PAPER

Simulation Study of Low-Latency Network Model with Orchestrator in MEC*

Krittin INTHARAWIJITR[†a)], ***Student Member***, **Katsuyoshi IIDA**[††b)], ***Senior Member***, **Hiroyuki KOGA**[†††c)], ***and*** **Katsunori YAMAOKA**[†d)], ***Members***

SUMMARY Most of latency-sensitive mobile applications depend on computational resources provided by a cloud computing service. The problem of relying on cloud computing is that, sometimes, the physical locations of cloud servers are distant from mobile users and the communication latency is long. As a result, the concept of distributed cloud service, called mobile edge computing (MEC), is being introduced in the 5G network. However, MEC can reduce only the communication latency. The computing latency in MEC must also be considered to satisfy the required total latency of services. In this research, we study the impact of both latencies in MEC architecture with regard to latency-sensitive services. We also consider a centralized model, in which we use a controller to manage flows between users and mobile edge resources to analyze MEC in a practical architecture. Simulations show that the interval and controller latency trigger some blocking and error in the system. However, the permissive system which relaxes latency constraints and chooses an edge server by the lowest total latency can improve the system performance impressively.

key words: *mobile edge computing, low-latency network, orchestrator, processor sharing, simulation*

1. Introduction

The fifth generation (5G) network will enhance communication through higher quality of services [2]. One 5G requirement is reduced service latency. A very short latency will enable or improve many latency-sensitive mobile applications in which excessive total latency is not acceptable. For example, an augmented reality application [3] showing additional data on a screen will be able to display more information in real-time. The latency issue in 5G networks is critical and must be resolved for this communication technology to move forward.

However, most mobile applications rely on cloud computing service because of limited mobile device resources. Mobile users have no way of knowing the physical location of cloud servers, though, which can be far from the users. A long distance between users and cloud servers leads to long network communication latency that will not satisfy the service time requirement in mobile applications.

One solution to the distance problem of cloud computing is *Mobile Edge Computing* (MEC) [4]–[6], which distributes cloud capability to the edge of networks where users connect to the entry point of core networks. MEC can provide services in a local area and support very short service communication latency.

Even though MEC can enable short communication latency, its resources are still limited. Unlike cloud computing, MEC is established through the cooperation of network devices acting as a virtual server inside the networks. Too many requests from users in the same area can go to one edge server and overload it. An excessive workload will mean that more time is needed to complete each job. Eventually, the computing latency will not be acceptable for low latency services.

However, there is not only one edge server along the edge of networks. Users are able to access other edge servers near the location, such as a sever in the same city. Their packets probably travel through networks but the distance is definitely shorter than the cloud. The nearby edge servers with lighter loads could probably provide shorter computing latency.

Surveys in [7], [8] address many challenges about MEC. The common issue is how to communicate between each component and how an edge sever handles load with their own capacity. Our previous studies [9], [10], consequently, consider an ideal MEC model where it is assumed that the system knows all information regarding sources and edge servers without any delay. However, that is unlike the real situation in network architecture. If all components send updated information every moment, the network would have a very high overhead. Furthermore, the system cannot instantaneously determine the necessary MEC network information.

Edge computing is a distributed computing model, which means that huge number of computers will be deployed in the network. How to manage them is one of important issues. To answer this, European Telecommunications Standards Institute (ETSI) [11] divides the MEC architecture to two levels; system level and host level. The host level that contains edge servers operates as a distributed

Manuscript received December 21, 2018.
Manuscript revised April 17, 2019.
Manuscript publicized May 16, 2019.

[†]The authors are with the Dept. of Information and Communications Engineering, Tokyo Institute of Technology, Tokyo, 152-8852 Japan.
[††]The author is with the Information Initiative Center, Hokkaido University, Sapporo-shi, 060-0811 Japan.
[†††]The author is with the Dept. of Information and Media Engineering, University of Kitakyushu, Kitakyushu-shi, 080-0135 Japan.

*Earlier version of this paper has been presented in [1].

a) E-mail: intharawijitr@net.ict.e.titech.ac.jp
b) E-mail: iida@iic.hokudai.ac.jp
c) E-mail: h.koga@kitakyu-u.ac.jp
d) E-mail: yamaoka@ict.e.titech.ac.jp

DOI: 10.1587/transcom.2018EBP3368

model. The system level as a control and management part is introduced as a centralized management model. It includes a component called *orchestrator* to communicate between users and edge servers. It is a control unit in system level to manage the components in host level. Furthermore, MEC must be implemented on network technology as Network Function Virtualization (NFV) [12] which also requires a NFV orchestrator for controlling the network. To deploy the orchestrator, there could be many considerable issues such as a single point failure, computational cost and resource requirement. However, it is very important to first study the real performance if we deploy the orchestrator in MEC system.

Consequently, an extension of the ideal model must be considered. A practical MEC model using a control unit (or an orchestrator) is needed where the interval necessary to collect system information is taken into account. The model will also have to take into consideration that a longer interval will require the system to work with information that is more outdated. This could lead the system to make poor decisions about network management and affect performance.

This paper studies the impact of computing and communication latency with a more practical model by cooperating with an orchestrator. We model MEC architecture and introduce the time interval needed to collect network information. Evaluation of the defined model through simulation is described, and numerical results are discussed with regard to how MEC can enable low latency services according to a practical model.

This manuscript is an extension of the conference paper [1] by investigating more parameters that would have an impact on the system. In [1], we studied the selection policy performance, orchestrator update interval and orchestrator communication latency but ignoring computation latency inside the orchestrator and always assumed it as zero. The manuscript, therefore, considers the latency from controller computation and expresses how it affects the system performance compared to other latencies.

The paper is organized as follows. Section 2 addresses definition and some literatures about MEC. In Sect. 3, we discuss the theoretical model. Section 4 depicts the simulation and evaluation metrics in this research. Section 5 presents all numerical results from the simulation. Finally, Sect. 6 concludes and highlight our work and future plan.

2. Mobile Edge Computing and Related Work

MEC is applied in a network architecture to distribute cloud computing service within the local area – especially at the edge of the core network [4]. The aim of MEC is to shorten the communication path from users to service resources and to provide services faster. MEC is also known as Cloudlet, fog computing, and edge cloud computing, depending on one's perspective [13].

The MEC architecture is illustrated in Fig. 1. Mobile devices, such as tablets, smart phones, and laptops, connect to a base station (BS) over a radio access network (RAN).

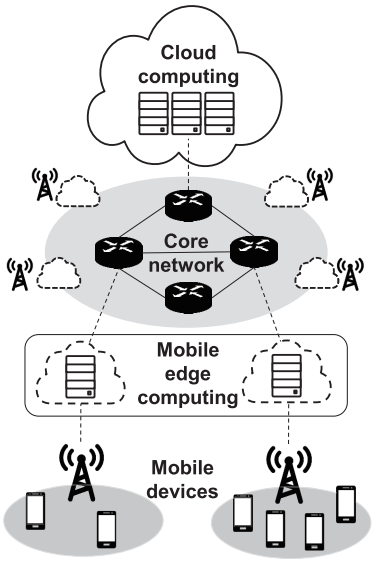


Fig. 1 Mobile edge computing domain.

The BS first connects to the edge of a network before entering the Internet. At the edge, network components having high computation and storage capabilities are gathered to create a virtual server offering mobile edge services. If any workload requests resources exceeding what the edge server can support, the request can be forwarded through the core network until it reaches cloud services on another side of the network.

Many works have focused on the optimal delay for saving more energy. In [4], Chen et al. study the optimization model to find the minimum overhead of distributing offload in MEC using game theory. M. Chen et al. in [14] and V. Chamola et al. in [15] modeled and simulate the task offloading. J. Liu and others in [16] also did the delay optimization of task scheduling when given a delay deadline. S. Sundar, B. Liang [17] also concerned a delay constrain and propsed their heuristic algorithm. While D. Satria and others [18] did not find the optimum but focus the procedure of when an edge server is overloaded.

Sarkar et al. in [19] and Deng et al. in [20] did not underline the time constrain but studied the suitability of MEC in cloud computing with regard to power usage.

Nevertheless, all of these studies assume that the model ideally knows the network information without being concerned with actual implementation.

In our previous studies [9], [10], we model MEC as an ideal system that can immediately get network information. A simple model is proposed in [9] and is used to study the MEC model mainly by considering the impact of both communication latency and computing latency. The communication latency is simply calculated by hop count while the computing latency is determined by a linear function. This preliminary work is still too ideal and simple for practical purposes, so we extend the model further in [10] by applying processor sharing [21] for the computing. However, the model still relies on the ideal assumption.

The ideal model is not practical in a real environment, which is more complex than the model recognizes. For example, without an orchestrator as a control unit, the model cannot say what network information will be available to the system or how the system will maintain that information. This paper, therefore, further extends the MEC model to make it more realistic. An interval for collecting network information is introduced to the model, as explained in Sect. 3.

3. Mathematical Model

To study MEC, we define a model of MEC to identify the important components in the architecture and address the relationship of each element in the model. Since this paper is proposing an extension of existing research, we will first explain the necessary points of the previous work [10].

3.1 MEC Model without Orchestrator

In this section, we describe a model of MEC when there is no an orchestrator. We assume that the system can know the component status i real-time as a ideal case.

We define a mathematical model of MEC in [10]. The essential components are illustrated in Fig. 2 and briefly explained below.

S_i: A source node gathering mobile devices in the same area, $1 \le i \le N$.
E_j: An edge node representing an mobile edge server, $1 \le j \le M$.
w_k: A workload produced from a source node and containing b_k as a size which is defined by amount of computational resources, $1 \le k \le K$.
λ_i: Average producing rate of a source node as a Poisson process [22].

The communication latency is denoted by $l_{i,j}$ for a link from S_i to E_j. The hop count is determined, in which each link has the same latency l_h. We suppose that every source node can access all edge nodes. The communication latency follows the model in [10]. The hop counts $H_{i,j}$ is calculated by the difference between node IDs where S_i and E_i have one hop distance, then plussing hop distance between E_i and E_j as shown in Fig. 3. The communication latency can be determined by $l_{i,j} = H_{i,j} \times l_h$.

For the computing latency, E_j computes workloads with a processing rate of μ_j instructions per time unit. We apply the concept of processor sharing (PS) [21], which allows a single server to simultaneously process many workloads with the resources equally divided for each workload and without waiting in a queue. We then estimate the computing latency by the current number of workloads and an additional new workload. The estimation function is defined as $P_{k,j}(t)$:

$$P_{k,j}(t) = \left(\frac{b_k}{\mu_j}\right)(n_j(t) + 1), \tag{1}$$

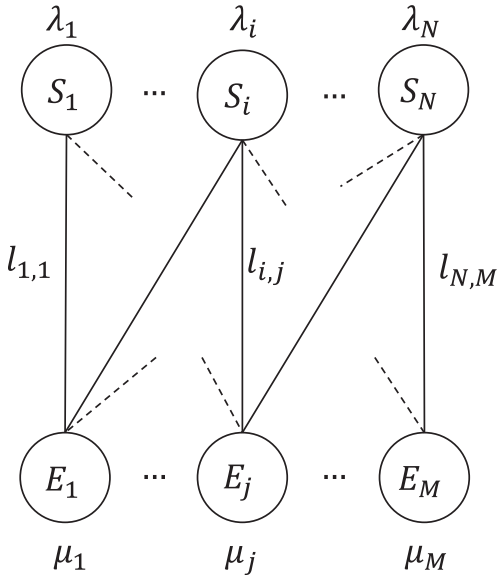


Fig. 2 Problem model.

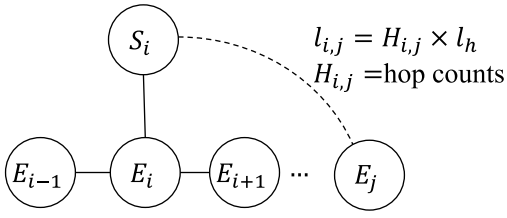


Fig. 3 Propagation latency model.

where $n_j(t)$ denotes the current number of workloads at time t, $t \ge 0$.

Finally, the total latency of workload w_k from S_i to E_j is defined as $L_{i,j,k}(t) = l_{i,j} + P_{k,j}(t)$.

Let θ be the maximum allowance of latency for a service. We propose the optimization model as follows.

$$\min_{b_r, b_k} \quad P_b = \frac{\sum_{w_r} b_r}{\sum_{w_k} b_k} \tag{2}$$

$$\text{subject to} \quad L_{i,j,a}(t) \le \theta \tag{3}$$

$$L_{i,j,e}(t') + \delta_{e,j} \le \theta \tag{4}$$

$$n_j(t) \ge 0 \tag{5}$$

$$l_h, \forall \mu_j, \forall b_k > 0 \tag{6}$$

$$t > t' \ge 0. \tag{7}$$

We set the objective function to find the minimum blocking probability, denoted by P_b, which is calculated from the ratio of rejected workloads w_r with size b_r in the system. Constraint Eq. (3) is to check whether the total latency of a new workload exceeds the maximum allowance. When estimating the computing latency, we might underestimate, which would lead to a wrong decision where some workloads are finished later than allowed.

According to PS, when a workload arrives at a server, resources of the server are shared to thae new workload

equally with other existing workloads. That makes each existing workloads extend the computing latency because of getting less resources. We define $\delta_{e,j}$ as the amount of that increased latency for an existing workload w_e when the server accept a new workload. The system can check whether the new workload will cause the finishing time of any existing workload to exceed the maximum allowance as in Eq. (4). We note that $\delta_{e,j}$ is a parameter that cannot be predefined or specified a certain value. We can only calculate the change of computing latency when a edge node accepts or finishes other workload.

We note that in this study b_k is identical for all workloads, and edge node E_j has a single processor with unlimited memory capacity.

The system using this optimization model is called a *"strict underestimation system"* because it checks latency of every existing workload with Eq. (4) and does not allow any excessive latency caused by underestimation.

In the real-time multimedia [23], the strict underestimation system is determined as a deterministic admission control for multimedia servers because it guarantees the worst case assumption of no errors. However, according to [23], checking every existing workload would cause very high overhead in the system and blocking probability becomes high. There is another service guarantee using statistical admission control. It increases the acceptance probability of workloads by allowing some service errors to more utilize the system resources. We therefore consider another system, called a *"permissive underestimation system,"* which ignores Eq. (4) and assumes that the service can tolerate some errors (or statistical services). For example, a real-time service, such as streaming video, augmented/virtual reality, on-line gaming, needs very low latency from edge nodes but it can ignore late reply packets and process only in-time feedback. The errors will be presented as lack or jitter to users. It might not be notices if they are very small.

The optimization model corresponds to an extension of the stochastic knapsack problem [24] that consider an edge node as a knapsack and a workload as an item in the bin. It optimizes the most effective packing items into bins. Our model is more complex because of the dynamic status from processing in an edge node. Since the optimization of the stochastic knapsack problem is NP-hard, our model is also difficult to find the solution. As a results, we consider three policies as one of the heuristic algorithm to select a target edge from candidate nodes which satisfy the constraints. The three policies are the followings.

- *Random policy* (Random): One simple approach is unpatterned selection where one edge node is randomly selected according to a uniform distribution.
- *Lowest latency policy* (Low. latency): This policy concerns fast computation and quick transmission. It selects the edge node providing the lowest total latency.
- *Minimum unfinished work policy* (Min. work): The unfinished work is determined as the unprocessed amount of all existing workloads in an edge node. The node with the least work is preferred as the most available edge node.

3.2 MEC Model with Orchestrator

In models of workload allocations for multiple edge nodes, the system must maintain information regarding sources, workloads, edge nodes, and links in the network. When a workload is produced from a source node, one edge node must be selected as a destination. To control flows and manage the network, we have to develop a way to ensure all the system information is known.

For the model without the controller in Sect. 3.1, we assume that every component in the network can know the status of other nodes immediately but, in practical, the system is not always aware of the current state of edge nodes. It can gather the status with an interval of time. As in the ETSI framework [6], MEC leaves that function to an entity called *orchestrator* which manages edge server resources and responds to users. This entity is very important to develop MEC system in the realistic system. The orchestrator actually operates many functions to manage the system. In this paper, we focused on the controller function of administration control and communication with sources/servers. Because this is one of MEC orchestrator functions, we will use the word *controller* instead of a orchestrator to represent a control unit in the system as a centralized model. The controller can manage and control a network very simply. It collects the necessary information for components, e.g. by probe messages, and makes autonomous decisions based on its collected data. In addition, we can deploy the controller with either a real or virtual server connecting around edge and source nodes.

In this study, we design a system that uses a controller to collect all information and control flows from sources to edges as in a centralized model. However, the controller cannot monitor every element all the time. It needs a time interval to collect data and control the system based on the information. As illustrated in Fig. 4, the controller performs as described below.

0. Controller updates information with interval.
1. Source node S_i produces a new workload w_k.
2. Source node S_i asks the controller for a destination.
3. Controller selects an edge node according to a policy regarding its current information.

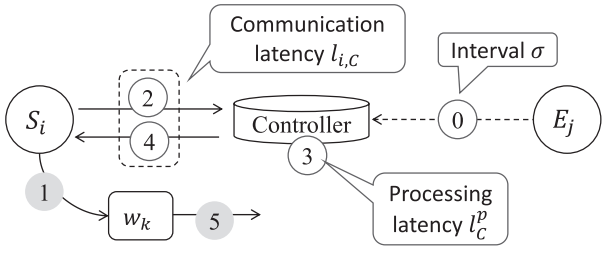


Fig. 4 Workload acceptance procedure with a controller

4. Controller sends back a decision to the source.
5. Source node submits the workload to the edge node.

The controller has an internal delay (arrows (0) to (5) in Fig. 4) resulting from processing and communication of the controller. This study concentrates on both the communication between the controller and nodes in the system (arrows (2) and (4)), and processing time inside the controller (arrow (3)). We also assume that the controller has enough potential to do its duty without any failure.

The controller collects updated information from all edge nodes in every time interval σ as in arrow (0). The controller communicates with source nodes as arrows (2) and (4). We suppose that the latencies of both arrows are the same. In addition, the controller processes an inquiry message from sources with time of arrow (3).

The controller communication latency ($l_{i,C}$) is then determined as the location of the controller between source nodes and the controller. We first consider this latency by hop count like the workloads. The hop delay of the controller, said (l_h^c), is defined in scope of workload hop delay (l_h). We also determine the location of the controller in between each source $S_{i\in\{1,N\}}$ and edge E_1. That means $l_h^c = 0$ when the controller is at the same place with source S_i and $l_h^c = l_h$ at E_1. As a result, we can state

$$l_{i,C} = H(i,1) \times l_h^c. \tag{8}$$

The controller processing latency, we define l_C^p as delay from receiving a query to replying message. Either more complexity of a process or less resources of controller lead a processing latency in the controller longer.

Note that such an interval and controller communication are also required in the distributed model. This analysis of the impact of the interval will be useful with regard to both centralized and distributed models.

4. Simulation and Evaluation Metrics

We describe the simulation implementation and evaluation metrics for the model in this section. We then show and discuss the numerical results from that simulation.

We developed a simulation of the problem model using C++ programming language to evaluate the MEC system under various parameters.

We consider a target application of the simulation to real-time interaction with human vision such as virtual/augmented reality, steaming and gaming. According to the concept of Tactile Internet merging with MEC, the latency scope should be roundly 10 ms; otherwise users will notice jagged process [25]. Therefore, the parameter values were carefully chosen by the application scope.

We fixed some parameters while we varied the number of edges (M), and controller interval (σ) as summarized in Table 1. The maximum allowance θ is set to 15 ms as a requirement for real-time virtual application. We assume that the MEC service is being operated in a city town where there are $N = 10$ source nodes requiring the service. Each source node generates independent workloads with Poisson Process as an event of service requests in networks. The producing rate is rate $\lambda = 2$ workloads per millisecond for all sources node. All workloads equally contain $b_k = 16$ million instructions of CPU cycles. We also suppose all edge nodes apply PS for execution with identical processing rate $\mu_j = 32$ billion instructions per second as a single server at the base station. A hop delay between each nodes is fixed to 2 ms as a proper delay between two base stations. In the simulation, we also implement a controller to collect the data with interval σ as mention in Sect. 3.2

Table 1 Simulation parameters.

Parameter	Description	Value
N	Number of source nodes	10
M	Number of edge nodes	6–14
λ	Producing work rate of a source node	2 workloads/ms
μ	Processing rate of an edge node	32 billion instructions/s
b	Size of a workload	16 million instructions
l_h	Hop delay	2 ms
θ	Maximum service latency allowance	15 ms
σ	Controller interval	0–2 ms
$l_{i,C}$	Controller hop delay	0–2 ms
l_{CT}	Controller processing latency	0–2 ms

The parameters were not changed during each simulation, which was allowed to run long enough to reach a steady state within which the results remained almost constant. To confirm that was the case, we ran each simulation for 500 s but did not collect results for the first 200 s, which we considered a warm-up phase.

Three metrics showing system performance were evaluated in this simulation for both a strict underestimation system and a permissive underestimation system. We explain each of them as follows.

- *Blocking probability* (P_b): Both systems are evaluated using Eq. (2) to represent how effective the system can support mobile users.
- *Decision error* (ϵ): The permissive underestimation system commits errors by ignoring Eq. (4) which accepted workloads could have excessive latency after execution. We call such an error as decision error. It is defined as a ratio of accepted workloads having total latency exceeding the allowance. It can be stated as

$$\epsilon = \frac{\sum_{w_u} b_u}{\sum_{w_a} b_a}, \tag{9}$$

  where w_u is a wrong accepted workload and w_a is an accepted workload.
- *Modified blocking probability* (P_b'): We cannot fairly compare the performance of both systems using the blocking probability since the strict system always achieves zero decision errors while another probably leads some errors. The permissive system should be evaluated using another metric able to measure its ac-

tual efficiency. To include the effect of decision error, we define modified blocking probability (P'_b) stated as

$$P'_b = P_b + \epsilon - P_b \cdot \epsilon. \quad (10)$$

This metric shows the real efficiency of the permissive system with respect to both blocking probability and decision error. Therefore, we can compare it with blocking probability of the strict system.

We note that the system evaluates only the blocking probability while the permissive system measures all three matrices.

5. Numerical Results

The numerical results from the simulation are presented and discussed in this section. First, we analyze the impact of the three polices with respect to different numbers of edge nodes in the system on the metrics described in Sect. 4. Then we analyze the impact of the interval, communication latency, processing latency of the controller (as respectively arrow (0), (2)&(4), and (3) in Fig. 4).

5.1 Policies

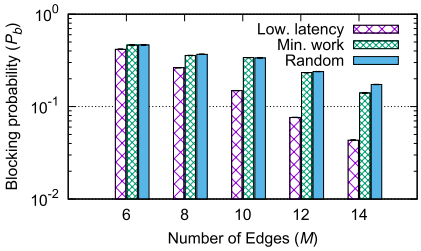
The three policies–random, low. latency, and min. work influence how to select edge nodes for arriving workloads. Fig. 5 shows the impact of the number of edge nodes (M) on the blocking probability and decision error when we fix the controller interval to 0.5 ms and $l_{i,C}$ always be 0. The Y-axis uses a logarithmic scale to show the results.

Generally, when we added more edge nodes in the system, the blocking probability and decision error were significantly improved (Fig. 5). It is clear that more edge nodes provide more resources in the network. That can support more workloads and make the system more efficient.

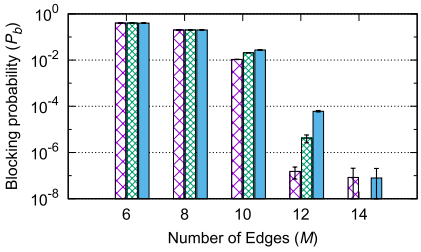
As Fig. 5 shows, the lowest latency policy provided the best performance and least error among the three polices in both systems. For fewer edge nodes ($M < N$), there was no clear difference among the policies. Because there were very limited resources in this case, each policy had few choices for allocating an edge node. On the other hand, when we increased the number of edge nodes, the lowest latency policy dropped the blocking probability below that of others.

The lowest latency policy usually gets the best solution because it enables very short transmission and very fast computation. The lowest latency policy helps an edge node complete workloads faster than other policies. The quick release of edge resources allow each edge node to accept more workloads and reduces the blocking probability.

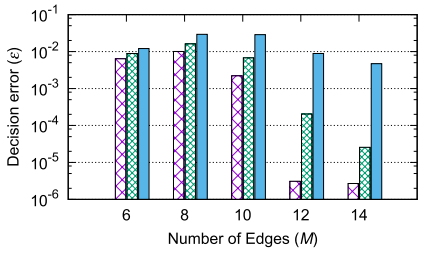
Figure 5(a) presents that the strict system still causes a higher rejection ratio. In addition, the random and minimum unfinished work policies had similar results. The minimum unfinished work policy has the potential to balance load in a system by selecting the most available edge node. The random policy also provides load balancing with a uniform



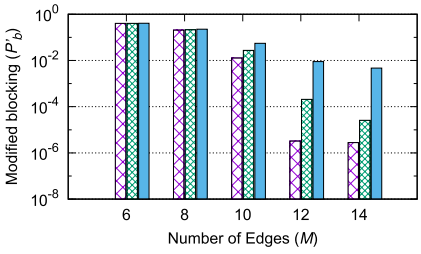
(a) Blocking probability of strict system



(b) Blocking probability of permissive system



(c) Decision error of permissive system



(d) Modified blocking probability of permissive system

Fig. 5 Impact of each policy.

distribution and less complexity. So, from a practical viewpoint, the two policies did not perform the same result.

The permissive system with the random policy leaded to the highest decision error (Fig. 5(c)) because it always makes decisions randomly without considering latency or available resources. The minimum unfinished work policy resulted in error that was quite low, but not lower than that of the lowest latency policy. It distributes workloads into edge nodes equally, and that helps ensure the number of workloads in each edge is nearly the same and the estimated latency is fairly precise. Nonetheless, the lowest latency policy is still the winner since it finishes a process so rapidly that underestimation is not over the maximum allowance.

The modified blocking probability of the permissive system is shown in Fig. 5(d). The permissive system provided a better solution than that of the strict system. Because of the high error, the random policy improved little, while the other policies performed extremely well.

The permissive system likely preferred a redundant supply of service ($M > N$). Considering Fig. 5(b), the system's performance was significantly enhanced when the number of edge nodes was greater than 10.

The lowest latency policy provides low blocking performance and very low decision error, especially for the permissive underestimation system. Additionally, more spare edge nodes in the system leads to an impressive blocking probability.

5.2 Controller Latency

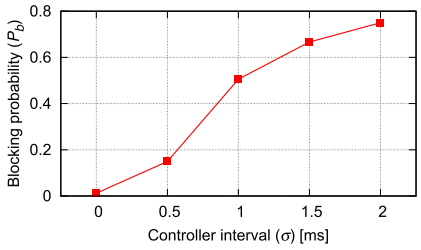
We will show the all latency impacts from using controller in the MEC. Firstly, we investigate the controller interval σ. Next, we analyze the impact of the communication latency from the controller ($l_{i,C}$) and a controller processing latency (l_{C^T}). Since we added more latencies to the system, we have to consider a higher allowable latency resulting from the communication with the controller.

5.2.1 Controller Interval

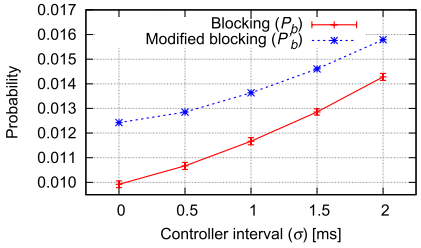
We analyzed the controller interval (σ) by assuming that the controller communication latency ($l_{i,c}$) always equal to 0 in order to see the real effect of this parameter. The results in Fig. 6 show the impact that an interval on blocking in both the strict and permissive underestimation systems. Note that both systems used the lowest latency policy and the number of edge nodes (M) was fixed at 10 nodes.

We showed the impact of using the controller in the system by increasing the controller interval (σ) from 0 to 2 ms. The interval $\sigma = 0$ represents the performance of an MEC model without the controller as described in Sect. 3.1 while the results of the model in Sect. 3.2 are presented where $\sigma > 0$. Figures 6(a) and 6(b) show the blocking probability for the strict and permissive underestimation systems respectively. The both systems had the same tendency that the controller interval where $\sigma > 0$ increased the blocking probability of the both systems from a case of the system without the controller ($\sigma = 0$). Clearly, when the controller collected data more frequently, it got information that is more current and controls the system with nearly real-time information. However, the both systems had the different impact from the controller latency. The analysis of each system is described separately as following.

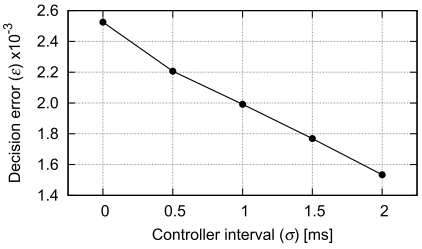
The blocking probability of the strict underestimation system in Fig. 6(a) greatly increased to almost 0.8 when we extended the controller interval σ to 2 ms. The strict system often rejects workloads due to Eq. (3), Eq. (4). When one workload has been rejected, the following workloads will be dropped until the controller receives an update from the edge nodes. That severely increases the blocking probability.



(a) Strict system



(b) Permissive system



(c) Decision error of permissive system

Fig. 6 Impact of controller interval, when edges (M) = 10.

The blocking probability of the permissive underestimation system in Fig. 6(b) was increased when the controller interval was prolonged, as it was for the strict system. However, the permissive system was more robust with respect to the controller interval than the strict system. Even though we inspected the modified blocking probability (P'_b) of this system, the combined performance was still impressive. For example, when the interval was 2 ms, the strict system reached a blocking probability of 0.8 while that of the permissive system was below 0.02. In fact, if an edge node has already finished some workloads, the controller in the strict system will not notice that until the next update interval, and resources available to process more workloads will probably go unused. Because the permissive system may accept a new workload without considering existing workloads, it can use any spare capability of an edge node by not waiting for updated information.

Another metric of the permissive system is the decision error ϵ (Fig. 6(c)). According to the simulation results, the decision error was not higher than 0.0026. When we increased the controller interval, the decision error of the permissive system surprisingly decreased. Decision error occurs when the controller permits too many workloads into one edge node and causes the latency of some existing work-

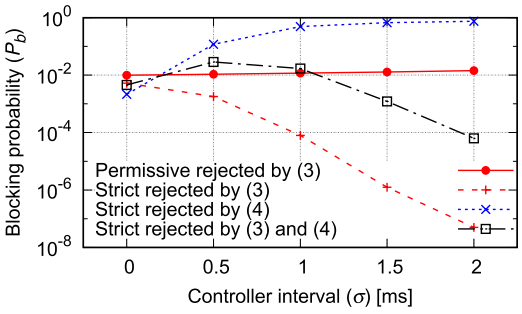


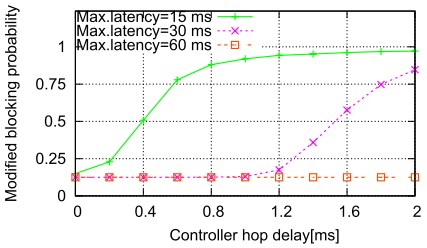
Fig. 7 Rejection cause, when edges (M) = 10.

loads to exceed the maximum. Decision error can be decreased if the system uses a longer interval. The controller updates the number of workloads of an edge node when it accepts one workload and will never decrease the number if it has not received an update. A longer updating period makes the number of workloads in the controller higher than the actual value in the edge node. The higher value by update interval leads the controller to further overestimate the computing latency and bring less underestimated latency. According to the impact of too much overestimation, the controller reasonably rejects more workloads and shows higher blocking probability. However, the accepted workloads are probably finished earlier than estimation and bring the ratio of error down.

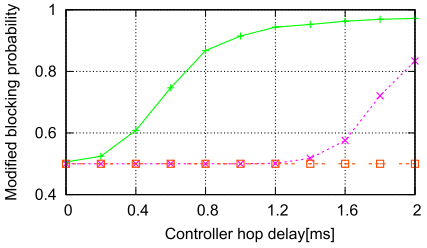
Consequently, the controller is prevented from sending too high a load to one node and provides precise decision.

As mentioned, the rejections result from two constraints. We then classified the causes of rejection into three kinds: 1 due to Eq. (3), 2) due to Eq. (4), 3) and due to both. To further address how the permissive system provides a better solution than the strict system, we show the blocking probability with respect to the causes of rejected workloads in Fig. 7. The permissive system, of course, rejects a workload only because of Eq. (3). The strict system, on the other hand, has all three kinds of rejection. Fig. 7 shows that the majority of rejection in the strict system came from Eq. (4) when the controller interval was extended. Because the permissive system ignores Eq. (4), it can accept more workloads than the strict system and instead induces decision error. On the other hand, there was a decrease in the blocking probability of the strict system due to Eq. (3). This was because the edge nodes blocked so many workloads because of Eq. (4) that they had fewer existing workloads and denoted a short time for a new workload by Eq. (3).

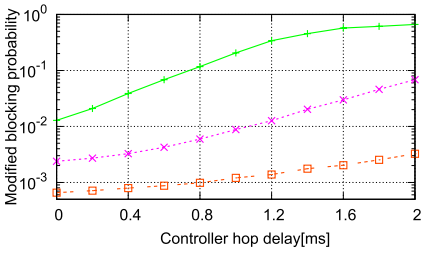
Although the longer controller interval can reduce the decision error in the permissive system, the blocking probability worsens. Furthermore, the modified blocking probability shows that the true performance of the permissive system is still better than that of the strict system (Fig. 6). However, if a service needs zero errors, the strict system should be considered, even though its blocking probability is so high. Additionally, the controller interval creates some overhead in the network due to probe messages or information packets. More consideration is needed to determine the proper interval and achieve balanced performance.



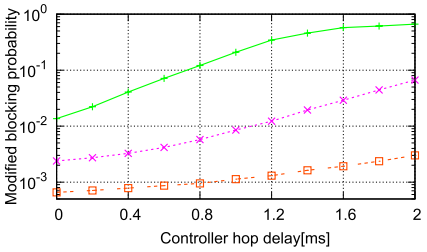
(a) Strict system with interval σ = 0.5 ms



(b) Strict system with interval σ = 1 ms



(c) Permissive system with interval σ = 0.5 ms



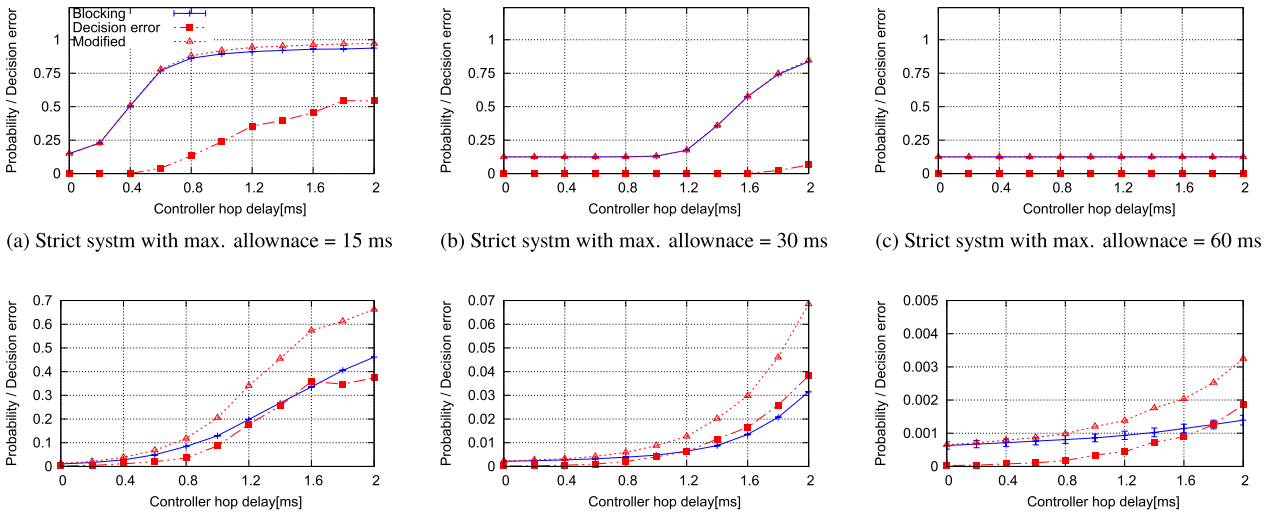
(d) Permissive system with interval σ = 1 ms

Fig. 8 Impact of controller hop delay on modified blocking probability, when sources (N) = 10.

5.2.2 Controller Communication Latency

We studied the controller communication latency by determining hop delay. Here, we ran simulations for maximum allowances of 15, 30, and 60 ms and increased the controller hop delay ($l_{i,C}$) from 0 to 2 ms with fixing the transmission delay to 0. We used the lowest latency policy to select an edge node and set 10 nodes as sources.

The numerical results in Fig. 8 show the impact of the controller hop delay on the modified blocking probability for different controller intervals, i.e., 0.5 and 1 ms. Because the strict system made decision errors, we calculated the modified blocking probability for both systems.



(a) Strict systm with max. allownace = 15 ms (b) Strict systm with max. allownace = 30 ms (c) Strict systm with max. allownace = 60 ms

(d) Permissive systm with max. allownace = 15 ms (e) Permissive systm with max. allownace = 30 ms (f) Permissive systm with max. allownace = 60 ms

Fig. 9 Impact of controller hop delay in different max. allowance, when sources (N) = 10 and interval (σ) = 0.5 ms.

Clearly, a longer controller hop delay increased the modified blocking probability in both systems. The controller propagation latency makes the controller serve an out-of-date decision to sources with some delay, so the source nodes have to wait for a while to start sending a workload to the designated edge node. This additional latency in the communication with the controller makes the total latency of each workload longer and could lead more rejections in the system. We observed such degradation in both systems when we added the hop delay of the controller.

The permissive system (Figs. 8(c), 8(d)) still performed better than the strict system (Figs. 8(a), 8(b)) on the whole, even though we added the communication latency between the sources and the controller. Without the controller latency, the permissive system had a very low blocking probability. When we increased the communication latency of the controller, the performance naturally deteriorated, but not much as it did in the strict system. This shows that the permissive system was tolerant to the impact of controller communication latency.

In the strict system (Figs. 8(a), 8(b)), the modified blocking stayed relatively constant when the allowable latency $\theta = 60$ ms but there is increment when $\theta = 15, 30$ ms caused by decision errors and the impact of the controller communication latency, which we will analyze in detail later.

Figures 8(c) and 8(d), the permissive system performance, show that a lower allowable latency clearly gave a higher modified blocking probability, but the tendency of the maximum allowance was the same.

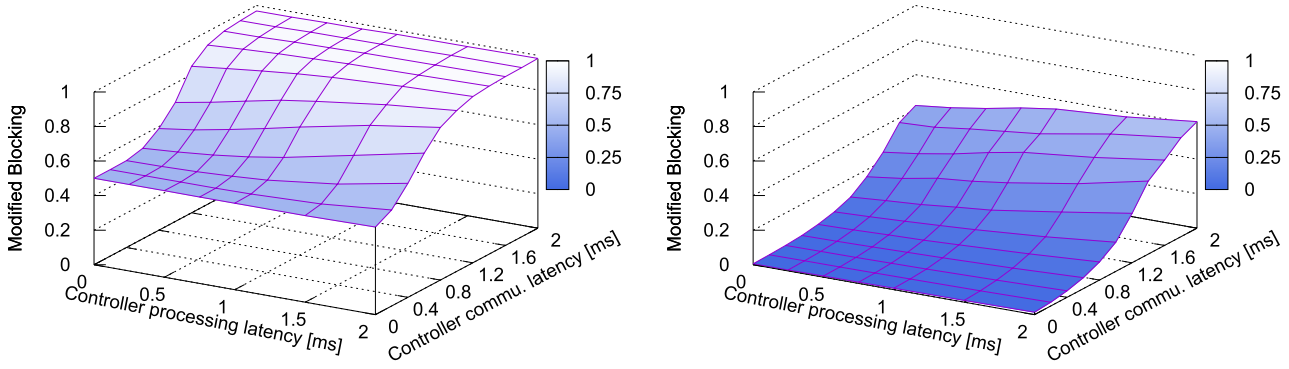
Figure 8 shows only the modified blocking probability. Analyzing the controller communication latency requires consideration of the blocking probability and of decision errors in the individual maximum allowance. Each panel of Fig. 9 shows the impact of the controller hop delay on the normal/modified blocking probability and decision error. We drew the probability and error on the same axes and with the same scale. The number of source nodes was 10, the controller interval was 0.5 ms, and the lowest latency policy was used.

Figures 9(a), 9(b) and 9(c) show the results for the strict system for maximum latencies of 15, 30 and 60 ms. Decision errors are apparent because, as mentioned in Sect. 4, underestimations occur when the controller uses out-of-date edge statuses. When we added more communication latency at the controller, more out-of-date decisions were delivered to source nodes, and this led to higher error rates.

Looking over the results for different maximum latencies, the higher allowances led to lower blocking rates and fewer errors. The high allowable latency endured the effect of underestimation because when the total latency was longer than expected, it did not exceed the allowance and was not considered to be a decision error. The results were quite stable in some part of Fig. 9(b) and in all of Fig. 9(c). Thus, if the maximum allowance is high enough, the system can maintain its performance despite communication latency of the controller, but once the allowance becomes too low, communicating with the controller will cause errors and degrade the system.

Figures 9(d), 9(e), and 9(e) show the results of the permissive system for maximum latencies of 15, 30 and 60 ms with different scales on the y axis. Obviously, the higher latency allowance gave better quality results for every metric. Although there were still decision errors even for the high allowable latency, the permissive system impressively outperformed the strict system.

The controller communication latency determined by the hop delay represents its distance from the source node. The above discussion indicates that the system performs well if we put the controller closer to the source. However, we should also think of the maximum allowance of latency to



(a) Modified blocking probability of strict systm

(b) Modified blocking probability of permissive system

Fig. 10 Impact of controller processing delay, when sources (N) = 10, interval (σ) = 1 ms, and max. latency (θ) = 20 ms.

achieve an acceptable level of system efficiency.

5.2.3 Controller Processing Latency

The processing latency in the controller is a time to receive an inquiry message and send back a results into the network. If the controller resources are enough to compute the controller process, the latency will be very short. In contrast, the limited resources requires more time to finish the contriller process. The complexity of controller process also affects this latency because a more complex process consumes more resources. This latency could the significant factor that we should concern when using the controller.

In the simulation, we ran the controller processing latency varied from 0 to 2 ms. We compared its impact with the controller communication latency as hop delay. We fix the number of source/egde nodes to 10, the controller interval to 1 ms, and the allowance latency to 20 ms. We ran two simulations with the strict and permissive systems using only the lowest latency policy.

The results is shown in Fig. 10 with 3D graph. Since we found the decision error in both systems like Sect. 5.2.2, we showed the modified blocking probability in the vertical axi. The horizontal axises are the controller processing and communication latency. Fig. 10(a) presents the impact of the controller processing latency in the strict system. The result shows that longer processing latency brought more blocking probability to the system. We could see clearer impact when we add more communication latency (see Fig. 10(a), communication latency = 0.8 ms). However, its effect was so less when compared to the communication latency. In the strict system, the controller processing latency could affect the blocking probability around ±0.1 from the ideal case but if we increase hop delay, it raised the blocking from 0.5 to 1. This is because the communication latency is a delay from the distance. It can be expanded to multiple hops regrading the source location. The processing latency is a delay inside the controller, so it is a fixed latency added to the network.

According to Fig. 10(b), the permissive system still provided the better performance than the strict system. It showed the same impact as the strict system does but within the smaller range of blocking probability.

Finally, the controller processing latency had some impact to the MEC system but less than the that of controller communication latency.

6. Conclusion

MEC is essential for the low-latency architecture of 5G services. It can provide very short communication path from users to edge servers. A practical system, however, also has to consider computational latency. In addition, MEC requires a more practical model to provide guidance regarding its design before implementation.

Consequently, we have defined a mathematical model of MEC with a controller that controls any flow in the system. The controller is responsible for selecting one of the edge servers that satisfy the system constraints. The simulation results show that the permissive underestimation system, which selects the destination with the lowest latency policy, provides an impressive MEC solution, although this system would result in some decision errors. When we include the controller communication latency in the system, the permissive system still has the low blocking probability while the strict system is affected to get high blocking probability.

This study leaves some questions open such as the complexity and scalability of the controller. If the controller creates the extra delay due to a heavy load, it may also introduce the extra delay for workloads. We therefore plan to analyze the impact of the internal delay and of the extra delay caused by heavy load of the controller.

Acknowledgments

This was supported in part by JSPS KAKENHI Grant-in-Aid for Scientific Research (B) Number 16H02806.

References

[1] K. Intharawijitr, K. Iida, H. Koga, and K. Yamaoka, “Practical en-

hancement and evaluation of a low-latency network model using mobile edge computing," Proc. IEEE 41st Annual Computer Software and Applications Conf. (COMPSAC2017), pp.567–574, Turin, Italy, July 2017. DOI: 10.1109/COMPSAC.2017.190
[2] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5G-enabled tactile internet," IEEE J. Sel. Areas Commun., vol.34, no.3, pp.460–473, March 2016. DOI: 10.1109/JSAC.2016.2525398
[3] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges," IEEE Commun. Surveys Tuts., vol.16, no.1, pp.337–368, July 2014. DOI: 10.1109/SURV.2013.070813.00285
[4] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," IEEE/ACM Trans. Netw., vol.24, no.5, pp.2795–2808, Oct. 2016. DOI: 10.1109/TNET.2015.2487344
[5] K. Bhardwaj, S. Sreepathy, A. Gavrilovska, and K. Schwan, "ECC: Edge cloud composites," Proc. IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud2014), pp.38–47, Oxford, UK, April 2014. DOI: 10.1109/MobileCloud.2014.18
[6] ETSI, "Mobile edge computing (MEC); technical requirements," Technical Report, MEC-002TechReq, ETSI, https://www.etsi.org/deliver/etsi_gs/MEC/001_099/002/01.01.01_60/gs_MEC002v010101p.pdf, accessed March 2017.
[7] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," IEEE Access, vol.5, pp.6757–6779, March 2017. DOI: 10.1109/ACCESS.2017.2685434
[8] C. Mouradian, D. Naboulsi, S. Yangui, R.H. Glitho, M.J. Morrow, and P.A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," IEEE Commun. Surveys Tuts., vol.20, no.1, pp.416–464, Firstquarter 2018. DOI: 10.1109/COMST.2017.2771153
[9] K. Intharawijitr, K. Iida, and H. Koga, "Analysis of fog model considering computing and communication latency in 5G cellular networks," Proc. IEEE International Conference on Pervasive Computing and Communication Workshops, 4 pages, Sydney, NSW, March 2016. DOI: 10.1109/PERCOMW.2016.7457059
[10] K. Intharawijitr, K. Iida, and H. Koga, "Simulation study of low latency network architecture using mobile edge computing," IEICE Trans. Inf & Syst., vol.E100-D, no.5, pp.963–972, May 2017. DOI: 10.1587/transinf.2016NTP0003
[11] ETSI MEC ISG, "Mobile edge computing (MEC); Framework and reference architecture," ETSI, GS MEC 003, https://www.etsi.org/deliver/etsi_gs/mec/001_099/003/01.01.01_60/gs_mec003v010101p.pdf, March 2016.
[12] V. Sciancalepore, F. Giust, K. Samdanis, and Z. Yousaf, "A double-tier MEC-NFV architecture: Design and optimisation," Proc. IEEE Conf. Standards for Communications and Networking (CSCN), 6 pages, Berlin, Germany, Nov. 2016. DOI: 10.1109/CSCN.2016.7785157
[13] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," Proc. ACM Workshop on Mobile Big Data (Mobidata2015), pp.37–42, Hangzhou, China, June 2015. DOI: 10.1145/2757384.2757397
[14] 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, March 2018. DOI: 10.1109/JSAC.2018.2815360
[15] V. Chamola, C. Tham, and G.S.S. Chalapathi, "Latency aware mobile task assignment and load balancing for edge cloudlets," Proc. IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp.587–592, HI, USA, March 2017. DOI: 10.1109/PERCOMW.2017.7917628
[16] J. Liu, Y. Mao, J. Zhang, and K.B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems" Proc. IEEE International Symposium on Information Theory (ISIT), pp.1451–1455, Barcelona, Spain, July 2016. DOI: 10.1109/ISIT.2016.7541539
[17] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," Proc. IEEE Conference on Computer Communications, pp.37–45, HI, USA, April 2018. DOI: 10.1109/INFOCOM.2018.8486305
[18] D. Satria, D. Park, and M. Jo, "Recovery for overloaded mobile edge computing," Future Generation Computer Systems, vol.70, pp.198–147, May 2017. DOI: 10.1016/j.future.2016.06.024
[19] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things," IEEE Trans. Cloud Comput., vol.6, no.1, pp.46–59, Jan.–March 2018. DOI: 10.1109/TCC.2015.2485206
[20] R. Deng, R. Lu, C. Lai, and T.H. Luan, "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing," Proc. IEEE International Conference on Communications (ICC2015), pp.3909–3914, London, UK, June 2015. DOI: 10.1109/ICC.2015.7248934
[21] S.F. Yashkov, "Processor-sharing queues: Some progress in analysis," Queuing Syst., vol.2, no.1, pp. 1–17, March 1987.
[22] Mathworld, "Poisson Process," Mathworld.wolfram.com, 2016, http://mathworld.wolfram.com/PoissonProcess.html, accessed: 10 Oct. 2016.
[23] H.M. Vin, P. Goyal, and A. Goyal, "A statistical admission control algorithm for multimedia servers," Proc. the second ACM international conference on Multimedia, pp.33–40, San Francisco, USA, Oct. 1994. DOI: 10.1145/192593.192616
[24] B.C. Dean, M.X. Goemans, and J. Vondrak, "Approximating the stochastic knapsack problem: The benefit of adaptivity," Mathematics of Operations Research, vol.33, no.4, pp.945–964, Nov. 2008. DOI: 10.1109/FOCS.2004.15
[25] G.P. Fettweis, "The tactile internet: Applications and challenges," IEEE Veh. Technol. Mag., vol.9, no.1, pp.64–70, March 2014. DOI: 10.1109/MVT.2013.2295069

Krittin Intharawijitr received the B.E., M.E. degrees in Computer Engineering from Chulalongkorn University, Bangkok, Thailand in 2013, Communications and Computer Engineering from Tokyo Institute of Technology, Tokyo, Japan in 2016 respectively. Presently, he is a Doctoral course student at Tokyo Institute of Technology, Japan. His research interests lie in the fields of network architecture, mobile networks, and cloud computing.

Katsuyoshi Iida received the B.E., M.E. and Ph.D. degrees in Computer Science and Systems Engineering from Kyushu Institute of Technology (KIT), Iizuka, Japan in 1996, in Information Science from Nara Institute of Science and Technology, Ikoma, Japan in 1998, and in Computer Science and Systems Engineering from KIT in 2001, respectively. Currently, he is an Associate Professor in the Information Initiative Center, Hokkaido University, Sapporo, Japan. His research interests include network systems engineering such as network architecture, performance evaluation, QoS, and mobile networks. He is a member of the WIDE project and IEEE. He received the 18th TELECOM System Technology Award, and Tokyo Tech young researcher's award in 2003, and 2010, respectively.

Hiroyuki Koga received the B.E., M.E. and D.E. degrees in computer science and electronics from Kyushu Institute of Technology, Japan, in 1998, 2000, and 2003, respectively. From 2003 to 2004, he was a postdoctoral researcher in the Graduate School of Information Science, Nara Institute of Science and Technology. From 2004 to 2006, he was a researcher in the Kitakyushu JGN2 Research Center, National Institute of Information and Communications Technology. From 2006 to 2009, he was an assistant professor in the Department of Information and Media Engineering, Faculty of Environmental Engineering, University of Kitakyushu, and then has been an associate professor in the same department since April 2009. His research interests include performance evaluation of computer networks, mobile networks, and communication protocols. He is a member of the ACM and IEEE.

Katsunori Yamaoka received the B.E., M.E., and Ph.D. degrees from the Tokyo Institute of Technology in 1991, 1993 and 2000, respectively. He left Ph.D program in 1994 and joined the Tokyo Institute of Technology as an assistant professor at that time. In 2000, he joined the National Institute of Multimedia Education (NIME) in Japan as an associate professor. Since 2001, he has been an associate professor at the Tokyo Institute of Technology. He has also been a visiting associate professor of the National Institute of Informatics (NII) in Japan since 2004. His research interests are network QoS control for multimedia communications.