applied sciences

MDPI

Article

Balancing Loads among MEC Servers by Task Redirection to Enhance the Resource Efficiency of MEC Systems

Jaesung Park [1] **and Yujin Lim** [2,*]

[1] School of Information Convergence, Kwangwoon University, Seoul 01897, Korea; jaesungpark@kw.ac.kr
[2] Department of IT Engineering, Sookmyung Women's University, Seoul 04310, Korea
* Correspondence: yujin91@sookmyung.ac.kr; Tel.: +82-2-2077-7305

Abstract: To improve the resource efficiency of multi-access edge computing (MEC) systems, it is important to distribute the imposed workload evenly among MEC servers (MECSs). To address this issue, we propose a task redirection method to balance loads among MECSs in a distributed manner. In conventional methods, a congested MECS selects only one MECS to which it redirects tasks. By contrast, the proposed method enables a congested MECS to distribute its tasks to a set of MECSs, the loads of which are lower than that of the congested MECS by determining the number of tasks that it redirects to each selected MECS. We prove that our task redirection method drives a MEC system to a state where the resulting MECS load vector is lexicographically minimal. Through extensive simulation studies, we show that compared with the conventional methods, the proposed method can achieve the smallest load difference between the load of the MECS, the load of which is the highest, and that of the MECS, the load of which is the smallest. By lexicographically minimizing the MECS load vector, the proposed method decreases the average task blocking rate when the task offload rate is high. In addition, we show that the proposed method outperforms the conventional methods in terms of the number of tasks, the delay requirements of which are not satisfied.

Keywords: task redirection; load balancing; lexicographically minimum; resource efficiency; distributed consensus

Citation: Park, J.; Lim, Y. Balancing Loads among MEC Servers by Task Redirection to Enhance the Resource Efficiency of MEC Systems. *Appl. Sci.* **2021**, *11*, 7589. https://doi.org/10.3390/app11167589

Academic Editor: Eui-Nam Huh

Received: 12 July 2021
Accepted: 17 August 2021
Published: 18 August 2021

1. Introduction

In a multi-access edge computing (MEC) system, a device offloads tasks to nearby MEC servers (MECSs). By serving offloaded tasks at the edge of a network, a MEC system can facilitate delay-sensitive and computing-intensive applications in devices that are limited with respect to energy, storage, and computing power [1,2]. However, devices are not distributed uniformly in MEC systems, and each device may have a different task offload rate. In addition, the service capacities of MECSs differ, and there is no central entity controlling the mapping between offloaded tasks and MECSs. Therefore, the number of tasks offloaded to a MEC system is likely not evenly distributed among MECSs, which results in the situation where some MECSs are heavily congested while other MECSs are lightly loaded. When tasks are offloaded to a congested MECS, it is highly probable that they will be blocked or their delay requirements will be violated. Furthermore, the capacity of a MEC system, in terms of the number of acceptable tasks, will be reduced because lightly loaded MECSs cannot serve the tasks blocked by congested MECSs. Therefore, to enhance the resource efficiency of MEC systems, tasks from overloaded MECSs must be redistributed to underloaded MECSs so that the workload imposed on the MEC system is evenly distributed among MECSs.

Various load balancing methods that allocate a task to the least-loaded MECS have been proposed. In [3], whenever a task is offloaded, a central controller assigns it to the least-loaded MECS sequentially. We note that a role transfer solution can be used to design a centralized method for solving the load balancing problem [4–6]. However, as the size of the MEC system, in terms of the number of offloaded tasks and MECSs, increases, it

becomes challenging for a central controller to assign tasks to the least-loaded MECS every time a task is offloaded. Distributed approaches were proposed in [7,8]. The method proposed in [7] transfers tasks from highly loaded MECSs to the least-loaded MECS. Similarly, to transfer the tasks offloaded from devices to a MECS, the authors in [8] forced an overloaded MECS to select two MECSs randomly from a set of neighboring MECSs and choose the least-loaded one for offloading. In the case of distributed methods, since all the overloaded MECSs redistribute their tasks to the least-loaded MECS simultaneously, the least-loaded MECS can easily become heavily loaded, which results in a further load unbalance among MECSs. In addition, most approaches do not explicitly consider the delay requirement of a task, nor the amount loads to be redirected to the least-loaded MECS. Load balancing methods based on machine learning were proposed in [9–11]. After predicting the state of the MEC system in terms of the MECSs loads, the authors proposed methods to balance loads among MECSs. However, the signaling cost of these methods is very high because a central controller must collect an enormous amount of data to analyze the state of the MEC system and transfer the learned model parameters to each MECS. In addition, a relatively long time is required for a controller to learn by analyzing big data, so these methods are not easily applied to short timescales. In [12], game theory was used to resolve the load balancing problem. The cost minimization problem was formulated as a transportation problem, and Vogel's approximation method was used to calculate the optimal solution. However, global information is needed to solve the optimization problem, and the static threshold required to determine the load state of a MECS was not systematically configured.

To address these issues, we propose a task redirection method to balance loads among MECSs using a decentralized consensus method [13,14]. We explicitly consider the delay requirement of a task when estimating the MECS load. In our task redirection method, each MECS determines whether to redirect tasks by considering its load state relative to that of the other MECSs. Once MECS i decides to redirect its tasks; instead of transferring tasks to the least-loaded MECS, i redistributes its tasks to a set of MECSs whose loads are smaller than its own load. In addition, MECS i determines the number of tasks to be redirected to each MECS in the set according to the difference between the load of each MECS in the set and its own load.

This paper is organized as follows. In Section 2, we introduce a system model and define the load balancing problem in a MEC system. In Section 3, we present our task redirection method and discuss its properties. In Section 4, we validate the proposed method by comparing its performance with those of the conventional methods. We conclude the paper and provide future works in Section 5. Before we proceed, in Table 1, we present the notations used in this paper for the readers' convenience.

Table 1. Notations used.

Notations	Meaning
N	A set of MECSs in the system.
C_i	CPU frequency of MECS i.
$O_i(t)$	A set of tasks in the waiting queue of MECS i at the end of time slot t.
$S_i(t)$	A set of tasks in the service queue of MECS i at the end of time slot t.
d_x	Data size of a task x.
w_x	Workload imposed by a task x in terms of the number of CPU cycles.
T_x	Maximum delay allowed to finish a task x.
$r_{i,x}(t)$	Uplink transmission rate to send a task x to MECS i during a time slot t.
$\rho_{i,w}(t)$	Load of MECS i imposed by the tasks in $O_i(t)$.
$\rho_{i,s}(t)$	Load of MECS i imposed by the tasks in $S_i(t)$.
$\rho_i(t)$	Load of MECS i at the end of time slot t.

Table 1. *Cont.*

Notations	Meaning
$\bar{\rho_i}(t)$	Avg. load of a MEC system at the end of time slot t (i.e., $\frac{1}{\|N\|}\sum_{j\in N}\rho_j(t)$).
$\Delta_i(t)$	A set of MECSs whose loads are lower than $\rho_i(t)$.
$\delta_{i,j}(t)$	Load difference between MECS i and j (specifically, $\frac{\rho_i(t)-\rho_j(t)}{\|N\|}$, $j \in \Delta_i(t)$).
$\alpha_{i,j}(t)$	The amount of workload that MECS i redirects to j.
$\Phi_{i,j}(t)$	A set of tasks that MECS i redirects to MECS j.

2. System Model and Problem Formulation

We considered a MEC system composed of a set N of MEC servers. We denote the capacity of MECS i in terms of the number of CPU cycles per second as C_i. We assumed that a MECS is installed in a base station (BS) and ignored the information transfer delay between the MECS and BS. Following [9], we assumed that each device offloads its tasks to the MECS installed in the BS, giving it the highest signal strength. Time is divided into slots whose length is assumed to be the frame time between a device and a BS.

A MECS maintains two queues: a waiting queue and a service queue. The set of tasks in the waiting queue of MECS i at the end of time slot t is denoted as $O_i(t)$, and the set of tasks in the service queue of MECS i at the end of the time slot t is denoted as $S_i(t)$. The waiting queue is used to temporarily buffer tasks offloaded from devices to a MECS during a time slot. At the end of a time slot, a MECS makes a task redirection decision to balance the loads among MECSs. Depending on the decision made by a MECS, the tasks in its waiting queue are either moved to its service queue or redirected to another MECS. A MECS uses its service queue to accommodate tasks until they are served in a FIFO manner. Thus, the tasks in the service queue of a MECS can be classified into two groups: a group composed of tasks moved from its waiting queue and a group composed of tasks redirected from other MECSs. Once a task is located in a service queue, it cannot be redirected to other MECSs to avoid unnecessary increases in the delay involved in transferring a task from one MECS to another.

During each time slot, MECS i receives tasks offloaded from devices and places them in its waiting queue. A task x is composed of three tuples (d_x, w_x, T_x), where d_x is the data size of the task, w_x is the workload of the task in terms of the number of CPU cycles required to process the task, and T_x is the maximum delay allowed to finish the task. We denote the uplink transmission rate to send a task x from a device to its serving MECS during a time slot t as $r_{i,x}(t)$. If we assume $r_{i,x}(t)$ does not change during a time slot, even though it can change across time slots to satisfy the delay requirement of task x, MECS i has to complete the task within:

$$b_{i,x} = T_x - \frac{d_x}{r_{i,x}(t)}. \tag{1}$$

Since the MECS is installed in a BS, it can be synchronized in time with the associated devices. Thus, $\frac{d_x}{r_{i,x}}$ can be obtained by subtracting the time when a device sends task x from the time when the MECS receives the task. Therefore, the CPU frequency (i.e., the number of CPU cycles per second) required to finish tasks x becomes:

$$f_{i,x} = \frac{w_x}{b_{i,x}}. \tag{2}$$

Thus, at the end of time slot t, the load of MECS i imposed by the tasks in $O_i(t)$ is:

$$\rho_{i,w}(t) = \frac{\sum_{x\in O_i(t)} f_{i,x}}{C_i}. \tag{3}$$

If a task x resides in the service queue of MECS i for $d_{i,x}$, MECS i has to finish x in $c_{i,x} = b_{i,x} - d_{i,x}$ to meet the deadline requested by x. MECS i removes tasks whose $c_{i,x} \leq 0$ from $S_i(t)$ because the delay requirement of the task is not satisfied. If we denote the set of tasks in $S_i(t)$ whose $c_{i,x} > 0$ as $S'_i(t)$, the load imposed by a task $x \in S'_i(t)$ in terms of the number of CPU cycles per second is given as:

$$f'_{i,x} = \frac{w_x}{c_{i,x}}. \tag{4}$$

Therefore, the load imposed by the tasks in $S_i(t)$ is obtained as follows:

$$\rho_{i,s}(t) = \frac{\sum_{x \in S'_i(t)} f'_{i,x}}{C_i}. \tag{5}$$

The first step to balance the loads among MECSs is to determine whether a MECS will have a higher load than the other MECSs. At the end of each time slot t, the set of tasks received by MECS i is $O_i(t) \cup S_i(t)$. Thus, if $\rho_{i,w}(t) + \rho_{i,s}(t)$ is larger than $\rho_{j,w}(t) + \rho_{j,s}(t)$, MECS i will have a higher load than MECS j if no other action is taken. Therefore, using Equations (3) and (5), we define the load of MECS i at the end of time slot t as:

$$\rho_i(t) = \rho_{i,w}(t) + \rho_{i,s}(t). \tag{6}$$

We can observe that $\rho_i(t)$ depends on many factors such as the attribute of a task (d_x, w_x, T_x), the uplink transmission rate $(r_{i,x}(t))$, the computing power of a MECS (C_i), and the number of tasks in a MECS $(|O_i(t)|, |S_i(t)|)$. The attribute of a task depends on the application services. Thus, $\rho_i(t) \neq \rho_j(t)$ even when all the other variables affecting $\rho_i(t)$ and $\rho_j(t)$ are the same. As we can see in Equation (1), $r_{i,x}(t)$ influences the heterogeneity of the task attribute by changing the delay requirement of a task when it arrives at a MECS. The number of tasks in a MECS i depends on the task input rate and the task service rate. The task input rate to a MECS i during a time slot t, which we denote by $a_i(t)$, is mainly affected by the number of devices offloading their tasks to a MECS i. Generally, $a_i(t) \neq a_j(t), (i, j \in N)$ because devices are not evenly distributed over the region that a MEC system serves. In addition, the association method that a device selects a MECS to which the device offloads its tasks also influences $a_i(t)$. Usually, it is difficult for a device to know the load situation of each MECS. Thus, following [9], we assumed that a device offloads tasks to the MECS colocated with the BS, which gives it the highest signal strength. By selecting the BS giving the highest signal strength, a task may obtain high $r_{i,x}(t)$. However, it was shown in [15,16] that the number of devices associated with each BS is not evenly distributed when a device determines its association according to the signal strength from a BS, which results in $a_i(t) \neq a_j(t), (i, j \in N)$. The task service rate of a MECS is determined by C_i and the workload imposed by a task (w_x), which are not the same for all tasks.

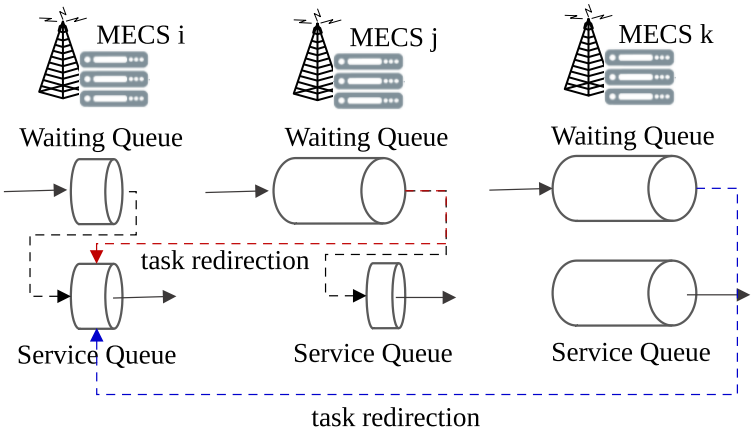
Therefore, $\rho_i(t) \neq \rho_j(t), (i, j \in N)$ in general, which means the $\rho_i(t)$s of some MECSs are high, while those of the other MECSs are low, as shown in Figure 1a. If a task is offloaded to a MECS whose load is high, it is highly probable that the delay requirement of the task is violated. However, we can avoid the situation if we redirect the tasks from a highly loaded MECS to a lightly loaded MECS, as we depict in Figure 1. Thus, our goal is to balance loads among MECSs by redirecting tasks so that:

$$\rho_i = \rho_j = \bar{\rho} = \frac{1}{|N|} \sum_{k \in N} \rho_k, \forall i, j \in N, \tag{7}$$

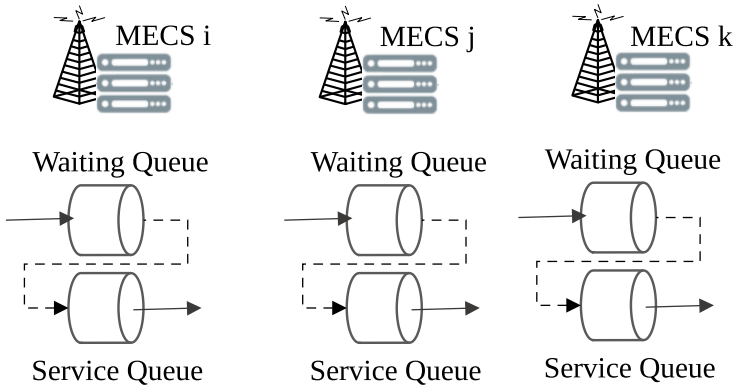
where $|N|$ is the cardinality of set N.

To balance the loads between a highly loaded MECS i and a lightly loaded MECS j, MECS i must determine the amount of workload to redirect to MECS j. Generally, to make such a decision, MECS i needs to know the local information of MECS j such as $O_j(t)$ and

$S_j(t)$. However, by including $\rho_{i,w}(t)$ in $\rho_i(t)$, we enable MECS i to determine the amount of workload to redirect to MECS j without the local information of MECS j. We detail our task redirection method in Section 3.



(**a**) Load balancing problem and task redirection



(**b**) Load balancing after task redirection

Figure 1. Load balancing problem and task redirection approach.

3. Task Redirection Method

In this section, we first present our task redirection algorithm that drives a MEC system to the state where loads among MECSs are balanced; then, we discuss the properties of the algorithm.

3.1. Task Redirection Algorithm

To balance the loads among MECSs in a distributed manner, each MECS i must be able to decide autonomously whether to redirect tasks in $O_i(t)$. If MECS i decides to redirect tasks, i must select MECS j to which it transfers tasks. In addition, MECS i has to determine the amount of workload to redirect to the selected target MECS j. To make such decisions, each MECS i exchanges $\rho_i(t)$ with other MECSs at the end of a time slot and calculates the average load.

$$\bar{\rho}(t) = \frac{1}{|N|} \sum_{j \in N} \rho_j(t). \tag{8}$$

If $\rho_i(t) \leq \bar{\rho}(t)$, i considers that it is relatively underloaded compared with the other MECSs. Thus, i does not redirect a task in $O_i(t)$ and moves all the tasks in $O_i(t)$ to $S_i(t)$. By contrast, if $\rho_i(t) > \bar{\rho}(t)$, the load of MECS i is higher than that of some other MECSs; therefore, i decides to redirect tasks in $O_i(t)$ to MECS j. We adopt the distributed consensus method in [14] for MECS i to determine not only a target MECS, but also the amount

of workload that i redistributes to the selected target MECS. The procedure is shown in Algorithm 1. MECS i collects a set $\Delta_i(t)$ of MECSs whose loads are lower than $\rho_i(t)$. For each MECS $j \in \Delta_i(t)$, a MECS i calculates the load difference:

$$\delta_{i,j}(t) = \frac{\rho_i(t) - \rho_j(t)}{|N|}, j \in \Delta_i(t). \tag{9}$$

Then, MECS i searches for the MECS $j^* \in \Delta_i(t)$ that gives the highest $\delta_{i,j}(t)$. Since the capacity of MECS i is C_i, the amount of workload corresponding to $\delta_{i,j^*}(t)$ becomes $\theta_{i,j^*}(t) = \delta_{i,j^*}(t)C_i$. Since the total workload imposed by the tasks in $O_i(t)$ is $o_i(t) = \sum_{x \in O_i(t)} f_{i,x}$, the amount of workload that i redirects to j^* is $\alpha_{i,j^*}(t) = \min(o_i(t), \theta_{i,j^*}(t))$. MECS i constructs a set $\Phi_{i,j^*}(t)$ of tasks to be transferred to j^* by randomly selecting tasks from $O_i(t)$. Specifically, MECS i randomly selects tasks from $O_i(t)$ as long as $\sum_{x \in \Phi_{i,j^*}(t)} f_{i,x} < \alpha_{i,j^*}(t)$. Then, i redirects all the tasks in $\Phi_{i,j^*}(t)$ to j^*.

After removing j^* from $\Delta_i(t)$, MECS i repeats the procedure until $\Delta_i(t)$ is empty or all the tasks in $O_i(t)$ are redirected, whichever comes first. If $\Delta_i(t)$ becomes empty before all the tasks in $O_i(t)$ are redirected, MECS i moves the remaining tasks to its service queue.

Algorithm 1 Task redirection algorithm.

$T_i(t) = \Delta_i(t)$
if $T_i(t) \neq \emptyset$ **then**
    find $j^* = \arg\max_{j \in T_i(t)} \delta_{i,j}(t)$
    $\theta_{i,j^*}(t) = \delta_{i,j^*}(t)C_i$
    $\alpha_{i,j^*}(t) = \min(o_i(t), \theta_{i,j^*}(k))$
    Construct $\Phi_{i,j^*}(t)$
    redistribute all the tasks in $\Phi_{i,j^*}(t)$ to j^*
    $o_i(t) = o_i(t) - \sum_{x \in \Phi_{i,j^*}(t)} f_{i,x}$
    if $o_i(t) \leq 0$ **then**
        break
    else
        $T_i(t) = T_i(t) - \{j^*\}$
else
    move remaining tasks in $O_i(t)$ to its service queue.

3.2. Properties of the Task Redirection Method

In [17], the load of each MECS was shown to converge to $\bar{\rho} = \sum_{i \in N} \rho_i / |N|$ in polynomial time if each MECS repeats Algorithm 1.

To state how the proposed method distributes tasks to MECSs, we first introduce the following definitions.

Definition 1. *A vector $\vec{a} \in X \subseteq R^n$ is said to be a min-max fair vector on X if and only if we cannot decrease a component in $a_i \in \vec{a}$ without increasing another component $a_j \geq a_i$. Formally, for all $\vec{b} \in X$, if there exists $i = \{1, \ldots, n\}$ such that $b_i \in \vec{b} < a_i \in \vec{a}$, then there exists $j = \{1, \ldots, n\}$ such that $b_j > a_j \geq a_i$.*

Definition 2. *Given a set of vectors $A \subset \mathbf{R}^n$, a vector $\vec{a} \in A$ is said to be leximax minimal if $< \vec{a} >$ is lexicographically less than or equal to $< \vec{b} >$ for any vector $\vec{b} \in A$, where $< \vec{a} >$ represents the vector obtained from vector $\vec{a}$ by rearranging its elements in nonincreasing order.*

We say that a MECS load vector $\vec{\rho} = (\rho_1, \ldots, \rho_n)$ is feasible if $\rho_i < 1$ for all $i \in N$ and denote the set of feasible $\vec{\rho}$s as $\Psi \subseteq R^n$. Then, we state the fairness of our task redirection method as Proposition 1.

Proposition 1. *The MECS load vector* $\vec{\rho} = (\rho_1, \ldots, \rho_n) = (\bar{\rho}, \ldots, \bar{\rho}) \in \Psi$ *obtained by our task redirection method is min-max fair.*

Proof. Let us suppose that there is a min-max fair load vector $\vec{a} = (a_1, \ldots, a_n) \in \Psi$ such that $a_i \neq a_j$ for some $i, j (i \neq j)$. Without loss of generality, we assumed that a_i is the smallest and a_j is the second smallest element in $\vec{a}$. If we choose ϵ such that $0 < \epsilon < a_j - a_i$, we have $a_i + \epsilon < a_j$. Let us consider another load vector $\vec{b} = (b_1, \ldots, b_n) \in \Psi$. If the loads of some MECSs decrease, the decreased load has to be accommodated by another MECSs. Specifically, for $0 < \delta_i < \epsilon$ and $0 \leq \delta_k < \epsilon$, the decrease in the load of MECS i from a_i to $a_i - \delta_i$ induces variation in the loads of other MECSs from a_k to $a_k + \delta_k$ and $\sum_{k \neq i} \delta_k = \delta_i$. When $b_i = a_i - \delta_i$ and $b_j = a_j + \delta_j$, $b_i < a_i$ and $b_j = a_j + \delta_j < a_i$, which contradicts that $\vec{a}$ is a min-max fair vector. □

Proposition 2. *The min-max fair MECS load vector* $\vec{\rho} = (\rho_1, \ldots, \rho_n) = (\bar{\rho}, \ldots, \bar{\rho})$ *is a unique leximax minimal load vector.*

Proof. It was shown in [18,19] that if a max-min fair vector exists on a set $X \subseteq R^n$, then it is the unique lexicographically maximal vector on X. Since $\vec{\rho}$ is min-max fair on Ψ, $-\vec{\rho}$ is a max-min fair load vector on $-\Psi$, which makes $-\vec{\rho}$ the unique lexicographically maximal vector on $-\Psi$. Therefore, $\vec{\rho}$ is the unique leximax minimal vector on Ψ. □

According to Proposition 2, by redirecting tasks among MECSs in a distributed manner, our task redirection method makes $\vec{\rho}$ lexicographically smallest given a set of tasks accommodated in a MEC system.

4. Performance Evaluation

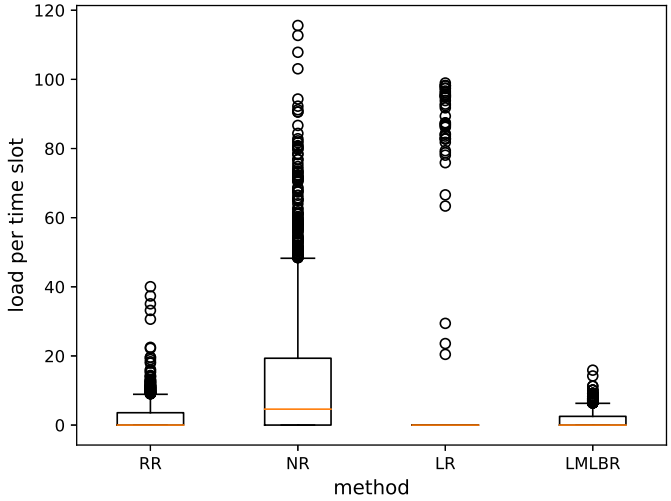
In this section, we verify the proposed method by comparing its performance with that of conventional schemes under the same environment. Henceforth, we call the proposed method, leximax minimal load balancing redirection (LMLBR). We selected the following three representative schemes for performance comparison: (1) **Random redirection (RR).** In RR, the overloaded MECS redirects its tasks to a randomly selected MECS; (2) **Nearest redirection (NR).** In NR, the overloaded MECS redirects its tasks to the MECS that is geographically closest; (3) **Least-loaded redirection (LR).** In LR, the overloaded MECS redirects its tasks to the least-loaded MECS.

We distributed 50 MECSs in an area of 1000 × 1000 m. According to [20–22], we set the simulation parameters as shown in Table 2. The capacity of each MECS (C_i) was randomly selected in [9 GHz, 11 GHz], according to a uniform distribution. We set the task arrival process at a MECS to follow a Poisson point process (PPP) with rate λ. When task x is generated, its d_x, w_x, and T_x are randomly chosen from the given ranges in Table 2 according to a uniform distribution. For example, the data size of a task is randomly selected in [3 kbits, 6 kbits]. We set the length of a time slot to 100 ms, which is the frame time between a device and a BS in an LTE system. We set the size of waiting queue and service queue of a MECS so that the number of offloaded tasks does not exceed the queue capacity of each MECS, when the number of devices associated with each MECS is evenly distributed and the number of offloaded tasks for each time slot is distributed uniformly.

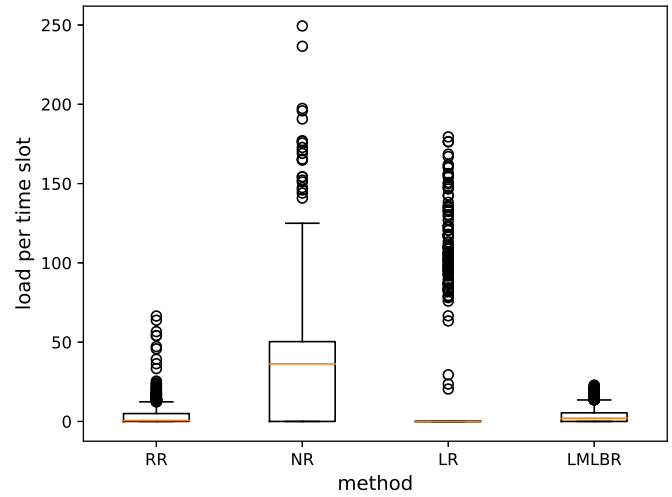
We first investigate the load variance of a MECS in Figure 2. The figure presents a box plot for the load changes of the 32nd MECS during the simulation time when the task offload rate is 0.5, 0.7, and 0.9. In NR and LR, all the overloaded MECSs select the best MECS in terms of the distance (NR) or the load (LR) as their target MECS and redirect their tasks to the best MECS. Therefore, the load of the best MECS increases sharply after it accepts the tasks redirected to it within a time slot. Therefore, the load of a MECS varies substantially depending on whether the MECS is chosen as the best MECS. However, in the proposed method, the variance is relatively small because tasks are redirected from a congested MECS to more than one target MECS according to the load difference between the congested MECS and the target MECS.

Table 2. Simulation parameters.

System Parameters	Values
Subcarrier bandwidth	15 kHz
Background noise	10^{-13} W
Data size of task x (d_x)	3000–6000 bits
Workload of task x (w_x)	0.1–1.0 GHz
Maximum delay allowed for task x (T_x)	0.1–0.2 s
CPU frequency of MECS i (C_i)	9–11 GHz
Waiting queue size of a MECS	5 tasks
Service queue size of a MECS	5 tasks

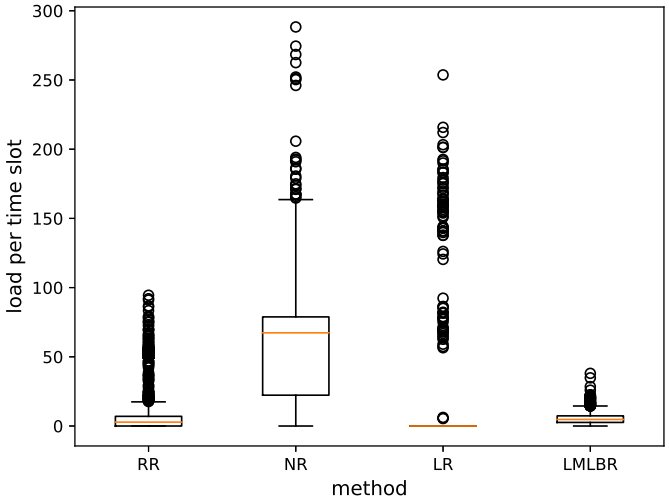


(**a**) task offload rate, λ = 0.5



(**b**) task offload rate, λ = 0.7

Figure 2. *Cont.*



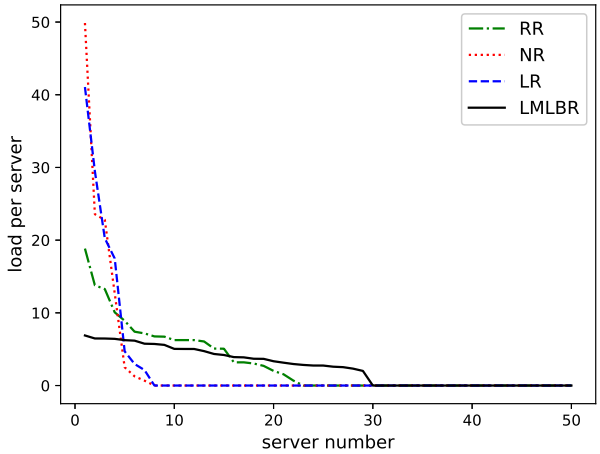
(**c**) task offload rate, $\lambda = 0.9$

Figure 2. Load per time slot of four methods.

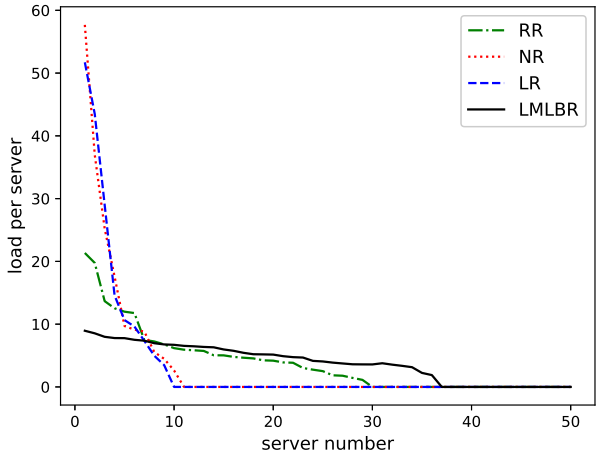
To scrutinize the load variance among MECSs, in Figure 3, we present the loads of all 50 MECSs at a given time slot when the task offload rate is 0.5, 0.7, and 0.9. LMLBR minimizes the MECS load vector lexicographically. In NR and LR, since the redirected tasks are absorbed into the best MECS at once, the loads of some MECSs are very high, while the loads of the other MECSs are low. However, in LMLBR, tasks are assigned to multiple MECSs, which makes the load difference among MECSs the smallest.

To compare the four methods in terms of the quality of service provided by the MEC system, we inspected the average task blocking rate and the rate of tasks completed within their delay constraints. When a task is offloaded to a MECS whose queue is full, the task is blocked by the MECS. The average task blocking rate is defined as the fraction of tasks that are blocked by the MECSs in the system. In Figure 4, the average task blocking rate is depicted with varying λ. In NR and LR, some MECSs are highly loaded; thus, if a task is offloaded to a highly loaded MECS, the task is likely to be blocked. Since the two lines representing RR and LMLBR look similar in Figure 4, we show and scrutinize the average blocking rate obtained by these two methods for $\lambda = 0.5, 0.7, 0.9$, separately, in Table 3. In our method, an overloaded MECS i considers the loads of other MECSs when it selects a MECS j to which it redirects its tasks. In addition, our method determines the amount of tasks to redirect to each j according to the load difference between i and j. On the contrary, in RR, an overloaded MECS redirects its tasks to one randomly selected MECS until it becomes underloaded, regardless of the load of the selected MECS. Therefore, two adverse cases can happen when RR is used. Firstly, an overloaded MECS may redirect its tasks to another overloaded MECS. Secondly, an overloaded MECS may redirect an excessive amount of tasks to the selected MECS, which results in overloading the selected MECS. Thirdly, multiple overloaded MECSs may select the same MECS i to which they redirect their tasks. In this case, it is very likely that the selected MECS i becomes overloaded even though it was underloaded before accommodating the redirected tasks. If the load imposed on a MEC system is not so high, the number of MECSs overloaded by the tasks offloaded from devices is small. Accordingly, the adverse cases occur rarely. Therefore, we can observe in Table 3 that the average blocking rate obtained by RR is smaller than that achieved by LMLBR when $\lambda \leq 0.7$. However, as the traffic offload rate increases, the two adverse cases by RR occur more frequently, which increases the average task blocking rate. Since our method avoids the two adverse cases, it can achieve a lower task blocking rate than RR when $\lambda = 0.9$. To further compare RR and LMLBR in terms of the blocking rate, we show not only the average blocking rate, but also the standard deviation of the blocking rate obtained by RR and our method in Figure 5. When we investigated the standard deviation of the blocking rate (denoted by σ_b) in Figure 5, our method achieved smaller

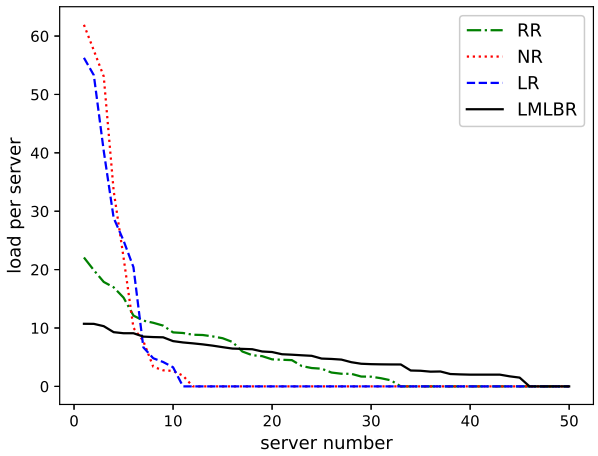
σ_b than RR for all $\lambda > 0.1$. This is attributed to the behavior of the proposed method. Unlike RR, our method redirects tasks from highly loaded MECSs to lightly loaded MECSs by considering the load difference between MECSs. Therefore, compared with RR, the proposed method achieves smaller load difference between the load of the MECS whose load is the highest and that of the MECS whose load is the smallest.



(**a**) task offload rate, λ = 0.5



(**b**) task offload rate, λ = 0.7



(**c**) task offload rate, λ = 0.9

Figure 3. Load per server for a given time slot (i.e., $\rho_i(t), \forall i \in N$).

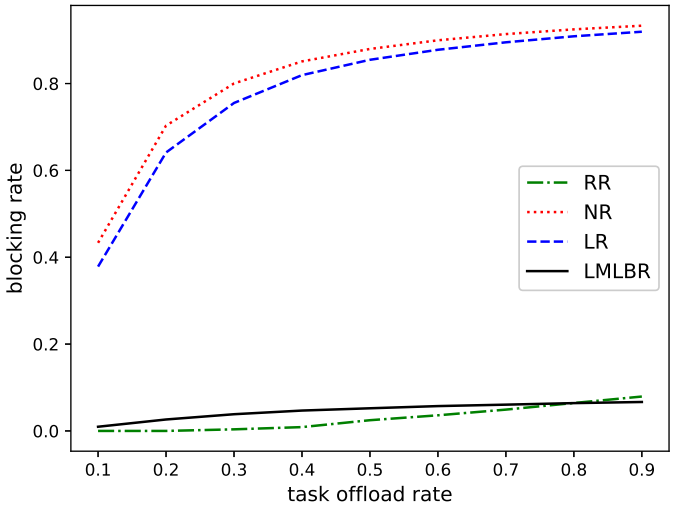


Figure 4. Average task blocking rate for various task offload rates.

Table 3. Comparison of RR and LMLBR in terms of the average task blocking rate (Difference (δ_{br}) represents the average blocking rate obtained by LMLBR minus the average blocking rate obtained by RR).

λ	**0.5**	**0.7**	**0.9**
RR	0.0247	0.0491	0.0792
LMLBR	0.0521	0.0605	0.0666
Difference (δ_{br})	0.0274	0.0114	−0.0126

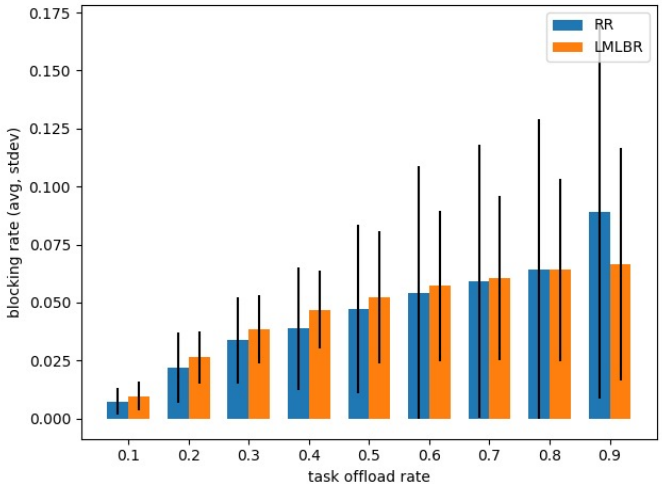


Figure 5. Comparison of RR and LMLBR in terms of the average task blocking rate. Each bar represents the average blocking rate at a given task offload rate. If we denote the average blocking rate as b and the standard deviation of the blocking rate as σ_b, each line in each bar indicates the range of blocking rates from $b - \sigma_b$ to $b + \sigma_b$.

The rate of tasks completed within their delay constraints is affected by the total delay from the beginning of the task offloading process to the end of task computing. The total delay is composed of three parts. The first component is the transmission delay between a device and the MECS to which the device offloads a task. The second part is the queuing and service delay experienced by a task in the MECS that serves the task. The third component is a redirection delay that is included in the total delay only when a task is redirected from one MECS to another MECS. According to Little's law, the queuing

delay is proportional to the queue length, which is proportional to the MECS load. Figure 6 shows that the rate of tasks completed within their delay constraints is the smallest for all λs when LMLBR is used. Since the MECS load vector obtained by LMLBR is leximax minimal, the queue length vector whose element is the queue length of each MECS is also leximax minimal, which results in the smallest total delay for a given λ. Overall, the rate of tasks completed within their delay constraint for LMLBR is 10–42% better than that achieved by other methods.

To inspect the influence of the number of redirection per task on the distribution of MECS load, we show in Figure 7 the loads of all 50 MECSs at the same time slot when $\lambda = 0.9$. As the number of task redirection increases, the load imposed on a MEC system is distributed among MECSs more fairly. However, the congestion level of a network connecting MECSs also increases with the number of task redirections. In addition, the time from when a task is first offloaded from a device to a MEC system to the time when it is finally served by a MECS also increases with the number of times that the task is redirected. Since the problem of determining the optimal number of task redirection in itself deserves to be investigated thoroughly, we set this problem as one of our future works.

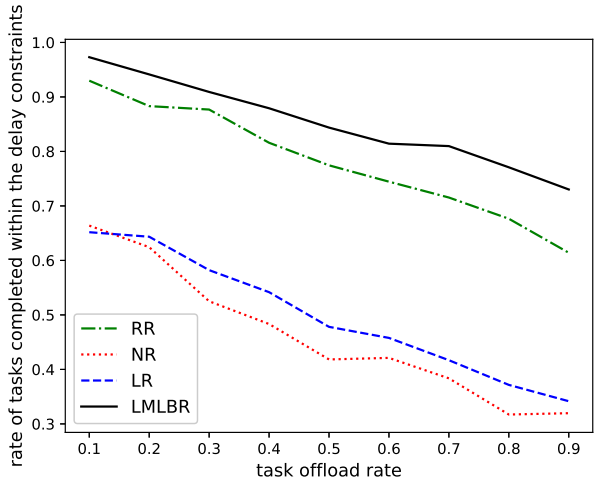


Figure 6. Rate of tasks completed within the delay constraints versus the task offload rate.

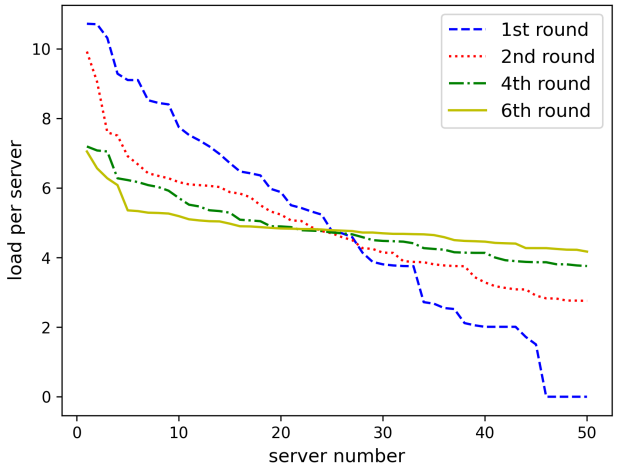


Figure 7. Influence of the number of redirect per task on the load distribution among MECSs when $\lambda = 0.9$.

To investigate the sensibility of performance with respect to the simulation parameters, we evaluated the performance by varying the queue size. In each MECS, we set the size of the service queue to be the same as that of the waiting queue. We show the

results in Tables 4 and 5. In Table 4, as the queue size increases, the task blocking rate decreases, but the difference is too small to mean much in practice. However, in Table 5, the performance comparison is meaningful in terms of the rate of tasks completed within the delay constraints. This is attributed to the following facts. In LMLBR, a MECS i redirects tasks to the MECSs whose loads are lower than ρ_i. In addition, the amount of tasks redirected from MECS i to MECS j is not excessive because LMLBR considers the load difference between i and j. However, in RR, an overloaded MECS redirects the tasks to the randomly selected MECS until it becomes underloaded. If tasks are redirected to one MECS excessively, the service queue of the MECS increases sharply, which results in the large waiting time and the decreases in the rate of tasks completed within the delay constraints.

Table 4. Task blocking rate versus queue size.

λ	Q Size = 20		Q Size = 10		Q Size = 5	
	RR	**LMLBR**	**RR**	**LMLBR**	**RR**	**LMLBR**
0.5	0	0	0	0	0.0247	0.0521
0.7	0	0	0.0021	0.0024	0.0491	0.0605
0.9	0.0004	0.0001	0.0051	0.0035	0.0792	0.0666

Table 5. Rate of tasks completed within the delay constraints versus queue size.

λ	Q Size = 20		Q Size = 10		Q Size = 5	
	RR	**LMLBR**	**RR**	**LMLBR**	**RR**	**LMLBR**
0.5	0.7641	0.8528	0.7892	0.8532	0.7745	0.8437
0.7	0.6652	0.7936	0.7213	0.8225	0.7154	0.8097
0.9	0.5596	0.7423	0.6113	0.7612	0.614	0.7301

Since the proposed method requires the exchange of information between MECSs, it incurs communication overhead. If a broadcast channel is used to exchange the load information among $|N|$ MECSs, the communication cost is $O(|N|)$. If an unicast channel is used to exchange the information, the overhead is $O(|N|^2)$. The factor determining whether the exchange of information between MECSs is required or not is the way that a MECS decides whether it is overloaded. If a MECS decides to redirect its tasks when its load is larger than a local threshold value, the communication cost of the method is zero. However, it is difficult to configure an optimal threshold value according to the dynamic network condition. If a MECS decides to redirect its tasks if its load is higher than $\bar{\rho}$, it requires exchanging its load information with neighboring MECSs. All four methods that are used for performance comparison in our experiments use $\bar{\rho}$ for the threshold value. Thus, each MECS exchanges its load information with other MECSs at each time slot and calculates the average load to decide whether to redirect its tasks. Therefore, they have the same communication overhead.

The computational complexity of our method can be derived as follows. In our method, when the load of a MECS is higher than the average load, the MECS selects a set of target MECSs whose loads are lower than its load. Then, the MECS redistributes its tasks to the target MECSs in order in proportion to the load differences between itself and the target MECSs. Thus, the computational complexity of LMLBR is $O(|N|log|N|)$ for sorting the target MECSs in order. In RR, when the load of a MECS is higher than the average load, the MECS randomly selects a target MECS to redistribute the tasks. Thus, the computational complexity of RR is $O(1)$. However, considering the results in terms of the quality of the service provided by a MEC system, the proposed method outperforms RR in terms of the rate of tasks completed within the delay constraints, while being comparable to RR with respect to the task blocking rate.

5. Conclusions and Future Works

In this paper, we presented a distributed task redirection method among MECSs to improve the resource efficiency of MEC systems. We proved that our method drives the MEC system to the state where the loads of the MECSs are lexicographically minimal. Through simulation studies, we showed that compared with other conventional load balancing methods, the proposed method achieves the smallest load difference between the load of the MECS whose load is the highest and that of the MECS whose load is the smallest. By obtaining the leximax minimal load vector, the proposed method improves the rate of tasks completed within their delay constraints. In addition, our method decreases the average task blocking rate when the task offload rate is high.

We are planning the following future works. Firstly, we will devise a task selection method that selects tasks to redirect from the waiting queue of a highly loaded MECS. By considering the delay requirements of the tasks in $O_i(t)$ when constructing the set Φ_{i,j^*}, we expect that the rate of tasks completed within the delay constraints will improve. Secondly, we will inspect the influence of the delay required to redirect a task from one MECS to another MECS. We will also scrutinize the optimal number of redirection per task. Finally, we will extend the proposed method so that it can be used in the environment where only a subset of MECSs in the system can exchange their load information.

Author Contributions: Conceptualization, J.P. and Y.L.; methodology, J.P.; software, Y.L. Both authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by stage 4 BK21 project in Sookmyung Women's Univ of the National Research Foundation of Korea Grant. This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2021R1F1A1047113).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mach, P.; Becvar, Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
2. Filali, A.; Abouaomar, A.; Cherkaoui, S.; Kobbane, A.; Guizani, M. Multi-Access Edge Computing: A Survey. *IEEE Access* **2020**, *8*, 197017–197046. [CrossRef]
3. Lai, S.; Fan, X.; Ye, Q.; Tan, Z.; Zhang, Y.; He, X.; Nanda, P. FairEdge: A Fairness-Oriented Task Offloading Scheme for Iot Applications in Mobile Cloudlet Networks. *IEEE Access* **2020**, *8*, 13516–13526. [CrossRef]
4. Zhu, H.; Zhou, M. Efficient Role Transfer Based on Kuhn–Munkres Algorithm. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2012**, *42*, 491–496. [CrossRef]
5. Zhu, H.; Zhou, M. M-M Role-Transfer Problems and Their Solutions. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2009**, *39*, 448–459.
6. Zhu, H.; Zhou, M. Roles in Information Systems: A Survey. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2008**, *38*, 377–396.
7. Sharmin, Z.; Malik, A.W.; Rahman, A.U.; Noor, R.M. Toward Sustainable Micro-Level Fog-Federated Load Sharing in Internet of Vehicles. *IEEE Internet Things* **2020**, *7*, 3614–3622. [CrossRef]
8. Liu, L.; Chan, S.; Han, G.; Guizani, M.; Bandai, M. Performance Modeling of Representative Load Sharing Schemes for Clustered Servers in Multiaccess Edge Computing. *IEEE Internet Things J.* **2019**, *6*, 4880–4888. [CrossRef]
9. Wang, Z.; Xue, G.; Qian, S.; Li, M. CampEdge: Distributed Computation Offloading Strategy under Large-Scale AP-based Edge Computing System for IoT Applications. *IEEE Internet Things J.* **2021**, *8*, 6733–6745. [CrossRef]
10. Li, J.; Luo, G.; Cheng, N.; Yuan, Q.; Wu, Z.; Gao, S.; Liu, Z. An End-to-End Load Balancer Based on Deep Learning for Vehicular Network Traffic Control. *IEEE Internet Things J.* **2019**, *6*, 953–966. [CrossRef]
11. Filali, A.; Mlika, Z.; Cherkaoui, S.; Kobbane, A. Preemptive SDN Load Balancing with Machine Learning for Delay Sensitive Applications. *IEEE Trans. Veh. Technol.* **2020**, *69*, 15947–15963. [CrossRef]
12. Abedin, S.F.; Bairagi, A.K.; Munir, M.S.; Tran, N.H.; Hong, C.S. Fog Load Balancing for Massive Machine Type Communications: A Game and Transport Theoretic Approach. *IEEE Access* **2018**, *7*, 4204–4218. [CrossRef]
13. Cybenko, G. Dynamic Load Balancing for Distributed Memory Multiprocessors. *J. Parallel Distrib. Comput.* **1989**, *7*, 279–301. [CrossRef]
14. Xiaoa, L.; Boyd, S.; Kim, S.-J. Distributed Average Consensus with Lead-Mean-Square Deviation. *J. Parallel Distrib. Comput.* **2007**, *67*, 33–46. [CrossRef]
15. Andrews, J.G.; Singh, S.; Ye, Q.; Lin, X.; Dhillon, H.S. An Overview of Load Balancing in HetNets: Old Myths and Open Problems. *IEEE Wirel. Commun.* **2014**, *21*, 18–25. [CrossRef]

16. Park, J. Association Game for Conflict Resolution between UEs and Small Cells. *Hindawi Wirel. Commun. Mob. Comput.* **2020**, *2020*, 5801217. [CrossRef]
17. Alex, O.; Tsitsiklis, J.N. Convergence Speed in Distributed Consensus and Averaging. *SIAM J. Control Optim.* **2009**, *48*, 33–55.
18. Sarkar, S.; Tassiulas, L. Fair Allocation of Discrete Bandwidth Layers in Multicast Networks. In Proceedings of the 2000 IEEE Conference on Computer Communications (INFOCOM'00), Tel Aviv, Israel, 26–30 March 2000; pp. 1491–1500.
19. Sarkar, S.; Tassiulas, L. Fair Bandwidth Allocation for Multicasting in Networks with Discrete Feasible Set. *IEEE Trans. Comput.* **2004**, *53*, 785–797. [CrossRef]
20. Yang, X.; Yu, W.; Huang, H.; Zhu, H. Energy Efficiency based Joint Computation Offloading and Resource Allocation in Multi-access MEC Systems. *IEEE Access* **2019**, *7*, 117054–117062. [CrossRef]
21. Zhang, K.; Mao, Y.; Leng, S.; Zhao, Q.; Li, L.; Peng, X.; Pan, L.; Maharjan, S.; Zhang, Y. Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks. *IEEE Access* **2016**, *4*, 5896–5907. [CrossRef]
22. 3GPP TR 38.912 Version 15.0.0 Release 15. 5G Study on New Radio (NR) Access Technology. Available online: https://www.etsi.org/deliver/etsi_tr/138900_138999/138912/15.00.00_60\/tr_138912v150000p.pdf (accessed on 15 August 2021).