Application of Cybertwin for Offloading in Mobile Multiaccess Edge Computing for 6G Networks

Tiago Koketsu Rodrigues, *Member, IEEE*, Jiajia Liu, *Senior Member, IEEE*, and Nei Kato, *Fellow, IEEE*

***Abstract*—Multiaccess edge computing is an essential technology that academia and industry have recognized as fundamental for the future of the Internet of Things. Current research on the subject utilizes virtual machines as the intermediary between end devices and cloud servers. However, recently a new framework was proposed that utilizes Cybertwins instead of virtual machines for the same function. Such framework comes with a myriad of advantages but, most importantly, in this case, it includes a control plane capable of enabling cooperation between the Cybertwins. In this article, we present a mathematical model of the total service delay of a Cybertwin-based multiaccess edge computing system that includes user mobility, migration of virtual servers, multiple physical servers at different network tiers, fronthaul and backhaul communication, processing, and content request/caching. We also propose algorithms for guiding the operation of Cybertwins and the control plane in a multiaccess edge computing scenario. Finally, a performance analysis between Cybertwin and a virtual machine-based scheme is offered. Simulations show that Cybertwin brings significant improvement for the assumed scenario in the form of a faster overall service due to the higher cooperation. The models and simulations here were designed with the characteristics of future networks, beyond the current 5G, in mind, making them likely relevant for future networks, where multiaccess edge computing and the Internet of Things should play an even more important role.**

***Index Terms*—6G, beyond 5G, cloudlets, cybertwin, mathematical analysis, multiaccess edge computing (MEC), offloading, virtual machine (VM).**

I. Introduction

MULTIACCESS edge computing (MEC) [1] is a technology that allows end devices to have fast access to resource-rich cloud servers. These servers are conventionally located in the core of the network, distant from the end users. MEC, however, introduced the concept of edge servers. User devices can offload their requests to servers called cloudlets that are on the edge of the network instead of the remote servers in the network core. Cloudlets have individually fewer resources than the remote cloud but are densely deployed in the network edge. With cloudlets, there is faster access to MEC, which real-time applications can benefit a lot from, as well as a decrease in the workload of both the remote cloud servers themselves as well as the access to the network cloud [2], [3]. These are substantial benefits that have elevated MEC as one of the foundations of future networks including beyond 5G and possibly even 6G [4]. Additionally, the Internet of Things (IoT) has much to gain from MEC as well [5], [6]. IoT devices should generate a massive amount of data (this is expected especially in 6G, given the expected number of IoT devices in the future), which will need to be processed quickly to satisfy the real-time requirements of the end users (e.g., smart factories, eHealth, and autonomous driving) [3]. Alternatively, a lot of content requests will be realized through IoT devices, such as smart homes and smart cars. For these service models, conventional cloud systems (due to the high latency) and local processing (due to battery and processing limitations) are not enough, making MEC an excellent solution.

Commonly, MEC systems are designed with one virtual machine (VM) server per user device [7]. This VM server is hosted by one of the cloudlets or the remote cloud. Each user device sends its requests (hereby called tasks) to its respective VM server, which will then utilize the resources available at its host to satisfy the tasks. Although this is a well-established model, it has a big drawback. MEC is dynamic by nature, with a volatile wireless medium as the first step of its access network and mobile users. Thus, it is often necessary to reconfigure the system to avoid idle servers or overloaded servers. However, a VM-based scheme makes workload balancing difficult as the only option for this is to migrate VM servers between cloud servers, a process that can be time consuming.

While MEC usually runs with VM servers, one major drawback of this system is a lack of information sharing among the main agents of the network. This makes it difficult to have a general view of the whole system and to make informed decisions such as where to offload user tasks and where to host VM servers. This can be done with local knowledge only but with only limited efficiency [8]. Luckily, a new framework that uses Cybertwins [9], [10] to replace VM servers has been recently proposed. This framework allows for a more coordinated sharing of information. With this in mind, in this article, we will evaluate the actual performance of a Cybertwin-based MEC and compare it to a VM-based MEC system model to determine how much the sharing of global information improves

Manuscript received January 29, 2021; revised May 27, 2021; accepted July 3, 2021. Date of publication July 7, 2021; date of current version November 5, 2021. This work was supported by the Research and Development on Intellectual ICT System for Disaster Response and Recovery, the Commissioned Research of the National Institute of Information and Communications Technology (NICT), Japan. *(Corresponding author: Tiago Koketsu Rodrigues.)*

Tiago Koketsu Rodrigues and Nei Kato are with the Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan (e-mail: tiago.gama.rodrigues@it.is.tohoku.ac.jp; kato@it.is.tohoku.ac.jp).

Jiajia Liu is with the National Engineering Laboratory for Integrated Aero-Space-Ground-Ocean Big Data Application Technology, School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, China (e-mail: liujiajia@nwpu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2021.3095308

the performance (measured here by the average service delay to finish user tasks). Results will be analyzed to provide insight for the design of future MEC systems to improve Quality of Experience (QoE) for users. Finally, the evaluation will be performed with a future networks (beyond 5G) IoT scenario in mind to keep them as close as possible to real-life performance.

The remainder of this article is organized as follows. Section II explains what is Cybertwin according to the scope of this article. Section III will present related works and how our article fits into the existing literature and addresses what is missing. Section IV has our assumed scenario for the Cybertwin-based MEC system and Section V contains a novel service delay calculation model for this scenario. Section VI details an algorithm for deciding where to deploy the Cybertwin of each user device and an algorithm for deciding where each Cybertwin should offload the tasks it receives. Section VII evaluates the performance of the Cybertwin system, comparing it to a VM-based system and benchmark methods. Finally, Section VIII concludes the article and presents possible future directions of this research.

II. Cybertwin System

Cybertwin [9], [10] is a recently designed new system for MEC. As such, the Cybertwin can be also seen as a sort of virtual server, built to be an avatar of the user in the MEC system. The Cybertwin was created as a replacement for VM servers. As such, it works between the MEC system and the users, receiving user tasks and giving them resources for their completion. This also means that under the Cybertwin system, cloud servers do not communicate with users, interfacing only with the Cybertwin instead. Because cloud servers and users do not interact directly with each other, even if the user moves and due to this mobility end up assuming a new network address (e.g., by connecting to a new access point), as long as the Cybertwin retain the same address, communication can continue with any problem. Moreover, this separation between user and server means that Cybertwin migration is easier than VM server migration, by just informing the user and the server of the new Cybertwin address in the network.

Additionally, the task of sharing and keeping information is also easier in Cybertwin-based frameworks due to the introduction of a control plane. Cybertwins are built with the idea of frequently reporting the control plane regarding the usage of the network around them, the usage of the server where they are hosted, and information regarding the tasks they receive from their users. In exchange, the control plane can aggregate all this data and use this global knowledge of the system to determine optimal configurations, such as when to migrate Cybertwins and where tasks should be executed. These decisions are then shared with the Cybertwins. This, together with the capability that Cybertwins have of offloading tasks to any cloud servers (whereas VM servers are usually limited to the cloudlet where they are hosted [7]) makes it possible to reach optimal performance in the form of the fastest execution of tasks through a balanced workload between all cloud servers, a very important metric for QoE at MEC [8], [11]. Fig. 1 shows

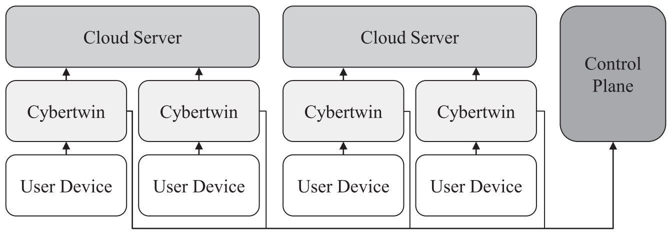

Fig. 1. Cybertwin-based MEC, which has a control plane for enabling system-wide coordination.

a simple representation of the Cybertwin-based MEC, with a focus on how the control plane aggregates information from all Cybertwins and thus, allows for enhanced coordination and distribution of workload.

Cybertwins have other important advantages, such as an activity log of important actions and the capability of negotiating the sale of information to interested companies if the user is interested in it. These points do not matter much for the study in this article, but readers interested in how Cybertwins work and these other benefits are referred to the bibliography [9], [10]. For this article, we are interested in measuring how much the control plane can improve performance in MEC. This is important as Cybertwin has some challenges when being deployed for MEC. First, there is overhead in the communication between the Cybertwin and the control plane. Additionally, VM server-based systems are already established, meaning that a transition to Cybertwin is difficult and costly due to the need of changing all cloud-based systems to the new standard. This change is only worthwhile if Cybertwins can bring a significant performance improvement. Based on this, in this article, we will measure the potential decrease in service delay brought by Cybertwin adoption.

III. Literature Review

Cybertwin is a new concept, having been proposed only recently. As such, there is no existing literature studying its application to MEC. On the other hand, there is plenty of work using VM-based MEC [2], [3], [7], [8], [11]–[19]. As mentioned before, workload balancing in such a scenario would mean migrating the VM, which is time consuming. As such, these works sometimes assume a static or quasistatic environment [2], [3], [11]–[14] so that reconfiguration and VM migration are not needed, but this is not realistic given the dynamic nature of MEC [20]. Some other works do mention migration [7], [8], [15], [16] but neglect the cost of migrating the VM server, which is not realistic either as a migration would mean stopping the service altogether, incurring high delay cost, and sacrificing QoE, i.e., a significant cost that cannot be ignored [21]. This could be avoided by having multiple copies of each VM server per user device [15], [17]–[19], deployed at different servers. However, this brings more problems as there is now a high resource cost as more VM servers are needed and you need to make sure that they are all synchronized.

The concept of a control plane and workload offloading to different servers was proposed outside of MEC before.

For example, the use of a control plane to facilitate the exchange of information, enable the application of machine learning techniques with more knowledge, and ultimately make better offloading decisions has been used quite extensively when doing network offloading in software-defined networks [22]–[24]. However, albeit the concept resembles what is done with Cybertwin in our assumed scenario, MEC has many characteristic features that make it unique, such as a tiered division of servers (with tasks being executed locally, at the edge, or at the remote cloud), the dynamic nature of the wireless medium, and the mobility of the user devices. All these points make MEC different enough that solutions used in other areas will not necessarily work well in MEC [20].

This just illustrates how important Cybertwin as a framework is for MEC, bringing higher flexibility for balancing workload while minimizing the costs of this reconfiguration, which is essential given how dynamic MEC is. Thus, in this article, we will measure its performance to provide numerical data as a comparison between Cybertwin and VM servers.

IV. Assumed Scenario

In this section, we will explain our assumed service model for a Cybertwin-based MEC, focusing on the most relevant aspects that might affect the overall performance.

A. Service Model

We will assume a scenario with multiple users, multiple cloudlets, and multiple base stations [4], [16]. Also, there is one massive remote server[1] [11], [12], [25]. There is one Cybertwin associated with each user that is hosted by one of the cloudlets. Users generate tasks and decide whether they will resolve such tasks locally or through the cloud system (depending on the state of the servers and the access network, it is possible that handling the task locally is faster). Tasks can be of one of two types: 1) content request or 2) processing. Content request tasks will require a specific content that must be retrieved by the cloud servers and sent back to the user. Meanwhile, processing tasks must be executed by the processors of the environment where the task is to be resolved. Content requests do not require any execution but they must be resolved in the cloud; these tasks cannot be satisfied purely locally. Meanwhile, processing requests can be executed locally if it is better to do so. If the task will be resolved in the cloud, then it will be first sent to a base station through a wireless channel. Then, the base station will forward the task to the cloud server that hosts the Cybertwin associated with that user through a wired medium. Once the Cybertwin receives the task, it will estimate which cloud server (one of the cloudlets or the remote server) is able of finishing the task faster. The Cybertwin then forwards the task to that server through a wired medium. The server will receive the task, resolve it by using its resources, and send the corresponding output back to the Cybertwin, which, in turn, will send it to a base station so that it can finally be sent to the user. Finally, it should be mentioned that the Cybertwin needs time to be ready when initially being set up or if it is migrated to a new host; during this time, it cannot resolve any task it receives. Fig. 2 shows our assumed system model with an emphasis on how the service delay is categorized into different intervals.

[1]Realistically, there are multiple servers located in a remote location on the network core. However, those servers are usually co-located and can share resources without much trouble, allowing us to consider them as one single massive server.

B. User Mobility

Users are capable of movement. If users move, they may move away from the cloudlet hosting they Cybertwin, meaning access to it now takes longer. The system should consider this when deciding where to host the Cybertwin, whether to migrate it, and where to send the tasks for execution. Disregarding mobility could lead to an unrealistic model and an inefficient solution due to changes in user location and how they access the network (e.g., through a new base station). We will divide time into timeslots and define movement through these timeslots thusly. Assume there are $\mathfrak{U}$ users represented by $u_0, \ldots, u_i, \ldots, u_{\mathfrak{U}}$ (i is used as an index for users). There are also $\mathfrak{T}$ timeslots represented by $t_0, \ldots, t_j, \ldots, u_{\mathfrak{T}}$ (j is used as an index for timeslots), each with a length of τ. Then, the coordinates of user u_i during timeslot t_j are $(x(u_i, t_j), y(u_i, t_j))$. If the user is moving, then this location changes for each timeslot. If the user is static, this location remains unchanged for all timeslots. It is not unrealistic to assume that such information is available to the system as most users follow determined routines with predictable paths and locations [12], [15], [16].

C. User Tasks

Each user generates tasks following a Poisson process with an average rate of λ_1 tasks per second. Each task has a ϖ chance of being a content request task and a $1 - \varpi$ chance of being a processing task. There are $\mathfrak{H}$ different contents that can be requested by the users, represented by $\sigma_0, \ldots, \sigma_l, \ldots, \sigma_{\mathfrak{H}}$ (l is used as an index for contents). $\alpha(\sigma_l)$ is the probability of content σ_l being requested, defined by a Zipf distribution [26]. With that, if we assume (without loss of generality) that the contents are sorted in descending popularity and if ϑ is the Zipf distribution exponent, the probability of content σ_l being requested comes straight from the definition of Zipf distributions

$$\delta(\sigma_l) = \frac{l^{-\vartheta}}{\sum_{q=1}^{\mathfrak{H}} q^{-\vartheta}}. \tag{1}$$

Here, and in other equations of this article, q is used as a general number for the summation. Let us say there are $\mathfrak{K}$ cloudlets represented by $k_0, \ldots, k_p, \ldots, k_{\mathfrak{K}}$ (p is used as an index for servers). Moreover, we will use k_{local} to represent the local environment and k_{remote} to represent the remote cloud. Define $a(u_i, k_p, t_j)$ as 1 if user u_i will send its processing tasks to cloudlet k_p during timeslot t_j and 0 otherwise. Additionally, $a(u_i, k_{\text{local}}, t_j)$ is used to define whether user u_i is using local execution for its processing tasks during timeslot t_j. As for content requests, they will be sent to the Cybertwin if it is set up or sent to the remote cloud if there is no Cybertwin set up. For this, we will define $\theta(u_i, k_p, t_j)$ as 1 if the Cybertwin of user u_i is hosted by cloudlet k_p during timeslot t_j and 0

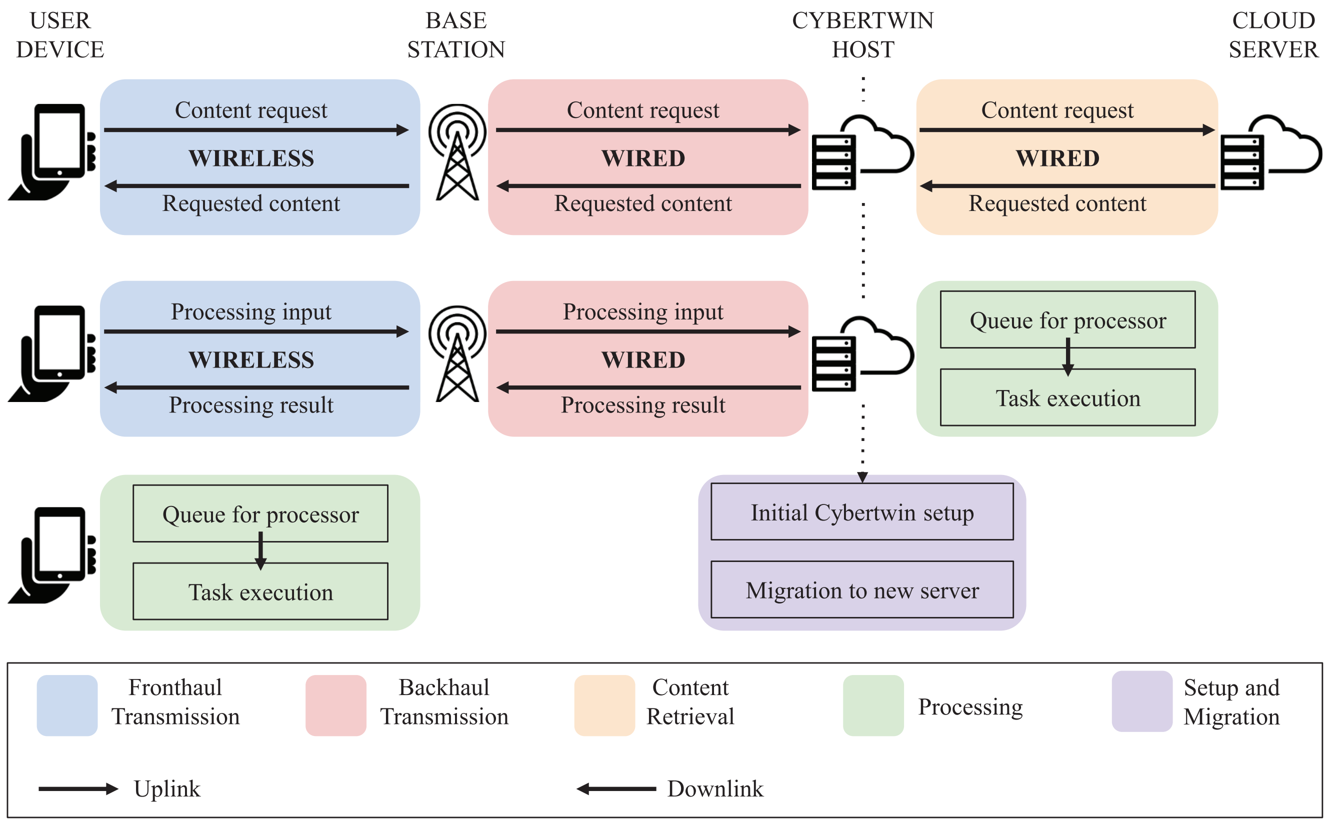

Fig. 2. Three variations of our service model, from top to bottom: content request, processing task executed in a cloud server, and processing task executed locally.

otherwise. $\theta(u_i, k_{\text{local}}, t_j) = 1$ signifies that no Cybertwin has been setup until timeslot t_j at the least. $\theta(u_i, \xi, t_j)$, $\xi = k_p, 0 \leq p \leq \mathfrak{K}$, or $\xi = k_{\text{local}}$, can be defined as the value of $a(u_i, \xi, t_0)$ in case of $t_j = t_0$. If $t_j \neq t_0$, then $\theta(u_i, \xi, t_j)$ will follow $a(u_i, \xi, t_j)$, moving the Cybertwin to a new server if the task target changes or keeping the Cybertwin in the same server as t_{j-1} if current task target is local execution. This is obtained through the equations as follows:

$$\theta(u_i, \xi, t_0) = a(u_i, \xi, t_0) \tag{2}$$

$$\theta\big(u_i, \xi, t_j\big) = a\big(u_i, k_{\text{local}}, t_j\big) \cdot \theta\big(u_i, \xi, t_{j-1}\big) + \big(1 - a\big(u_i, k_{\text{local}}, t_j\big)\big) \cdot a\big(u_i, \xi, t_j\big). \tag{3}$$

Note that if the user is switched to local execution of its processing tasks after setting up a Cybertwin, the Cybertwin is not migrated to a different cloudlet, staying in the same place instead. The Cybertwin will only be migrated if it is decided that the user will send its processing tasks to a different cloudlet.

V. Mathematical Model

In this section, we present a mathematical model for deriving the average service delay of the Cybertwin-based MEC system. The model is divided following the interval-based categories shown in Fig. 2.

A. Fronthaul Transmission Delay

Fronthaul refers to the communication made in the wireless medium, between the user and the access point. The access point here will be a base station. Let us say there are $\mathfrak{V}$ base stations presented by $v_0, \ldots, v_r, \ldots, v_{\mathfrak{V}}$. Additionally, base stations are divided into either uplink base stations (i.e., base stations that receive data from users) or downlink base stations (i.e., base stations that send data to users) [10]. Let us define $\hat{\psi}(v_r)$ as 1 if v_r is an uplink base station and 0 otherwise and let us define $\check{\psi}(v_r)$ analogously for downlink base station. Each user will associate at each timeslot to two base stations: 1) one uplink and 2) one downlink. This association will be made to the base station at each category, which offers the highest signal power [7]. Let us define $b(u_i, v_r, t_j)$ as 1 if user u_i is associated to base station v_r during timeslot t_j and 0 otherwise. Finally, if $\omega(q)$ is the transmission power of q, $G(q)$ is the antenna gain at q, h is the Rayleigh fading coefficient, and $L(\hat{q}, \check{q}, t_j)$ is the path loss between $\hat{q}$ and $\check{q}$ during timeslot t_j, we will assume the signal power between a transmitter TX and a receiver RX during timeslot t_j is calculated by adding the transmission power, the antenna gains at both ends, and then applying the fading coefficient and path loss. Additionally, this formula gives us the signal power in decibels, but we will convert that to Watts for our model, resulting in the equation as follows:

$$H\big(\text{TX}, \text{RX}, t_j\big) = \frac{10^{(\omega(\text{TX})+G(\text{TX})+G(\text{RX})+h-L(\text{TX},\text{RX},t_j))\cdot 10^{-1}}}{1000}. \tag{4}$$

Regarding path loss, a millimeter-wave communication system is expected for 6G networks [4], which means that if $d(\hat{q}, \check{q}, t_j)$ is the distance between $\hat{q}$ and $\check{q}$ during timeslot t_j, and ι_{int} and ι_{exp} are the path-loss intercept and exponent, the path loss is proven (by millimeter-wave models) to be found through the formula as follows:

$$L\big(\hat{q}, \check{q}, t_j\big) = \iota_{\text{int}} + 10 \cdot \iota_{\text{exp}} \cdot \log_{10}\big(d\big(\hat{q}, \check{q}, t_j\big)\big). \quad (5)$$

We will assume that the total amount of beams available to a base station is divided equally among the users communicating with it at any given time [4]. This includes users that use local execution for processing tasks but need to use the base station for content request tasks and users that need to send both types of tasks to the base station. With these assumptions, the total number of concurrent accesses at base station v_r during timeslot t_j is found by calculating, for all users associated with the base station, how many tasks are sent depending on whether or not local execution is chosen. This leads to the equation as follows:

$$f(v_r, t_j) = \sum_{i=0}^{\mathfrak{U}} \big(b(u_i, v_r, t_j) \cdot \lambda_1 \cdot \big(\varpi + \big(1 - a\big(u_i, k_{\text{local}}, t_j\big)\big) \cdot (1 - \varpi)\big)\big). \quad (6)$$

If N is the noise spectral density, $\hat{\Omega}$ is the bandwidth for the uplink wireless channel (and $\check{\Omega}$ is its downlink counterpart), and $\hat{\mathfrak{B}}$ is the number of beams at an uplink base station (and $\check{\mathfrak{B}}$ is its downlink counterpart), we can use Shannon–Hartley's theorem to tell us the send rate for the uplink and downlink channels for user u_i during timeslot t_j. This is done by finding the signal-to-noise ratio of the communication channel and multiplying this value by the number of beams used (we assume that the total number of beams is divided equally among concurrent transmissions). This results in the equations as follows:

$$\hat{R}\big(u_i, t_j\big) = \hat{\Omega} \cdot \sum_{r=0}^{\mathfrak{V}} \left(\hat{\psi}(v_r) \cdot b\big(u_i, v_r, t_j\big) \cdot \log_2\left(\frac{\hat{\mathfrak{B}}}{f\big(v_r, t_j\big)} \cdot \frac{H\big(u_i, v_r, t_j\big)}{N \cdot \hat{\Omega}} \right) \right) \quad (7)$$

$$\check{R}\big(u_i, t_j\big) = \check{\Omega} \cdot \sum_{r=0}^{\mathfrak{V}} \left(\check{\psi}(v_r) \cdot b\big(u_i, v_r, t_j\big) \cdot \log_2\left(\frac{\check{\mathfrak{B}}}{f\big(v_r, t_j\big)} \cdot \frac{H\big(u_i, v_r, t_j\big)}{N \cdot \check{\Omega}} \right) \right). \quad (8)$$

Regarding packet size, if $\hat{g}_{\text{pro}}$ is the uplink packet size for processing tasks, $\hat{g}_{\text{con}}$ is the uplink packet size for content request tasks, $\check{g}_{\text{pro}}$ is the downlink packet size for processing tasks, and $\check{g}_{\text{con}}(\sigma_l)$ is the size of content σ_l, then the average packet size can be found by considering the ratio between processing and content tasks (and the Zipf frequency of the contents). Therefore, for the uplink and downlink channels, the average packet sizes are found, respectively, by

$$\hat{n} = \varpi \cdot \hat{g}_{\text{con}} + (1 - \varpi) \cdot \hat{g}_{\text{pro}} \quad (9)$$

$$\check{n} = \varpi \cdot \sum_{l=0}^{\mathfrak{H}} \big(\delta(\sigma_l) \cdot \check{g}_{\text{con}}(\sigma_l)\big) + (1 - \varpi) \cdot \check{g}_{\text{pro}}. \quad (10)$$

The packet transmission time is obtained by dividing the packet size by the transmission rate. Thus, by adding wireless propagation time, if γ is the speed of light, we have, respectively, the uplink fronthaul transmission delay and downlink fronthaul transmission delay of user u_i during timeslot t_j as

$$\hat{F}\big(u_i, t_j\big) = \frac{\hat{n}}{\hat{R}\big(u_i, t_j\big)} + \sum_{r=0}^{\mathfrak{V}} \left(b\big(u_i, v_r, t_j\big) \cdot \hat{\psi}(v_r) \cdot \frac{d\big(u_i, v_r, t_j\big)}{\gamma} \right) \quad (11)$$

$$\check{F}\big(u_i, t_j\big) = \frac{\check{n}}{\check{R}\big(u_i, t_j\big)} + \sum_{r=0}^{\mathfrak{V}} \left(b\big(u_i, v_r, t_j\big) \cdot \check{\psi}(v_r) \cdot \frac{d\big(u_i, v_r, t_j\big)}{\gamma} \right). \quad (12)$$

Finally, by joining the downlink and the uplink, we end up with a formula to find the fronthaul transmission delay for tasks generated by user u_i during timeslot t_j that is

$$F\big(u_i, t_j\big) = \hat{F}\big(u_i, t_j\big) + \check{F}\big(u_i, t_j\big). \quad (13)$$

B. Backhaul Transmission Delay

Backhaul refers to the communication made in the wireless medium, between base stations and cloudlets. The wired medium usually has so much bandwidth in comparison to the packet size that propagation dominates in the delay calculation [16]. Thus, if $\Delta(v_r, k_p)$ is the wired link propagation delay between base station v_r and cloudlet k_p, we can find the uplink backhaul transmission delay and downlink backhaul transmission delay for tasks generated by user u_i during timeslot t_j as

$$\hat{B}\big(u_i, t_j\big) = \sum_{p=0}^{\mathfrak{K}} \left(\theta\big(u_i, k_p, t_j\big) \cdot \sum_{r=0}^{\mathfrak{V}} \Big(b\big(u_i, v_r, t_j\big) \cdot \hat{\psi}(v_r) \cdot \Delta\big(v_r, k_p\big) \Big) \right) \quad (14)$$

$$\check{B}\big(u_i, t_j\big) = \sum_{p=0}^{\mathfrak{K}} \left(\theta\big(u_i, k_p, t_j\big) \cdot \sum_{r=0}^{\mathfrak{V}} \Big(b\big(u_i, v_r, t_j\big) \cdot \check{\psi}(v_r) \cdot \Delta\big(v_r, k_p\big) \Big) \right). \quad (15)$$

By simply joining those equations together, we have the backhaul transmission delay for tasks generated by user u_i during timeslot t_j through

$$B\big(u_i, t_j\big) = \hat{B}\big(u_i, t_j\big) + \check{B}\big(u_i, t_j\big). \quad (16)$$

C. Processing Delay

Processing tasks can be executed either locally, at a cloudlet, or at the remote cloud [11], [16]. This means that the task must be granted access to one of the processors of the executing environment. It is commonly considered that the remote cloud has a wealth of resources and thus can allocate a processor to tasks as soon as they arrive, without any wait [15]. Consequently, the processing delay of processing tasks executed in the remote cloud is simply the processing task execution time in the remote cloud, which will be denoted as $\mu(k_{\text{remote}})$. For the local and cloudlet environments, if all processors are busy, the task then has to enter a queue and wait for one of the processors to be free. Thus, processing delay in these environments will follow an M/M/c queue model [8]. Consequently, we first must define the rate of processing task arrival at each environment. For local environments, this means the processing task generation of a single user. Therefore, the task arrival rate at the local environment of a user doing local processing during timeslot t_j is obtained from the ratio of processing tasks and the total number of tasks

$$\lambda\big(k_{\text{local}}, t_j\big) = (1 - \varpi) \cdot \lambda_1. \tag{17}$$

For cloudlet environments, this depends on how many users are sending their tasks to the cloudlet. Thus, the processing task arrival rate at cloudlet k_j during timeslot t_j is

$$\lambda\big(k_p, t_j\big) = \sum_{i=0}^{\mathfrak{U}} \big(a\big(u_i, k_p, t_j\big) \cdot (1 - \varpi) \cdot \lambda_1\big). \tag{18}$$

With this, if $\mu(\xi)$ is the processing task execution time at environment ξ and $c(\xi)$ is the number of processors at environment ξ, the occupation rate of the processors at environment ξ during timeslot t_j is given by queueing theory to be

$$\varrho\big(\xi, t_j\big) = \frac{\lambda\big(\xi, t_j\big) \cdot \mu(\xi)}{c(\xi)}. \tag{19}$$

Then, the queueing theory defines the probability that tasks have to wait when arriving at environment ξ during timeslot t_j as

$$\begin{aligned} W\big(\xi, t_j\big) = {} & \frac{\big(c(\xi) \cdot \varrho\big(\xi, t_j\big)\big)^{c(\xi)}}{c(\xi)!} \cdot \\ & \times \left(\big(1 - \varrho\big(\xi, t_j\big)\big) \cdot \sum_{q=0}^{c(\xi)-1} \frac{\big(c(\xi) \cdot \varrho\big(\xi, t_j\big)\big)^q}{q!} \right. \\ & \left. + \frac{\big(c(\xi) \cdot \varrho\big(\xi, t_j\big)\big)^{c(\xi)}}{c(\xi)!} \right)^{-1}. \end{aligned} \tag{20}$$

Thus, for tasks that are processed at environment ξ during t_j, queueing theory also says that the average processing delay is

$$\dot{P}\big(\xi, t_j\big) = \frac{W\big(\xi, t_j\big) \cdot \mu(\xi)}{\big(1 - \varrho\big(\xi, t_j\big)\big)^{c(\xi)} \cdot c(\xi)} + \mu(\xi). \tag{21}$$

Finally, the processing delay of the processing tasks generated by user u_i during timeslot t_j is

$$\begin{aligned} P\big(u_i, t_j\big) = {} & \sum_{p=0}^{\mathfrak{K}} \big(a\big(u_i, k_p, t_j\big) \cdot \dot{P}\big(k_p, t_j\big)\big) + a\big(u_i, k_{\text{local}}, t_j\big) \\ & \cdot \dot{P}\big(k_{\text{local}}, t_j\big) + a\big(u_i, k_{\text{remote}}, t_j\big) \cdot \mu(k_{\text{remote}}). \end{aligned} \tag{22}$$

D. Content Retrieval Delay

Let us define $\beta(\sigma_l, k_p)$ as 1 if cloudlet k_p has the content σ_l cached or 0 otherwise. We will also assume that the remote cloud has all the contents in its memory. Without a Cybertwin set up, the user would have to request the content to the remote cloud directly. Otherwise, the user will always request the content to its Cybertwin, which can either deliver the content directly or request it at another cloudlet or even request it at the remote cloud if no cloudlet has it. We will assume that there is coordination among the cloudlets so that no content is stored in more than one cloudlet, i.e.,

$$\sum_{p=0}^{\mathfrak{K}} \beta\big(\sigma_l, k_p\big) \le 1, \quad 0 \le l \le \mathfrak{H}. \tag{23}$$

With these assumptions, if $m(k_{\hat{p}}, k_{\check{p}})$ is the propagation delay between cloudlets $k_{\hat{p}}$ and $k_{\check{p}}$, we can define the average round-trip content retrieval delay for requests sent to a Cybertwin hosted by cloudlet k_p through the probability of each content being requested and the distance between k_p and the cloudlet that has such content (or the remote cloud if not cloudlet has it)

$$\begin{aligned} \dot{Q}\big(k_p\big) = 2 \cdot \sum_{l=0}^{\mathfrak{L}} \Bigg(\delta(\sigma_l) \cdot \Bigg(& \sum_{\dot{p}=0}^{\mathfrak{K}} \big(\beta\big(\sigma_l, k_p\big) \cdot m\big(k_p, k_{\dot{p}}\big)\big) \\ & + \left(1 - \sum_{\dot{p}=0}^{\mathfrak{K}} \beta\big(\sigma_l, k_p\big)\right) \\ & \cdot m\big(k_p, k_{\text{remote}}\big) \Bigg) \Bigg). \end{aligned} \tag{24}$$

Let us define $\Delta(v_r, k_{\text{remote}})$ as the wired link propagation delay between base station v_r and the remote cloud. The content retrieval delay of content request tasks generated by u_i during timeslot t_j is the content retrieval delay of the cloudlet that holds the Cybertwin or if no cloudlet has the Cybertwin for this user, the delay between the base station associated with this user and the remote cloud (as the content will be taken from the remote cloud directly)

$$\begin{aligned} Q\big(u_i, t_j\big) = {} & \sum_{p=0}^{\mathfrak{K}} \big(\theta\big(u_i, k_p, t_j\big) \cdot \dot{Q}\big(k_p\big)\big) + \theta\big(u_i, k_{\text{local}}, t_j\big) \\ & \cdot \sum_{r=0}^{\mathfrak{V}} \big(b\big(u_i, v_r, t_j\big) \cdot \Delta(v_r, k_{\text{remote}})\big). \end{aligned} \tag{25}$$

E. Setup and Migration Delay

When first creating the Cybertwin, there is some time needed to set up the environment in the cloudlet. During this time, any tasks that arrive will have to wait. Additionally, there may be situations where it is desirable to migrate the Cybertwin to a different cloudlet to reflect the corresponding user's movement [25]. This migration also takes time and tasks that arrive before the migration is completed will also have to wait. Let us define $\mathfrak{S}$ as the time needed to set up a Cybertwin. To define the time needed for the Cybertwin of user u_i to be ready, i.e., have concluded migration or setup, at timeslot t_0, we simply check whether the Cybertwin will be set up in this first timeslot or not. For other timeslots, we compare the location of the Cybertwin in t_j and t_{j-1}. If the locations are different, then we must perform a migration if the Cybertwin is already set up or a setup otherwise. Thus, the time needed for the Cybertwin to be ready is given by the equations as follows:

$$z(u_i, t_0) = (1 - \theta(u_i, k_{\text{local}}, t_0)) \cdot \mathfrak{S} \quad (26)$$

$$\begin{aligned} z(u_i, t_j) = {} & (1 - \theta(u_i, k_{\text{local}}, t_j)) \cdot \theta(u_i, k_{\text{local}}, t_{j-1}) \cdot \mathfrak{S} \\ & + \sum_{\hat{p}=0}^{\mathfrak{K}} \sum_{\check{p}=0}^{\mathfrak{K}} \big(\theta(u_i, k_{\hat{p}}, t_{j-1}) \cdot \theta(u_i, k_{\check{p}}, t_{j-1}) \\ & \cdot m(k_{\hat{p}}, k_{\check{p}})\big). \end{aligned} \quad (27)$$

Additionally, it is also important to find out how many tasks the user sends to the Cybertwin. This must exclude processing tasks that are executed locally at the user device and exclude all tasks of users that have not set up a Cybertwin at that timeslot. Thus, the number of tasks that user u_i sends to its Cybertwin during timeslot t_j is

$$\begin{aligned} \Lambda(u_i, t_j) = {} & \lambda_1 \cdot \big((1 - \theta(u_i, k_{\text{local}}, t_j)) \cdot \varpi \\ & + (1 - a(u_i, k_{\text{local}}, t_j)) \cdot (1 - \varpi)\big). \end{aligned} \quad (28)$$

With this, if q^+ is the maximum between q and 0, and $\lfloor q \rfloor$ is the floor of q (q being used here as a stand-in for any number), we can estimate the total number of tasks that will have to wait for the Cybertwin of u_i to be ready at timeslot t_j by considering the time needed for the Cybertwin to be ready and the time needed for the tasks to reach this Cybertwin (i.e., the uplink parts of the fronthaul and the backhaul). Only tasks that traverse the uplink before the Cybertwin is ready are counted here

$$Z(u_i, t_j) = \Big\lfloor \Big(z(u_i, t_j) - \big(\hat{F}(u_i, t_j) + \hat{B}(u_i, t_j)\big) \Big) \cdot \Lambda(u_i, t_j) \Big\rfloor^+. \quad (29)$$

Finally, because we are interested in the average delay, we must take into consideration that not all tasks will experience this setup or migration delay. To account for this, we can divide the total migration and setup delay felt by all tasks generated by a user during a timeslot by the total amount of tasks created by that user during that timeslot. We should also consider that there is an interarrival time between tasks caused by the interval being task generation at the user that decreases the waiting time for each subsequent task. Thus, the average setup/migration delay for tasks generated by user u_i during timeslot t_j is given by

$$M(u_i, t_j) = \frac{\sum_{q=0}^{Z(u_i,t_j)} \Big(z(u_i, t_j) - \big(\hat{F}(u_i, t_j) + \hat{B}(u_i, t_j)\big) - \frac{q}{\Lambda(u_i,t_j)} \Big)}{\lambda_1 \cdot \tau}. \quad (30)$$

F. Service Delay

To find the service delay, we can add all the other delays described in this section. Because we want the average service delay, it is important to take into consideration that not all tasks will experience all types of delay. For example, content request tasks will not have processing delay, and locally processed tasks will not have transmission delay. After accounting for such possibilities, the average service delay for tasks generated by user u_i during t_j can be estimated as

$$\begin{aligned} S(u_i, t_j) = {} & \big(\varpi + (1 - a(u_i, k_{\text{local}}, t_j)) \cdot (1 - \varpi)\big) \cdot F(u_i, t_j) \\ & + \frac{\Lambda(u_i, t_j)}{\lambda(u_i, t_j)} \cdot B(u_i, t_j) + \varpi \cdot Q(u_i, t_j) \\ & + (1 - \varpi) \cdot P(u_i, t_j) + M(u_i, t_j). \end{aligned} \quad (31)$$

VI. Resource Allocation

In this section, we will define the main problem of resource allocation (i.e., how to execute each user-generated task) based on the mathematical model of the previous section. Solving this problem is NP-hard. So, instead, we will divide resource allocation into three subproblems: 1) choosing where Cybertwins should be hosted; 2) choosing where contents should be cached; and 3) choosing which server Cybertwins should send incoming tasks. Finally, we will offer algorithms for each subproblem. Executing each algorithm will lead to the configuration of the MEC system.

A. Objective Problem

We will focus on average service delay as that would raise the QoE given to users while allowing for a wide range of applications in the MEC system. Thus, we will focus on minimizing the average value of $S(u_i, t_j)$. As for decision variables, we will control where the target of the processing tasks of each user at each timeslot, i.e., $a(u_i, \xi, t_j)$. This allows controlling the location of the Cybertwin of each user, where content requests are sent to, the workload of the base stations, and the workload at the cloud servers. As for constraints, we must force $a(u_i, \xi, t_j)$ to be binary, naturally, and as well as force users to only have one processing target per timeslot. Beyond that, we will assume that the other variables are either derived from $a(u_i, \xi, t_j)$ or have their values given as input. Thus, the objective function is defined as

$$\begin{aligned} \min_{a(u_i,\xi,t_j)} \quad & \frac{\sum_{i=0}^{\mathfrak{U}} \sum_{j=0}^{\mathfrak{T}} S(u_i, t_j)}{\mathfrak{U} \cdot \mathfrak{T}} \\ \text{s.t.} \quad & a(u_i, k_p, t_j) \in \{0, 1\} \\ & a(u_i, k_{\text{local}}, t_j) \in \{0, 1\} \\ & \sum_{p=0}^{\mathfrak{K}} a(u_i, k_p, t_j) + a(u_i, k_{\text{local}}, t_j) = 1. \end{aligned} \quad (32)$$

Our target problem is solving (32). That being said, the objective function above is a binary linear programming problem whose solution is nonconvex and NP-hard [11]. Thus, optimization is nontrivial.

B. Cybertwin Location Selection

Let us first assume that cloudlets have a capacity of A Cybertwins each [3], [12], [15], [16]. Thus, for each timeslot,

Algorithm 1 Cybertwin Location Selection: Decide Which Server Should Host the Cybertwin of Each User

```
1: for all $t_j$ do
2:     for all $u_i$ do
3:         Find closest $k_p$ to $u_i$ that is not at full capacity
4:         if all cloudlets are at full capacity then
5:             Set $u_i$ to local execution
6:         else
7:             Set $u_i$ to send tasks to $k_p$
8:             Update remaining capacity of $k_p$
```

Algorithm 2 Content Caching Decision: Decide Which Contents Will be Cached at the Edge and at Which Cloudlet

```
1: Set $p$ to 0
2: for all $\delta_l$ do
3:     Starting from $k_p$, check each cloudlet
4:     Cache $\delta_l$ at the first cloudlet with enough memory
5:     if no cloudlet has enough memory then
6:         Do not cache $\delta_l$
7:     $p = p + 1$
8:     If $p > \mathfrak{K}$ then $p = 0$
```

we will check users one by one and put their Cybertwin in the closest cloudlet that is not at full capacity. Here, we define the closest cloudlet as the cloudlet that with less backhaul delay to reach from the base stations that user u_i is associated to. If all cloudlets are at full capacity, the user will be set to local execution of processing tasks. This gives priority to the edge cloud as it should offer faster execution than the local environments and faster content delivery than the remote cloud. Additionally, we try to minimize the backhaul transmission delay by connecting to the closest cloudlet. Finally, this is not optimal, but it is feasible and it allows us to purely compare Cybertwins with VMs in MEC, which is our goal. Algorithm 1 describes the steps of this routine.

C. Content Caching Decision

As mentioned in the previous section, we assume that cloudlets coordinate so that no content is cached at more than one cloudlet. This is done to increase the amount of the content that can be delivered from the edge, without accessing the farther away remote cloud. To decide which cloudlet caches which content, we will iterate through the list of contents sorted on popularity and cache each content at the first cloudlet we find with enough memory available for that content. This is done while always rotating the cloudlets so that there is no situation where one cloudlet has all the most popular contents cached. By prioritizing higher popularity content, we can increase the chance of a cache hit at the edge. Algorithm 2 describes the steps of this routine.

D. Cybertwin Offloading

One of the main advantages of Cybertwin is being able to divert the tasks it receives to other cloud servers. To make

Algorithm 3 *Cybertwin Offloading:* Execute Every Time a Processing Task Is Received at the Cybertwin

```
1: for all $k_p$ and the remote cloud do
2:     Estimate backhaul delay from Cybertwin to server
3:     Estimate processing delay at server
4:     Update lowest found service delay accordingly
5: Send processing task to server with lowest service delay
```

this possible, we will assume that there is enough coordination between cloudlets and the control plane so that they have close estimates of the task workload at all cloudlets. By using this information and the equations in the previous section, the Cybertwin can estimate, for all processing tasks that arrive at it, what would the service delay be if the task was offloaded to each of the cloudlets and even the remote cloud. These estimates are then used to decide where the task should be executed, always choosing the lowest service delay. Algorithm 3 describes the steps of this routine.

VII. Results and Discussion

In this section, we will evaluate the performance of a Cybertwin-based MEC system. There will be discussions around a comparison of Cybertwin with a couple of benchmark solutions and, more importantly, a VM-based scheme.

A. Evaluation Scenario

For the parameters of our simulation, we will utilize values that are estimated in the literature to be close to what will be the norm for future networks (beyond 5G and even what is estimated for 6G) [4]. This will allow us to be close to a realistic performance in such circumstances. These parameters can be found in Table I.[2] Unless specifically stated, all results were obtained under such values. For comparison, besides the Cybertwin-based method explained in the previous section, we also evaluated the performance of the other three methods. The first method (local Only) will execute all processing tasks in the local environment at the user device while requesting content directly at the remote cloud. The second method (remote only) will send all tasks, processing or content request, to the remote cloud server for them to be resolved there. Finally, the final method (VM) will resemble the Cybertwin scheme but utilize VMs instead of Cybertwins; the main differences are that VMs always execute the processing tasks they receive at the cloud server that hosts them (i.e., no offloading to other cloud servers), and cloudlets have no cooperation so all they will host the most popular content and requests that are not on cache are passed to the remote cloud. The first two methods are here more as a benchmark comparison while the third method resembles closely what is seen in the literature [2], [3], [7], [8], [11]–[19]. Finally, all results shown here are the average of 100 different simulation runs.

[2]20% user mobility means that 20% of all users move, with a random location at each timeslot that respects the maximum user velocity. This ratio is reasonable because in IoT many devices have a fixed location and never move but it is still high enough to allow us to measure the impact of mobility on the performance of the system.

TABLE I
SCENARIO PARAMETERS

Number of timeslots	6
Timeslot length	100 s
Number of users	10000
Number of cloudlets	3
Number of uplink base stations	15
Number of downlink base stations	5
Number of contents	10000
Area size	10000 m^2
User mobility probability	20 %
Maximum user velocity	1 m/s
User task generation rate	0.1 tasks/s
Processing task execution time (local)	700 ms
Processing task execution time (cloudlet)	50 ms
Processing task execution time (remote)	10 ms
Number of processors (local)	1
Number of processors (cloudlet)	16
Cloudlet capacity	3100 users
Content size	[5,15] MB
Ratio of content request tasks	50 %
Content Zipf distribution exponent	0.81
Cloudlet memory	[10,50] GB
Number of antennas (uplink base station)	32
Number of antennas (downlink base station)	128
Uplink base station transmission power	25 dBm
Downlink base station transmission power	35 dBm
Path loss floating intercept	75.85
Path loss average exponent	3.73
Rayleigh coefficient	-1.59175
Base station antenna gain	24.5 dBi
User device antenna gain	8.35 dBi
User transmission power	27 dBm
Noise density	$4 \cdot 10^{-19}$ W/Hz
Uplink wireless bandwidth	1 MHz
Downlink wireless bandwidth	10 MHz
Processing task uplink packet size	256 KB
Processing task downlink packet size	256 KB
Content request uplink packet size	8 KB
Remote cloud access time	100 ms
Backhaul delay base station/cloudlet	[0,10] ms
Backhaul delay cloudlet/cloudlet	[0,50] ms
Cybertwin setup delay	500 ms

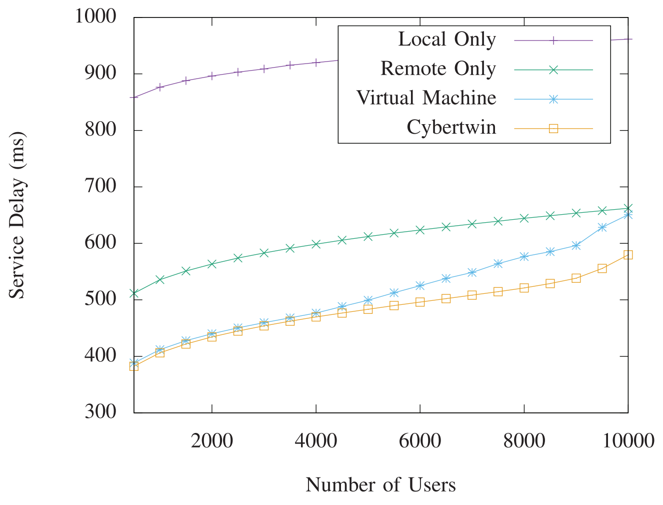

Fig. 3. Comparison of all methods while varying the number of users.

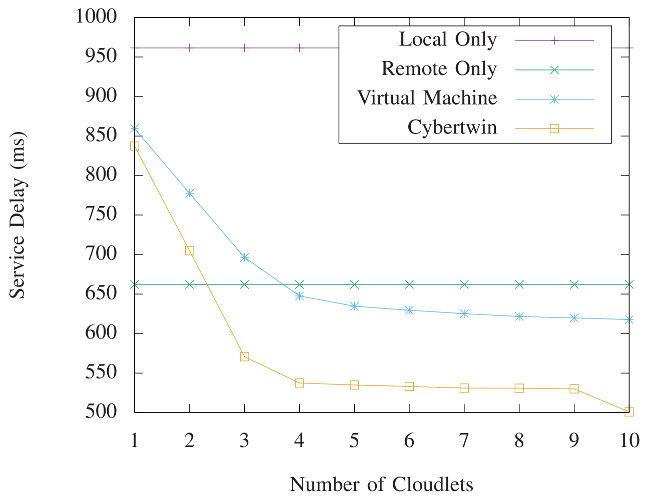

Fig. 4. Comparison of all methods while varying the number of cloudlets.

B. Results

Fig. 3 shows the service delay for all methods while the number of users increases. Because more users mean more tasks, which means more competition for resources both in processing and transmission, we see an increase in the service delay for all methods. This is true even for the local only and remote only methods where, although there is no competition for processing resources, the users are still sharing the resources of the base stations. In the case of local only, this happens when requesting content at the remote cloud. As expected, local execution by itself has bad performance and does worse than offloading to cloud servers. Moreover, the edge servers offer a better QoE than purely using the remote server. When comparing VMs to Cybertwin, we can see that in situations with few users, there is barely any difference. This is because there is so little competition for resources that there is not much room for improvement. However, as the number of users increases, the cooperation brought on by the control plane in the Cybertwin-based scheme starts doing better at workload balancing and we see a significant difference in performance in favor of Cybertwin.

Fig. 4 shows the service delay for all methods while the number of cloudlets increases. More cloudlets mean more resources, which should decrease the service delay. Of course, this is not true for the local only and remote only methods as they do not use the edge servers at all. As before, local only offers the worst performance. In scenarios with few cloudlets, the remote only method is better than the VM and Cybertwin-based schemes. This is understandable as those methods utilize the cloudlets up until the maximum capacity and then let the other users do local execution. Thus, few cloudlets mean more users doing local execution for those methods. However, with more cloudlets, those methods significantly improve upon

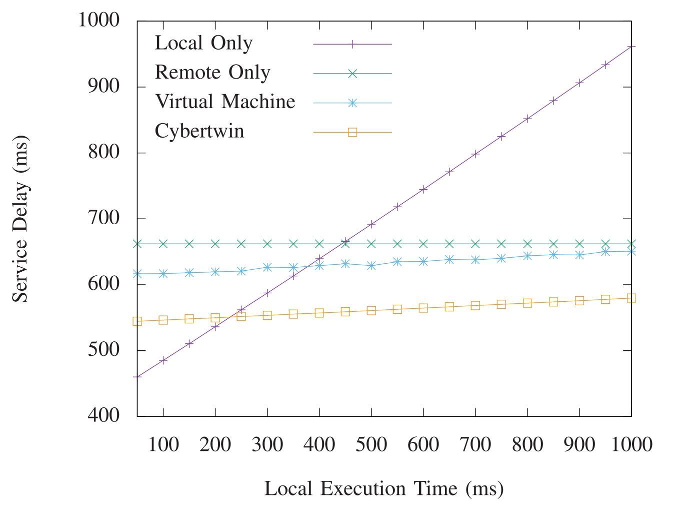

Fig. 5. Comparison of all methods while varying the local execution time.

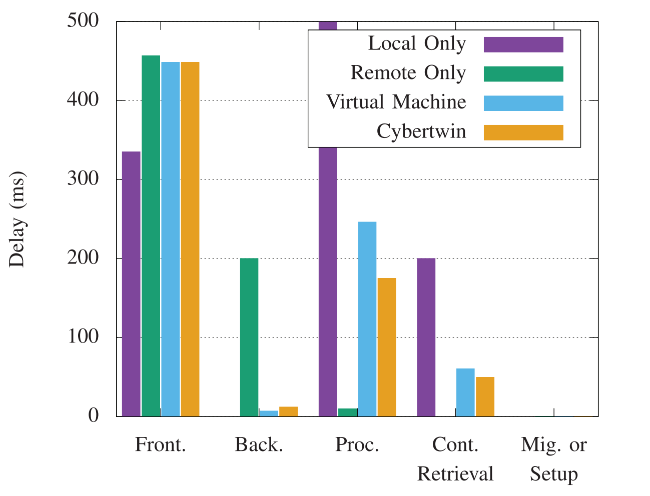

Fig. 7. Level of the individual delay categories for all methods.

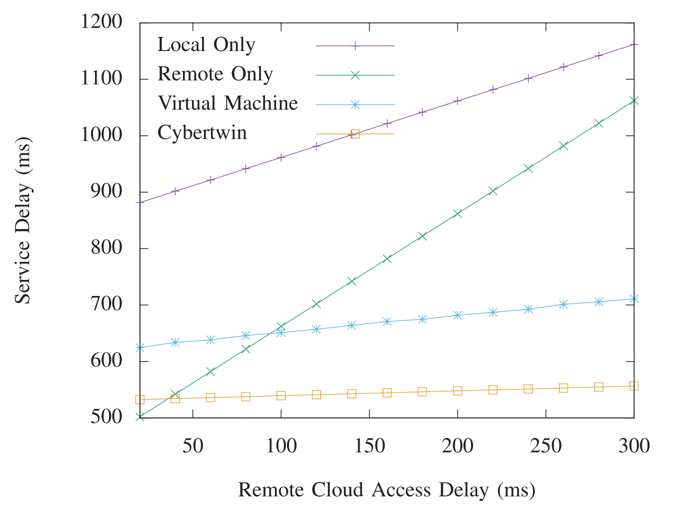

Fig. 6. Comparison of all methods while varying the remote cloud access delay.

remote only. Special note to Cybertwin, which is more efficient at using the extra resources of more cloudlets than VM, making it decrease the service delay at a steeper rate with each additional cloudlet.

Fig. 5 shows the service delay for all methods while the local execution time increases. This does not affect the remote only method because it never uses local execution. The VM and Cybertwin methods see small variation due to the fraction of users that cannot be put in the cloudlets due to capacity. Finally, as expected, the local only solution sees the bigger variation, to the point of it being better than cloud servers at fast enough local execution times. Regardless of all this, it can be seen that Cybertwin keeps a relevant gap between it and the VM-based scheme, giving extra evidence to how the control plane brings enough improvement with its additional coordination.

Fig. 6 shows the service delay for all methods while the remote cloud access delay increases. All methods eventually need to access the remote cloud to retrieve requested contents at the very least. For the local only method, this happens for all contents, while the VM and Cybertwin methods do this only for contents that do not see a cloudlet cache hit. The remote only method is the only one where all tasks (processing and content request ones) need to go to the remote cloud, thus justifying its big variation. In fact, with long enough access delay, remote only is even worse than local only. The local only method has a less steep increase in service delay but is still worse than the other methods most of the time. The VM method has a small variation in its service delay due to its use of the edge but it still behaves worse than the Cybertwin-based scheme, this time mostly due to how the control plane enables a higher rate of cache hit at the edge with Cybertwin.

Finally, to provide more insightful data, Fig. 7 shows, for all methods, what is the value of individual delays. It is important to say that the average processing and content retrieval delays shown here are calculated among only the processing tasks and the content request tasks, respectively, while the other delays are an average across all tasks. For *fronthaul transmission*, we can see that all methods have a high level of delay. This is caused by competition for the resources at the base station and the size of the requested contents, which justifies the high number even on the local only method. It is interesting to see how the difference between methods matches how much they local execution: local only uses it the most so it has the lowest fronthaul delay, remote only uses it the least and has the highest fronthaul delay, while both VM and Cybertwin use it at the same rate and have similar fronthaul delay. This is also justified as more local execution means less processing tasks using up base station resources. For *backhaul transmission*, most methods see a small delay. Local only has 0 delay here (as we consider the time spent in the backhaul by the content request tasks as content retrieval delay in cases without VM/Cybertwin). VM and Cybertwin have a small backhaul delay as cloudlets are located close enough to base stations to make this type of delay insignificant. It is interesting to see how Cybertwin is slightly worse here, justified by the extra backhaul delay when redirecting tasks between cloudlets, a feature nonexistent in the VM-based scheme. Finally, the remote only method has high backhaul delay as all its tasks must be sent to the distant remote cloud. For *processing*, local only has by far the highest delay (even going beyond the limit of the graph) due to the slow execution of the local environment. Analogously, remote only has a very good performance due to the fast execution of the resource-rich remote cloud. Both VM and Cybertwin methods not only have a slower processor as they also need queues for processors, making them worse in this regard when compared to remote only. However, it is noteworthy how Cybertwin has a significant edge here brought on by the workload balancing enabled by the control plane. For *content retrieval*, local only has the highest delay due to always requesting content at the distant remote cloud. Remote only has 0 delay due to always having cache hits (a fact that is counterbalanced by its high backhaul delay). Both VM and Cybertwin methods have a low delay but once again we see an advantage toward Cybertwin, once again explained by the extra coordination brought on by the control plane and the higher rate of cache hits it brings. Finally, for *migration/setup*, local only has 0 delay due to never setting up or migrating anything. All the other methods have non-0 values of delay but they are insignificantly low to the point where they do not show up in the graph. Thus, under our assumed parameter values, setup and migration do not have a relevant effect on the total delay. This allows us to conclude that Cybertwin is better solely for its improvements on the processing delay and content request delay areas.

VIII. Conclusion and Future Works

In this article, we studied the application of Cybertwin in place of VMs for MEC. The Cybertwin framework comes with a control plane that enables higher cooperation between the agents of the MEC system when compared to VM-based schemes. We provided a mathematical model for estimating service delay under these assumptions and algorithms for deciding Cybertwin location, content caching, and inter-cloudlet offloading in a scenario with mobile users. Finally, simulations show that Cybertwin is capable of providing a faster service when compared to VM due to more efficient resource usage and better workload balancing. More importantly, this gap in performance in favor of Cybertwin widens under scenarios with higher workload (i.e., more users), more resources (i.e., more cloudlets), and a worse network (i.e., longer remote access delay), further evidencing the big advantage brought by the newer framework.

For future works, we would like to explore the performance of Cybertwin in scenarios with partial adoption. That is, how much advantage does the control plane bring if part of the users is still using VMs. We believe this to be important due to the current prevalence of VMs, which make a full adoption of Cybertwin at the very least distant. Additionally, the control plane brings additional overhead to enable the higher cooperation for Cybertwins. We would like to measure this overhead closely and gauge the performance at different levels of Cybertwin reporting, possibly detailing the tradeoff relationship between the rate of knowledge shared by the Cybertwins and the resulting service delay of the system.

References

[1] *Multi-Access Edge Computing (MEC)*. Accessed: Nov. 29, 2020. [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing

[2] A. Samanta and Y. Li, "Latency-oblivious incentive service offloading in mobile edge computing," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 351–353.

[3] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[4] T. K. Rodrigues, K. Suto, and N. Kato, "Edge cloud server deployment with transmission power control through machine learning for 6G Internet of Things," *IEEE Trans. Emerg. Topics Comput.*, early access, Dec. 31, 2019, doi: 10.1109/TETC.2019.2963091.

[5] A. Yousafzai, I. Yaqoob, M. Imran, A. Gani, and R. M. Noor, "Process migration-based computational offloading framework for IoT-supported mobile edge/cloud computing," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4171–4182, May 2020.

[6] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based IoT," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2146–2153, Jun. 2018.

[7] T. G. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and K. Temma, "Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1287–1300, Sep. 2018.

[8] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017.

[9] Q. Yu, J. Ren, Y. Fu, Y. Li, and W. Zhang, "CybertWin: An origin of next generation network architecture," *IEEE Wireless Commun.*, vol. 26, no. 6, pp. 111–117, Nov. 2019.

[10] Q. Yu, J. Ren, H. Zhou, and W. Zhang, "A cybertwin based network architecture for 6G," in *Proc. 2nd 6G Wireless Summit (6G SUMMIT)*, May 2020, pp. 1–5.

[11] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.

[12] D. Sarddar and E. Nandi, "Optimization of edge server selection technique using local server and system manager in content delivery network," *Int. J. Grid Distrib. Comput.*, vol. 8, no. 4, pp. 83–90, Aug. 2015.

[13] J. Y. Hwang, L. Nkenyereye, N. M. Sung, J. H. Kim, and J. S. Song, "IoT service slicing and task offloading for edge computing," *IEEE Internet Things J.*, vol. 8, no. 14, pp. 11526–11547, Jul. 2021.

[14] Z. Yang, B. Liang, and W. Ji, "An intelligent end-edge-cloud architecture for visual IoT assisted healthcare systems," *IEEE Internet Things J.*, early access, Jan. 19, 2021, doi: 10.1109/JIOT.2021.3052778.

[15] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo, "Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks," *IEEE Trans. Mobile Comput.*, early access, Jul. 2, 2020, doi: 10.1109/TMC.2020.3006507.

[16] S. Ma, S. Song, J. Zhao, L. Zhai, and F. Yang, "Joint network selection and service placement based on particle swarm optimization for multi-access edge computing," *IEEE Access*, vol. 8, pp. 160871–160881, 2020.

[17] W. Zhao, J. Liu, and T. Hara, "Optimal replica distribution in edge-node-assisted cloud-P2P platforms for real-time streaming," *IEEE Trans. Veh. Technol.*, vol. 67, no. 9, pp. 8637–8646, May 2018.

[18] Y. Sun, S. Zhou, and Z. Niu, "Distributed task replication for vehicular edge computing: Performance analysis and learning-based algorithm," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1138–1151, Feb. 2021.

[19] X. Sun and N. Ansari, "Adaptive avatar handoff in the cloudlet network," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 664–676, Sep. 2019.

[20] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 38–67, 1st Quart., 2020.

[21] X. Sun and N. Ansari, "Green cloudlet network: A sustainable platform for mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 180–192, Mar. 2020.

[22] N. S. Kagami, R. I. T. da Costa Filho, and L. P. Gaspary, "CAPEST: Offloading network capacity and available bandwidth estimation to programmable data planes," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 175–189, Mar. 2020.

[23] B. Fan, Z. He, Y. Wu, J. He, Y. Chen, and L. Jiang, "Deep learning empowered traffic offloading in intelligent software defined cellular V2X networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 11, pp. 13328–13340, Nov. 2020.

[24] R. Shah, M. Vutukuru, and P. Kulkarni, "Cuttlefish: Hierarchical SDN controllers with adaptive offload," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2018, pp. 198–208.

[25] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 938–951, Mar. 2021.

[26] Q. Li, Y. Zhang, A. Pandharipande, X. Ge, and J. Zhang, "D2D-assisted caching on truncated ZIPF distribution," *IEEE Access*, vol. 7, pp. 13411–13421, 2019.

Tiago Koketsu Rodrigues (Member, IEEE) received the M.Sc. and Ph.D. degrees from Tohoku University, Sendai, Japan, in 2017 and 2020, respectively.

He is previously Tiago Gama Rodrigues, is currently an Assistant Professor with Tohoku University. His research interests include artificial intelligence, machine learning, network modeling and simulation, and cloud systems.

Mr. Rodrigues serves as an Editor for the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY and the IEEE NETWORK. From 2017 to 2020, he was the Lead System Administrator of the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, overviewing the review process of all submissions and the submission system as a whole.

Jiajia Liu (Senior Member, IEEE) received the M.Sc. degree from Xidian University, Xi'an, China, in 2009, and the Ph.D. degree from Tohoku University, Sendai, Japan, in 2012.

He is currently a Full Professor with the School of Cybersecurity, Northwestern Polytechnical University, Xi'an. He has published more than 180 peer-reviewed articles in many prestigious IEEE journals and conferences. His research interests include wireless mobile communications, FiWi, and the Internet of Things.

Prof. Liu serves as an Associate Editor for the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, an Editor for the IEEE NETWORK, and a Guest Editor for the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING and the IEEE INTERNET OF THINGS JOURNAL. He is a Distinguished Lecturer of the IEEE ComSoc.

Nei Kato (Fellow, IEEE) received the M.D. and Ph.D. degrees from Tohoku University, Sendai, Japan, in 1988 and 1991, respectively.

He is a Full Professor and the Dean with the Graduate School of Information Sciences and the Director of Research Organization of Electrical Communication from 2015 to 2019, and the Strategic Adviser to the President, Tohoku University, Sendai, Japan, in 2013. He has published more than 450 papers in prestigious peer-reviewed journals and conferences. He has researched on computer networking, wireless mobile communications, satellite communications, *ad hoc* sensor and mesh networks, UAV networks, smart grid, AI, IoT, Big Data, and pattern recognition.

Prof. Kato has been the Vice-President (Member and Global Activities) of IEEE Communications Society since 2018, and the Editor-in-Chief of IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY since 2017. He is a Clarivate Analytics Highly Cited Researcher in 2019 and 2020 and a Fellow of the Engineering Academy of Japan and IEICE.