Stateful Versus Stateless Selection of Edge or Cloud Servers Under Latency Constraints

Vincenzo Mancuso*, Paolo Castagno†, Matteo Sereno†‡, Marco Ajmone Marsan*
*IMDEA Networks Institute, Madrid, Spain
†Università di Torino, Turin, Italy
‡Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy

***Abstract*—We consider a radio access network slice serving mobile users whose requests imply computing requirements. Service is virtualized over either a powerful but distant cloud infrastructure or an edge computing host. The latter provides less computing and storage capacity with respect to the cloud, but can be reached with much lower delay. A tradeoff thus naturally arises between computing capacity and data transfer latency. We investigate the performance of this service model, discussing how service requests should be routed to edge or cloud servers. We look at the performance of various classes of online algorithms based on different levels of information about the system state. Our investigation is based on analytical models, simulations in OMNeT++, and a prototype implementation over operational cellular networks. First of all, we observe that distributing the load of service requests over edge and cloud is in general beneficial for performance, and simple to implement with a *stateless* online server selection policy that can be easily configured with near-optimal performance. Second, we shed light on the limited improvements that *stateful* polices can offer, notwithstanding they base their decisions on the knowledge of server congestion levels or round-trip latency conditions. Third, we unveil that stateful policies are dangerously prone to errors, which may make stateless policies preferable.**

***Index Terms*—Edge computing, Radio access network, Performance evaluation.**

I. Introduction

Mobile networks are evolving towards complex distributed connect-compute systems where end users receive services through data storage and processing elements that are located in different positions. Such elements are what used to implement fog, edge and cloud computing [1]. With an extreme simplification, we can describe in-network computing as comprising processing at the network periphery and/or in the network core. In this paper we refer to the former with the term *edge* and to the latter with the term *cloud* [2], [3].

The two key differences between edge and cloud lay in processing power and latency. The computing power and storage capacity of cloud infrastructures are normally much larger than in edge deployments. However, the delay incurred to access the computing resources of the cloud is normally much longer than to access those at the edge. For these reasons, whenever a user application is both computing-intensive and delay-insensitive, selecting a cloud server is a wise choice. On the contrary, applications that are at the same time latency-critical, and computing-parsimonious, are normally deployed on edge facilities. However, a wide variety of user applications is not easily classified in one of the two extreme categories above, and the question about which is the best server, and on what elements to base the choice becomes relevant.

The work was supported by the Region of Madrid through the TAPIR-CM project (S2018/TCS-4496).

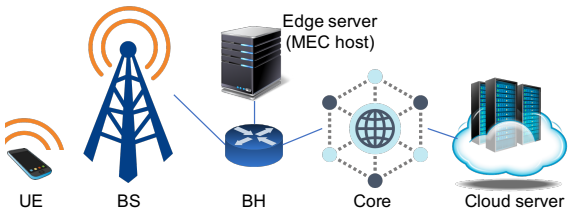

Fig. 1. Network architecture

In this paper we consider application scenarios in which end users exploiting a slice of a mobile network intermittently issue service requests to the network computing facilities and need an answer within a specified fixed deadline. Each user request can be processed either in the edge or in the cloud. For each request, a choice must be made about where to best serve it, in order to meet the specified deadline. The choice must be made by an element at the network edge, close to the edge resources, where partial information about the network state can be available. We look at several online algorithms for the implementation of the server selection choice, with variable degree of complexity and information on the system state. We study the probability that a request can be satisfied within the specified deadline, as a function of the system parameters, using an analytical model, a simulator based on OMNeT++ [4], and a prototype implementation over the MONROE platform [5].

Our results show that distributing the load of service requests over edge and cloud is simple to implement with a *stateless* online policy, which we name `RANDALPH`, and which can be easily configured with near-optimal performance by means of simple load balancing considerations. The results also shed light on the limited improvements that *stateful* online polices can offer when they use edge state information (`MCCLOUD` policy), or edge and cloud state information (`LENNON` policy). Moreover, we show that stateful policies can be jeopardized by state estimation errors, which makes stateless server selection preferable. Our prototype shows that the analytical model is robust with respect to a number of assumptions.

This work contributes to: (i) defining a framework for the virtualization of mobile services over edge and cloud; (ii) defining stateful and stateless online server selection policies; (iii) studying the policies analytically, in terms of throughput, loss probability and latency distribution (hence for any quantile); and (iv) shedding light on the viability of edge and cloud service virtualization in real systems.

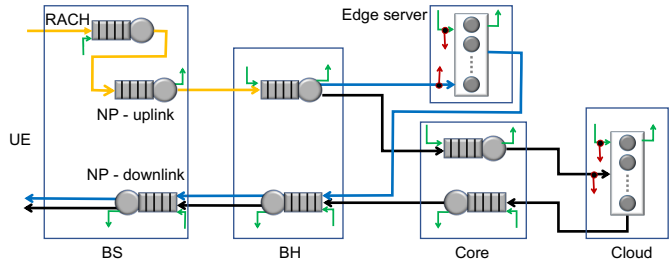

Fig. 2. System overview with background traffic (green) and overflow (red)

II. The System

We consider the system illustrated in Fig. 1, whose components are listed in the second column of Table I. The notation used in the system models is summarized in Table II.

Mobile user terminals (UEs) attach to a base station (BS), which is connected to a backhaul subsystem (BH). An edge server is also attached to the same BH (for instance, it can be a MEC host.) A transport/core network connects BH and the Internet, where a cloud server resides. UEs generate service requests with rate λ, which are sent for computation to a server running at either the edge or the cloud. A request can also be dropped rather than forwarded to a server. The server selection or dropping decision is made according to the state of edge and cloud servers, i.e., the number of requests at the two queues, indicated as j ($0 \leq j \leq k_E$) and i ($0 \leq i \leq k_C$), respectively, and in some cases also considering the delays incurred to reach either the edge or the cloud server. Note that in this paper we only consider that one edge and one cloud server are available, but the extension to the case of multiple edge or cloud servers can be studied with a straightforward extension of our approach. We use α_{ij} to refer to the probability that a request is sent to the edge, and β_{ij} for the cloud.

Specifically, we consider the following classes of policies:

1) Randomized alpha (`RANDALPH`, stateless): selection probabilities are fixed a-priory, i.e., $\alpha_{ij} = \alpha$, independently of queue states, and $\beta_{ij} = 1 - \alpha$. The requests reaching the edge or cloud when no space is available in their queues are discarded. Operating `RANDALPH` does not require knowledge about the state of the system.
2) Mobile computing plus cloud (`MCCLOUD`, stateful): a service request is always sent to the edge unless it has no space, in which case it is routed to the cloud. Operating `MCCLOUD` requires instantaneous knowledge about the state of the edge server only. In other words, the information necessary to implement the `MCCLOUD` policy only concerns the state of the queue at the edge.
3) Length of next round-trip (`LENNON`, stateful): a server with space available is selected depending on which site offers the lowest expected round-trip time at the selection instant. Requests are dropped at the backhaul only if both edge and cloud have no space left in their queues. Operating `LENNON` requires instantaneous knowledge on the state of both edge and cloud servers. In other words, the information necessary to implement the LENNON policy concerns the state of the queues at the edge and at the cloud as well as the expected round trip delays to reach either the edge or the cloud servers.

More complex policies have been defined and analyzed, that account for delay distributions and probability to miss the latency deadline, but results are not reported here because of lack of space and because the key findings of the analysis are the same as what is concluded by considering the policies introduced above.

A request consists of a message of P_R bits, on average. When a service request reaches the edge or cloud server, some computation is carried out, and a response is sent, consisting of P_S bits, on average. The intensity of served requests that are delivered to UEs is indicated as $\xi \leq \lambda$ (expressed in served requests per second). We denote the total service time by T and the application timeout by T_o. The service time is the round trip latency, which includes the transfer of the request to a server, the waiting and computation time at the server, and the transport of the server response back to the UE that originated the request. The performance of the system is characterized in terms of (i) service loss probability π_L, (ii) latency T and (iii) failure probability π_F, which is the probability that a request is not served within the timeout T_o.

Incoming service requests follow the network path depicted in Fig. 2. For each subsystem we denote by $\lambda_{(\cdot)}$, $\mu_{(\cdot)}$, and $\rho_{(\cdot)}$ the arrival rate, the service rate and the load, respectively, where the index identifies the subsystem (see Table I).

The **RACH** models transmissions from UE to BS. It uses a set of dedicated resources, fully isolated from other potential tenants and slices running on the same BS. We assume that RACH does not generate losses of service requests, and operates independently from the state of the rest of the network.

The RACH feeds a **network processor** in uplink, in which the service request is buffered and then injected in the packet network that lies between the BS and the destination server. Vice versa, service responses are queued at the downlink network processor for delivery to users. The network processor is assumed to introduce no losses.

The **backhaul** is an optical ring that connects several BSs and edge servers, and connects to the transport/core network [7]. It is *virtually* composed of point-to-point links with dedicated resources. Moreover, we assume that a slice of the backhaul is dedicated to the service we are studying, and that no losses occur in the backhaul because of its capacity.

The **transport network** lies between the backhaul ring and the cloud. It consists of multiple high-speed links connected sequentially, whose resources we consider not to be sliced.

Servers process user requests. They run on VMs and have a finite buffer, and use compute power and storage so as to generate downlink traffic in response to the received service requests. At both edge and cloud, the studied slice has a dedicated set of VMs, each pinned to a CPU core and used to serve the aggregate load of that slice. The number of VMs is n_E for the edge and n_C for the cloud. The total number of requests that can be queued at the edge server (including requests under service) is k_E, and k_C at the cloud.

Of course, in the case of resource abundance, the system behavior can be approximated by considering an infinite number of VMs at the cloud (and possibly also at the edge), with a significant simplification of the analysis the will be presented in the next section.

TABLE I
BASIC SYSTEM NOTATION, AND SYNOPSIS OF MODELING, SIMULATION AND TESTBED

Notation	Description	Model	Simulation	Testbed
R	RACH	Custom from [6]	3GPP-compliant	Real cellular network
U, D	Up-/Down-link BS network processor	M/M/1, M/M/1-PS, or M/D/1-PS	Slotted M/M/k/n	Real BS scheduler (tested with various commercial operators)
B_{u}, B_{d}	Backhaul (uplink and downlink)	Decoupled M/M/1 queues feed by constant-speed links	Decoupled slotted M/M/1/n queues feed by constant-speed links, with limited buffer space n	Cellular network plus access to proxy in a lab, implementing the allocation policy (Country A)
T_{u}, T_{d}	Transport (to and from the cloud)	Decoupled M/M/1 queues feed by constant-speed links	Decoupled slotted M/M/1/n queues feed by constant-speed links, with limited buffer space n	Internet in between a proxy (Country A) and a VM in a data center (Country B)
E	Edge server	M/M/n_E/k_E	Slotted M/M/n_E/k_E	Docker container implementing a slotted M/M/n_E/k_E (located in Country A)
C	Cloud server	M/M/n_C/k_C	Slotted M/M/n_C/k_C	VM in a Data center (located in Country B) implementing a slotted M/M/n_E/k_E queue.

TABLE II
MODELING NOTATION

Notation	Description
$f, F, \widehat{f}$	pdf and CDF of a latency r.v., and associated LST
λ, μ, ρ, ξ	Arrival rate (requests/s), service rate (requests/s), load (in Erlang units) and throughput (requests/s)
p_R	RACH attempt success probability
n_E, n_C	Number of dedicated processors at edge and cloud
k_E, k_C	Max total number of requests at edge and cloud
p_E, p_C	Distribution of customers in edge and cloud servers
T, T_o	System latency (round trip service latency) and timeout
α, β	Request allocation probabilities towards edge and cloud
π_E, π_C	Overflow probabilities at edge and cloud
π_L, π_F	System loss and failure probabilities

III. BASIC ANALYSIS

We study flows and latency distributions at each network node modeled as a queue. We use f to indicate the pdf of latency, F for the CDF, and $\widehat{f}$ for the corresponding Laplace-Stieltjes transform (LST). Loss probabilities are denoted by π, with a subscript that indicates the subsystem they refer to. We use ℓ to indicate background traffic load and assume that all queues in the system are work-conserving FIFO queues.

Next, we analyze the behavior of the system for fixed values of $\alpha_{ij} = \alpha$ and with $\beta_{ij} = 1 - \alpha_{ij}$, which corresponds to the analysis of `RANDALPH`. Successively, in Section IV, we will show how to use continuous-time Markov chains (CTMCs) to extend the analysis to analyze the proposed stateful policies.

A. RACH

From [6], the LST of the sojourn time distribution, given that the request is eventually served is as follows:

$$\widehat{f}_R(s) = \frac{1-e^{-s(T_x+W_x)}}{s(T_x+W_x)} \sum_{i=1}^{k_x} p_R(\lambda+\ell_R,i) \left(\frac{e^{-sT_x}}{1+sE\left[B_R\right]} \right)^{i-1}, \quad (1)$$

where $p_R(\lambda+\ell_R, i)$ is the probability that a request succeeds in exactly $i \leq k_x$ RACH attempts (where k_x is the maximum number of permitted attempts), with $\sum_{i=1}^{k_x} p_R(\lambda+\ell_R, i) = 1$, since there are no losses. T_x and W_x are RACH configuration parameters, and $E\left[B_R\right]$ is the average RACH backoff duration.

Assuming negligible RACH losses, we can also neglect RACH timeouts, finite number of RACH retries, and network processor back-pressure. Therefore, by imposing $k_x = \infty$ and approximating the RACH per-state success probability as the effect of power ramping (see [6]), p_R in (1) becomes independent on traffic and can be simplified as

$$p_R(\lambda+\ell_R, i) = e^{-\frac{i(i-1)}{2}} \left(1 - e^{-i}\right). \quad (2)$$

B. Network processor

The arrival rate at the uplink network processor is $\lambda_U = \lambda + \ell_U$, because we have assumed that the RACH introduces no losses. The service rate is μ_U, so that the load is $\rho_U = (\lambda + \ell_U)/\mu_U$, which accounts for background traffic. We model this component with a processor sharing queue with fixed service time (M/D/1-PS), whose sojourn time distribution is [8]:

$$\widehat{f}_U(s) = \frac{(1-\rho_U)\,(\lambda_U+s)^2\, e^{-(\lambda_U+s)/\mu_U}}{s^2 + \lambda_U\left[s + (\lambda_U+s)\,(1-\rho_U)\right] e^{-(\lambda_U+s)/\mu_U}}. \quad (3)$$

When the network processor load is low, an M/M/1 approximation can be used as well, since, if the uplink processor is not the system bottleneck, results change very little if we use an M/M/1 queue or a more complex M/M/1/PS queue.

The arrival rate at the network processor in downlink is the system throughput $\xi = (1 - \pi_L)\lambda$ plus the background downlink traffic, where the system loss π_L is a function of λ and α. By defining $\rho_D = ((1 - \pi_L)\lambda + \ell_D)/\mu_D$, we have an expression for $\widehat{f}_D$, the LST of the latency in the downlink network processor, similar to the one for the uplink.

C. Backhaul

We use an M/M/1 representation of the backhaul, with two independent FIFO queues serving requests and replies, respectively. The LST of the sojourn time in an M/M/1 has the form $a/(a+s)$, where a is the difference between service rate and arrival rate. For the backhaul, the uplink has service rate $\mu_{B_{\mathrm{u}}}$ and sees the entire offered traffic λ plus some background traffic $\ell_{B_{\mathrm{u}}}$, while the downlink serves at rate $\mu_{B_{\mathrm{d}}}$ and sees only the throughput ξ and the downlink background traffic $\ell_{B_{\mathrm{d}}}$.

Note that only the downlink case is affected by the allocation probability α, since the uplink sees the full traffic λ, while ξ is affected by server selection decisions.

D. Transport network uplink and downlink

To model the transport network, we use M/M/1 queues with service rates μ_{T_u} and μ_{T_d} to and from the cloud, respectively. The corresponding arrival rates, which include background traffic, are $\beta\lambda + \ell_{T_\mathrm{u}}$ and $(1-\pi_C)\beta\lambda + \ell_{T_\mathrm{d}}$.

E. Edge server

To model the edge behavior we use an M/M/n_E/k_E queue with arrival rate $\alpha\,\lambda + \ell_E$. The load is $\rho_E = \frac{\alpha\,\lambda+\ell_E}{\mu_E}$. The probability to have j requests in the edge server is

$$p_E(j) = p_E(0)\ \rho_E^j / \left(\min(j, n_E)!\ n_E^{[j-n_E]^+}\right), \qquad (4)$$

where $p_E(0)$ is found by solving $\sum_j p_E(j) = 1$, the loss probability relative to arrivals at the edge server is $\pi_E = p_E(k_E)$, and $[x]^+$ indicates the max between 0 and x.

For the computation of the distribution of the latency, consider that when a customer arrives and finds $j < k_E$ customers in the system, the customer is directly served if $j < n_E$, while it has to wait for $j - n_E + 1$ service completions before being served if $j \geq n_E$. The time to complete a service when all servers are busy is exponentially distributed with rate $n_E\,\mu_E$, while the rate at which the waiting customer is finally served is μ_E (again, with exponential distribution). Therefore, the total time in the edge server is the sum of an exponential r.v. (the service) and an independent random waiting time, which, conditional to finding j customers in the system, is 0 if $j < n_E$ and otherwise it is Erlang distributed with $j - n_E + 1$ stages and rate $n_E\,\mu_E$ if $n_E \leq j < k_E$. The conditional expression is:

$$\widehat{f}_E(s|j) = \frac{\mu_E}{s+\mu_E}\left(\frac{n_E\,\mu_E}{s+n_E\,\mu_E}\right)^{[j-n_E+1]^+}. \qquad (5)$$

Removing the condition, the LST of the distribution of the total time spent in the edge system is as follows:

$$\widehat{f}_E(s) = \frac{1}{1-\pi_E}\frac{\mu_E}{s+\mu_E}\sum_{j=0}^{n_E-1} p_E(j)\left(\frac{n_E\,\mu_E}{s+n_E\,\mu_E}\right)^{[j-n_E+1]^+}. \qquad (6)$$

F. Cloud server

This case is similar to the case of the edge server, with a queue M/M/n_C/k_C with arrival rate $\beta\,\lambda + \ell_C$ and service rate μ_C, resulting in a queue size distribution indicated as $p_C(i)$ and loss probability $\pi_C = p_C(k_C)$.

G. RANDALPH's performance indicators

The system throughput, with RANDALPH, is $\xi = \lambda\,(1-\pi_L)$, where the system loss probability is $\pi_L = \alpha\,\pi_E + (1-\alpha)\,\pi_C$.

Latency T has a distribution with LST obtained by multiplying the LSTs of system components, because the latency in a subsystem does not depend on the latency in another subsystem. However, we need to consider only successful requests, so that latency has to be conditioned on success:

$$\widehat{f}_T(s) = \frac{1}{1-\pi_L}\widehat{f}_R(s)\widehat{f}_U(s)\widehat{f}_D(s)\widehat{f}_{B_\mathrm{u}}(s)\widehat{f}_{B_\mathrm{d}}(s) \cdot\Big(\alpha(1-\pi_E)\widehat{f}_E(s) + (1-\alpha)(1-\pi_C)\widehat{f}_{T_\mathrm{u}}(s)\widehat{f}_{T_\mathrm{d}}(s)\widehat{f}_C(s)\Big). \qquad (7)$$

A failure is either caused by a message drop (or discard) or by the expiration of the timeout. Hence, the failure probability of RANDALPH depends on the loss probability and on the CDF of T computed at T_o:

$$\pi_F = \pi_L + (1-\pi_L)\ (1 - F_T(T_o)) = 1 - (1-\pi_L)\,F_T(T_o). \qquad (8)$$

However, computing $F_T(t)$ in $t = T_o$ requires inverting the function $\widehat{f}_T(s)/s$, the division by s representing the integral operator in the time domain, so as to pass from pdf to CDF. This can be done numerically.

IV. Stateful Policies

The analysis presented for RANDALPH applies also to the *stateful* policies that we have defined, if observed in a given, fixed state of the edge and cloud queues. However, there is a key difference in the way we consider the edge and cloud servers, since in a given state the delay incurred by a request entering any of those queues sees a "simpler" latency distribution, i.e., (5) in the case of the edge.

As described in what follows, we characterize the stateful policies by evaluating the state probabilities and taking the average over all queue states. To compute such average, we solve a CTMC that describes the joint evolution of the queues of edge and cloud servers.

A. Description of the system dynamics through a CTMC

All policies see the state of edge and cloud queues evolve as described in Fig. 3. The CTMC depicted in the figure is bi-dimensional; horizontal transitions represent edge events, while vertical transitions refer to the cloud. The use of a CTMC comes with an approximation that we will validate with experiments. Specifically, we are considering that a state transition happens as soon as a new request is allocated, and when a request is served. Assuming that the state of a queue changes at server selection time is however not exact, because it neglects the time needed for the request to actually reach the server. Therefore, the CTMC of Fig. 3 is inconsistent with the actual state of edge and cloud when a request is queued. In particular, we assume that the allocation decision is taken at the backhaul, i.e., very close to the edge, but not to the cloud.

Services occur as in two independent M/M/n/k queues. Arrivals have a total intensity λ in each state, although the state-dependent server selection splits arrivals into edge arrivals, with intensity $\alpha_{ij}\lambda$ in state (i,j), and cloud arrivals, with intensity $\beta_{ij}\lambda$. Note that coefficients $\alpha_{i\,k_E}$ and $\beta_{k_C\,j}$ must be set to 0 in the CTMC, since in the corresponding states it is not possible to accept additional jobs at the edge or cloud, respectively, but the server selection probability can be larger than 0 in those states, in which case requests are lost due to overflow. However, to avoid unnecessarily heavy notation, we use α_{ij} and β_{ij} for both the allocation probabilities and the coefficients of the CTMC. Of course, CTMC coefficients are also 0 if any of the indices is negative (i.e., in the figure, there are no arrows entering from the left at the left edge, and no arrows entering from the top at the top edge).

The coefficients are such that $\alpha_{ij} + \beta_{ij} \in [0,1]$, and the quantity $1 - (\alpha_{ij} + \beta_{ij})$ is the loss in state (i,j) due to policy dropping (in all nodes of the CTMC, except the right and bottom edges) or overflow (right and bottom edges of the

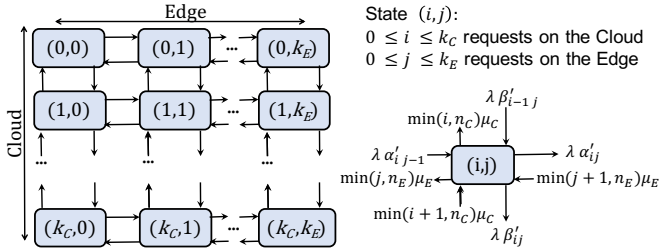

Fig. 3. Markov chain representing the state of edge and cloud

CTMC, where the loss is $\alpha_{i\,k_C}$ and $\beta_{k_E\,j}$, respectively, and it is 1 in state (k_C, k_E)).

B. General expressions

Let $\{\pi_{ij}\}$ be the steady state solution of the CTMC representing our edge/cloud system. The fraction of traffic routed towards the edge and the cloud, and the associated loss probabilities as well as the overall loss probability π_L and the LST of the latency are computed as per-state averages, i.e.:

$$\alpha = \sum_{i\,j} \pi_{ij}\alpha_{ij}; \qquad \beta = \sum_{i\,j} \pi_{ij}\beta_{ij}; \tag{9}$$

$$\pi_E = \frac{1}{\alpha}\sum_{i} \pi_{i\,k_E}\alpha_{i\,k_E}; \qquad \pi_C = \frac{1}{\beta}\sum_{j} \pi_{k_C\,j}\beta_{k_C\,j}; \tag{10}$$

$$\pi_L = \pi_{k_C\,k_E} + \sum_{(i\,j)\neq(k_C\,k_E)} \pi_{ij}\,(1-\alpha_{ij}-\beta_{ij}) + \sum_{i=0}^{k_C-1} \pi_{i\,k_E}\alpha_{i\,k_E} + \sum_{j=0}^{k_E-1} \pi_{k_C\,j}\beta_{k_C\,j}. \tag{11}$$

$$\widehat{f}_T(s) = \frac{1}{1-\pi_L}\widehat{f}_R(s)\widehat{f}_U(s)\widehat{f}_D(s)\widehat{f}_{B_\mathrm{u}}(s)\widehat{f}_{B_\mathrm{d}}(s) \cdot \left(\sum_{i,j<k_E}\pi_{ij}\alpha_{ij}\widehat{f}_E(s\,|\,j) + \sum_{i<k_C,j}\beta_{ij}\widehat{f}_C(s\,|\,i)\widehat{f}_{T_\mathrm{u}}(s)\widehat{f}_{T_\mathrm{d}}(s)\right). \tag{12}$$

The four terms in (11) are losses due to no space left in the system, the request dropping decided by the policy, the overflow at the edge when the cloud has still space left, and the one at the cloud when the edge has space, respectively.

Note that, in case α_{ij} does not depend on i and j, and $\beta_{ij} = 1-\alpha_{ij}$, the above expressions reduce to the ones computed for RANDALPH. Note also that the expression for the failure probability given by (8) holds with the system loss and LST of latency computed as in (11) and (12), respectively.

C. MCCLOUD

MCCLOUD behaves similarly to RANDALPH with $\alpha = 1$. However, here the overflow of the edge goes to the cloud, so that a loss occurs only if arrivals find the queues in state (k_C, k_E). So, the CTMC coefficients are as follows:

$$\begin{cases} \alpha_{ij} = 1, & \forall i \le k_C,\, j < k_E; \\ \beta_{ij} = 1, & \forall i < k_C,\, j = k_E. \\ \alpha_{ij} = \beta_{ij} = 0, & \text{otherwise.} \end{cases} \tag{13}$$

The probability to select the cloud server is the overflow probability observed by sending all requests to the edge, i.e.:

$$\alpha = 1 - \sum\nolimits_i \pi_{i\,k_E}, \qquad \beta = 1-\alpha. \tag{14}$$

The loss probability of the edge server is $\pi_E = 0$, because no request is sent to the edge in case its queue is full. The cloud server suffers losses with probability $\pi_C = \pi_{k_C\,k_E}/\beta$, where $\pi_L = \pi_{k_C\,k_E}$ is the overall MCCLOUD loss probability.

D. LENNON

With LENNON, no request is allocated to either the edge or the cloud if there is no space. Hence, $\pi_E = \pi_C = 0$, and losses only occur because of policy dropping when both queues are full: $\pi_L = \pi_{k_C\,k_E}$. As a consequence, server selection probabilities and CTMC coefficients coincide in all states. Specifically, the probability to allocate a request to the edge in state (i,j) is a function of the edge server queue occupancy and the expected latency over edge and cloud. Denoting by d_C and d_E the round-trip propagation time between the backhaul and the cloud or the edge server, respectively, we have:

$$\alpha_{ij} = \begin{cases} 1, & i = k_C, \forall j < k_E \text{ (i.e., no space at the cloud);} \\ 1, & \forall i < k_C,\, j < k_E \text{ such that} \\ & \frac{1}{\mu_E} + \max(j - n_E + 1, 0)\,\frac{1}{n_E\,\mu_E} + d_E \\ & \le \frac{1}{\mu_C} + \max(i - n_C + 1, 0)\,\frac{1}{n_C\,\mu_C} + d_C; \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

The probability to select the cloud server is $\beta_{ij} = 1-\alpha_{ij}$, except in state (k_C, k_E), where it is 0 because the request is dropped. This is possible thanks to the fact that the entity that enforces the LENNON policy knows the status of both queues.

E. Modeling the error on queueing state estimation

So far we have presented our models assuming that the state of the queues at edge and cloud servers is exactly known. However, as we have already remarked, there is a lag between allocation decisions and queueing. Moreover, while it is reasonable to assume that the state of the edge queue can be estimated or communicated to the backhaul, the state of the cloud can be only roughly estimated in a real implementation. Indeed, even if a monitor (e.g., a resource orchestrator) collects information about the system state, given the very fast system dynamics and the unavoidable information transfer delays, these informations risk not to be accurate. For instance, with realistic cloud round-trip times of the order of several tens of milliseconds and network virtual functions requiring a few milliseconds to parse a service request, the status of the cloud queue changes very quickly in the time between the status is sampled and when the monitor receives the sample. In practice, the difference between the cloud status used at the monitor for making a decision and the status observed by the service request when reaching the cloud is a random variable with variance equal to the sum of the variance of the cloud arrival process in the elapsed time and the variance of the cloud service process in the same interval. The exact shape of the distribution of such a random variable depends on how status samples are gathered and on the service discipline used at the cloud. For these reasons, we generically consider queueing state estimate errors as random variables which yield the *belief* probabilities $q((i',j') \mid (i,j)) = \Pr\left((i',j') \text{ estimated} \mid (i,j)\right)$.

Accordingly, the stateful policies MCCLOUD and LENNON make server selection decisions in state (i,j) as if they were observing state (i',j') with probability $q((i',j') \mid (i,j))$. The

belief probabilities have hence the effect of modifying the per-state transition probabilities. For instance, the edge server selection probability becomes:

$$\alpha_{ij}^{(e)} = \sum_{i'} \sum_{j'} q((i',j') \mid (i,j))\, \alpha_{i'j'}, \qquad (16)$$

and the same applies to compute $\beta_{ij}^{(e)}$. The expressions are identical for all stateful policies, while the error does not affect RANDALPH. CTMC transitions change because of errors, which can lead to losses due to policy dropping also for states in which the policy would normally not drop. E.g., LENNON will suffer a loss if it believes that the edge and/or the cloud server queues are not full when they are, or if it believes that both queues are full when at least one is not.

In the presence of state estimation errors, we can use the general expressions for stateful policies presented in Section IV-B, with $\alpha_{ij}^{(e)}$ and $\beta_{ij}^{(e)}$ replacing α_{ij} and β_{ij}.

Belief probabilities are therefore easy to account for. However, their actual structure depends on the estimate mechanism adopted for conveying queue state information to the backhaul. A discussion of estimation approaches is not included in this paper for the sake of brevity. We will rather consider simple cases of independent, discretized, zero-mean Gaussian errors:

$$q((i',j') \mid (i,j)) = G(i',i,\sigma_C)\; G(j',j,\sigma_E), \qquad (17)$$

where $G(y,x,\sigma)$ denotes the probability that a Gaussian random variable with average x and variance σ^2 takes a value in the interval $[y-1/2, y+1/2]$, except when $x=0$ or $x=k$, in which case we extend the left or right limit of the interval to (minus) infinite. For MCCLOUD we simply use $q(j' \mid j) = G(j',j,\sigma_E)$ because the cloud state is not used for server selection.

V. Simulation and Real Experiments

The model presented above is validated by comparing its results with a discrete-event simulator developed in OMNeT++ [4] and with a network overlay testbed with two server sites in two different countries (both the BS and the edge server are in Spain, while the cloud server is in Italy). In both the simulator and the testbed, we have reproduced all the subsystems used in the model, although in the testbed we did not have access to resources in an edge infrastructure, which was replaced with a dedicated server in a lab, as explained later in this section and annotated in Table I.

A. Simulator

In the simulator, all subsystems but the RACH make use of pre-defined modules offered by OMNeT++. The RACH is implemented with c++ code according the specification given in [9]: it implements the four-way handshake required to set-up a RRC connection between UE and gNB, accounting for the contention-based access procedure and its subsequent contention resolution. BS and UEs have a RACH module that handles timers as specified by the standard, causing the access procedure to fail in case an answer is not received within the required timeout. The RACH module for a UE accounts for the preamble selection, the power ramping strategy—to select an appropriate transmission power—and for the maximum number of retrials before a connection failure. On the BS side, the RACH module handles contention in the preamble transmissions and accounts for limited processing capabilities. Such limitations concern the maximum number of RAR responses sent by the gNB during the standard RACH four-way handshake and the maximum number of active RRC connections. Both The BS and the UE implement the RRC connection timeout, causing a UE to repeat the random access procedure in case of long periods of inactivity.

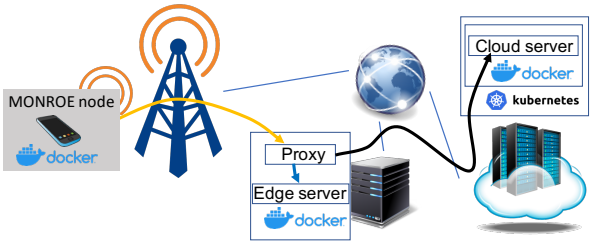

Fig. 4. Testbed architecture. Requests are QUIC flows that reach the proxy through the BS (orange path). The proxy routes requests to edge (blue) or cloud (black). Replies are QUIC flows that travel back over the request's path.

All other subsystems of Section II are defined as routers or processing units linked together by wired links with a given capacity. The simulator considers a number of devices generating UDP requests with uniform interarrivals, while service times at edge and cloud are exponential, and their state is estimated as explained later in the testbed description. Thus, the simulator allows us to verify the accuracy of our model's approximations on inter-request times and state transitions.

B. Testbed

For real experiments, we use the MONROE platform [5], accessing a BS in Spain, a server in our IMDEA Networks lab in Spain and a server in a data center in Italy, at the University of Turin. The testbed architecture is illustrated in Fig. 4. The testbed design includes three main device types, namely UE, server and proxy. Each device type is fully specified within a Docker image, to be easily deployed and migrated. Device images are placed around the network reflecting a specific setting, and exploit QUIC as transport protocol. Our UEs are static MONROE nodes located in Spain, and connected to operational cellular networks, over which we have no control. The cloud server runs in a Docker container in a remote data center and the edge server and the proxy run in Docker containers on the same machine in our lab. UEs contact the proxy through their cellular connectivity and the proxy makes server selection decisions. All server replies are QUIC flows sent to the proxy and afterwards redirected to the UEs. Proxy and cloud server talk over a wired transnational Internet backbone, which is out of our control.

The network components of the testbed are real operational networks, so that we run our experiments in the wild, except for what concerns the proxy, in which we are free to implement any server selection policy. Note however that the proxy does not know instantaneous server state values, which have to be estimated. The estimation mechanism we implemented is quite simple: the proxy monitors forwarded requests and replies, and computes the queue states by looking at request IDs, which are progressive numbers, and by discounting from the outstanding requests the expected number of services completed in a round trip time between the proxy and the server. Thus, the testbed

allows us to validate the *ensemble* of network approximations we have introduced in the model, and evaluate our policies robustness to state estimation errors.

VI. Evaluation

The numerical evaluation of the three server selection policies that we have described and analyzed above (RANDALPH, MCCLOUD and LENNON) considers as a reference scenario a network slice that is allocated one processor at the edge computing facility and 10 processors at the cloud infrastructure. Up to 10 service requests can be queued at the edge, and 50 at the cloud, including requests being processed. Each processor is capable of serving 200 requests per second. Hence, the total service capacity of edge+cloud is 2200 requests per second. These values reflect the capabilities of state of the art processors. The network delay parameters are derived from the testbed and plugged into both the simulator and the analytical model.

In addition to the three policies above, we also study two algorithms that will be considered as baseline, and that are special cases of the RANDALPH algorithm. The algorithm called STAY assumes the availability of only the edge server, so that all requests are routed to the edge (i.e., we use $\alpha = 1$ in RANDALPH). On the contrary, the algorithm called GO assumes that only the cloud server is available, and all requested are routed to it (i.e., we use $\alpha = 0$ in RANDALPH). Note that STAY can process at most 200 requests per second, while GO can process up to 2000. Finally, we also look at an algorithm named R-BAL, which represents the load-balancing version of RANDALPH considering the edge and cloud nominal processing capacities (i.e., $\alpha = n_E\mu_E/(n_E\mu_E + n_C\mu_C)$). Note that R-BAL only requires static information about the system configuration, and is independent of the dynamics of the system state.

In Fig. 5 we plot the curves of the loss and failure probabilities for the different algorithms, as functions of the load offered to the overall processing capacity provided by edge and cloud, assuming it takes 27 ms to cross the links between UE and edge (RTT, not considering queueing and processing), and 66 ms for the cloud, in line with our experimental observations. By loss we mean a discarded request, while by failure we mean that the reply is not returned to the end user within the maximum admitted latency (which in the figure is taken to be 100 ms). Hence, failures are a superset of losses (i.e., lost requests + late replies). We can see that STAY exhibits the highest loss and failure probabilities, as expected, due to its small processing capacity. The other algorithms show comparable performance, with failure probabilities below 5%, except in the case of MCCLOUD, that does not exploit the cloud processors until the edge processor becomes full. The figure also reports results obtained with RANDALPH with the values of α that yield the lowest loss (R-LOSS) or the lowest failure probability (R-FAIL). In those curves, α is different for each value of λ. The curves do not correspond to practical algorithms, but rather to ideal algorithms that can adapt to traffic conditions. However, as shown in the figure, adaptation only leads to a small gain with respect to R-BAL and does not outperform MCCLOUD and LENNON in terms of loss.

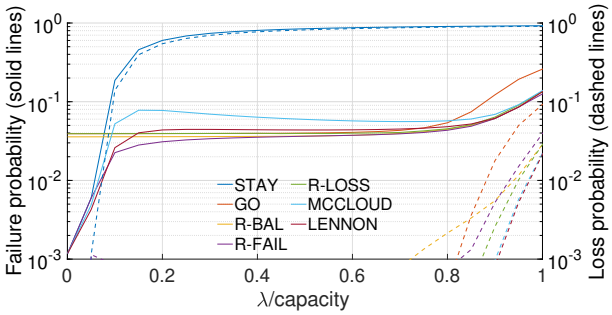

Fig. 5. Failure (at 100 ms) and loss ($n_C = 10$, $k_C = 50$, $n_E = 1$, $k_E = 10$)

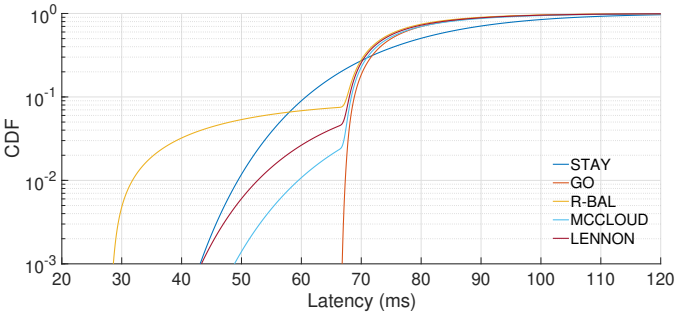

Fig. 6. CDF of T for $\rho = 0.75$, $n_C = 10$, $k_C = 50$, $n_E = 1$, $k_E = 10$

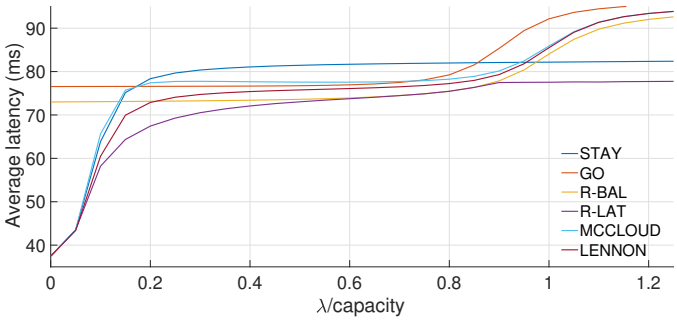

Fig. 7. Average latency T for $n_C = 10$, $k_C = 50$, $n_E = 1$, $k_E = 10$

To better compare the performance of the policies under evaluation, in Fig. 6 we plot the curves of the CDF of latency T (i.e., the time between the issue of a request by the end user, and the reception of the corresponding reply – considering only requests that are not lost) with a load $\rho = 0.75$. In Fig. 7 we plot the corresponding curves of the average latency. The CDFs show a bi-modal behaviour, which corresponds to requests sent to edge and cloud, except for STAY and GO that only use one option. The latency CDF curves show that R-BAL is the only algorithm capable of providing latencies of 40-50 ms with non-negligible probabilities (remember that this CDF is computed over replies to non-lost requests; hence, the case of STAY must be considered accounting for its high loss probability). Stateful policies exhibit higher latency, which is what allows them to achieve a lower loss rate. Indeed, LENNON, which is the most complex stateful policy we consider, achieves lower latency (and therefore lower failure rates) with respect to MCCLOUD, while they see almost the same loss. The corresponding average values, which are visible in Fig. 7 are however not so different. Indeed, from the figure we can see that in the most common range of loads (say between 0.2 and 0.8) all algorithms provide average latency values

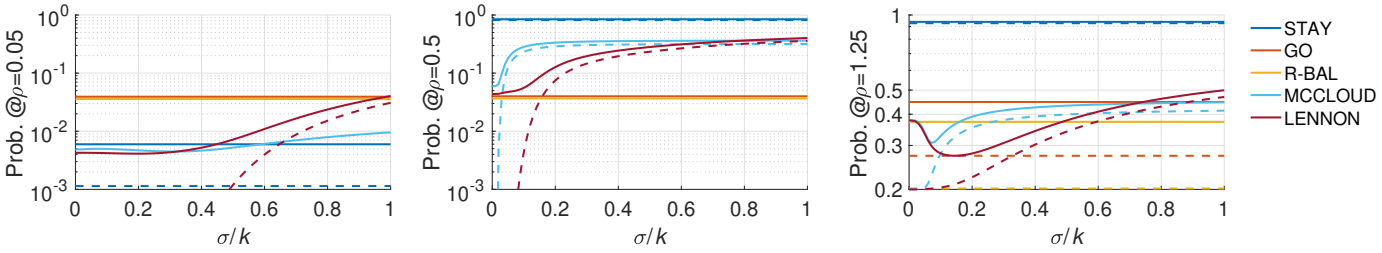

Fig. 8. Failure (solid lines) and loss (dotted lines) probability with $T_o = 100$ ms, $n_C = 10$, $k_C = 50$, $n_E = 1$, $k_E = 10$. Impact of errors.

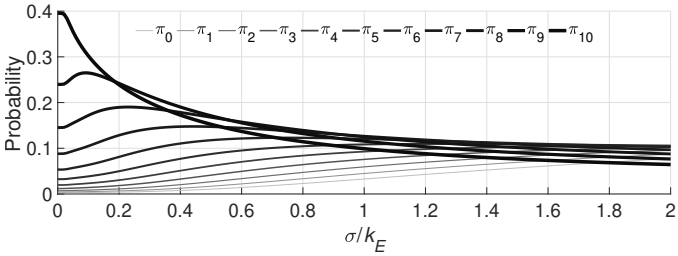

Fig. 9. Impact of estimate errors on edge state probabilities with MCCLOUD at $\rho = 0.15$ (for the overall system, while edge's capacity only suffices for $\rho \leq 1/11$), with $n_C = 10$, $k_C = 50$, $n_E = 1$, $k_E = 10$.

of a few tens of ms. STAY has the highest average latency because of its lower capacity. The curve labeled R−LAT shows results for RANDALPH, using a value of α that adapts to λ so as to minimize latency. Of course, implementing R−LAT is impractical as for the cases of R−FAIL and R−LOSS, but we report it to give an idea of what latency is achievable by a genie-based algorithm.

All of the results presented so far assume that the state of the queues of requests at edge and cloud processors are exactly known with no delay. In this (admittedly unrealistic) situation, we can draw two important conclusions from results: i) the joint exploitation of edge and cloud processors is important, and ii) a state-oblivious algorithm like R−BAL is capable of achieving performance comparable to algorithms that exploit the instantaneous system state.

As a next step, we investigate the effect of imprecise information about the system state on the algorithm performance. In Fig. 8 we report the loss and failure probabilities versus the standard deviation of state estimate error, normalized to the state size k (i.e., either to k_E for the edge or k_C for the cloud) – for each point of the curves we inject the same normalized error magnitude to both edge and cloud state estimations) for the different algorithms at three values of the load requested from the edge and cloud processors. Left we plot results for load equal to 5% of the overall capacity (110 requests/s). The load grows to 50% in the middle plot, and to 125% in the right plot. Only in the left plot algorithm STAY is in normal operating conditions, and can provide acceptable loss and failure probabilities. In the middle and right plots, STAY is in sustained overload, and its loss probability corresponds to the excess requests with respect to its capacity. LENNON and R−BAL perform very well at load 0.5, and R−BAL shows good performance also in overload conditions.

Looking in more detail, we can observe that errors affect loss and failure in a different way, depending on the used policy. In fact, loss and failure curves under low-medium loads evolve almost in parallel for LENNON, which means that LENNON suffers more losses due to errors, while the loss of MCCLOUD is barely affected at low loads and its failure grows faster than loss in general. However, it is interesting to notice that stateful policies under very high loads ($\rho \geq 1$), experience some benefit from the presence of small errors, and their failure probability shows a minimum as the error variance changes. For instance, with a sustainable load for the edge (left plot), MCCLOUD suffers no impact of errors on loss because both edge and cloud are underutilized; however, delay increases because some of the traffic that could be served at the edge is sent to the cloud. This explains why the failure probability of MCCLOUD worsens with the magnitude of the error. Instead, LENNON can experience more losses because it may erroneously estimate the overall system as full.

At medium/high load for the edge, the impact of errors on MCCLOUD is twofold. On the one hand, the error causes unnecessary losses at the edge, which receives requests that cannot be queued. This however reduces the load of the cloud, so that, on the other hand, the overall failure probability eventually starts dropping. Asymptotically, for large error magnitudes, the edge receives half of the requests (and only serves 1/11 of the capacity of the system), while the remaining half is served at the cloud with little loss (except if the system is largely overloaded). The combination of these two effects produces a maximum in the curves of the loss of MCCLOUD versus the error variance. The behaviour of LENNON is similar, except errors on the estimation of the cloud state can partially compensate errors on the edge, so that the maximum of the curves is reached for larger error variances.

In order to visualize the effect of errors in the estimation of the state of the edge processor when using MCCLOUD, in Fig. 9 we report the probabilities of states at the edge queue for a load equal to 15%, which saturates the edge server. The element π_0 is the probability that the edge processor is idle, while the other elements correspond to one request in service and from 0 to 9 in the queue (i.e., π_{10} is the probability that the edge queue is full and a request is being served).

We see that the probability π_{10}, which is close to 0.4 in the absence of errors, drops significantly with the error variance. This means that the effect of errors is to lower the load of the edge progressively, which occurs because i) the edge queue size is overestimated by MCCLOUD and ii) some requests are retained by the edge when the edge queue is full (and therefore those requests are lost). However, when there are small errors, MCCLOUD in practice offloads some traffic to the cloud only when the edge queue is almost full (e.g., the figure shows that the queue's level is 80% or more with about 80-90%

TABLE III
VALIDATION: MODEL (M) VS SIMULATION (S) VS TESTBED (T)

Policy		π_F (%)	π_L (%)	$E[T]$ (ms)	Q1 (ms)	Q2 (ms)	Q3 (ms)
GO @{ρ=0.5,	M	1.6	1e-10	66.8	60.1	63.7	70.4
RTT_E=32.5 ms,	S	2.1	0	64.7	57	61	68
RTT_C=56 ms}.	T	3.7	0.2	68.4	59	63	69
GO @{ρ=0.75,	M	1.3	4e-3	64.1	57.1	61.2	68.1
RTT_E=28.4 ms,	S	1.8	2e-3	62.8	56	57	68
RTT_C=51.9 ms}.	T	1.6	1e-3	62.8	56	60	66
R-BAL @{ρ=0.5,	M	1.5	4e-3	64.5	58.8	62.4	69.1
RTT_E=31.9 ms,	S	1.9	4e-3	65.5	55	59	66
RTT_C=55.4 ms}.	T	4.6	6e-2	67.8	58	63	70
R-BAL @{ρ=0.75,	M	1.2	0.1	60.8	54.5	58.4	65.3
RTT_E=27 ms,	S	1.6	0.1	60.2	51	59	63
RTT_C=50.4 ms}.	T	1.2	3e-3	67.8	55	59	65
MCCLOUD@{ρ=0.5,	M	4.2	0.2	66.6	56.7	61.6	72.1
RTT_E=28.7 ms,	S	4.6	0	64	53	58	69
RTT_C=52.2 ms}.	T	4.3	0.1	66.3	56	60	67
MCCLOUD@{ρ=0.75,	M	3.1	0.2	65.1	56.3	60.8	69.9
RTT_E=27.8 ms,	S	3.6	4e-4	63.3	53	58	67
RTT_C=51.4 ms}.	T	4.2	0.3	66.6	57	61	68
LENNON @{ρ=0.5,	M	1.1	1e-13	61.3	54.9	58.7	65.8
RTT_E=28 ms,	S	2.6	0	61.1	52	56	66
RTT_C=51.4 ms}.	T	2.0	0	61.9	55	59	64
LENNON @{ρ=0.75,	M	1.2	3e-5	61.7	55	59	66
RTT_E=27.3 ms,	S	1.8	1e-4	60.5	52	56	64
RTT_C=50.8 ms}.	T	1.2	8e-3	61.2	55	59	64

probability when $\sigma/k < 0.2$, which corresponds to frequent error estimates up to 2 units, for an edge with $k_E = 10$). This can be convenient in terms of latency because the cloud queue is not necessarily long. Therefore the failure probability drops for small errors. However, when the normalized error variance increases above 0.2, MCCLOUD observes more and more losses because the probability to wrongly estimate the edge as non-full becomes high, asymptotically half of the times. So, with $\rho = 1.25$, the loss starts at about 20% of 1.25 (i.e., the part of ρ exceeding 1) in the absence of error, while asymptotically the edge will receive $1.25/2 \cdot 2200$ requests/s, and only serve up to 200, which corresponds to a loss of about 43% of the total offered load, as shown in the rightmost plot of Fig. 8.

The investigation of the effect of errors leads us to conclude that in order to reap the (small) performance benefits of algorithms that leverage the information about the edge and cloud state, it is necessary to guarantee the accuracy and timeliness of the information used by the algorithm. In the many practical cases in which accuracy and timeliness of the information about the system state are not possible (or too costly) to obtain, state oblivious algorithms, such as R-BAL allow almost equal performance to be reached.

Finally, Table III presents results for failure and loss probabilities, average latency, and first, second and third quantiles of latency, obtained with the analytical model (M), simulation (S), and testbed (T) for the policies GO, R-BAL, MCCLOUD and LENNON, at loads ρ=0.5 and ρ=0.75.

We can observe that, in spite of the differences among the approaches, results are in very good agreement. Note that, to compute these results, we have first run testbed experiments and recorded the latency while sending very few requests, so as to capture the average time to reach the servers in absence of congestion (reported in the first column of the table). Afterward, we have simulated and modeled the behavior of the policies under the same operational conditions. The matching between model and simulation is extremely good, which validates our model under ideally stable network conditions. However, some discrepancies can be observed when considering LENNON and, in general, testbed results. These differences are due to two main causes. The first is that the testbed includes the effects of background traffic, which is ignored in model and simulations because, of course, we could not measure the background in the cell and in the Internet in general. We have minimized the impact of background traffic by running experiments at night[1]. The second and most important cause is that in simulations and testbed experiments, the state of edge and cloud is only estimated by observing what flows through the proxy. The magnitude of the error due to state estimation depends on the load, which we have in turn estimated from the RTT between proxy and servers and used in the model. For instance, with a typical RTT of 0.15 ms between proxy and edge, the error resulting from considering exactly 200 services per second instead of the real value—which is a Poisson number with average up to 200—is a r.v. with variance equal at most to 0.03, i.e., $\sigma/k_E = 0.005$, which explains the good match for MCCLOUD results even if our estimate of the error parameter were not too precise. At the cloud, instead, with an RTT of about 25 ms and a capacity of 2000 services/s, the variance of the error can go up to 50, hence $\sigma/k_C = 0.142$, with some uncertainty that justifies a small mismatch in the table.

VII. RELATED WORK

Edge computing aims to bring computing capabilities within the radio access network as close as possible to UEs (see [2], [3] and the references therein). This technology expands the computational capabilities of mobile devices by moving their computations to either cloud servers (usually far from the place where the request is generated) or servers that are close to the end users. Edge and Fog servers are different instances of these servers located close to base stations.

The problem addressed in our paper is called in different ways in the edge/cloud computing literature: computation offloading, server selection, request routing. Different names emphasize different aspects of the problem. The authors of [10] provide an interesting taxonomy of different approaches to computation offloading in edge/cloud systems that allows a precise positioning of our proposal. In particular, our algorithms can be seen as instances of centralized offloading schemes with a cloud infrastructure and a single edge server. Our scheme is centralized because the offloading destination, that is, either edge or cloud, is determined by a decision maker (that in some cases uses information on the status of the servers). Although the literature on offloading strategies is quite wide and varied (see for instance the references in [10]),

[1]Minimizing the impact of background traffic is necessary, although our model and simulator can account for it, because we cannot measure or control background traffic in the testbed, and thus we cannot study equivalent configurations with model, simulation and testbed, unless the background traffic is close to zero.

due to lack of space, in this section we will mostly refer to papers dealing with centralized offloading.

Several approaches were proposed to optimize offloading decisions. In particular, [11] proposes an algorithm to optimize tasks scheduling on the edge server with the objective of minimizing the execution delay. Other proposals, such as [12] and [13] jointly optimize offloading and power allocation. Other studies take into account different aspects, such as for instance [14], that proposes a joint optimization of computing resources, channel allocation, and transmission power to exploit the trade-off between energy consumption and execution latency. The papers [15] and [10] use queuing models for an optimization of energy consumption, and execution delay.

Scenarios with several edge servers were also studied with the aim of optimizing execution delay, energy consumption (due to both edge servers and mobile devices), and in some cases costs for mobile devices. A non-exhaustive list of such proposals includes, for example, [15] that proposes queueing models for a (multi-objective) optimization of (average) latency and energy consumption. The paper [16] proposes a game theory-based framework that exploits the energy harvesting capabilities of mobile devices and their "social" relationships to derive a computation offloading scheme that optimizes execution costs. Also the paper [17] proposes an offloading scheme aiming at minimizing energy consumption. The main peculiarity of this research effort concerns the exploitation of the heterogeneity of edge nodes.

In [18] a joint optimization of the service placements (i.e., the association of edge servers with base stations) and on the routing of the service requests is presented. The decisions (i.e., service placement and request routing) are taken by assuming that demand is fixed and predicted. The variability in service demand can be accounted for by means a set of changes with respect to the original set of decisions.

Furthermore, it is worth to point out that most of the proposals that appeared in the literature assume that service request patterns are known, and based on this knowledge implement different optimizations. Unlike our work, these approaches are however not suitable for the *online* management of arriving requests. Another remarkable difference between the approaches proposed in the literature and the content of this paper concerns the possibility of managing deadlines (or timeouts, or any latency quantile), and this is certainly one of the novelties of our approach.

One fundamental point that we address in our study concerns the relation and the trade-offs between server selection strategies, necessary knowledge and metrics used by those strategies, and the unavoidable inaccuracy of such metrics. Similar goals inspired the papers [19] and [20].

VIII. Conclusions

We studied a slice of a radio access network serving mobile users that issue latency-constrained service requests implying computing requirements. Service can be provided by either a powerful but distant cloud infrastructure or a closer but less powerful edge computing host. We investigated the performance of three families of online algorithms for the routing of service requests to either edge or cloud servers. The three families of algorithms assume different levels of information about the system state, and we discussed the impact of inaccuracy on such information. Our study is based on detailed analytical models, which are validated through OMNET++ simulations and experiments in the wild, exploiting the facilities of the MONROE testbed.

The most important takeaway of our paper is that state-oblivious algorithms, that only require an estimate of the long-term load of requests, can provide comparable results to more complex state-based algorithms in the case of perfect information, but superior performance in the case of inaccurate state information.

References

[1] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing," in *Proceedings Global Internet of Things Summit (GIoTS)*, 2017.

[2] M. Patel and et al, "Mobile-edge computing introductory technical white paper," ETSI White paper, 2014. [Online]. Available: https://portal.etsi.org

[3] S. Kekki and et al, "MEC in 5G networks," ETSI White paper, 2018. [Online]. Available: https://portal.etsi.org

[4] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proc. of Simutools '08*, 2008.

[5] M. Rajiullah and et al, "Web Experience in Mobile Networks: Lessons from Two Million Page Visits," in *Proc. of the World Wide Web Conference (WWW)*, 2019.

[6] P. Castagno, V. Mancuso, M. Sereno, and M. Ajmone Marsan, "A Simple Model of MTC Flows Applied to Smart Factories," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[7] L. Cominardi and et al, "Understanding QoS Applicability in 5G Transport Networks," in *IEEE Int. Sym. on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2018.

[8] T. J. Ott, "The Sojourn-Time Distribution in the M/G/1 Queue with Processor Sharing," *Journal of Applied Probability*, vol. 21, no. 2, pp. 360–378, 1984. [Online]. Available: http://www.jstor.org/stable/3213646

[9] 3rd Generation Partnership Project, "Physical layer procedures for control (Release 16)," 3GPP TS 38.213, Jul. 2020. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3215

[10] F. Sufyan and A. Banerjee, "Computation Offloading for Distributed Mobile Edge Computing Network: A Multiobjective Approach," *IEEE Access*, vol. 8, pp. 149 915–149 930, 2020.

[11] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. of IEEE Int. Symp. Inf. Theory (ISIT)*, 2016.

[12] Z. Kuang and et al, "Partial Offloading Scheduling and Power Allocation for Mobile Edge Computing Systems," *IEEE IoT J*, 2019.

[13] F. Shan, J. Luo, J. Jin, and W. Wu, "Offloading Delay Constrained Transparent Computing Tasks With Energy-Efficient Transmission Power Scheduling in Wireless IoT Environment," *IEEE Internet of Things J*, vol. 6, no. 3, pp. 4411–4422, 2019.

[14] J. Zhang and et al, "Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks," *IEEE Internet of Things J*, vol. 5, no. 4, pp. 2633–2645, 2018.

[15] L. Liu and et al, "Multiobjective Optimization for Computation Offloading in Fog Computing," *IEEE Internet of Things J*, vol. 5, no. 1, pp. 283–294, 2018.

[16] L. Liu, Z. Chang, and X. Guo, "Socially Aware Dynamic Computation Offloading Scheme for Fog Computing System With Energy Harvesting Devices," *IEEE IoT J*, 2018.

[17] G. Zhang and et al, "FEMTO: Fair and Energy-Minimized Task Offloading for Fog-Enabled IoT Networks," *IEEE IoT J*, 2019.

[18] K. Poularakis and et al, "Service Placement and Request Routing in MEC Networks With Storage, Computation, and Communication Constraints," *IEEE/ACM Trans. on Net.*, vol. 28, no. 3, 2020.

[19] B. Gao and et al, "An Online Framework for Joint Network Selection and Service Placement in Mobile Edge Computing," *IEEE Trans on Mobile Comp*, 2021.

[20] G. Zou and et al, "Towards the optimality of service instance selection in mobile edge computing," *Knowledge-Based Systems*, 2021.