Service Placement for Real-Time Applications: Rate-Adaptation and Load-Balancing at the Network Edge

Saadallah Kassir* and Gustavo de Veciana* Nannan Wang†, Xi Wang†, Paparao Palacharla†
* Electrical and Computer Engineering Department, The University of Texas at Austin, TX, USA.
† Fujitsu Laboratories of America, Richardson, TX, USA.
Email: {skassir, deveciana}@utexas.edu, {nannan.wang, xi.wang, paparao.palacharla}@fujitsu.com

***Abstract*—Mobile Edge Computing may become a prevalent platform to support applications where mobile devices have limited compute, storage, energy and/or data privacy concerns. In this paper, we study the efficient provisioning and management of compute resources in the *Edge-to-Cloud continuum* for different types of real-time applications with timeliness requirements depending on application-level update rates and communication/compute delays. We begin by introducing a highly stylized network model allowing us to study the salient features of this problem including its sensitivity to compute vs. communication costs, application requirements, and traffic load variability. We then propose an online decentralized service placement algorithm, based on estimating network delays and adapting application update rates, which achieves high service availability. Our results exhibit how placement can be optimized and how a load-balancing strategy can achieve near-optimal service availability in large networks.**

***Index Terms*—Edge Computing, Fog Network Dimensioning, Rate Adaptation, Service Placement, Real-Time Applications**

I. Introduction

Many of the emerging mobile applications require unprecedented compute power, e.g., autonomous vehicles, remotely controlled robots, Augmented Reality (AR) technologies, unmanned aerial vehicles, cloud gaming platforms, etc. Equipping mobile devices with the compute resources needed can be a considerable challenge for manufacturers due to cost, complexity, battery longevity, weight, and size constraints. A solution to overcome this challenge and bring to market such computation-hungry services is to (partially) offload compute to the cloud via wireless connectivity to remote servers.

A flexible approach to support mobile devices with remote compute resources is through a server-side process running on a Virtual Machine (VM). If kept up-to-date, the process can keep track of a device's state in real-time, perform computations, and possibly send back control commands. Such processes are expected to become prevalent to support the management and control of mobile devices, e.g., robots, self-driving cars, smart cities devices [1]. However, in real-time settings, associating remote processes to devices poses several technical challenges. In particular, in order to maintain safety or offer an appropriate Quality of Service (QoS), the process needs to closely track the state of its device. In other words, updates among mobile device and server-side processes should not have "aged" too much to remain relevant. Maintaining such timeliness depends both on the update rate as well as communication/compute delays.

To support possibly stringent timeliness requirements, edge computing architectures have been proposed as means to reduce network delays, by moving the servers closer to the devices. By contrast, the alternative of hosting VMs in the cloud, typically further away from the devices, provides an attractive solution leveraging large pools of shared resources. Deploying mobile services at scale will require careful study of cost/performance tradeoffs of edge/cloud infrastructure based solutions.

Contributions. In this paper, we first explore the fundamental characteristics of provisioning edge/cloud compute resources for real-time mobile services. To that end we propose a stylized network model allowing us to capture the salient features of the network dimensioning problem. Based on the initial insights developed from studying the resource provisioning problem, we propose an online, adaptive and distributed joint service-placement and rate-adaptation policy that is more generally applicable, and that describes how the network is ought to be managed while operating.

The framework introduced in this paper allows us to reach multiple conclusions. First, we identify key tradeoffs between cloud and edge computing, and show how the optimal provisioning and placement depend on the application's characteristics. Second, we show how the relative cost of compute vs. communication impacts the optimal location of compute resources. In particular the most cost effective placement may not be in the cloud or at the edge, but rather at an intermediate level. Third, we show that for any use-case, as the density of mobile devices grows placing compute resources at the edge becomes more cost effective. However, perhaps counter-intuitively, stricter timeliness guarantees makes it beneficial to shift compute resources further away in the cloud. Finally, we introduce a device-side online distributed algorithm to manage dynamic mobile device loads by determining both the device's update rate and a server to host its VM. Our approach adapts its decisions to measured network congestion, which may not be under service provider's control. We show that under the proposed joint placement and rate-adaptation policy, near-optimal service availability can be achieved in large networks, and show the benefit over load balancing policies when applications choose fixed update rates.

Related Work. There has been substantial work in this area. We identify two relevant classes of work. The first class focuses on the need for mobile edge computing. The natural way to introduce the concept is to compare the characteristics of edge and cloud computing, as in [2]–[4]. In this paper, we take this one step further by characterizing precisely the tradeoffs for real-time applications. Additionally, we propose an intuitive hierarchical network model materializing the idea of "Cloud-to-Thing continuum", or Fog-to-Cloud, suggested in [3] and [5], where service providers can place compute resources anywhere in the network. This softens the dichotomy between edge and cloud, leading naturally to the optimal placement problem.

The second line of work focuses on service placement, i.e., where to instantiate VMs once a provider has dimensioned a graph of compute resources, e.g., [6]–[12]. These works propose various policies to optimize placement based on different performance metrics. For instance, [9] considers power consumption and transmission delay, [10] examines the number of services placed, [11] focuses on minimizing the violation in QoS, i.e., latency, while [12] uses user-specific reward functions. Other studies suggest approximation or genetic algorithms to solve the service placement problem. Furthermore, [13] suggests to solve a Mixed Integer Linear Program to minimize capital and operating expenditures to dimension the network, but the authors to not analyze the communication vs. compute tradeoff explicitly, and do not address heterogeneity in the device requirements (e.g., latency constraints and compute job size). In this work, we follow a different approach. We simplify the network model which allows us to extract basic insights, that we leverage to propose a more general service placement algorithm. Unlike the above-mentioned work, we propose a service placement policy that addresses the need to adapt to network congestion by adapting the update rates associated with mobile devices supporting real-time applications.

Paper Organization. This paper is organized as follows. Section II describes four mobile applications serving as running examples throughout the paper. Section III proposes a highly stylized system model and network architecture, as well as an appropriate timeliness metric. Section IV includes our problem formulation and result analysis. In Section V, we study a more general setting, and analyze the performance of our online device-side joint service placement and rate-adaptation algorithm. We conclude the paper in Section VI.

II. Mobile Edge Computing Services: Use Cases

Our work is motivated by several emerging applications/use cases including those developed in the context of 5G networks [23], [24]; specifically we focus on four types of applications:

- **XR Traffic:** Augmented Reality, Virtual Reality and Mixed Reality, generally referred to as extended reality (XR), have been the subject of extensive study in industry and academia as it is considered one of the innovative services to be supported by next generation wireless networks. XR devices have the particularity of requiring both considerable bandwidth and low latency, making the design of networks supporting such services challenging [25]–[27].
- **Vehicular Network Traffic:** Supporting self-driving and/or coordination amongst next generation vehicles may be based on exchanging basic safety messages or localization data. To be relevant, update messages will typically require tight timeliness constraints, but may require relatively little compute and communication resources.
- **Cloud Gaming Traffic:** In the near term cloud gaming may become the leading use-case. It has the potential to reduce the compute requirements on the gaming devices by performing computations in a remote server, enabling complex multiplayer games to be more accessible on-demand. Similarly to XR traffic, considerable data may be streamed from the server to the devices to enable high-quality graphics, but timeliness constraints may be looser.
- **IoT Device Traffic:** We shall also consider IoT devices that do not have strict and tight latency budgets, but that can potentially be massively deployed, e.g. smart home devices, or agricultural sensor networks. Typical traffic for such use cases consists of short and sporadic packets.

We summarize the requirements for these use-cases in Table I.

III. System Model

In this section, we introduce a network architecture and performance metrics that we use to explore the characteristics and tradeoffs associated with service placement and provisioning decisions for real-time applications.

A. Network Model

We shall initially take the perspective of a *(virtual) service provider* who pays for communication and compute resources from one or more *infrastructure providers*. Initially we assume that the service provider provides custom services to a homogeneous customer base of *mobile devices*, i.e., with the same application requirements.

We consider a setting where the network resources lie on a binary tree where compute resources could be made available on any node, while the edges correspond to communication links carrying traffic between compute nodes and mobile devices, see Figure 1. The root of the tree is at height h_c and will be interpreted to correspond to a cloud compute service provider, while the leaves at height 1 will be viewed as edge

TABLE I: Network Requirements and Parameters per Use-Case

Use Case	Timeliness Constraint (τ_0, in ms)	Devices per BS (η, in devices/BS)	UL/DL Update Size (p_u, p_d, in kB/update)		Compute per update (ψ^{-1}, in Ops/update)	References
XR	15	10	50	50	1e9	[14]–[16]
Vehicular Networks	10	40	0.4	0.4	1e7	[16]–[19]
Cloud Gaming	100	5	5	100	1e10	[16], [20], [21]
IoT	10,000	500	0.2	0	1e4	[16], [22]

compute nodes co-located with cellular Base Stations (BS). Meanwhile intermediate levels are introduced to study the potential benefits of placing compute resources between the two extremes.

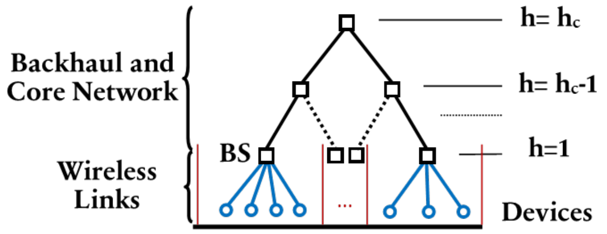


Fig. 1: Tree Topology Model

We model mobile device service requests as arriving at each BS as a Poisson Point Process (PPP) with intensity λ. Each request corresponds to a server-side process running on a VM hosted in a compute node, and has a random duration of mean μ^{-1} seconds. Hence, if sufficient resources were provisioned the number of active mobile devices at each BS, illustrated in blue in Figure 1, follows a Poisson distribution with mean $\eta = \lambda/\mu$ devices, corresponding to the stationary distribution of an $M/GI/\infty$ queue. However, if limited resources are provisioned mobile devices may experience blocking. In particular, suppose the service provider provisions sufficient compute resources to k simultaneous sessions at each node resources at level h of the tree. Assuming requests at all leaf nodes are served by the parent node at level h, the total offered mean load on such a node will be $\eta 2^{h-1}$ and the blocking probability probability ϵ is given by the Erlang function $E(\eta 2^{h-1}, k)$ associated with an $M/GI/k/k$ queue. The service availability, i.e., $1 - \epsilon$, thus depends not only on the resources provisioned k but also on the level h at which they are located – more on this later.

When a session is active, we assume the device sends updates of size p_u bits at a fixed rate ρ updates/sec. to its associated process, which in turn performs a fixed number of operations ψ^{-1} and may send back an update of size p_d bits to the device – see Figure 2. As discussed below, we consider applications where these tasks must be performed in a timely manner.

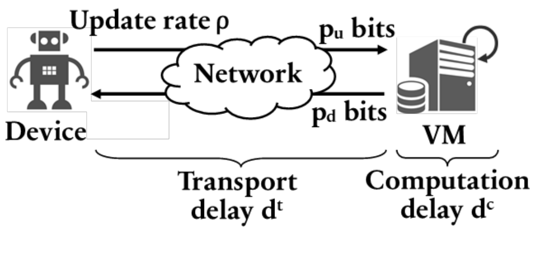


Fig. 2: Interaction between a device and its server-side process.

We assume for now that the devices supported by the service provider share the same application and their VMs are hosted at the same level in the tree. Section V revisits this assumption by examining a general service and network model.

B. Delay Models

To achieve timely service, three key metrics need to be considered: (1) the device update rate ρ affecting the amount of compute needed and the load on communication links, (2) the transport delay d^t experienced by updates depending on the level the VM is placed; and (3) the compute delay d^c that depending on the amount of compute resources allocated to a VM.

To get at the main characteristics of such systems we shall consider a simple delay model. If compute resources are placed at level h the round-trip transport delay $d^{\mathrm{t}}(h)$ is given by:

$$d^{\mathrm{t}}(h) = \frac{p_u + p_d}{l} + 2\phi h \quad \text{sec.,} \tag{1}$$

where l is the bottleneck link capacity, likely the wireless link, and ϕ is a constant forwarding delay per hop to the compute resources. The first term captures the overall transmission delay of a file while the second term captures the forwarding delay. This idealized model assumes that the service providers have access to uncongested links with minimal queuing from their infrastructure provider, i.e., by over-provisioning their communication resources, see [28], or prioritizing such traffic.

Meanwhile, the delay to process an update depends on application specific tasks such as database look-ups, GPS-coordinates processing, video frames rendering, etc. Given that an update requires ψ^{-1} operations and assuming perfect parallelism, c CPU cores each able to deliver ν operations/sec would complete the update task in

$$d^{\mathrm{c}}(c) = \frac{1}{c\psi\nu} \quad \text{sec.} \tag{2}$$

Note that we will allow c take fractional values.

C. Timeliness Metric

In this paper we adopt an end-to-end timeliness metric based on the *Age-of-Information* (AoI), see e.g., [29]–[31]. The key difference with traditional end-to-end or round-trip delay, is capturing the difference between the current time (at the mobile device) and the time at which the server has completed processing the last update, i.e., acted upon and possibly delivered back to the mobile device. Hence, the AoI requires factoring both the update rate and compute/communication delays, and captures the tension between these two variables. For instance, if the device's update rate is low, then the remote process may often be out of sync even if the transport and compute delays are low. Conversely, if the update rate is high, but updates experience large delays, the server-side process decisions would be outdated most of the time.

Several works have studied ways of characterizing the AoI in multi-user settings, see e.g., [32]–[34]. For the most part, they use variations of Theorem 3 in [32] which captures the AoI for a specific device. For simplicity we shall use a natural variant of this timeliness metric τ, given by:

$$\tau = \frac{1}{2\rho} + d^{\mathrm{t}}(h) + d^{\mathrm{c}}(c) \tag{3}$$

As can be seen, low delays and high update rates improve timeliness. Further support for this performance metric can be found in Appendix A. We denote by τ_0 the application specific timeliness constraint, i.e., resources need to be provisioned so as to ensure $\tau \leq \tau_0$ for the devices subscribed to the service.

Putting Equations 1 and 3 together, one can observe the dependence of the timeliness τ on the device update rate ρ and the level h at which the service provider rents/places its compute resources. It is clear that both $d^{\text{t}}(h)$ and τ increase with h. However, fixing a timeliness constraint τ_0 forces ρ to increase with h to compensate for the additional delay, increasing the compute resources required at the compute node side to process the additional updates. Therefore, one can distinguish two clear tradeoffs, one between the VM level and timeliness, the other between the VM level and compute resources.

IV. Problem Formulation and Results

The service provisioning problem reduces to determining (1) the optimal level h^* at which to place the VMs, (2) the required number of cores c^* per VM, (3) the minimum number of VMs k^* that can be hosted per compute node, as well as (4) the minimum device update rate ρ^*, that will dictate the amount of traffic, i.e., the communication cost, on the links below level h^*.

A. Problem Formulation

Given the simple system model proposed in the previous section, the service provider's cost $\mathcal{C}_P$ in the network resource provisioning phase can be approximated as the sum of the communication and compute cost:

$$\mathcal{C}_P(\rho, c, k, h) = \theta^{\text{t}} \eta 2^{h_c - 1} h (p_u + p_d) \rho + \theta^{\text{c}} 2^{h_c - h} k c \quad (4)$$

where θ^{t} is the communication cost (in \$/Mbps/hop/link) and θ^{c} is the compute cost (in \$/core). Note that $\eta 2^{h_c - 1}$ is the mean number of devices in the network assuming high availability (no blocking) and $h(p_u + p_d)\rho$ is the mean load $\times$ links per device if compute is placed at level h. Meanwhile $2^{h_c - h}$ is the number of nodes at level h where compute resources are placed and kc is number of cores per node if each VM requires c cores. Now given a timeliness constraint τ_0 and an availability requirement, i.e., blocking probability ϵ, one can characterize the minimum cost level at which to dimension the network in four steps.

First, given Equation 3 and the timeliness constraint τ_0 one can determine the smallest feasible update rate, when compute resources are placed at level h, i.e., that with the smallest communication/compute cost. Specifically, to ensure no queuing at the VM we need $d^c \leq \frac{1}{\rho}$ so setting this to equality we get:

$$\rho(h) = \frac{3/2}{\tau_0 - d^{\text{t}}(h)}. \quad (5)$$

Second, given the compute delay constraint, the number of CPU cores allocated per VM can be found from Equation 2:

$$c(h) = \frac{\rho(h)}{\psi \nu}. \quad (6)$$

Third, the number of VMs $k(h)$ compute nodes at level h would need to support to limit blocking to ϵ can be obtained via the Erlang-B formula, see [35], i.e., solving $\epsilon = E(\eta 2^{h-1}, k(h))$, where $\eta 2^{h-1}$ is the mean load such nodes would see. Note service providers benefit from statistical multiplexing gains when compute resources are placed higher in the tree, allowing increased aggregation of traffic on shared resources. As h increases, the compute load variability per node decreases, hence placing compute at the top of the tree, i.e., in the cloud, reduces the need for slack compute resources in order to ensure high availability, and thus reduces compute costs.

Finally, the optimal service level can be trivially found by evaluating $h^* = \arg\min_{h \in \{1, \cdots, h_c\}} \mathcal{C}_P(\rho(h), c(h), k(h), h)$. We can then obtain $\rho^* = \rho(h^*)$, $c^* = c(h^*)$ and $k^* = k(h^*)$. This is tractable since h_c is reasonably small.

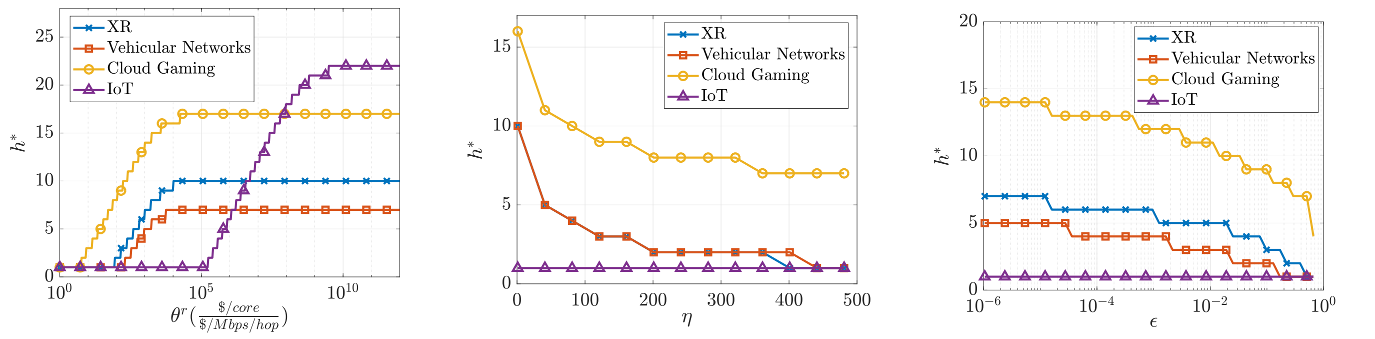
B. Analysis of Results

We now present and analyze our results for different application parameters shown in Table I. We analyze successively the effect of the relative costs of compute and communication, density of devices, and service availability constraint.

1) Effect of Compute and Communication Costs: Figure 3a shows the optimal VM level h^* as a function of the relative cost of compute to communication $\theta^{\text{r}} = \theta^{\text{c}}/\theta^{\text{t}}$. Clearly, as compute becomes more expensive with respect to communication, it is preferable for compute resources to be placed higher up in the tree so as to benefit from statistical multiplexing. It is also worth noticing that the optimal VM placement depends on the use-case. While the optimal decision for all applications would be to place compute resources as close to the mobile devices as possible, i.e., at the edge, when compute is cheap, each use case has a different behavior as θ^{r} grows. For large θ^{r}, the cost effective placement for each use-case is as far from the edge as possible. The highest level possible for large θ^{r} is dictated by the application's timeliness constraint. More specifically, the looser τ_0, the larger h^* is for large θ^{r}. Any value of h larger than this level would be infeasible, as the transport delay $d^{\text{t}}(h)$ would exceed the timeliness requirement τ_0, for any ρ.

2) Effect of Device Density: As discussed earlier, having multiple VMs share compute resources leads to statistical multiplexing gains. Resource pooling can either be achieved by placing compute resources higher up in the tree, or by increasing the device load per BS. Therefore, as η grows, we expect to need fewer compute resources per unit demand, pulling the optimal service level down closer to the edge. This is indeed what is exhibited in Figure 3b, where θ^{r} has been estimated based on realistic compute and communication cost values [36], [37].

3) Effect of Service Availability Requirement: The network is dimensioned so as to guarantee an availability of $1 - \epsilon$. This naturally leads to higher dimensioning cost versus mean load provisioning, as slack resources will need to be allocated to address load variations. In fact, the smaller ϵ is, the more compute resources will need to be provisioned to ensure the



(a) Effect of the relative cost of compute to communication θ^{r}; $\epsilon = 1 \times 10^{-5}$.

(b) Effect of the number of devices η attached to each BS, $\theta^{\mathrm{r}} = 1 \times 10^3$; $\epsilon = 1 \times 10^{-5}$.

(c) Effect of the blocking probability ϵ, $\theta^{\mathrm{r}} = 1 \times 10^3$.

Fig. 3: Optimal service level h^*, for $h_c = 25$, $\phi = 100\mu$s/hop, $\nu = 30 \times 10^3$ MIPS/core, $l = 1$ Gbps.

desired service availability level is met. Since more compute resources need to be reserved, the most cost effective strategy is to place compute resources higher in the tree. Figure 3c illustrates this trend, showing that one can afford to place the resources closer to the edge under relaxed constraints.

V. Online Service Placement of Heterogeneous Traffic in the Fog

So far, we have tackled the problem of service placement and dimensioning for real-time applications on mobile devices. The analysis presented in Section IV was based on a simple network topology, delay models and timeliness constraints. These assumptions were necessary to abstract what in practice is quite a complex system. In this section, we explore the design of an algorithm that jointly adapts devices' update rates and VM placement in a heterogeneous network, using delay measurements instead of model-based predictions. We assume a service provider has already provisioned resources on a general network topology and consider the case where mobile devices running different applications are co-hosted on shared resources. We emphasize that the problem addressed in the previous sections is a joint network dimensioning and service placement problem faced by the service provider, while the one presented in this section is a joint network management (through device rate-adaptation) and service placement problem faced by the devices requesting the service. The former problem is hence faced during the network deployment phase, while the latter is faced when the network is operating.

A. Network Model and Algorithm Description

In our general network model we let $\mathcal{S}$ denote the set of compute nodes, where each $s \in \mathcal{S}$ has capacity κ_s. These nodes shared by mobile devices of different types where $\mathcal{A}$ denotes the set of types. Requests of Type $a \in \mathcal{A}$ arrive as a PPP of intensity λ_a, and are active for a random time with mean μ_a^{-1} seconds. The types also have potentially different compute requirements per update ψ_a^{-1} and application timeliness requirement τ_a. Note that request types capture devices' requests generated at different locations and associated with different application requirements, whence Type a requests are restricted to be served by a subset of compute nodes $\mathcal{S}_a \subseteq \mathcal{S}$.

In practice, delays experienced by updates might be roughly quasistatic or constant over time, congestion dependent, i.e., depend on previous placement decisions made by the algorithm, or vary due to exogenous traffic which is not under the service provider's control. Hence, in the sequel several metrics including network delays are denoted as depending on time.

Our proposed Algorithm 1 extends traditional Least Ratio Routing (LRR) based algorithms, see [38], to realize joint service placement and rate-adaptation along with possibly service migration. Thus it is executed when new mobile devices arrive to the network, but also subsequently if a device moves and/or observes changes in network congestion that warrant the migration of its VM to another location. As devices execute Algorithm 1 more frequently, they will be able to react to more sudden changes in the delay profile, but at the cost of more frequent compute node pings. For simplicity, we focus on the algorithm's behavior upon arrival of a new request.

Algorithm 1 Rate-Adaptive Least Ratio Routing

```
1: procedure LRR($a, t$)                              ▷ Device Type $a$, time $t$
2:     Ping/measure $d^{\mathrm{t}}_{a,s}(t)$, $r_s(t)$, $\kappa_s$, $f_s(\cdot)$, $\forall s \in \mathcal{S}_a$
3:     $\rho_{a,s}(t) = 1.5/(\tau_a - d^{\mathrm{t}}_{a,s}(t))$, $\forall s \in \mathcal{S}_a$
4:     $\Delta_{a,s}(t) = \psi_a^{-1}\rho_{a,s}(t)$, $\forall s \in \mathcal{S}_a$
5:     $u_s(t) = r_s(t)/\kappa_s$, $\forall s \in \mathcal{S}_a$
6:     $u'_{a,s}(t) = (r_s(t) + \Delta_{a,s}(t))/\kappa_s$, $\forall s \in \mathcal{S}_a$
7:     $\tilde{\mathcal{S}}_a = \{s \in \mathcal{S}_a | u'_{a,s}(t) \leq 1\}$
8:     $s^* = \arg\min_{s \in \tilde{\mathcal{S}}_a} \int_{u_s(t)}^{u'_{a,s}(t)} f_s(u)\,\mathrm{d}u$
9:     return $s^*, \rho^*_{a,s^*}(t)$
```

When a Type a device arrives at time t, it first pings the compute nodes that can potentially host its VM to *estimate* the current transport delays $d^{\mathrm{t}}_{a,s}(t)$ to all $s \in \mathcal{S}_a$. It also gathers the amount of resource $r_s(t)$ currently allocated at s. Given $d^{\mathrm{t}}_{a,s}(t)$, the device can use Equation 5 to determine the update rate $\rho_{a,s}(t)$ it would currently require if its VM was instantiated on node s while satisfying its timeliness constraint τ_a. It then deduces its compute requirements $\Delta_{a,s}(t) = \rho_{a,s}(t)\psi_a^{-1}$. The device can then determine the current utilization $u_s(t) = \frac{r_s(t)}{\kappa_s} \in [0, 1)$ and projected utilisation $u'_{a,s}(t) = \frac{r_s(t)+\Delta_{a,s}(t)}{\kappa_s}$ if the VM was placed on $s \in \mathcal{S}_a$.

The nodes that can support this request at time t are given by $\tilde{\mathcal{S}}_a = \{s \in \mathcal{S}_a | u'_{a,s}(t) \leq 1\}$. If none are available, the request is blocked. Otherwise, each compute node has a strictly increasing function $f_s : [0,1] \rightarrow \mathbb{R}_+$ which we refer to as Marginal Utilization Cost Function (MUCF) capturing the cost of using an extra compute resource unit at a given utilization. The algorithm greedily places the VM on the feasible node having the smallest marginal cost at time t, defined as the integral of the MUCF from $u_s(t)$ to $u'_{a,s}(t)$.

The MUCF can be designed with different objectives in mind. For instance, a natural objective would be to balance the compute nodes' loads. This strategy ensures that there are as much available resources as possible in all the nodes, which may help reducing the blocking rate. Different MUCFs would attempt to greedily balance the loads on the compute nodes. In fact, any convex function would achieve this goal, and the function convexity would control the extent to which the service provider wants to balance the load, at the cost of risking to consume more compute resources. In light of these observation, being proportionally fair with respect to the available resources among them is a reasonable policy, i.e., greedily maximizing the network-level utility function $\sum_{s\in\mathcal{S}} \log(1-u_s(t))$ for an arrival at time t. Theorem 1, proved in Appendix C, confirms this objective can be achieved by properly selecting the MUCF.

Theorem 1: Proportional Fairness MUCF. *Choosing $f_s(u) = \frac{1}{1-u}, u \in [0,1)$ for a compute node s is equivalent to greedily maximizing the proportional fairness utility function.*

B. Algorithm Performance Analysis

In our framework, we aim at minimizing device blockage. Since we are studying a heterogeneous system, where devices running different applications can request service from the same pool of resources, we assign a reward w_a to Type a devices per unit time spent in the network, which can represent, e.g., revenue generated by serving a Type a device.

We observe that this problem reduces to the Multiple Knapsack Problem (MKP) for fixed compute requirements Δ. This problem has been thoroughly studied in the literature and several strategies based on approximation algorithms and heuristics have been proposed to solve it [39]. In Theorem 1, we propose an MUCF that greedily balances the loads across the compute nodes to solve a variant of the MKP where the item sizes $\Delta_{a,s}$, depend on the knapsacks $s \in \mathcal{S}$.

A natural definition for the cost function of the device-centered problem $\mathcal{C}_D$ in the network operation phase is the expected rate of loss in revenues due to blockage, for a fixed Δ, defined as:

$$\mathcal{C}_D(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa}) = \sum_{a\in\mathcal{A}} w_a \lambda_a \mu_a^{-1} P(B_a; \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa}) \quad (7)$$

where $P(B_a; \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa})$ captures the probability that a typical Type a device is blocked. $\mathcal{C}_D$ can be lower-bounded by solving a relaxed MKP as stated in Theorem 2, proved in Appendix D.

Theorem 2: Lower-Bound on the Rate of Loss in Revenue. *Let A be an assignment matrix representing the mean number of Type a devices assigned to node s, such that $A \in \mathcal{B}(\boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa}) = \{A \in \mathbb{R}^{|\mathcal{A}|\times|\mathcal{S}|} | \sum_{s\in\mathcal{S}} A_{a,s} \leq \lambda_a \mu_a^{-1}, \forall a \in \mathcal{A}\ ,\ \sum_{a\in\mathcal{A}} \Delta_{a,s} A_{a,s} \leq \kappa_s, \forall s \in \mathcal{S}\}$.*

Let A^* be a feasible assignment, solution of the Linear Program relaxed Multiple Knapsack Problem (LP-MKP):

$$\text{LP-MKP}(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa}) : \qquad \max_A \sum_{a\in\mathcal{A}} \sum_{s\in\mathcal{S}} w_a A_{a,s} \quad \text{s.t. } A \in \mathcal{B}(\boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa})$$

Then, $$\underline{\mathcal{C}}_D(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa}) = \sum_{a\in\mathcal{A}} w_a (\lambda_a \mu_a^{-1} - \sum_{s\in\mathcal{S}} A^*_{a,s}) \leq \mathcal{C}_D(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa})$$

The authors in [38] discuss a special case of the suggested framework, where $w_a = 1$, $\mu_a^{-1} = 1, \forall a \in \mathcal{A}$, and $\Delta_{a,s} = 1, \forall (a,s) \in \mathcal{A} \times \mathcal{S}$. In this specific setting, $\mathcal{C}_D$ was proven to converge to $\underline{\mathcal{C}}_D$ in the fluid limit, i.e., when both the arrival rate vector λ and capacity vector κ are scaled by a large fluid-scale factor γ. In this paper we study whether our proposed local and adaptive LRR policy can asymptotically drive the value of the network-wide cost function to its theoretical lower bound in large systems in more general settings than in [38].

C. Algorithm Performance Evaluation

We now evaluate the performance of our joint service placement and rate-adaptation algorithm via simulation in the fluid-scaled network with factor γ. In these simulations, we assumed a more general underlying network model than Figure 1, depicted in Figure 4.

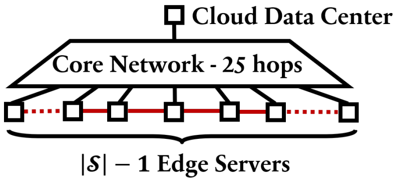


Fig. 4: General Topology Model

This network features a cloud compute node located 25 hops away from the edge, the cloud compute node having a compute capacity set to be 100 times the one of an edge node. Each of the $|\mathcal{A}|$ device types is attached to a random BS colocated with an edge compute node. A type is a collection of devices of one of the use-cases described in Section II, and having technical requirements given in Table I. Each device can place its service in its closest compute node, or any adjacent node including the cloud at the cost of higher transport delay, as modeled in Equation 1. Moreover, Type a's reward w_a is set to be proportional to ψ_a^{-1} depicting a pricing model based on the amount of compute a Type a update requires.

We compare the performance of our joint placement and rate-adaptation policy to a similar placement algorithm, i.e., using the MUCF suggested in Theorem 1, but for devices having static update rates, set such that compute nodes up to five hops away can be reached without violating their timeliness constraint.

In Figure 5, we show that the rate of loss in revenue $\mathcal{C}_D$ converges to the lower-bound $\underline{\mathcal{C}}_D$ as γ increases for the rate-adaptive algorithm, but not for the static-rate one. In this simulation, network delays are assumed to be static, and $\boldsymbol{\kappa}$ has been set so as to ensure that the total capacity in the network is larger than the expected network load.

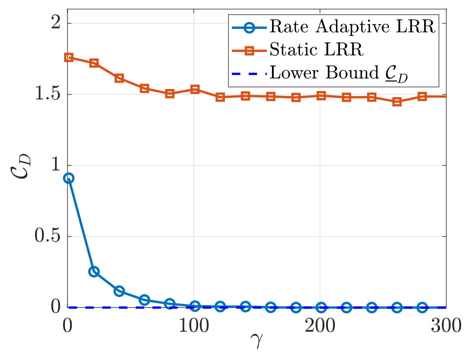


Fig. 5: Performance comparison of the rate-adaptive and static LRR algorithms in the fluid-limit and the theoretical lower-bound on the mean rate of loss in revenue; $|\mathcal{A}| = 50$, $|\mathcal{S}| = 20$.

We note that the value of the lower-bound is 0, meaning that in large-scale systems a zero-blocking regime can be achieved.

Figure 5 may, however, look different if the fixed update rate is designed differently. Figure 6 shows the value of $\mathcal{C}_D$ in the fluid limit for the static LRR algorithm as a function of the devices' rates. Note that the figure only shows the rate of XR devices, but all the devices' rates increase along with ρ_{XR}. We observe that $\mathcal{C}_D$ first drops quickly once the update rate is large enough to reach nodes two hops away, as it gives the devices the ability to balance the load among edge nodes. This effect is not as beneficial for larger rates as the VM compute requirements also increase, leading to a higher blocking rate. The cost function value then dramatically drops once the update rate allows the devices to reach the cloud node and benefit from its substantial capacity, yet without reaching zero-blocking as the rate-adaptive LRR. Finally, larger update rates are associated with larger values of $\mathcal{C}_D$ as the devices' VM requirements keep on increasing while not gaining any load-balancing opportunity.

The key takeaway is that manufacturers can design devices with slow fixed update rates, requiring little compute resources at the server side and little power at the device side, but at the cost of reducing the set of reachable nodes given the timeliness constraint, hence reducing the placement algorithm's balancing ability. Conversely, for large rates, farther nodes such as powerful cloud servers can be reached leading to better load-balancing, but devices may occupy unnecessary resources if their VMs are placed at the edge, leading to wasted compute resources and reduced service availability. Manufacturers may optimize for an optimal update rate balancing these two

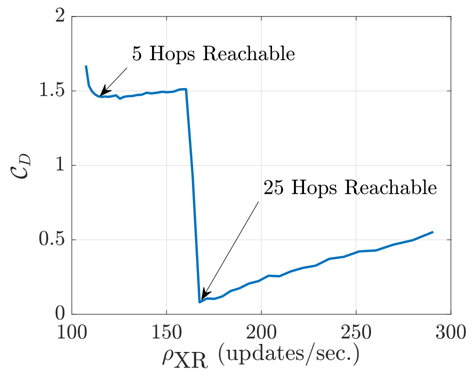


Fig. 6: Plot of the mean rate of loss in revenue as a function of the devices' fixed update rate; $|\mathcal{A}| = 50$, $|\mathcal{S}| = 20$, $\gamma = 300$.

effects, but it is unlikely to perform well in arbitrary network topologies.

Rate-adaptation allows for more flexibility in the service placement, without occupying unnecessary compute resources, explaining the better performance of the rate-adaptive LRR algorithm in Figure 5 over static rate policies.

Another major strength of Algorithm 1 is the fact that it can closely adapt to changes in network delays. Figure 7 shows the performance over time of the joint placement and rate-adaptation policy under stochastic delays and compares it to the one of the static algorithm. The delay process experienced by the devices is now assumed to depend on previous decisions taken by the algorithm, whose mean is simulated as an increasing and convex function in the congestion level. In this simulation, the compute node capacities $\boldsymbol{\kappa}$ were slightly under-provisioned to exhibit the policies' performances in a compute-limited settings.

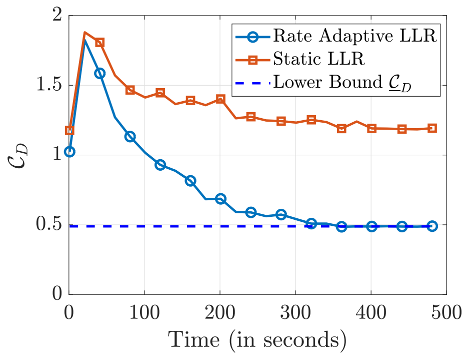


Fig. 7: Performance comparison of the rate-adaptive and static LRR algorithms, and the theoretical lower-bound on the mean rate of loss in revenue under stochastic network delays; $|\mathcal{A}| = 20$, $|\mathcal{S}| = 10$, $\gamma = 500$.

One can clearly observe that the rate-adaptive LLR policy's performance eventually moves very close to the steady-state lower-bound $\underline{\mathcal{C}}_D$, while the static policy does not perform as well. Here again, the rate-adaptive LRR algorithm has the advantage of being able to reach further compute nodes when needed than the static rate algorithm, explaining the gap between the two curves. Moreover, when performing rate adaptation, all the devices currently served in the network can quickly react to changes in the network delay they experience, making use of the available compute nodes' resources more efficiently, hence reducing blocking.

The presented results indicate that rate-adaptation is a requirement and needs to be associated with load-balancing policies to achieve low blocking rates, i.e., high service availability, and serve real-time timeliness constrained devices.

VI. Conclusions

In this paper, we studied the service placement and dimensioning problem in the fog network. We introduced a simple framework allowing us to identify the most cost-effective VM placement and network resource dimensioning strategies, and understand the fundamental tradeoffs associated with this problem. Unlike results presented in related work, we use the notion of *Age-of-Information* as a timeliness metric, as it demonstrated to be more relevant than network delay when devices send real-time updates. We showcased that different "forces" influence the optimal VM placement and resource dimensioning decisions in the *Cloud-to-Thing continuum*, which may greatly vary from use-case to use-case. We then proposed an online and decentralized joint service placement and rate adaptation policy based on delay measurements. This algorithm is aware of stochasticity in the network delays, ensuring near-optimal availability in large-scale networks by balancing the load on the different compute nodes that can host the devices' service. Our algorithm showed to outperform static rates policies, revealing that rate-adaptation is a requirement in the design of real-time applications.

References

[1] N. Mohammadi and J. E. Taylor, "Smart city digital twins," in *2017 IEEE SSCI*, November 2017.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, October 2016.

[3] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, December 2016.

[4] Y. Mao *et al.*, "A survey on mobile edge computing: The communication perspective," *IEEE COMST*, Fourth quarter 2017.

[5] X. Masip-Bruin *et al.*, "Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems," *IEEE Wireless Communications*, October 2016.

[6] A. Zhou *et al.*, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE TSC*, November 2017.

[7] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *IEEE INFOCOM 2018*, April 2018.

[8] A. Karamoozian, A. Hafid, and E. M. Aboulhamid, "On the Fog-Cloud Cooperation: How Fog Computing can address latency concerns of IoT applications," in *FMEC 2019*, June 2019.

[9] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet of Things Journal*, December 2016.

[10] O. Skarlat *et al.*, "Optimized IoT service placement in the fog," *Service Oriented Computing and Applications*, 2017.

[11] A. M. Maia *et al.*, "Optimized placement of scalable iot services in edge computing," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, April 2019.

[12] S. Pasteris *et al.*, "Service Placement with Provable Guarantees in Heterogeneous Edge Computing Systems," in *IEEE INFOCOM 2019*, April 2019.

[13] A. Santoyo Gonzlez and C. Cervell Pastor, "Edge Computing Node Placement in 5G Networks: A Latency and Reliability Constrained Framework," in *2019 6th IEEE International Conference on CSCloud/ 2019 5th IEEE EdgeCom*, 2019.

[14] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov, "Head tracking for the oculus rift," in *2014 IEEE ICRA*, 2014.

[15] S. Mangiante *et al.*, "VR is on the Edge: How to Deliver 360° Videos in Mobile Networks," in *ACM SIGCOMM 2017*.

[16] R. Ramaswamy, N. Weng, and T. Wolf, "Characterizing network processing delay," in *GLOBECOM '04.*, November 2004.

[17] K. Hong *et al.*, "Evaluation of multi-channel schemes for vehicular safety communications," in *IEEE 71st VTC*, May 2010.

[18] 5GPP, "5G Automotive Vision," 2015.

[19] NGMN Alliance, "V2X white paper," 2018.

[20] M. Claypool and K. Claypool, "Latency and player actions in online games," *Commun. ACM*, November 2006.

[21] M. Jarschel *et al.*, "An Evaluation of QoE in Cloud Gaming Based on Subjective Tests," in *IMIS 2011*, June 2011.

[22] R. Ratasuk *et al.*, "NB-IoT system for M2M communication," in *IEEE Wireless Communications and Networking Conference*, April 2016.

[23] NGMN Alliance, "5G white paper," 2015.

[24] 5GPP, "Cloud-native and verticals services," August 2019.

[25] A. ElGamal, J. Mammen, B. Prabhakar, and D. Shah, "Throughput-delay trade-off in wireless networks," in *IEEE INFOCOM 2004*, March 2004.

[26] B. Soret *et al.*, "Fundamental tradeoffs among reliability, latency and throughput in cellular networks," in *Globecom '14 Workshops*, 2014.

[27] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward low-latency and ultra-reliable virtual reality," *IEEE Network*, March 2018.

[28] C. Fraleigh, F. Tobagi, and C. Diot, "Provisioning ip backbone networks to support latency sensitive traffic," in *IEEE INFOCOM 2003*, March 2003.

[29] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *IEEE INFOCOM 2012*, March 2012.

[30] M. Costa, M. Codreanu, and A. Ephremides, "Age of information with packet management," in *IEEE ISIT 2014*, June 2014.

[31] A. Kosta, N. Pappas, V. Angelakis *et al.*, "Age of information: A new concept, metric, and tool," *Foundations and Trends in Networking*, 2017.

[32] R. D. Yates and S. K. Kaul, "The Age of Information: Real-Time Status Updating by Multiple Sources," *IEEE TIT*, March 2019.

[33] L. Huang and E. Modiano, "Optimizing age-of-information in a multi-class queueing system," in *IEEE ISIT 2015*, June 2015.

[34] H. B. Beytur and E. Uysal-Bykoglu, "Minimizing age of information on multi-flow networks," in *SIU 2018*, May 2018.

[35] S. Berezner, A. Krzesinski, and P. G. Taylor, "On the inverse of Erlang's function," *Journal of applied probability*, 1998.

[36] "Amazon EC2 Pricing." [Online]. Available: https://aws.amazon.com /ec2/pricing/on-demand/, Accessed on 2019-10-05

[37] B. Boudreau, "Global Bandwidth & IP Pricing Trends." [Online]. Available: https://www.telegeography.com/hubfs/2017/presentations/ telegeography-ptc17-pricing.pdf, Accessed on 2019-10-13

[38] M. Alanyali and B. Hajek, "Analysis of simple algorithms for dynamic load balancing," *Mathematics of Operations Research*, 1997.

[39] A. Fréville, "The multidimensional 0–1 knapsack problem: An overview," *European Journal of Operational Research*, 2004.

[40] C. Fortuin, P. Kasteleyn, and J. Ginibre, "Correlation inequalities on some partially ordered sets," *Communications in Mathematical Physics*, 1971.

Acknowledgement

This work was made possible by the support of Fujitsu Laboratories of America, and the NSF Grant ECC-1809327. The statements made herein are solely the responsibility of the authors.

Appendix

A. Timeliness metric motivation

We show that the timeliness metric $\frac{1}{2\rho}+d^t+d^c$ is reasonable in the setting under study. In [32], the authors show in

Theorem 3 that the mean AoI τ_m of device m is:

$$\tau_m = \frac{\mathbb{E}[I_m D_m] + \mathbb{E}[I_m^2]/2}{\mathbb{E}[I_m]}$$

where I_m represents the inter-arrival time between updates originating from m, and D_m is the system delay experienced by its updates. Intuitively, I_m and D_m are correlated, as a long inter-arrival time would be associated with the compute node having more time to process the tasks currently queued. More formally, we have the following result proved in Appendix B:

Lemma 1. $\mathbb{E}[I_m D_m] \leq \mathbb{E}[I_m] \cdot \mathbb{E}[D_m]$

Therefore, for deterministic I_m , $\mathbb{E}[I_m] = \frac{1}{\rho}$, $\mathbb{E}[I_m^2] = \frac{1}{\rho^2}$ and $\mathbb{E}[D_m] = d^t + d^c$, we get $\tau_m \leq \frac{1}{2\rho} + d^t + d^c$. Now, with increasing number of devices (our regime of interest), the incremental impact of an individual device m on the delay experienced by its own packets becomes negligible. Hence, by separation of time scales, one can conclude that this bound becomes tight in the limit, motivating our timeliness metric.

B. Proof of Lemma 1:

Proof. Let N_m be the number of arrivals to the queue during the inter-arrival time I_m, and let $\{R_i\}_i$ be the residual times of the update packets in the compute node queue upon arrival of the update from m. By the law of total covariance, we have:

$$\begin{aligned}\mathrm{Cov}(I_m, D_m) &= \mathbb{E}[\mathrm{Cov}(I_m, D_m | \{R_i\}_i, N_m)] \\ &\quad + \mathrm{Cov}(\mathbb{E}[I_m | \{R_i\}_i, N], \mathbb{E}[D_m | \{R_i\}_i, N_m])\end{aligned}$$

By observing that D_m is a deterministic function of $\{R_i\}_i$ and N_m, we conclude that the first term must be 0 as the covariance of two random variables is 0 if one of them is deterministic. Moreover, we note that $\mathbb{E}[I_m|\{R_i\}_i, N_m] = f(\{R_i\}_i, N_m)$ and $\mathbb{E}[D_m|\{R_i\}_i, N_m] = g(\{R_i\}_i, N_m)$, where f and g are deterministic functions. Clearly, f and g are respectively nonincreasing and nondecreasing in $\{R_i\}_i$ and N_m. Hence, from the FKG inequality [40], $\mathrm{Cov}(\mathbb{E}[I_m|\{R_i\}_i, N_m], \mathbb{E}[D_m|\{R_i\}_i, N_m]) \leq 0$, thus $\mathrm{Cov}(I_m, D_m) \leq 0$, hence $\mathbb{E}[I_m D_m] \leq \mathbb{E}[I_m]\mathbb{E}[D_m]$. □

C. Proof of Theorem 1

Proof. We start from Algorithm 1's decision policy. We have:

$$\begin{aligned}&\arg\min_{s\in\tilde{\mathcal{S}}_a} \int_{u_s(t)}^{u'_{a,s}(t)} f(u)du \\ &= \arg\min_{s\in\tilde{\mathcal{S}}_a} \int_{u_s(t)}^{u'_{a,s}(t)} \frac{1}{1-u}du \\ &= \arg\max_{s\in\tilde{\mathcal{S}}_a} \log(1-u'_{a,s}(t)) - \log(1-u_s(t)) \\ &= \arg\max_{s\in\tilde{\mathcal{S}}_a} \log(1-u'_{a,s}(t)) - \log(1-u_s(t)) \\ &\qquad\qquad + \sum_{s'\in\tilde{\mathcal{S}}_a} \log(1-u_{s'}(t)) \\ &= \arg\max_{s\in\tilde{\mathcal{S}}_a} \log(1-u'_{a,s}(t)) + \sum_{s'\in\tilde{\mathcal{S}}_a\setminus s} \log(1-u_{s'}(t))\end{aligned}$$

where the third step consists in adding a constant w.r.t. s. □

D. Proof of Theorem 2

This lower-bound proof is a generalization of the one proposed in [38]. As in [38], we introduce a virtual overflow compute node s_o of infinite capacity hosting the VMs of devices that have been blocked. Define $\beta_a(t)$ to be the number of Type a customers that have been blocked, i.e., that have been hosted in s_o, in $[0,t)$, T_m to be the random holding time of device m, and $X_{a,s}(t)$ to be the state of the network at time t, i.e., the number of Type a customers served by s. We have successively:

$$\begin{aligned}&\mathcal{C}_D(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa}) \\ &\quad= \sum_{a\in\mathcal{A}} w_a \mu_a^{-1} \lambda_a P(B_a; \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa}) \\ &\quad= \sum_{a\in\mathcal{A}} w_a \mu_a^{-1} \lambda_a \lim_{t\to\infty} \frac{\mathbb{E}[\beta_a(t)]}{t\lambda_a} \\ &\quad\overset{\text{(a)}}{=} \lim_{t\to\infty} \sum_{a\in\mathcal{A}} \frac{w_a\mu_a^{-1}}{t} \mathbb{E}[\sum_{m=1}^{\beta_a(t)} \frac{T_m}{\mu_a^{-1}}] \\ &\quad\overset{\text{(b)}}{\geq} \lim_{t\to\infty} \sum_{a\in\mathcal{A}} \frac{w_a}{t} \mathbb{E}[\int_0^t X_{a,s_o}(y)\,\mathrm{d}y] \\ &\quad= \lim_{t\to\infty} \sum_{a\in\mathcal{A}} \frac{w_a}{t} (\mathbb{E}[\int_0^t \sum_{s\in\mathcal{S}\cup s_o} X_{a,s}(y)\,\mathrm{d}y] \\ &\qquad\quad - \mathbb{E}[\int_0^t \sum_{s\in\mathcal{S}} w_a X_{a,s}(y)\,\mathrm{d}y]) \\ &\quad\overset{\text{(c)}}{=} \lim_{t\to\infty} \sum_{a\in\mathcal{A}} \frac{w_a\lambda_a}{\mu_a t} \int_0^t 1 - e^{-y\mu_a}\,\mathrm{d}y \\ &\qquad\quad - \lim_{t\to\infty} \frac{1}{t} \int_0^t \mathbb{E}[\sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}} w_a X_{a,s}(y)]\,\mathrm{d}y \\ &\quad\overset{\text{(d)}}{\geq} \sum_{a\in\mathcal{A}} \frac{w_a\lambda_a}{\mu_a} \lim_{t\to\infty} \frac{t + e^{-t\mu_a} - 1}{t} \\ &\qquad\quad - (\sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}} w_a A^*_{a,s}) \lim_{t\to\infty} \frac{1}{t} \int_0^t 1\,\mathrm{d}y) \\ &\quad= \sum_{a\in\mathcal{A}} w_a(\lambda_a\mu_a^{-1} - \sum_{s\in\mathcal{S}} A^*_{a,s}) \\ &\quad= \underline{\mathcal{C}}_D(\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa})\end{aligned}$$

Step (a) follows from the algebraic limit theorem as the number of types is finite and from the fact that T_m are i.i.d. of mean μ_a^{-1}, i.e., $\frac{T_m}{\mu_a^{-1}}$ have unit mean. Step (b) is a bound because some customers that arrived to s_o before time t may still be in the system at time t. Step (c) follows from the idea that the augmented network can be viewed as an $M/M/\infty$ queue, using the expression of the mean number of users in such a system at time t starting from the empty state at $t = 0$, as well as the Fubini-Tonelli theorem. In step (d), we use the fact that at every time y, $\mathbb{E}[\sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}} w_a X_{a,s}(y)] \leq \sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}} w_a A^*_{a,s}$ by definition of LP-MKP($\mathbf{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \Delta, \boldsymbol{\kappa}$).