms). At these rates, VMs with higher service rates spend less time in the high-energy processing state, leading to energy savings of up to 55 W. As the request arrival rate increases, this trend continues, especially in configurations with lower container service rates. For example, at a request rate of 20 requests/ms, systems with a container service rate of 0.5 (blue and light blue) and higher VM service rates exhibit energy consumption differences of up to 107 W. However, when the request arrival rate exceeds the MEC node's service capacity, the system's resources remain in the processing state continuously, causing energy consumption to converge toward the maximum of 466 W.

Regarding the response time in Fig. 3c, higher VM service rates consistently reduce response times across all configurations. This effect is particularly pronounced at lower request arrival rates. For instance, in configurations with lower



(a) Availability. (b) Energy consumption. (c) Response Time.

Fig. 3. Effects of varying the VM (μ_{VM}) and Container (μ_{CTN}) Service Rates.

container service rates (light and dark blue), increasing the VM service rate from 1 to 2 reduces response time by 53% when the request arrival rate reaches 20 requests/ms. At higher arrival rates, a higher container service rate becomes more critical for minimizing response times, particularly when the system's VM processing capacity is exceeded. This is evident in configurations with a VM service rate of 2 and container service rates of 0.5 and 2 (light blue and red). This analysis highlights the importance of balancing VM and container service rates to optimize system performance, particularly in environments with high request arrival rates.

C. Varying the Container Setup Rate and Number of VMs

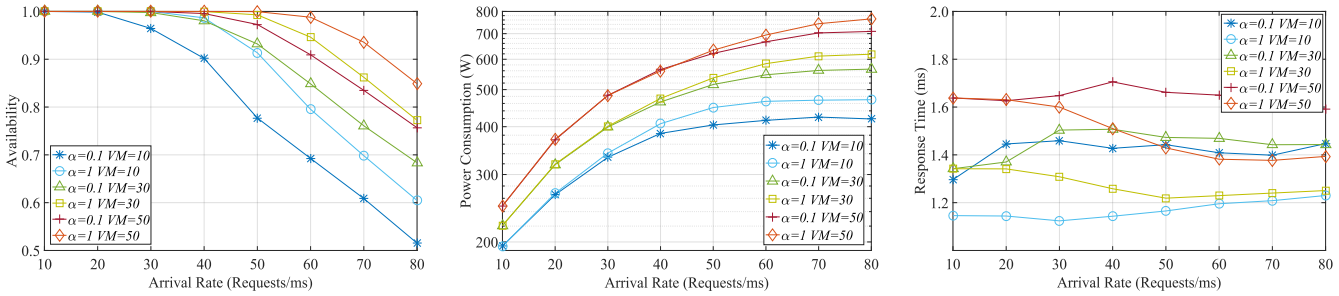
This section evaluates the influence of different container setup rates associated with different numbers of VMs on the MEC node. Fig. 4a illustrates that a higher container setup rate leads to higher system availability, as it makes containers available for service more quickly, reducing the number of users waiting in the queue. This can be observed in all configurations of the number of VMs observed, where, for example, at the point where the request arrival rate reaches 70/ms, all configurations with setup rates of 1 and 0.1 showed availability differences of 10%. In addition, improvements in the number of VMs mainly imply a delay in the drop in availability in relation to the growth in the number of requests/ms, in which for configurations with only 10 VMs available, availability begins to decrease with 20 requests/ms, while for scenarios with 50 VMs, availability remains close to 100% up to 40 requests/ms.

Regarding energy consumption, in Fig. 4b the experiment shows that the better availability caused by the higher setup rate for containers also brings with it an increase in energy consumption. This can be seen when we analyze the arrival rates at which the curves of the configurations with the highest setup rate begin to differ from the curves with the lowest setup rate for configurations with the same number of VMs; these points coincide with the points at which availability begins to differ for these same scenarios. On the other hand, the lower energy consumption presented by the configurations with the lowest setup rates does not necessarily represent a good sign, since this reduction in consumption is due to the containers spending less time in the processing phase and more time in the setup phase, which has lower energy consumption, but does not perform any service while energy is being consumed.

By examining the response time depicted in Fig. 4c, it is clear that a higher container setup rate leads to a reduced response time, as expected. This reduction in response time can be observed mainly when the request arrival rate exceeds the processing capacity of the VMs available in the system. Therefore, for smaller numbers of VMs, this is evident closer to the beginning of the curves due to the lower utilization of VMs, which increases availability but suffers performance degradation as their number increases in the system, as previously mentioned. By analyzing the graph, it is easy to infer that the configurations with a configuration rate of 0.1, a number of VMs equal to 10 and 30, and a configuration rate equal to 1 with a number of VMs equal to 50 (dark blue, dark green, and orange) are equivalent to each other for arrival rates greater than 50. However, despite similar response times ($\sim$ 1.4 ms), these configurations differ significantly in availability, as previously discussed.

V. Conclusions and Future Directions

This work presented a CPN-based model for dynamic virtual resource allocation in MEC-NFV systems supporting URLLC services. The model captures key aspects of a hybrid NFV-MEC system, including virtualization overhead, failure costs, service buffering, and dynamic resource scaling based on demand. The results highlight that at high request arrival rates, improving container service rates is more effective than adding VMs, and higher setup rates significantly reduce response time when VM capacity is exceeded. Additionally, while increasing processing resources enhances performance, it also leads to higher energy consumption. VMs contribute more to energy usage at low workloads, whereas containers offer more efficient scaling at higher workloads. Response time is initially higher with more VMs due to OS overhead, but improves under heavy loads as containers activate. In summary, the proposed model provides valuable insights into the operational dynamics of MEC-NFV nodes, helping network operators optimize resource allocation. Using this model, operators can ensure URLLC requirements are met without excessive power consumption caused by overprovisioning.



(a) Availability. (b) Energy consumption. (c) Response Time.

Fig. 4. Effects of varying the the Container Setup Rate (α) and Number of VMs (n).

ACKNOWLEDGMENT

This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Code 001 and by the UFPE/ Propesqi via Edital Nº 06/2024

REFERENCES

[1] M. U. A. Siddiqui, H. Abumarshoud, L. Bariah, S. Muhaidat, M. A. Imran, and L. Mohjazi, "Urllc in beyond 5g and 6g networks: An interference management perspective," *IEEe Access*, vol. 11, pp. 54 639–54 663, 2023.

[2] P. Xue and Z. Jiang, "Secrouting: Secure routing for network functions virtualization (nfv) technology," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1727–1731, 2022.

[3] R. Morabito, "Power consumption of virtualization technologies: an empirical investigation," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 522–527.

[4] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance overhead comparison between hypervisor and container based virtualization," in *2017 IEEE 31st International Conference on advanced information networking and applications (AINA)*. IEEE, 2017, pp. 955–962.

[5] M. Falcao, C. B. Souza, A. Balieiro, and K. Dias, "An analytical framework for urllc in hybrid mec environments," *The Journal of Supercomputing*, pp. 1–20, 2022.

[6] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P.-V. Mekikis, A. Antonopoulos, and C. Verikoukis, "Online vnf lifecycle management in an mec-enabled 5g iot architecture," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4183–4194, 2019.

[7] Z. Tong, T. Zhang, Y. Zhu, and R. Huang, "Communication and computation resource allocation for end-to-end slicing in mobile networks," in *2020 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2020, pp. 1286–1291.

[8] W. Li and S. Jin, "Performance evaluation and optimization of a task offloading strategy on the mobile edge computing with edge heterogeneity," *The Journal of Supercomputing*, vol. 77, no. 11, 2021.

[9] L. Yala, P. A. Frangoudis, and A. Ksentini, "Latency and availability driven vnf placement in a mec-nfv environment," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–7.

[10] M. Emara, H. ElSawy, M. C. Filippou, and G. Bauch, "Spatiotemporal dependable task execution services in mec-enabled wireless systems," *IEEE wireless communications letters*, vol. 10, no. 2, pp. 211–215, 2020.

[11] N. Kherraf, H. A. Alameddine, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, "Optimized provisioning of edge computing resources with heterogeneous workload in iot networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 459–474, 2019.

[12] A. Samanta and J. Tang, "Dyme: Dynamic microservice scheduling in edge computing enabled iot," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6164–6174, 2020.

[13] C. Souza, M. Falcao, A. Balieiro, and K. Dias, "Modelling and analysis of 5g networks based on mec-nfv for urllc services," *IEEE Latin America Transactions*, vol. 19, no. 10, pp. 1745–1753, 2021.

[14] C. Souza, M. Falcao, A. Balieiro, T. Taleb, and E. Alves, "Enabling the embb and urllc coexistence in mec-nfv networks," in *IEEE International Conference on Communications*, 2024, pp. 159–164.

[15] C. Souza, R. Dos Reis, M. L. Damasceno, M. Falcão, and A. Balieiro, "A cpn-based model for resource allocation in multi-access edge computing supporting urllc," in *EEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2024, pp. 1–6.

[16] M. Falcão, C. Souza, A. Balieiro, and K. Dias, "Dynamic resource allocation for urllc in uav-enabled multi-access edge computing," in *2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2023, pp. 293–298.

[17] C. Gong, W. He, T. Wang, A. Gani, and H. Qi, "Dynamic resource allocation scheme for mobile edge computing," *The Journal of Supercomputing*, vol. 79, no. 15, pp. 17 187–17 207, 2023.

[18] A. Rago, G. Piro, G. Boggia, and P. Dini, "Anticipatory allocation of communication and computational resources at the edge using spatio-temporal dynamics of mobile users," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4548–4562, 2021.

[19] M. Awais, H. Pervaiz, Q. Ni, and W. Yu, "Task dependency aware optimal resource allocation for urllc edge network: A digital twin approach using finite blocklength," *IEEE Transactions on Green Communications and Networking*, 2024.

[20] Q. Duan, "Modeling and performance analysis for service function chaining in the sdn/nfv architecture," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 476–481.

[21] M. Di Mauro, G. Galatro, M. Longo, F. Postiglione, and M. Tambasco, "Ip multimedia subsystem in a containerized environment: availability and sensitivity evaluation," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 42–47.

[22] T. Fautrel, L. George, F. Fauberteau, and T. Grandpierre, "An hypervisor approach for mixed critical real-time uav applications," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019, pp. 985–991.

[23] I. Mavridis and H. Karatza, "Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing," *Future Generation Computer Systems*, vol. 94, pp. 674–696, 2019.

[24] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Comm.*, vol. 24, no. 3, 2017.

[25] C. Tools. (2025) Cpn tools; a tool for editing, simulating, and analyzing colored petri nets. [Online]. Available: http://cpntools.org/

[26] A. Shahidinejad, M. Ghobaei-Arani, and L. Esmaeili, "An elastic controller using colored petri nets in cloud computing environment," *Cluster Computing*, vol. 23, no. 2, pp. 1045–1071, 2020.

[27] S. Kekki and W. Featherstone, "Mec in 5g networks," *ETSI White Paper*, no. 28, p. 1–28, 2018.

[28] K. Nguyen, F. Simonovski, F. Loh, T. Hoßfeld, and N. H. Thanh, "Investigation of container network function deployment costs in the edge cloud," in *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE, 2024, pp. 9–16.

[29] 3GPP, "System architecture for the 5g system (5gs)," *White Paper*, 2020.