Optimal Scaling and Offloading for Sustainable Provision of Reliable V2N Services in Dynamic and Static Scenarios

Livia Elena Chatzieleftheriou, *Member, IEEE,* Jesús Pérez-Valero, Jorge Martín-Pérez, Pablo Serrano, *Senior Member, IEEE*

***Abstract*—The rising popularity of Vehicle-to-Network (V2N) applications is driven by the Ultra-Reliable Low-Latency Communications (URLLC) service offered by 5G. Distributed resources can help manage heavy traffic from these applications, but complicate traffic routing under URLLC's strict delay requirements. In this paper, we introduce the V2N Computation Offloading and CPU Activation (V2N-COCA) problem, aiming at the monetary/energetic cost minimization via computation offloading and edge/cloud CPU activation decisions, under stringent latency constraints. Some challenges are the proven non-monotonicity of the objective function and the no-existence of closed-formulas for the sojourn time of tasks. We present a provably tight approximation for the latter, and we design BiQui, a provably asymptotically optimal and computationally efficient algorithm for the V2N-COCA problem. We then study dynamic scenarios, introducing the Swap-Prevention problem, to account for changes in the traffic load and minimize the switching on/off of CPUs without incurring into overcosts. We prove the problem's structural properties and exploit them to design Min-Swap, a provably correct and computationally effective algorithm for the Swap-Prevention Problem. We assess both BiQui and Min-Swap over real-world vehicular traffic traces, performing a sensitivity analysis and a stress-test. Results show that (i) BiQui is near-optimal and significantly outperforms existing solutions; and (ii) Min-Swap reduces by a $\geq 90\%$ the CPU swapping incurring into just $\leq 0.14\%$ extra cost.**

***Index Terms*—Vehicle-to-Network, V2N, Ultra-reliable Low-Latency Communications, URLLC, Queueing Theory, Algorithm design, Optimization problem, Asymptotic optimality.**

I. Introduction

Network intelligence has emerged as a pivotal goal for 6G. In conjunction with recent advancements in the automotive sector, Vehicle-to-Network (V2N) applications attract significant interest from both academia and industry [2]. A notable instance of V2N communication is Tele-operated Driving (ToD), wherein vehicles are remotely controlled by operators who rely on inputs from the vehicles, including augmented video feeds from the vehicle's front camera that highlight recognized objects or obstacles to be avoided.

For safety-critical applications like these, an Ultra-Reliable Low Latency Communication (URLLC) service is indispensable [3] because it will ensure that the delay experienced by any task between the vehicle and the network remains below a specific threshold with a probability exceeding a predefined reliability threshold. For example, ToD services mandate that transmissions are completed within 100 ms with a 99.999% reliability [3], encompassing transmission and propagation delays, as well as the sojourn time (*i.e.*, waiting plus service time) at the servers. Given the stringent nature of these requirements, any delay could potentially result in vehicular crashes, including those involving pedestrians.

To ensure timely processing of vehicular tasks, these can be offloaded to servers located either in the cloud or at the network edge. While cloud servers offer greater computational power, their relative distance from the vehicles could introduce significant delays. On the other hand, edge servers, colocated with Road-Side Units (RSUs) along roads, can provide more immediate services, albeit at a potentially higher cost or with less computational power [4].

This presents a challenging trade-off due to the dynamic scaling of computing resources. Dimensioning the system to handle peak traffic ensures URLLC service availability but leads to resource wastage during off-peak hours. Adapting resources based on demand could result in significant savings for service providers, yet maintaining URLLC guarantees for vehicular applications remains paramount. The challenge is scaling computing resources effectively, determining the appropriate number of processing units at both edge and cloud that guarantees an appropriate performance. Existing approaches often fall short in meeting the stringent latency and reliability constraints of URLLC services. Many works focus on average performance metrics, such as average delay or utilization, not capturing the tail latency, and others fail to provide tractable methods to guarantee URLLC requirements under dynamic load conditions. A detailed discussion of related work is provided in Section VIII.

Our work models the system as an $M/\Gamma/k$ queue, where Γ a mixture of Gamma distributions, and provides a closed-formula approximation of the service time distribution with provably high accuracy. We rigorously characterize the sojourn

L. E. Chatzieleftheriou is with the IMDEA Networks Institute, Spain. J. Perez-Valero is with Universidad de Murcia, Spain. J. Martín-Pérez is with Universidad Politécnica de Madrid, Spain. P. Serrano is with Universidad Carlos III de Madrid, Spain.

This work is an extension of our work in [1].

This is the author's accepted version of the article. The final version published by IEEE is: L. E. Chatzieleftheriou, Jesús Pérez-Valero, J. Martín-Pérez and P. Serrano, "Optimal Scaling and Offloading for Sustainable Provision of Reliable V2N Services in Dynamic and Static Scenarios," in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2025.3605408.

time distribution and develop efficient algorithms that ensure URLLC guarantees while minimizing operational costs, successfully tackling the challenges above. In nutshell:

- We introduce the V2N Computation Offloading and CPU Activation (V2N-COCA) Problem. The problem consists in minimizing operational costs while deciding CPU scaling and edge offloading for URLLC V2N video tasks. We study the structural properties of the V2N-COCA Problem and leverage them to propose BiQui, a provably asymptotically optimal algorithm with low complexity that we extensively evaluate over real-traces.
- We introduce an approximation of the $M/\Gamma/k$ sojourn (*i.e.*, waiting plus service) time, that is required to ensure URLLC requirements but up to now lacked a closed-form expression. We rigorously demonstrate that our approximation works perfectly when targeting V2N video tasks, and we discuss its applicability to other domains.

This paper also presents a substantial extension of our previous conference publication [1]. More specifically:

- We analyze dynamic scenarios where computing load and resource availability can change drastically. We formulate the Swap-Prevention problem to minimize CPU on/off switching frequency, which in turn extends CPU lifetime and reduces costs.
- We study the problem's structural properties, and use these insights to design the Min-Swap algorithm. We also prove its correctness and computational efficiency.
- We extend our previous performance evaluation to analyze Min-Swap's performance under sudden system changes. We also assess V2N-COCA's performance in additional scenarios and conduct a sensitivity analysis.

In §II we present our system model. In §III we introduce and analyse the V2N-COCA problem. In §IV we propose and evaluate our approximation of the tasks' sojourn time at the servers. In §V we design BiQui and analyze its complexity and approximation properties. In §VI we introduce and analyze the CPU Swap-Prevention problem and Min-Swap. In §VII we evaluate BiQui and Min-Swap. In §VIII we discuss the related work, and in §IX we conclude the paper.

II. Static Scenarios: System Model

We illustrate in Fig. 1 the considered scenario: passing vehicles have wireless connectivity through Road Side Units (RSUs) deployed along the road. We provide notation in Tab. I.

Vehicles use a remote driving service where highly-accurate computationally-intensive Artificial Intelligent (AI) algorithms decide which actions vehicles should take based on their surroundings [3].

Vehicles hold a CPU in which some computations can be performed, *e.g.*, to preprocess frames before actually performing heavier AI-related tasks [5]. However, given the complexity of these algorithms and the possible inter-vehicle interactions, the complex AI tasks are offloaded to external resources, either at the edge or the cloud: edge resources might provide shorter Round-Trip Times (RTTs), but there might not be enough computing capacity for peak-hour conditions or they may become too expensive, and therefore cloud resources might be preferable despite their longer RTTs. The main objective of the service provider is to activate the computation resources that minimize the operational costs while ensuring the URLLC service guarantees.

Notation	Description
V	Number of vehicles
T	Maximum delay
P_G	Reliability requirement
l_i	Frame length of task i
λ_v	Frame rate from vehicle v
λ_C and λ_E	Incoming rate at the cloud and the edge
s	Task average service time
$S_c(\cdot)$ and $S_e(\cdot)$	Service time at the cloud and the edge
D_{sojo}	Sojourn time to process the task
D_{tran} and D_{prop}	Transmission and propagation delay
D_i	Total delay experienced by task i
C and E	Maximum available CPUs at cloud and edge
c_{0c} and c_{0e}	Subscription cost at the cloud and edge
c_{1c} and c_{1e}	Usage cost at the cloud and edge
x and y	CPUs activated at the edge and cloud
z	Offloading policy
Ω	Feasibility region

TABLE I: Notation Table

URLLC application and offloading decisions. Each vehicle v generates a flow of computing tasks at a rate λ_v, *e.g.*, a flow of video frames taken from a front camera to be processed. For simplicity, we assume a single URLLC service, which implies that all vehicles generate traffic following the same model and there is a single service requirement (however, our analysis can be generalised to capture multiple services with different rates and requirements). We assume that the URLLC service requires that for each task i, its total delay $D_i(\cdot)$ must be less than a maximum delay T with at least P_G probability [3]. This can be formalised as

$$\mathbb{P}(D_i(\cdot) \leq T) \geq P_G, \forall \text{ task } i. \tag{1}$$

We will refer to T as the delay requirement and to P_G as the reliability requirement. Following [6], the length l_i of frame i follows a distribution that is specific to the service type and video format. The terms "frames" and "tasks" will be used interchangeably. In order to avoid conflicts with apps with less stringent delay applications, URLLC services in practice can be assigned to a dedicated slice in the network [7].

To ensure a high-quality URLLC vehicular service, we must preserve the order of the packets within each flow (*i.e.*, for each vehicle). This can be ensured by implementing a per-flow traffic split between the edge and the cloud at the RSU, using *e.g.*, flow hashing [8]. In practice, this implies that the computing load generated by a portion $z \in [0, 1]$ of flows is offloaded to the cloud, and the computing load of the remaining portion $(1 - z)$ of flows is executed at the edge[1].

This work focuses on Vehicle-to-Network (V2N) services, which rely on external network resources for computation. Accordingly, we adopt a two-tier architecture for offloading decisions, excluding a third tier that would account for local vehicle-based computing. A representative example is

[1]For cases that the offloading decision may not result in integer solutions, *e.g.*, when $z = 0.5$ and the number N of vehicles is odd, it will be needed to round z to the closest value that results in an integer split of vehicles to the edge or to the cloud, and a confirmation or adjustment that the activated CPUs can handle the incoming computational load after this rounding.

infrastructure-assisted driving, which uses roadside infrastructure to improve autonomous vehicle performance and safety. These services require global road information, which is unavailable within the vehicle itself. Therefore, we focus on data that must be offloaded to external resources, rather than locally processed information.

Let λ_E and λ_C be the incoming computing demand at the edge and the cloud server, respectively. Both λ_E and λ_C are, naturally, functions of our offloading decisions z and the number V of vehicles, and can be computed as:

$$\lambda_E(z,V) = V\lambda_v(1-z) \quad \text{and} \quad \lambda_C(z,V) = V\lambda_v z. \tag{2}$$

Computing resources, activation decisions, service model. In practice, edge facilities comprise areas of 10-20km [9], hence one edge pool covers a whole urban area. We assume that there are up to E CPUs available at the edge, and up to C CPUs available at the cloud, whose computing capacity is dedicated to the URLLC vehicular app. We denote by $x \in \{1,\dots,E\}$ and $y \in \{1,\dots,C\}$ the number of CPUs to be activated at the edge and at the cloud, respectively, which correspond to our activation decisions. Assuming that the number of cycles required to process a task is linearly proportional to its length [10], [11] with constant c cycles/bit, the service time to process a task of length l_i in a CPU with computational capacity c_0 cycles per unit of time equals

$$S(l_i) = {}^{l_i \cdot c}/_{c_0}. \tag{3}$$

Edge and cloud servers as M/G/k queues. We assume that there are a *large enough* group of V vehicles using the URLLC service, each one generating an independent flow of tasks, *e.g.*, a video flow. In practice, this will be the most common scenario in a few years, since most of the vehicles are now manufactured with such features. Under these conditions, the Palm-Khintchine Theorem ensures that the aggregated video arrival process follows a Poisson process at a rate $\lambda = V\lambda_v$. Given that the offloading mechanism is based on hashing [8], the resulting flows towards the edge and the cloud are also two Poisson process (at rates λ_E and λ_C, respectively) since they are the result of a random *thinning* of a Poisson process. Frames at each server are enqueued using a publish/subscribe protocol, and thus each CPU processes frames in a sequential fashion. As a result, both the edge and the cloud servers can be modeled as two different $M/G/k$ systems [12]. More specifically, since x and y denote our CPU activation decisions at the edge server and the cloud server, respectively, the queuing system at the edge as is an $M/G/x$ queue with arrival rate $\lambda_E = \lambda(1-z)$ and service rate μ_E, and the queueing system at the cloud is an an $M/G/y$ queue with arrival rate $\lambda_C = \lambda z$ and service rate μ_C, where μ_E and μ_C depend on the computational capacity of each location's processing units. The $M/G/k$ queue is one of the most general models and does not have closed-form expressions to characterize the tasks sojourn time (apart from approximations, *e.g.*, *Kingman's law*). In § III we present our approximation to characterize the CDF of the sojourn time.

Total delay D_i experienced by a task i. It is defined as the total time since the task is generated, processed, and sent back to the vehicle. It can be expressed as a function of the number of vehicles V in the system, and the service provider's activation and offloading decisions, *i.e.*, variables x, y and z, respectively. Formally:

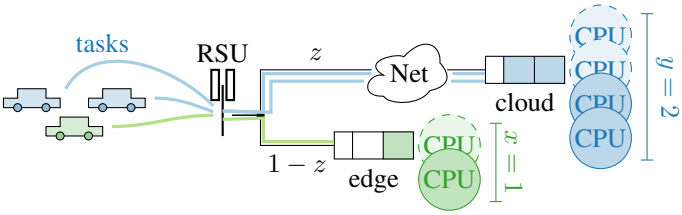


Fig. 1: Vehicles produce tasks. We offload the task flows either to the cloud (with probability z) or to the edge (with probability $1-z$). We process the frames by activating $x = 1$ CPU at the edge or $y = 2$ CPUs at the cloud. Maximum available CPUs: $E = 2$ (edge) and $C = 4$ (cloud).

$$D_i(V,x,y,z) = D_{tran}(l_i) + D_{prop} + D_{sojo}(V,x,y,z), \tag{4}$$

where $D_{tran}, D_{prop}, D_{sojo}$ the radio transmission delay, radio-to-server propagation time (back & forth), and task sojourn time (*i.e.*, waiting plus service time), respectively.

We ensure transmission latencies D_{tran} in the order of few millisenconds through a 3GPP-compliant NR deployment – see [13, Sec. 7] and [14, Sec. 5]. Moreover, we guarantee communication reliability with Type B frame repetitions [15, Sec. 6.1.2.3.2]. Using a dedicated V2X slice for the service provider, propagation delays D_{prop} can also be bounded. Thus, we focus our analysis on the complex $D_{sojo}(V,x,y,z)$, and we perform a sensitivity analysis w.r.t. D_{prop} and D_{tran} in Sec. VII.

Infrastructure costs. Our analytical framework is able to capture both different economic models for the cost of the infrastructure usage, and its energy consumption. Regarding the energy consumption, the literature [16] identifies two main components: (i) an energy consumption term caused by the activation of the servers, which is proportional to the number of activated CPUs, and (ii) an energy consumption term that is proportional to the time the servers are busy with executing tasks, *i.e.*, the service time.

Both terms depend on the type of CPU (In Section VII, we analyze the impact of using faster, more expensive cloud CPUs in terms of activation and energy consumption) Regarding the cost of the infrastructure usage, current pricing plans [17] also take into account two terms: (i) a "subscription" cost for accessing a number of resources, which is proportional to the number of activated CPUs, and (ii) a "usage" cost that is proportional to the time the servers are used. Based on the above, we define the total cost K as a linear combination of the number of activated CPUs and their service time as follows:

$$K(V,x,y,z) := c_{0e}x + c_{0c}y + c_{1c}\lambda_C(z,V)s + c_{1e}\lambda_E(z,V)s, \tag{5}$$

where the constants c_{0e} and c_{0c} capture the subscription cost per CPU at the edge and the cloud, respectively, the constants c_{1e} and c_{1c} capture the usage cost per time unit at the edge and the cloud, respectively, and $\lambda_C(z,V)s, \lambda_E(z,V)s$ represent the product of the incoming rate (at the cloud and edge) by the service time. Specifically, $s = S(\overline{l_i})$ denotes the service time for the average packet length $\overline{l_i}$, *i.e.*, the average service

time. Our cost function is specific enough to accurately capture existing energetic and monetary costs. At the same time, it is generic enough to allow for specific particularizations to be incorporated in it, *e.g.*, by considering a variety of specific functions for the service time, such as a that in eq. (3) or a fixed time per task (irrespectively of its length).

Operational expenses. In (5) we capture the infrastructure cost for a given configuration x, y, z. Later, in Section VI, we also capture the long-term operational expenses introducing the time dependency in the configuration variables $x(t), y(t), z(t)$. Specifically, in Section VI we consider (i) the operational expenses due to switching on/off CPUs $|x(t) - x(t-1)| + |y(t) - y(t-1)|$; and (ii) the infrastructure cost over time $K(x(t), y(t), z(t))$.

Remark. In our model we traded simplicity and the ability to accurately capture the intricacies of real-world applications, judiciously deciding to keep our model as lightweight as possible. We provide more specific details regarding a wide range of relevant factors in Section VII, where we use real-world [18] infrastructure delays D_{tran} and D_{prop} and vary the maximum delay T, the reliability requirement P_G, the CPUs' computational capacity K, and the incoming demand $\lambda_v V$.

III. The V2N Computation Offloading and CPU Activation Problem

We now introduce our optimization problem, and next analyse its structural properties and associated challenges.

Optimization problem. We formalize it as follows:

Problem 1 (V2N Computation Offloading and CPU Activation (V2N-COCA) Problem)**.**

$$\min_{x,y,z} \quad K(V,x,y,z) \tag{6}$$

$$\text{s.t.} \quad \textit{Eq.}\ (1),$$

$$x \in \{0,1,\dots,E\}, \quad \textit{and} \quad y \in \{0,1,\dots,C\}, \tag{7}$$

$$z \in [0,1]. \tag{8}$$

The objective in (6) corresponds to finding the CPU activation decisions x and y and offloading decision z that minimise the total operational cost defined in (5). Eq. (1) ensures the URLLC requirements of vehicular applications are met. Eq. (7) capture the upper bounds on the available CPU resources at the edge and the cloud. Finally, Eq. (8) describes the offloading decision z as a ratio in $[0, 1]$.

V2N-COCA Problem's Structural Properties. We use these in Sec. V, to design and analyse our computationally efficient and asymptotically optimal algorithm. It holds that:

Proposition 1. *The cost function in* (5) *is increasingly monotone w.r.t. CPU activation decisions x, y.*

This proposition implies that the minimum cost will be in the boundary of the feasible region w.r.t. activation decisions. We provide the proof of Proposition 1 in Appendix A.

We next study the feasibility region, with our motivation being to better understand which points (*i.e.*, decisions) would be preferable as solutions, to drive the design of our algorithm. The feasibility region Ω of Problem 1 is defined as

Param	Value
T	100 ms
P_G	99.999%
D^e_{prop}	18.2 ms
D^c_{prop}	22.8 ms
c_{0c}, c_{1c}	15, 2
c_{0e}, c_{1e}	30, 4
s_i, λ	22.5, 1.87 ppms

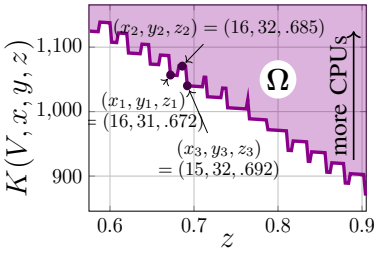


Fig. 2: Total costs *vs.* offloading decision z for the feasibility region and boundary, depicted for an instance considering a ToD service [3] and real-world propagation delays [18], characterized by the values of the table at the left.

$\Omega := \Big\{(x,y,z) : (1), (7), (8)\Big\}$, *i.e.*, intuitively, as the set of all those combinations of decisions (x, y, z) that meet the URLLC requirements, upper bounds on available computing resources, and percentage of offloaded computing load. We illustrate in Fig. 2 the feasibility region for the scenario described in its caption. In general, it holds that:

Proposition 2. *For the feasibility region Ω dictated by eqs.* (1)*,* (7)*, and* (8)*, it holds that:*

1) *It is not continuous w.r.t. the offloading policy z. In fact, its boundary is a step function.*
2) *It is not monotone w.r.t. the offloading policy z.*
3) *Between two consecutive steps, the boundary is linear w.r.t. any offloading policy z.*

Prop. 2 implies that we cannot exploit the continuous nature of the offloading decisions z to find the optimal solution. Still, point (3) will be useful to design an asymptotically optimal algorithm. We provide the proof of Prop. 2 in Appendix B.

Challenges for the use of the optimal solution and for the design of an efficient solution. The V2N-COCA Problem is a mixed integer programming problem, with non-monotone objective function w.r.t. one of its decisions (Prop. 2). An additional major challenge for finding the optimal solution or for the design of efficient algorithms to solve it, is the quantification of the total delay $D_i(V, x, y, z)$ experienced by any task i (defined in Eq. (4)), which is the first of the problem's constraints, ensuring that URLLC requirements are met. More specifically, the challenge stems from the fact that, to the best of our knowledge, no closed-form expressions are available to quantify the sojourn time $D_{sojo}(V, x, y, z)$ in M/G/k systems. Using the optimal solution or designing an efficient one requires knowing the sojourn times of tasks in the M/G/k queues at the edge and cloud servers. As there are not closed-formulas for these quantities, using the optimal solution implies obtaining the actual sojourn times for all scenarios and configurations, to then do an exhaustive search over them in order to conclude to a decision. Indicatively, as we mention in Section VII, in our experimentation, obtaining the sojourn times for all configurations of a scenario took in the order of hours. Clearly, this is not a practical approach to follow in real systems, because of their dynamic nature, in which the traffic load may change throughout the day or

Type	w_i	α_i	β_i
I	$1/12$	16.487	21499
B	$5/12$	15.584	14608
P	$6/12$	17	15895

TABLE II: Parameters for frame length distribution l_i [6]

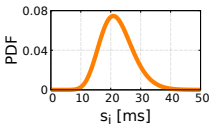


Fig. 3: PDF of the service time s_i

computing resources outages may occur. At the same time, the existing approximations for the M/G/k average waiting time do not suffice in URLLC, and we will compare against them in section VII. However, due to the safety concerns involved in V2N applications, the distribution of the sojourn time in the M/G/k system to ensure that V2N tasks are timely processed, respecting both the target delay and the reliability requirements. We now present and validate our approximation for the M/G/k sojourn time distribution.

IV. Our Sojourn Time Approximation

We first present initial thoughts stemming from experimental evidence, we then discuss the structure of the URLLC requirement and our approximation proposal, and we validate it both w.r.t. accuracy and impact when solving Prob. 1.

Remark: We focus on V2N computing tasks performed over video frames taken *e.g.*, from the front/back camera of autonomous vehicles [5], and we rely on evidence from the real-world traces in [6]. Our sojourn time approximation can be generalized and transferred to any scenario and domain whose data has similar characteristics. Given the generality of our model, it could also apply in different settings. For instance, it could be used for augmented reality applications, such as a remote surgery operation.

Initial thoughts stemming from experimental evidence. Each vehicle v generates an H.264/AVC flow, i.e., a flow of I, B, and P frames arranged in a Group of Pictures (GOP) [6]. The frame length of each type $i \in \{I, B, P\}$ follows a Gamma distribution with shape α_i and scale β_i in Tab. II, and thus the frame length of the flow follows a mixture of them where each type i is weighted with w_i therein. We will call this distribution Γ. Then, the average video frame length is $l = w_I a_I \beta_I + w_B a_B \beta_B + w_P a_P \beta_P \approx 260$ kb. The number of cycles to process a video frame is proportional to its length, with a constant of approx. 21.42 cycles/bit [10]. The service time to process a video frame of length l_i is given by $s_i = l_i \times 21.42/250\,\mu$s, and therefore the average service time per task for a CPU operating at 250 MHz is $s = 22.3$ ms. We depict s_i's Probability Density Function (PDF) in Fig. 3.

The structure of the URLLC requirement in eq. (1) and our approximation of the sojourn time. M/G/k systems are one of the most general frameworks for modelling queuing systems, but there are no closed-form expressions to characterise the distribution of their total sojourn time f_{sojo}, which captures the sum of the waiting and service time. This time equals to the addition of waiting and service times, and therefore its distribution is given by the convolution of the distribution of the waiting time f_W and the distribution of the service time f_S, *i.e.*, , $f_{sojo} = f_W * f_S$. In V2N applications, the service time could relate to the time that is needed to process video frames captured by the vehicle cameras. Such time is directly proportional to the video frame length which, as discussed above, is distributed as a mixture of gamma distributions. Motivated by the small variance of such tasks (see Fig. 3) we make the following proposition to approximate the sojourn times based on the waiting times of an M/D/k:

Proposition 3. *Let Γ be a mixture of Gamma distributions as defined above and with the parameters of Table II. Then, the distribution f_{sojo} of the sojourn time D_{sojo} in an M/Γ/k queue, for video tasks [6], can be approximated with a significance factor $\alpha = 0.01$ by the convolution of the queuing time distribution f_W of an M/D/k queue with deterministic service times, and the service time distribution f_S defined by [6] (i.e., a mixture of Gamma distributions). We formulate this as:*

$$f_{sojo} := f_{W(M/\Gamma/k)} * f_{S(M/\Gamma/k)} \approx f_{W(M/D/k)} * f_{S(M/\Gamma/k)}. \quad (9)$$

The importance of this proposition is two-fold. First, it provides a closed-form expression to approximate with accuracy $\alpha = 0.01$ the sojourn time, *i.e.*, with a 99% confidence level in the estimation. Second, it opens the way for solving Problem 1 by using existing results in the literature. Indeed, we can get $f_{W(M/D/k)}(t) = {}^{d}/_{dt} F_{W(M/D/k)}(t)$ using the following closed-form expression [19, Eq. (4.4)]

$$F_{W(M/D/k)}(t) = e^{-\lambda(kD-t)} \sum_{j=0}^{kc-1} Q_{kc-j-1} \frac{\lambda^j (kD-t)^j}{j!},$$

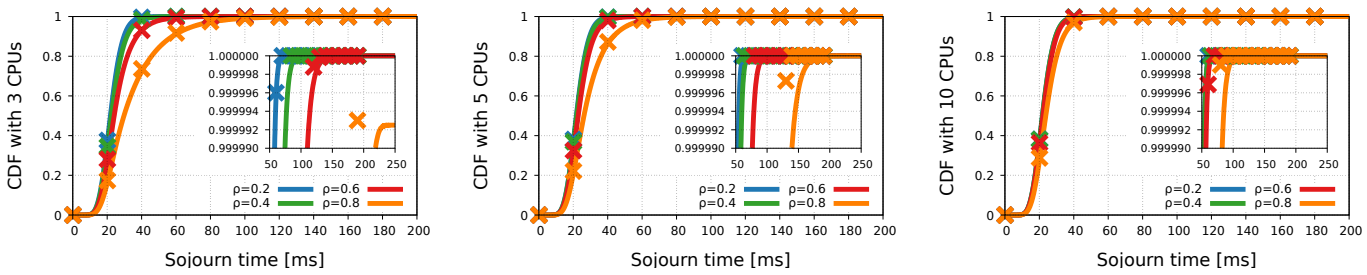
where $k \in \mathbb{N}, t \in [(k-1)D, kD)$, and $D = \mathbb{E}[S(l_i)]$, and Q_j the probability of having up to j tasks in the queue in an M/D/k system [19]. To compute Q_j we leverage the probability p_j of having j tasks in the system, *i.e.*, $Q_j = \sum_{i=0}^{k+j} p_i$. As shown in [20, Sec. 4.5.2], such probabilities exhibit a geometric tail behaviour, meaning $p_j \approx p_M \eta^{j-M}$ for $j \geq M$ and M sufficiently large. In particular, η is the unique solution of $e^{\lambda D(1-1/\eta)} \eta^{-k} = 1$. We compute p_j using (10) below, an expression from [19, (2.2)]. Knowing that $\sum_j^{\infty} p_j = 1$, we have an infinite system of linear equations to solve. We leverage the geometric tail behaviour of p_j to truncate the infinite system to a finite system of M linear equations. According to [20, Sec. 2.3.3], for any $j < M$, the truncated linear system is:

$$p_j = e^{-\lambda D} \frac{(\lambda D)^j}{j!} \sum_{i=0}^{k} p_i + \sum_{i=k+1}^{k+j} p_i e^{-\lambda D} \frac{(\lambda D)^{j-i+k}}{(j-i+k)!} \quad (10)$$

$$1 = \frac{p_M}{1-\eta} + \sum_{j=0}^{M-1} p_j. \quad (11)$$

When $M = 100$, this results in 7-digit precisions for p_j. Having p_j, we obtain Q_j and $f_{W(M/D/k)}(t)$, and we compute the sojourn time approximation f_{sojo} in Prop. 3.

Proposition 3: practical considerations and constraints. Our M/Γ/k model applies to scenarios with a large number of V2N tasks consisting on H.264/AVC video frames, processed sequentially by k CPUs. Under these conditions, task arrivals follow a Poisson process (by the Palm-Khinchin theorem), and service times follow a mixture of gamma distributions, denoted by Γ. Thus, Prop. 3 supports dynamically scaling the computing resources as needed in order to ensure URLLC in (1) based on the computing load, thus resulting in minimization of their



(a) (b) (c)

Fig. 4: Comparison between our sojourn time approximation from Proposition 3 (lines) and $M/\Gamma/k$ simulations (markers). Our approximation is tight and conservative for all loads and CPUs.

operational costs over time. We introduce, analyze, and solve the dynamic scenario in Section VI. We now validate Prop. 3.

Kolmogorov-Smirnov test to rigorously validate Proposition 3. We compare the Cumulative Distribution Function (CDF) of the sojourn time obtained by convoluting the service times of M/Γ/k queueing systems with the waiting times of (i) M/Γ/k systems (obtained through exhaustive simulations), and (ii) M/D/k systems (obtained using eq. (4.4) from [19]). We perform the comparison for different values of the intensity factor $I = \lambda/\mu$ (Erlangs) and number of servers k.

We note that the values considered for k are in the order of magnitude of those required when experimenting with data from a real-trace dataset (see Section VII). We employ the well-established non-parametric statistical Kolmogorov-Smirnov (KS) test, which determines whether two samples originate from the same distribution or not. Our null hypothesis is H_0 : "the CDF from our approximation is slightly below the actual CDF." It implies that our analysis could be more conservative in terms of the URLLC, possibly resulting in a higher probability of being within the target delay, thus possibly being even more stringent than the URLLC requirement – and never resulting in latency not within the URLLC threshold. In the KS test, we accept the null hypothesis H_0 when the p-value is greater than the selected threshold α, which indicates that there is no significant difference between the two CDFs.

Results are presented in Table III, for $k = \{3, 5, 10, 15, 20\}$ servers. In all cases, the p-value remains above the significance level (α), leading to acceptance of H_0. In Fig. 4 we plot our approximation (lines) and the exhaustive simulations (markers), for $k = 3, 5, 10$ servers at the edge and cloud. Note that smaller values of k correspond to larger errors between our sojourn time approximation in Proposition 3 and the $M/\Gamma/k$ simulations. Nevertheless, we observe that our approximation matches the CDF of the sojourn time that is obtained in the $M/\Gamma/k$ simulations for all combinations of CPUs and load in the system. Moreover, Fig. 4 shows that the approximation of Proposition 3 is tight at high percentiles, but also conservative as approximation, resulting in the activation of at most 1 additional CPU (and this only for high loads and the stringiest reliability constraints). This is certified also in Table IV.

The impact of the sojourn time approximation. We evaluate the effectiveness of our approximation of the sojourn time of $M/\Gamma/k$ in Prop. 3, *vs.* the actual sojourn time of the $M/\Gamma/k$, in terms of its results when solving Problem 1.

I (Erlang)	k (servers) $k=3$	$k=5$	$k=10$	$k=15$	$k=20$
$I=1$	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)
$I=2$	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)
$I=4$	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)
$I=8$	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)

Take-away: in all scenarios H_0: "The CDF of Prop. 3 is slightly below the CDF of $M/\Gamma/k$ simulations" is accepted.

TABLE III: Comparison of (KS Statistics, P-value) of the Kolmogorov-Smirnov test for different loads (I) and number of servers (k) with significance level $\alpha = 0.01$.

In Table IV we present the results of different combinations of URLLC requirements expressed in terms of the maximum accepted delay T and the reliability P_G [3] (rows), and different traffic intensities captured as $I = \lambda s$, where s the average service time of tasks (columns). Each pair (k_1, k_2) captures the minimum number of servers needed to guarantee the corresponding requirements in the first row. The results of Table IV suggest that the optimal solution of the V2N-COCA problem found using our $M/\Gamma/k$ approximation of Prop. 3 requires the same number of CPUs as when exhaustive $M/\Gamma/k$ simulations are used, *i.e.*, when having perfect knowledge of the sojourn times in advance. In only a few cases it is conservative, overestimating the number of CPUs to activate by one, which in all cases results in difference ∼1% compared to the oracle. These cases (marked in the table with bold) have high load, high reliability requirement, and low target delay. From the above, we conclude that using the approximation in Prop. 3 leads to near-optimal results and, given that is a closed formula, opens the way for the design of our efficient solution.

Discussion. An alternative would have been to create tables with the solution of the Problem 1, and consult the tables every time that is needed. However, the time that is required by the algorithm to construct the table of results is impractical, as the procedure necessitates evaluating and storing results for all possible scenarios. Even for a single, predefined scenario, identifying the optimal solution is time-intensive and impractical to maintain in table form. Indicatively, the duration for a single scenario is on the order of several hours, highlighting its limitations for real-time or near-real-time applications. On the contrary, our approximation uses a closed-form expression,

Reliability Requirement	Load (Erlang) I=1	I=2	I=4	I=8	I=16	I=32
100ms, 99.999%	(3,3)	(4,4)	(6,6)	(10,10)	(18,18)	(34,34)
100ms, 99.9%	(2,2)	(3,3)	(5,5)	(9,9)	**(18,17)**	(34,34)
50ms, 99.9%	(3,3)	(5,5)	(7,7)	(11,11)	**(20,19)**	**(36,35)**
50ms, 99%	(3,3)	(4,4)	(6,6)	(10,10)	(18,18)	**(35,34)**

TABLE IV: Comparison (k_1, k_2) of the minimum CPUs required by Proposition 3 (k_1) and $M/\Gamma/k$ simulations (k_2) to meet V2N service requirements upon different loads $I = \lambda s$, where λ ranges from 44.44 pkt/sec to 1422.22 pkt/sec and the service time $s = 22.3$ ms [6].

allowing for the solution of Problem 1 within milliseconds (see Sec. V). This rapid computation, combined with our approximation's excellent performance, makes it suitable for highly dynamic scenarios where traffic load fluctuates throughout the day or in situations of computing resource outages. In such cases, resources can be dynamically adjusted to match actual demand, ensuring continuous and URLLC services for vehicles. We will study such scenarios in Section VI.

V. Our Joint Offloading and CPU Activation Algorithm

We now design **BiQui**: an efficient **BI**nary search solution over the **Q**ue**UI**ng theory approximation of Prop. 3 for solving the V2N-COCA Problem. Then we analyse its correctness, computational complexity, and prove its asymptotic optimality.

A. Algorithm Design and Intuitive Explanation

BiQui exploits the objective function's and the feasible set's properties (Prop. 1 and 2), and relies on our closed-form approximation of the task sojourn time in the system (Prop. 3). We now present in detail its steps, which are divided in three phases. For each we provide a high-level rationale and intuition behind it, and refer to the specific lines in Alg. 1.

Phase 1: Initialization. (lines 1). Initialize offloading as $z = 1$ and number of edge CPU activation as $x = 0$, motivated by the typically lower prices of cloud servers vs. those of edge servers [18]. If the opposite holds, the initialization is inverted: $z = 0$ and $y = 0$. We also initialize as empty the set $\mathcal{E} = \emptyset$ of the explored configurations for different offloading policies.

Phase 2: Binary search. We perform it in the number of available cloud CPUs (line 2), to find the minimum number that should be activated to obtain a feasible solution, *i.e.*, a solution that satisfies the reliability constraint. Binary search ensures minimum worst-case computational complexity.

Phase 3: walking down the feasibility region. We exploit Prop. 1, which implies that the minimum cost will be in this boundary. Although offloading decisions z are continuous, we sample for a finite number of discretized values: over $[0, 1]$ and in steps of $z_{gran} \in (0, 1]$, *i.e.*, with a granularity z_{gran}. In Sec. V-B we detail the trade-off that emerges by this discretization choice, and prove BiQui's asymptotic optimality. For each of these $1/z_{gran}$ values we run a `for` loop (line 3), to: (i) increase the activated edge CPUs until the reliability requirement is met (line 4), (ii) decrease the activated cloud CPUs while the reliability requirement is met (line 5). If the current configuration is better than the provisional one, we update the latter (line 6). Lastly, we update all the explored configurations $\mathcal{E}$ that are optimal for every offloading decision z (line 7 in Alg. 1).

Algorithm 1 BiQui

Input: Granularity z_{gran} for partitioning offloading space, number V of vehicles, target delay T, reliability requirement P_G

Output: number x_0 and y_0 of CPUs to be activated at the edge and the cloud, offloading policy z_0, Set $\mathcal{E}$ of explored points in the feasibility boundary

1: Initialize: $z_0 = 1, x_0 = 0$ and $\mathcal{E} = \emptyset$
2: Binary search on y using our $M/\Gamma/k$ approx. in eq. (9)

$$y_0 = \arg\min_{y=1,\dots,C} \{K(V, x_0, y, z_0) : \mathbb{P}(D_i \leq T) \geq P_G\}$$

3: **for** z = 1, . . ., $1/z_{gran}$ **do**
4: **while** (x,y,z) not feasible & $x \in \{0,\dots,$ E-1$\}$ **do** x=x+1
5: **while** (x,y-1,z) feasible and $y \in \{1,\dots,$ C$\}$ **do** y=y-1
6: **if** $K(V, x, y, z) < K(V, x_0, y_0, z_0)$ **then** $(x_0, y_0, z_0) = (x, y, z)$
7: **if** (x, y, z) feasible **then** $\mathcal{E} = \mathcal{E} \cup \{(x, y, z)\}$
8: **end for**

B. Correctness, Computational complexity, and Approximation properties

We now discuss BiQui's correctness, computational complexity, and we finally prove its asymptotic optimality.

Correctness. BiQui provides a correct (*i.e.*, feasible) solution for Problem 1. The reliability requirement imposed in eq. (1) is ensured by Line 2. The upper and lower bounds on the available resources are ensured in lines 4 and 5. The offloading decision z is guaranteed to lie with in $[0, 1]$ because of the for-loop range in line 3.

Computational complexity. We examine the computational complexity of each step of the algorithm, and then conclude to BiQui's total computational complexity. Phase 1 has a time complexity of $\mathcal{O}(1)$. Phase 2 has $\mathcal{O}(\log C)$ (line 2). Note that Ineq. (1) captures the URLLC requirement and its satisfaction is required for feasibility. It involves probability, but our search is not probabilistic: It is a binary search over $\{0, 1, 2, \dots, C\}$, where C the max number of CPUs at the cloud, with $O(\log C)$ worst-case complexity. Every query of the binary search sees if all constraints are satisfied, including (1). Phase 3 is a `for`-loop with $1/z_{gran}$ iterations. Although lines 4-5 are two `while`-loops nested within it, the total number of times that they will run is bounded from the number E and C of max available servers at the edge and at the cloud. The reason is that these lines increase/decrease the number of used resources at the edge and the cloud, respectively. For lines 6 and 7, the complexity of the action to be taken should the condition be positive is $\mathcal{O}(1)$, and it will be run $1/z_{gran}$ times. Thus, in total, for Phase 3 we have $\mathcal{O}(E) + \mathcal{O}(C) + \mathcal{O}(1/z_{gran}) = \mathcal{O}(E + C + 1/z_{gran})$. That is, the total computational complexity of BiQui equals $\mathcal{O}(\log C + C + E + 1/z_{gran})$, *i.e.*, $\mathcal{O}(C + E + 1/z_{gran})$.

Asymptotic optimality. Let $K_{OPT}(V, x^*, y^*, z^*)$ be the cost achieved by the optimal solution of Problem 1 and $K_{BiQui}(V, x, y, z)$ be that achieved by BiQui. The computational complexity of the optimal solution is high, as it requires exhaustively searching the entire solution space. By changing the granularity z_{gran} of partitioning of the decision space for the offloading decisions z, we can create a trade-off between the computational complexity of BiQui and how much it approximates the optimum solution. It holds that:

Proposition 4. *Given the CDF of the sojourn time of tasks in the queuing system, BiQui is asymptotically optimal w.r.t. the offloading decisions z. That is,*

$$\lim_{z_{gran}\to 0} K_{BiQui}(V, x, y, z) = K_{OPT}(V, x^*, y^*, z^*). \quad (12)$$

The sojourn time CDF ensures that the reliability requirement can be handled accordingly. For scarce sojourn time approximation, both the optimal solution and BiQui will deviate from the respective solutions under perfect approximations.

We provide the proof for Proposition 4 in Appendix C.

Remark. Prop. 4 states that BiQui will be optimal in terms of cost as long as $z_{gran} \to 0$, *i.e.*, no optimality gap exists for BiQui. The exact value of z_{gran} in practice is a choice of the application provider (or to whoever uses BiQui), who may want to trade cost for computational complexity. We discuss this also in Fig. 10 and Sec. VII-D. In Prop. 4 the sojourn time CDF is input to BiQui, being either our approximation of Prop. 3 or exhaustive $M/\Gamma/k$ simulations. Hence, both sides of the equation use the same sojourn time CDF. If BiQui used the (unknown in practice) $M/\Gamma/k$ sojourn time CDF, it would be optimal. Using our sojourn time approximation, BiQui activates +1 CPU since, as shown by the KS test and experimental evidence, our sojourn time approximation is conservative. Note that, as we proved above, regardless z_{gran}, **safety** will always be ensured for BiQui from constraint (1) in line 2 of Alg. 1, thus guaranteeing to ensure the safety requirements.

BiQui's adaptability to traffic changes. Although BiQui is a static algorithm—designed for fixed traffic—it is executed periodically in practice to adapt to inevitable traffic variations. As discussed in Section VII-D, we run BiQui every 5 minutes, matching the granularity of our real-world traces. If finer-grained data is available, BiQui can run more frequently, improving its responsiveness. The main limitation on adaptability is computational complexity. We prove that BiQui's complexity is upper-bounded by $O(C + E + 1/z_{gran})$, where C and E are the number of CPUs at the cloud and edge, respectively, and z_{gran} is the decision granularity—indicating linear complexity in system parameters. Execution time is very low in practice: on a 12th Gen Intel(R) Core(TM) i7-1260P, it ranges from 6.29 ms to 119.19 ms, depending on z_{gran} and vehicle–cloud RTT. See Table VI and Section VII.D for details. We do not address triggering mechanisms for executing BiQui outside its periodic schedule, as these are covered in the drift detection literature and are beyond the scope of this work.

We now proceed with the study of dynamic scenarios.

VI. Dynamic Scenarios: Problem Formulation and Algorithm Adaptation

Since continuously switching on/off a CPU might dramatically shorten its lifespan and substantially increase maintenance costs [21], we now study a dynamic scenario and define a swap-preventing problem to be solved after the V2N-COCA, and we design an algorithm to solve the swap-preserving problem. We first present additional model considerations, then proceed with the swap-preventing problem formulation, and we conclude with our algorithm, its computational and correctness properties, and a toy example that makes it clearer.

A. Slotted time and swap tolerance parameter τ

Let time be slotted, and fix the number of vehicles $V(t)$ within each slot t, but let it vary among slots $t = 1, 2, \ldots$. This results in time-varying traffic and, thus, varying computing load, $\lambda(t) = \lambda_v V(t)$. This change in the input when solving Problem 1 may result in a very different CPU activation policy between slots $t-1$ and t. Although this may be optimal in terms of cost, it is not in terms of the lifespan of CPUs.

We define a tolerance parameter $\tau > 0$ in order to control the difference in the cost obtained without and with accounting for the change of the activation status of CPUs, *i.e.*, by solving Problem 1 and by additionally considering the swap-preventing aspect, respectively. The tolerance parameter introduces a trade-off to the problem in the total cost to be payed and the difference in activation decisions: When $\tau = 0$, the tolerance to the allowed difference in the cost is zero, which essentially implies that we will not account for the swap-preventing aspect and we will proceed with the (de)activations that are dictated by BiQui after solving Problem 1. On the other hand, as the value of the parameter τ increases and $\tau \to +\infty$, more costly solutions are allowed as long as they do not change the activation status of the CPUs. In the following we present the swap-preventing problem formulation and analysis.

B. Swap-preventing problem formulation and analysis

Our aim is to minimize the changes in the activation status of the CPUs, provided that it is convenient in terms of cost.

For example, consider that the edge and the cloud costs are the same, and that all the load is being sent to $y(t-1) = 10$ cloud CPUs during slot $t-1$. Let a demand increase occur in the following slot, *i.e.*, $\lambda(t) > \lambda(t-1)$, and let the optimal solution in terms of cost be to send all the load to the edge, *i.e.*, $(x(t), y(t)) = (10, 0)$. An intuitive explanation for this is that lower Round-Trip-Times (RTTs) to send the traffic to the edge instead of the cloud would compensate for the higher service times therein. However, this would imply a large CPU deactivations at the cloud and CPU activation at the edge, *i.e.*, $|x(t) - x(t-1)| + |y(t) - y(t-1)| = 10 + 10 = 20$ CPUs whose activation status will change from slot $t-1$ to slot t.

When taking into account the CPU lifespans and the additional activation delays that in reality CPUs have, it may be better to use an additional CPU at the cloud and none at the edge, *i.e.*, $x(t) = 0$ and $y(t) = 11$.

We formulate the above in the following problem:

Problem 2 (Swap-Prevention Problem)**.** *Let $x^*(t)$ and $y^*(t)$ be the CPU activation decisions at the edge and at the cloud, respectively, and $z^*(t)$ be the offloading policy after solving Problem 1. Let $x(t)$ and $y(t)$ be the CPU activation decisions at the edge and at the cloud, respectively, and $z(t)$ be the offloading policy to be taken in order to prevent CPU activation swaps between slot $t-1$ and slot t.*

Then, we formally aim at:

$$\min_{x(t),y(t),z(t)} |x(t)-x(t-1)|+|y(t)-y(t-1)| \tag{13}$$

s.t.

$$\frac{|K(x(t),y(t),z(t))-K(x^*(t),y^*(t),z^*(t))|}{K(x(t),y(t),z(t))} \leq \tau \tag{14}$$

$Eq.$ (1), (7), (8)

The objective function in (13) captures the sum of the absolute number of CPUs whose status will need to change. Ineq. (14) bounds by τ the cost difference up to which the system is allowed to minimize the CPUs' change of status. Regarding eqs. (1), (7) and (8), they are the same as in Problem 1, and their rationale and interpretation as well.

Swap-Prevention Problem: Structural properties. We focus on the relation of the objective function in (13) w.r.t the activation decisions $x(t)$ and $y(t)$, on the cardinality of the feasibility region. Regarding the objective function, it holds:

Proposition 5. *The objective function of Prob. 2 in eq.* (13) *is non-monotone w.r.t. the activation decisions $x(t)$ and $y(t)$.*

We provide the proof for Proposition 5 in Appendix D.

Although the objective function in eq. (13) is non-monotone w.r.t. activation decisions, the cardinality of the feasibility region of Problem 2 is monotone increasing w.r.t. the tolerance parameter τ. Indeed, in Appendix E we prove that:

Proposition 6. *Let $\Omega'(\tau)$ be the set of configurations from Problem 1 whose cost differ less than a τ percent from the optimal solution. Let $\tau_1, \tau_2 \geq 0$ two tolerance parameters for Problem 2. Then, for the respective feasible regions it holds that:*

$$\tau_1 \leq \tau_2 \Leftrightarrow \Omega'(\tau_1) \subseteq \Omega'(\tau_2). \tag{15}$$

Intuitively, Proposition 6 states that the feasible set $\Omega'(\tau)$ of Problem 2 contains more points as we increase τ. This is because as τ increases we accept more solutions from the feasibility set Ω of Problem 1, because we accept also more expensive in terms of operational costs solutions. As an example, in Fig. 5 we plot the feasible space for the specific parametrization described in its legend. In this case, $\Omega'(0.03)$ contains only 7 feasible points, while $\Omega'(0.12)$ contains 70 points (of which 7 are those contained in $\Omega'(0.03)$).

CPU switching costs. In Problem 2 we aim at minimizing switching on/off CPUs. It is easy to extend the problem formulation to consider the cost of switching on a CPU at the edge/cloud $\nu_x, \nu_y \geq 0$ and switching off a CPU at the edge/cloud $\phi_x, \phi_y \geq 0$. In particular, for $[x]^+ = \max\{0, x\}$, one can choose the following objective function

$$\begin{aligned} U(x(t),x(t-1),y(t),y(t-1)) \\ = \nu_x[x(t)-x(t-1)]^+ + \phi_x[x(t-1)-x(t)]^+ \\ + \nu_y[y(t)-y(t-1)]^+ + \phi_y[y(t-1)-y(t)]^+ \end{aligned} \tag{16}$$

to minimize the long-term CPU switching costs. Note that considering unitary costs $\nu_x = \nu_y = \phi_x = \phi_y = 1$ results in Problem 2. We remark that any instance of Problem 2 using objective function (16) shares the same structural properties, as discussed in Appendix F.

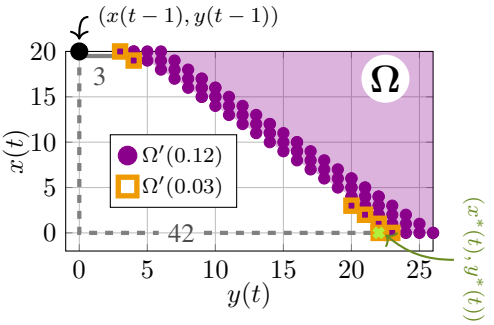


Fig. 5: Feasibility set Ω of Problem 1 and feasible set $\Omega'(\tau)$ of Problem 2 for $\tau \in \{0.03, 0.12\}$. The Manhattan distance is 42 between the solution at time $t-1$ (black dot) and the best in terms of cost solution at the next slot $(x^*(t), y^*(t))$. Whereas the Manhattan distance is 3 with $(x, y) = (3, 20)$, thus, it is minimizes CPU swap.

Algorithm 2 Min-Swap

Input: Tolerance τ, cloud and edge CPUs $x(t-1), y(t-1)$ activated during previous slot, current load $\lambda(t)$, z_{gran}, current CPU and offloading decisions $x^*(t), y^*(t), z^*(t)$ of BiQui, and the boundary set $\mathcal{E}(t)$ explored by BiQui for Problem 1 for slot t.

Output: CPU activation and offloading decisions $x(t), y(t), z(t)$ for slot t

$\Omega = \emptyset$
for $(x, y, z) \in \mathcal{E}(t)$ **do**
  for $y' \in \{y, \ldots, C\}$ **do**
    $z' = z_0 \in \{1, \ldots, 1/z_{gran}\} : (x, y', z_0)$ is feasible
    $\Omega = \Omega \cup \{(x, y', z')\}$
  end for
end for
Find $\Omega'(\tau) := \left\{(x, y, z) \in \Omega : \frac{|K(x,y,z)-K(x^*,y^*,z^*)|}{K(x,y,z)} \leq \tau\right\}$
$(x(t), y(t), z(t)) = \underset{(x,y,z)\in\Omega'(\tau)}{\arg\min} \{|x - x(t-1)| + |y - y(t-1)|\}$

C. Min-swap Algorithm

We aim to minimizing the number of CPUs whose activation status would change from slot $t-1$ to slot t, based on the decisions of BiQui for slot t. The Min-Swap algorithm that we propose takes as its input the outputs of BiQui, *i.e.*, it is a routine that is executed after BiQui solves Problem 1. That is, we first invoke BiQui with the new load $\lambda(t)$ –

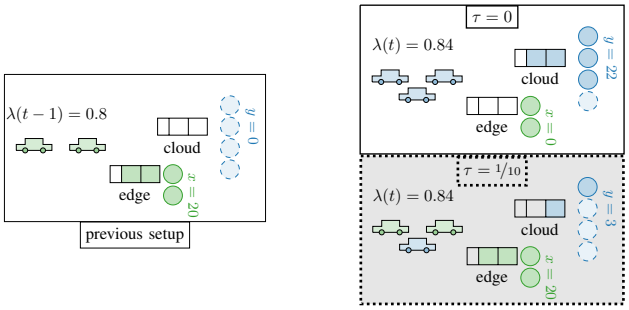


Fig. 6: With same edge/cloud costs, when the load λ changes from 0.8 to 0.84 [pkt/ms] the best is to turn off the $y' = 20$ CPUs at the edge (left) and turn on $x = 22$ CPUs at the cloud (right solid). With tolerances $\tau > 0$, BiQui selects solutions minimizing the CPU (de)activation (right dotted).

respectively $V(t)$ – and obtain the asymptotically optimal in terms of cost solution $x^*(t), y^*(t), z^*(t)$ and the boundary set $\mathcal{E}$ for Problem 1. Then, we invoke Min-swap. We present its pseudocode in Algorithm 2. Min-swap consists of three steps.

Step: 0. Given the boundary set $\mathcal{E}$ explored by BiQui for Problem 1, Min-Swap builds its feasibility set Ω. Specifically, the loop in line 1 checks every configuration in the boundary $\mathcal{E}$ and increases the number of cloud CPUs to $y' \leq C$. For each edge/cloud CPU pair (x, y'), Min-Swap runs a binary search on the offloading decision z to find a feasible configuration (x, y', z') for Problem 1 (Lines 1-7 in Alg. 2).

Step 1: Starting from the feasibility set Ω from Step 0, find the feasible set $\Omega'(\tau)$ for Problem 2 as:

$$\Omega'(\tau) := \big\{(x,y,z) \in \Omega : \frac{|K(x,y,z) - K(x^*(t),y^*(t),z^*(t))|}{K(x,y,z)} \leq \tau\big\}. \quad (17)$$

The feasibility set $\Omega'(\tau)$ consists of all configurations $(x, y, z) \in \Omega$ whose cost differs, at most, a $\tau \geq 0$ percent with respect to the optimal solution (line 8). Due to the monotonicity of the operational cost in eq. (5) w.r.t. activation decisions (Proposition 1), starting from the boundary $\mathcal{E}$ of Problem 1 allows to find faster the feasible region $\Omega'(\tau)$ of Problem 2.

Step 2: Find the Min-swap solution $x(t), y(t), z(t)$ searching at $\Omega'(\tau)$. That is, set $x(t), y(t), z(t)$ to

$$\underset{(x,y,z)\in\Omega'(\tau)}{\arg\min} \{|x - x(t-1)| + |y - y(t-1)|\} \quad (18)$$

That is, Min-swap selects among such configurations $\Omega'(\tau)$ the one minimizing the number of CPUs whose activation status will change (line 9).

D. Min-Swap: correctness and computational complexity

We now discuss the correctness and computational complexity of the Min-Swap algorithm.

Correctness. Min-Swap produces solutions that are correct, *i.e.*, both feasible and ensuring the safety requirements of V2N applications. Indeed, eq. (14) is ensured through Line 1 in Alg. 2. Moreover, because Min-Swap searches its solutions in Ω, also eqs. (1), (7), (8) are ensured from Line 4 of Alg. 2.

Computational Complexity. The computational complexity of Min-Swap depends on: (i) the size of the feasibility region Ω of Problem 1; and (ii) the granularity in the offloading decisions z_{gran}. Building Ω in line 1 takes $\mathcal{O}(C \cdot E \cdot \log(1/z_{gran}))$ because for every edge/cloud CPU pair (x, y'), line 4 executes a binary search on the offloading decision to find a feasible solution (x, y', z'). Then, in line 8, Min-Swaps check every solution of the feasibility region Ω to check whether they excess the cost of the optimal solution by a τ percent. This operation is linear on the size of Ω, *i.e.*, $\mathcal{O}(C \cdot E)$. Lastly, in line 9 Min-Swap selects the solution from $\Omega'(\tau)$ minimizing the CPU swap. By implementing $\Omega'(\tau)$ with a heap data structure [22] and using the min CPU swap as sorting criteria, we ensure the best solution is found in $\mathcal{O}(C \cdot E \cdot \log(CE))$. This is because we look for the min CPU swap solution in a space whose size is $|\Omega'(\tau)| \leq CE$, and the heapify operation takes $\mathcal{O}(\log(C \cdot E))$ run-time for each element in such space. Overall, the worst-case run-time of Min-Swap is governed by either building Ω or looking for the minimum CPU swapping solution. That is, Min-Swap run-time is $\mathcal{O}(C \cdot E \cdot \log(\min\{C \cdot E, 1/z_{gran}\}))$.

E. Toy example: BiQui vs. Min-Swap in dynamic scenarios

Now we present an example of BiQui resulting in a large number of CPUs whose activation status would change from slot to slot if BiQui is used in a real-time. In our example, this happens due to changes in the total computational load λ. Then, we show how our Min-swap algorithm, through solving Problem 2 given a tolerance parameter τ, prevents this effect.

Toy example setup. Let $C = 40$ and $E = 20$ be the maximum available CPUs at the cloud and at the edge, respectively. Let the RTTs and the subscription/usage costs for the edge and the cloud servers be as in Fig. 2. In Fig. 6 we illustrate the optimal solution of Problem 1 for load $\lambda(t-1) = 0.8$ (left), and both and several feasible solutions for $\lambda(t)$ (right).

Change in load leads to high CPU (de)activations. When $\lambda(t-1) = 0.8$, the optimal solution in terms of cost is to send all to the edge, *i.e.*, $(x^*(t-1), y^*(t-1), z^*(t-1)) = (20, 0, 0)$. Without loss of generality, let us assume that this is the decision taken by Min-Swap as well during slot $t-1$. If the load increases to $\lambda(t) = 0.84$ it cannot be handled entirely at the edge, because there are not additional available CPUs therein to activate. It could be intuitively expected that the best action to take is to turn on additional CPUs in the cloud and set $(x(t), y(t), z(t)) = (20, 3, 0.05)$, because of its cheaper price compared to the additional CPU at the edge. However, Fig. 6 shows that the best solution in terms of cost is to turn off all edge CPUs and send all the computational load to the cloud, *i.e.*, $(x^*(t), y^*(t), z^*(t)) = (0, 22, 1)$.

This is a result of the multiplexing gains at the cloud, which will reduce the waiting time and thus will result in the demand being dispatched with less CPUs when all is processed at the cloud. The above results in $|x^*(t) - x^*(t-1)| + |y^*(t) - y^*(t-1)| = 42$ CPUs (de)activations.

What Min-Swap offers. To prevent the high (de)activations explained above, the Min-Swap algorithm searches for solutions in the feasible space $\Omega'(\tau)$ of Problem 2, whose

cost differs at most τ percent from the optimal cost when solving Problem 1. In Fig. 6 (right) we illustrate some of the solutions contained in $\Omega(\tau)$ for $\tau = 1/10$. Among them all, Min-Swap selects the candidate solution that minimizes the CPU (de)activations, which turns out to be $(x(t), y(t), z(t)) = (20, 3, 0.05)$. As a result, the tolerance $\tau = 1/10$ only (de)activates $|x(t) - x^*(t-1)| + |y(t) - y^*(t-1)| = 3$ additional CPUs, *i.e.*, 92.86% less (de)activations compared to the optimal solution, at the expense of being at most 10% worse in terms of cost.

F. Min-Swap with CPU switching costs

We remark Min-Swap is a suitable algorithm to solve Problem 2 considering CPU switching costs, *i.e.*, taking the objective function (16). It suffices to replace line 9 by $\arg\min_{(x,y,z)\in\Omega'(\tau)}\{U(x, x(t-1), y, y(t-1))\}$, *i.e.*, to return the explored feasible solution with smallest CPU switching cost. Still, the loops in lines 2 and 3 must remain given the non-monotonicity of $U(x(t), x(t-1), y(t), y(t-1))$ (see Appendix F). That is, the Min-Swap Alg. 2 must still leverage the boundary set $\mathcal{E}(t)$ and exhaustive search on the number of cloud CPUs. The above description ensures Alg. 2 correctness while considering CPU switching costs. Specifically: (i) the reliability requirement (1) is met for line 2 remains; (ii) the CPU activation (7) and offloading requirements (8) are met for line 4 remains; and (iii) the tolerance requirement (14) is also met for line 8 remains as well.

VII. Performance evaluation

We assess BiQui's performance using real-world traffic traces, pricing plans currently used in the market, and propagation delay setups drawn from the literature. We consider reliability constraints imposed by 5G-Americas [3] for V2N.

A. Setting and Traces

Setting. Unless otherwise specified, we consider $z_{\text{gran}} = 10^{-2}$ granularity, target delay $T_G = 100\,\text{ms}$, and $P_G = 99.999$ % reliability [3]. We consider propagation delays from real-world providers [18]: edge Round Trip Time (RTT) $D^e_{\text{prop}} = 18.2\,\text{ms}$ and cloud RTT $D^c_{\text{prop}} = 22.8\,\text{ms}$. We assume that the vehicular H.264 video flows are processed on AWS EC2 G4 instances, optimised for intense video-processing, and with pricing the hourly cost of EC2 G4 [17]: $c_{0e} = 0.0052, c_{1e} = 1.363\,\$/\text{h}$. Similarly, we consider the EC2 G4 instances price in Regional premises as reference for the cloud pricing, *i.e.*, we take $c_{0c} = 0.0052, c_{1c} = 0.94\,\$/\text{h}$. Unless otherwise specified, the computing resources at the edge and at the cloud are considered to have the same computational capacity, and therefore what essentially distinguishes the edge from the cloud is the number of available CPUs [18, Sec. 2] and their distances to the vehicles.

Real-world Traces. The traffic data [23], recorded by 6 road probes in Torino, includes traffic flow measurements aggregated over 5 minutes. Fig. 13a shows the geographical distribution, with streets represented by points sized according to their maximum traffic intensities. The spots correspond to Corso Belgio, L. Einaudi, Giambone, G. Ferraris, G. Cesare, Francia and G. Agnelli. The impact of the prots' size on the results is that the larger is the spot, the larger is the traffic demand of the respective street, and thus its demand for CPU resources. Fig. 13b depicts the traffic flows over the streets for one entire day. The aggregated demand achieves peaks $\lambda \leq 2.5\,\text{pkt/ms}$ (Fig. 13b), and we thus for our evaluation consider demands λ within $[0, 2.5]$.

B. Benchmarks

We compare BiQui performance against:

OPT: performs an exhaustive search on (x, y, z) simulating $M/\Gamma/k$ queues. Computationally inefficient (needs hours to give a solution), but provides the best achievable performance as benchmark.

AVG: offloads and scales to meet, on average, the delay requirement $\forall$ task i, *i.e.*, $\mathbb{E}[D_i(V, x, y, z)] \leq T$ [24].

KNG: it estimates the average waiting time using Kingman law of congestion [25]. It decides (x, y, z) so the waiting time and P_G percentile of the Γ service time $S(l_{P_G})$ meet the target delay, *i.e.*, $\mathbb{E}[W(x, y, z)] + S(l_{P_G}) \leq T$.

SNC: it adapts [26] to account for Γ distributed service times using Stochastic Network Calculus (SNC). As in [27], [28], we use affine arrival/service curves [29] to bound the arrival/service excess/deficit and take (x, y, z) to meet target delay T and reliability P_G constraints – see Appendix G.

OffAll: it offloads all tasks to the cloud $z = 1$ until it exhausts cloud CPUs. Then, it sends tasks to the edge.

LocAll: it process tasks locally at the edge $z = 0$ until it exhausts edge CPUs. Then, it offloads tasks to the cloud.

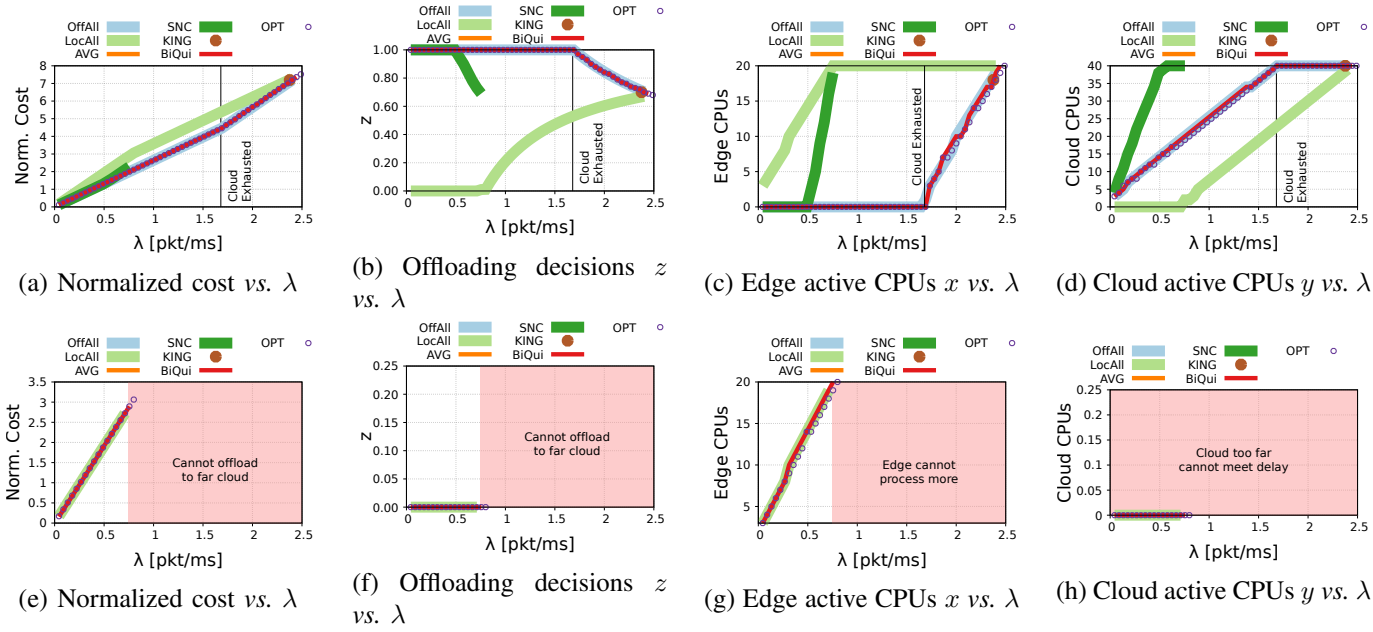
C. Results

We evaluate BiQui against the benchmarks above by doing a sensitivity analysis on the problem's parameters, and over the real traces presented above. We investigate the impact of varying Round-Trip-Times (RTTs), of the target delay T, of the granularity z_{gran} of the offloading decisions, of the reliability requirement P_G, of varying the subscription and usage costs over the total cost, and of the heterogeneity of the computing resources. We also evaluate BiQui over more dynamic scenarios, and finally we study BiQui's execution time under various scenarios.

The impact of varying RTTs. We consider high and low Round-Trip-Times (RTTs) for cloud servers. Fig. 7 depicts normalized costs, offloading decisions z, number of activated edge-cloud CPUs, *vs.* different load λ. Our main observations and insights per scheme are:

OPT: The max. supported computing load is up to $2.5\,\text{pkt/ms}$ for low (Fig. 7a) and up to $0.75\,\text{pkt/ms}$ for large (Fig. 7e) cloud RTTs. High RTTs at some point lead the total delay to exceed the target delay T, leading to infeasible solutions, as the reliability requirement is not met anymore, thus indicating the system's limits.

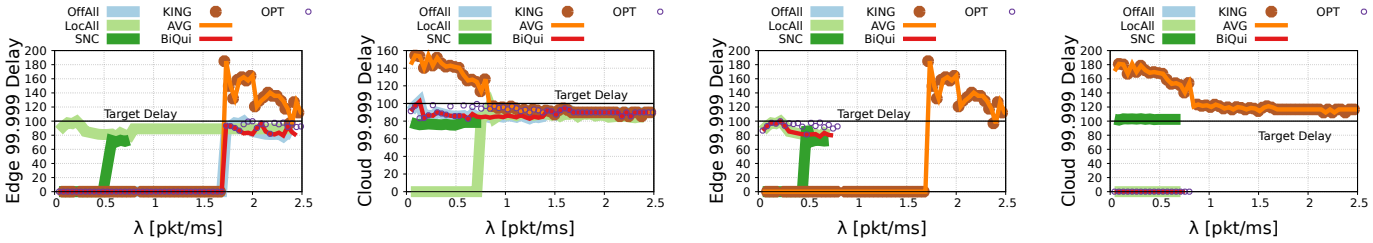
BiQui: it matches the costs and decisions taken by the OPT, for both smaller and larger cloud RTTs. This happens for all the possible computing loads except for those at the very high



(a) Normalized cost *vs.* λ (b) Offloading decisions z *vs.* λ (c) Edge active CPUs x *vs.* λ (d) Cloud active CPUs y *vs.* λ

(e) Normalized cost *vs.* λ (f) Offloading decisions z *vs.* λ (g) Edge active CPUs x *vs.* λ (h) Cloud active CPUs y *vs.* λ

Fig. 7: **Impact of varying server RTT.** Edge RTT: $D^e_{\text{prop}} = 18.2$ ms. Cloud RTT: $D^c_{\text{prop}} = 22.8$ ms (top); and $D^c_{\text{prop}} = 49.1$ ms (bottom). All RTTs are from [18]. Target delay: $T = 100$ ms. Reliability requirement: $P_G = 99.999\%$.



(a) Delay at edge - low cloud RTT (b) Delay at cloud - low cloud RTT (c) Delay at edge - high cloud RTT (d) Delay at cloud - high cloud RTT

Fig. 8: 99.999-th percentile of the delay experienced by a ToD service [3] when it is processed at the edge and cloud. For the edge: $D^e_{\text{prop}} = 18.2$. For low cloud RTT: $D^c_{\text{prop}} = 22.8$ ms. For high cloud RTT: $D^c_{\text{prop}} = 49.1$ ms [18].

limit BiQui, *i.e.*, for $\lambda \to 2.5$ for low RTTs (Fig. 7a) and for $\lambda \to 0.75$ for large RTTs (Fig. 7b). However, the reason for this is the conservativeness of the sojourn time approximation of Proposition 3 (discussed above).

AVG: it is infeasible for all RTTs. From Fig. 8, the 99.999% delay experienced by the tasks processed at the edge or cloud exceed the $T = 100$ ms target delay. This is the main drawback of the existing approaches in the literature, *e.g.*, [24], which fail to capture the strict latency and reliability requirements of V2X services.

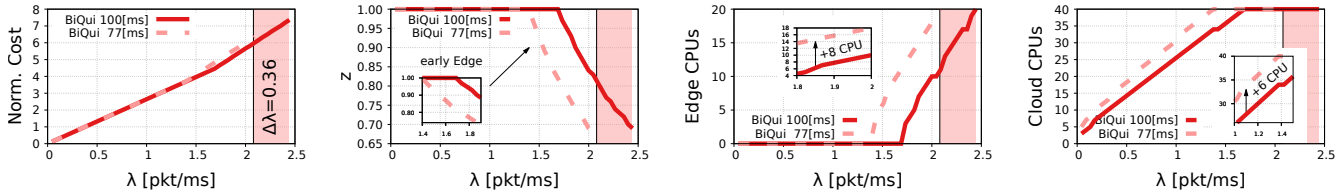
KNG: only finds solutions under small RTT and $\lambda = 2.376$ (see Fig. 7a-d). This is due to the decreasing saw-tooth delay behaviour (see Fig. 8). KNG turns out to be a loose and optimistic approximation of the 99.999% delay, and it reduces the error as λ increases. For accommodating demands $\lambda > 2.5$, CPU setups $C > 40$ are needed.

SNC: it appears too conservative. For low RTT (Fig. 7a-d), it eats up all CPUs with loads $\lambda \leq 0.71$, not finding feasible solutions for higher loads. For high RTT (Fig. 7e-h), it never finds feasible solutions. It provides rather loose bounds for the reliability P_G, hence pitfalls into resource over-provisioning.

OffAll: its behaviour highly depends on RTTs. From Figs. 7a-d (low RTT) it matches OPT, as cloud is cheaper. However, in Figs. 7e-f (high RTT), feasible solutions are not possible due to delay violations stemming from high RTTs.

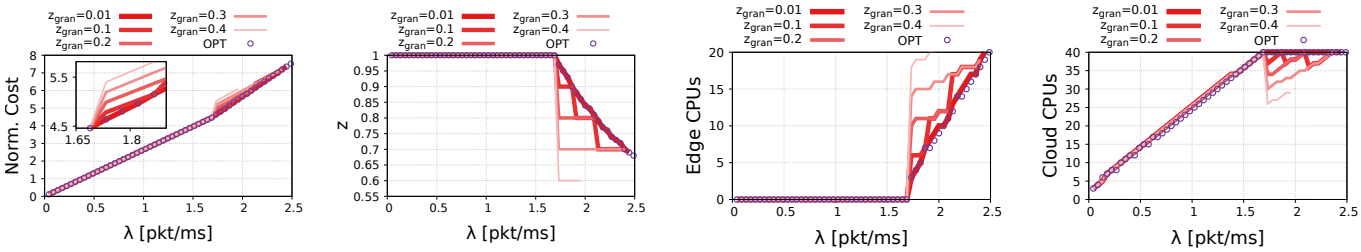
LocAll: is an optimal approach with large cloud RTT (as in Fig. 7e-h), for the only feasible solution is to locally process all tasks at the edge. Upon small cloud RTT (as in Fig. 7a-d), it leads to suboptimal deployments because it does not use first cheap CPUs at the cloud.

The impact of the target delay, T. We investigate on the lower possible target delay that BiQui could handle. This allows us to reveal the potential of our algorithm, while being relevant for more stringent scenarios that may be considered in future 6G applications. In Fig. 9 we depict the normalized cost, our decisions (CPU activation x at the cloud and y at the edge, and the offloading policy z), *vs.* the generated computing load λ. In particular, we depict BiQui for two different target delays: $T = 100$ [ms] and $T = 77$ [ms]. The latter is the minimum target delay for which BiQui can leverage the cloud,



(a) Normalized costs *vs.* λ (b) Offloading z *vs.* λ (c) Activation x (edge) *vs.* λ (d) Activation y (cloud) *vs.* λ

Fig. 9: We stress BiQui to meet a target delay of $T = 77$ [ms] (dashed) with target delay $T = 100$ [ms] (solid) and reliability requirement $P_G = 99.999\%$. Following [18], we use $D^e_{\text{prop}} = 18.2$ [ms] and $D^c_{\text{prop}} = 22.8$ [ms] for the propagation delay at the edge and cloud, respectively.



(a) Normalized costs *vs.* λ (b) Offloading z *vs.* λ (c) Activation x (edge) *vs.* λ (d) Activation y (cloud) *vs.* λ

Fig. 10: Impact of BiQui granularity z_{gran}. Target delay $T = 100\,\text{ms}$ and reliability $P_G = 99.999\%$. We use $D^e_{\text{prop}} = 18.2$ (edge), $D^c_{\text{prop}} = 22.8\,\text{ms}$ (cloud) for propagation delays [18].

λ (pkt/ms)	x (CPU)	y (CPU)	z (%)	K ($/hour)
1	(0, 0)	(31, 26)	(100, 100)	(2.71, 2.70)
1.5	(6, 0)	(40, 36)	(91, 100)	(4.13, 3.99)
2.25	(–, 16)	(–, 40)	(–, 74)	(–, 6.69)

TABLE V: Comparison (a_1, a_2) BiQui's decisions x, y, z and obtained cost K under target delay 77 ms (a_1) and 100 ms (a_2), for different load λ. Reliability: $P_G = 99.999\%$, propagation delays: $D^e_{\text{prop}} = 18.2, D^c_{\text{prop}} = 22.8\,\text{ms}$ [18].

and it is a limit imposed by the system and not by BiQui's performance. That is, with target delays $T < 77$ [ms] in our setting, it is impossible for any algorithm to offload traffic to the cloud while meeting the 99.999% reliability requirement. The intuition behind this is as follows. The cloud RTT for the experiments is 22.8 [ms], hence the sojourn time cannot consume more than $77 - 22.8 = 54.2$ [ms] to meet the target delay. However, it is impossible that the sojourn time remains smaller than 54.2 [ms] with a 99.999% probability.

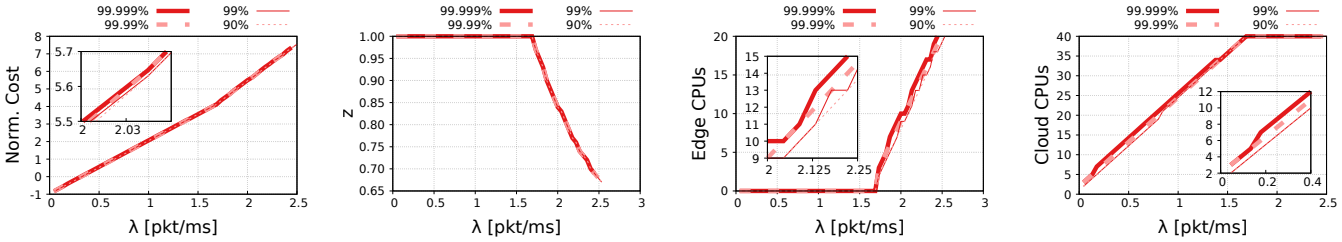
The impact of the offloading granularity, z_{gran}. We use the same experimental setup as that for low cloud RTT. Fig. 10 shows BiQui for $z_{\text{gran}} = \{0.01, 0.1, 0.2, 0.3, 0.4\}$ *vs.* varying computing load λ. See that for $z_{\text{gran}} = 0.01$ BiQui matches OPT, confirming Prop. 4. For $\lambda < 1.73$ BiQui sends all traffic to the cloud until it exhausts the available CPUs $C = 40$. With $\lambda \geq 1.73$ BiQui offloads traffic to the edge ($z < 1$) and turns on edge CPUs. Fig. 10 (c-d) show worse granularities (e.g. $z_{gran} = 0.4$) lead to sudden jumps in the offloading and additional edge CPUs. Such jumps in the number of (expensive) edge CPUs are reflected in the cost increase in Fig. 10 inset. BiQui gives near optimal results and a smooth offloading for finer granularity $z_{gran} \to 0$, confirming again Prop. 4. However, finer granularities lead to larger execution times – see Sec. VII-D.

The impact of the reliability requirement, P_G. We now lower the reliability requirement of the ToD service from $P_G = 99.999\%$ to $P_G = 99\%$ and $P_G = 99.99\%, 99.9\%$, which is the reliability imposed by infrastructure-assisted environment perception and HD map collection [3]. Fig. 11 shows BiQui performance using a near-cloud topology ($D^c_{prop} = 22.8$ms [18]) to process V2N traffic. Results in Fig. 11b show the offloading decision z does not change regardless of the reliability requirement P_G. However, with smaller P_G BiQui saves around 2 CPUs at the edge and cloud (see Fig. 11c-d), thus the subtle cost savings in Fig. 11a inset.

The impact of the heterogeneity of the computing resources. We now study the effect of having heterogeneous CPUs at the edge and cloud, by considering that cloud CPUs have 2x, 4x, 8x and 16x more cycles per time unit than edge CPUs – *i.e.*, parameter c_0 in (3) is larger for cloud CPUs. Hence, cloud CPUs have smaller service time to process a task $s = S(\overline{l_i})$. We also assume that edge cloud costs increase proportionally with the CPU speed, that is, a cloud CPU with 2x more cycles per time unit has subscription and usage costs $2c_{0c}, 2c_{1c}$. Fig. 12a-d show the impact of having cloud CPUs with more cycles per time unit vs. an increasing demand λ.

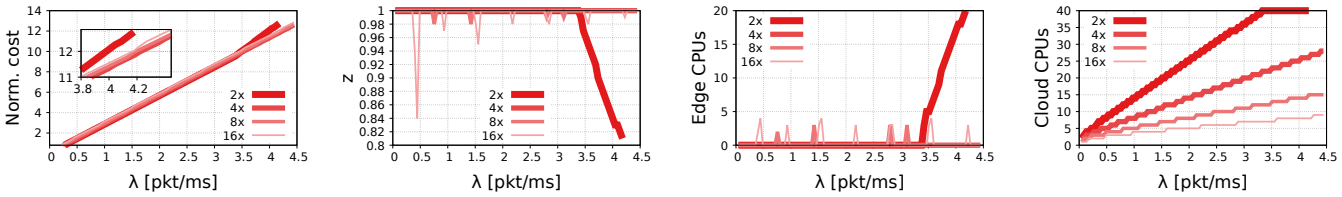
From Fig. 12a we infer that cheap and slow cloud CPUs lead to higher costs with $\lambda = 4$ [pkt/ms]. Specifically, Fig. 12a inset shows that it is cheaper having 16x faster and more expensive cloud CPUs than just 2x faster and more expensive cloud CPUs. The latter case exhausts the 40 and 20 CPUs available at the cloud and edge – see Fig. 12c-d –, thus being more expensive. Additionally, Fig. 12a evidences that the cost difference is not significant when we use 4x, 8x and 16x faster and more expensive cloud CPUs. Note that turning on fewer, yet more powerful, CPUs can compensate for their higher cost.

With 8x and 16x faster cloud CPUs, Fig. 12b-c evidences



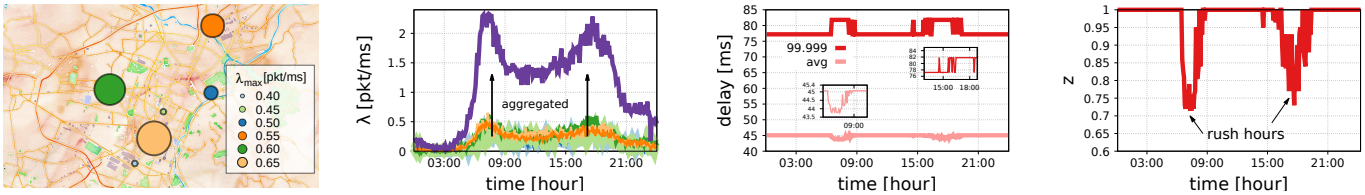
(a) Normalized costs *vs.* λ (b) Offloading z *vs.* λ (c) Activation x (edge) *vs.* λ (d) Activation y (cloud) *vs.* λ

Fig. 11: We vary the reliability requirement and consider $P_G = 99.999$, $99.99, 99$, and 90% imposed to BiQui to meet a target delay of $T = 100$ [ms] for V2N services. Following [18], we use $D^e_{\text{prop}} = 18.2$ [ms] and $D^c_{\text{prop}} = 22.8$ [ms] for the propagation delay at the edge and cloud, respectively.



(a) Normalized costs *vs.* λ (b) Offloading z *vs.* λ (c) Activation x (edge) *vs.* λ (d) Activation y (cloud) *vs.* λ

Fig. 12: **Impact of the heterogeneity of the computing resources.** We increase 2x, 4x, 8x and 16x the cloud CPU computational capacity c_0 and ask BiQui to meet a target delay of $T = 100$ [ms] and reliability $P_G = 99.999\%$ for V2N services. Following [18], we use $D^e_{\text{prop}} = 18.2$ [ms] and $D^c_{\text{prop}} = 22.8$ [ms] for the propagation delay at the edge and cloud, respectively. Cloud costs c_{0c}, c_{1c} increase proportionally with the CPU capacity.



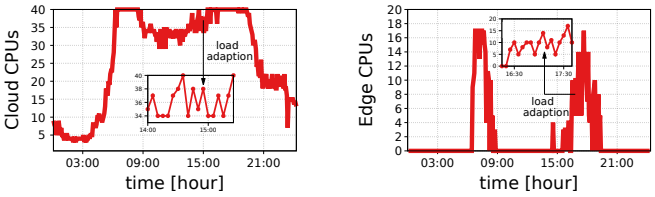
(a) Torino streets (b) Streets demand (c) Average & 99.999 delay (d) Offloading decision z

Fig. 13: BiQui over real traces: The traffic demand (b) corresponds to that of five streets from Torino (a). Results show the delay experienced (c) by a ToD service [3] as BiQui changes the offloading decision (d) throughout a day.

offloading spikes that lead to on/off patterns in edge CPUs. The reason is that BiQui finds cheaper solutions turning on edge CPUs before turning on cloud CPUs that are 8x and 16x more expensive. We recall Proposition 2 to emphasize the observed offloading spikes are inline to the non-monotonicity of the cost function.

Last, for a given demand λ BiQui turns on less cloud CPUs when they are faster and more expensive, since less cloud CPUs meet the target reliability $P_G = 99.999\%$ (Fig. 12d).

BiQui over a day. We assess BiQui performance using the traffic load of Torino city – see Fig. 13a-b. The used dataset reports the traffic load of each road (Fig. 13a) every 5 min and BiQui runs right after each traffic load report. Hereof, BiQui updates the offloading z (Fig. 13c) and CPU activation decisions x, y (Fig. 14a-b) upon reported load changes. Results show that during rush hours (8:00 and 18:00) BiQui exhausts cloud CPU resources (Fig. 14a) and offloads the demand to the edge (Fig. 13d and Fig. 14b) to meet the target delay (Fig. 13c). Leveraging Proposition 3, we evaluate the



(a) Cloud active CPUs (b) Edge active CPUs

Fig. 14: **BiQui over a day.** CPU scaling over the street demand in Fig. 13 (b). BiQui adapts to the load changes, which are reported each 5 min in the dataset.

offloading z and CPU activation decisions x, y and observe in Fig. 13c an increase in the experienced average and 99,999$^{\text{th}}$ delay percentile during rush hours. Still, BiQui meets the 99,999$^{\text{th}}$ percentile of the target delay $T = 100$ ms even during rush hours. We remark BiQui makes decisions in < 120 ms (see TABLE VI) and adapts the CPU activation and offloading

RTTs	z_{gran} 0.01	0.05	0.1	0.2	0.3	0.4
22.8 ms	81.12 ms	17.72 ms	12.72 ms	12.91 ms	10.63 ms	7.24 ms
49.1 ms	119.19 ms	25.05 ms	21.18 ms	7.89 ms	6.67 ms	6.29 ms

TABLE VI: BiQui execution times (in milliseconds) for different values of the granularity parameter z_{gran} and two RTTs between the vehicle and the cloud server.

decisions at the same pace of the traffic load reports. With load reports each 5 mimutess, BiQui performs 12 updates per hour and activates 17 edge CPUs as the 18:00 rush hour approaches (see Fig. 14b inset). Overall, BiQui adapts to load changes and offloads traffic to the edge to meet the target delay during rush hours.

D. BiQui's execution time

We run our experiments over a 12th Gen Intel(R) Core(TM) i7-1260P. Tab. VI shows the maximum observed execution times of BiQui considering different granularities z_{gran} and RTTs [18]: near cloud (22.8 ms) and far cloud (49.1 ms).

The impact of z_{gran} on execution time. The execution time monotonically decreases with the offloading granularity z_{gran}, as larger values of z_{gran} result into less iterations looking for feasible solutions – see line 3 of Alg. 1.

The impact of the RTTs on execution time. High RTTs require more CPUs to process traffic on time and meet the target delay T. However, BiQui's runtime is non-monotone w.r.t. the RTT – see Tab. VI. Increased RTTs result in increased execution times for smaller offloading granularity values, while for the opposite holds for higher ones. We accredit the different behavior for large granularity values to the fact that the binary search in line 2 of Alg. 1 terminated earlier, thus resulting in lower execution time for BiQui.

Take-away. According to Tab. VI, BiQui takes less than 120 ms to find the optimal CPU and offloading setup in the considered configurations. Thus, just one frame risks violating the ToD URLLC requirement (100 ms) due to BiQui execution. The rest of the frames in the video flow will meet the URLLC requirements.

E. The impact of τ on Min-swap's performance

In the following, we evaluate the performance of the Min-Swap algorithm when we change the tolerance parameter τ. We use the same setting specified in Sec. VII-A but having same edge and cloud costs $c_{0c} = c_{0e} = c_{1c} = c_{1e}$.

The impact of τ with increasing demand. In Fig. 15 we study the performance of Min-Swap when the demand increases with time. That is, the t-th xtick of Fig. 15 represents the demand $\lambda(t)$, and $\lambda(t) < \lambda(t+1), \forall t$. The values in the y axis of Fig. 15 relate to the solutions $x(t), y(t), z(t)$ obtained by Min-Swap when it receives as input $x(t-1), y(t-1)$ and BiQui's solution for $\lambda(t)$. That is, for every time step t we feed Min-Swap with its CPU configuration for the prior time step $t-1$. This mimics the behaviour of running Min-Swap in a scenario with time-varying load $\lambda(t)$ that increases.

In Fig. 15(a) we plot the relative cost error, *i.e.*, how much Min-Swap cost differs with BiQui solution, which is captured by the left hand side of eq. (14). Results show that the relative cost error always remains below τ. The relative cost error presents peaks when the BiQui optimal solution incurs into large CPU (de)activations. For instance, as discussed in Sec. VI-E, with $\lambda(t) \simeq 0.8$ [pkt/ms] BiQui ($\tau = 0$ in Fig. 15(b)) sends all the load to the cloud to get the optimal solution, thus the peak in Fig. 15(a).

Fig. 15(b-d) evidence that small values of τ lead to more oscillations in the offloading and CPU activation decisions. Such oscillations occur because a demand increase may lead to optimal configurations $x^*(t), y^*(t), z^*(t)$ with large CPU (de)activations. Consequently, we have to increase τ to select more expensive configurations that minimize the CPU (de)activations. In particular, Fig. 15(b-d) show that the tolerance should increase to $\tau = 0.05$ so that Min-Swap operates without oscillations. Moreover, with $\tau = 0.05$, the offloading – see Fig. 15(b) – smoothly increases as the demand goes up, thus Min-Swap prevents oscillations in the CPU (de)activations – see Fig. 15(c-d).

The impact of τ on real-world traffic data. We now analyze the performance of Min-Swap algorithm by using one day of real data from the Torino traffic trace described in Sec. VII-A. For the sake of readability, in Fig. 16 we illustrate just the performance of Min-Swap with $\tau = 0.05$ and the performance of BiQui, *i.e.*, Min-Swap with $\tau = 0$. As in Fig. 15 experiments, Min-Swap uses the solution from the prior timestamp $x(t-1), y(t-1)$ to find the best configuration for the new load $\lambda(t)$.

Fig. 16(a) shows that the relative cost error for Min-Swap $\tau = 0.05$ always remains smaller than a 5%. Moreover, during the early morning hours there is no relative cost error for there is barely road traffic – recall Fig. 13(b).

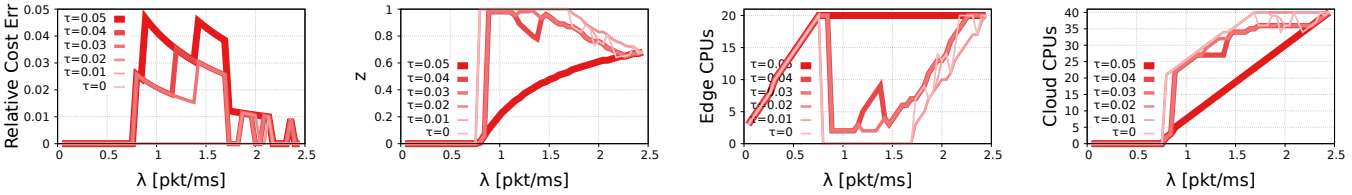
Fig. 16(b-d) plot the decisions $x(t), y(t), z(t)$ taken by BiQui ($\tau = 0$) and Min-Swap ($\tau = 0.05$) over time. Fig. 16(b) shows that BiQui leads to severe oscillations in the offloading decisions before 6:00 and around 21:00. The offloading oscillations before 6:00 result in 5 to 10 CPU swaps in Fig. 16(c-d), and the oscillations around 21:00 result in 20 to 25 CPU swaps. Note that Min-Swap with $\tau = 0.05$ does not incur into prominent CPU swaps before 6:00 nor at 21:00. Indeed, Min-Swap with $\tau = 0.05$ presents a smooth tendency in both offloading and CPU activations.

Min-Swap with $\tau = 0.05$ ends up having less CPU swaps over time than BiQui. To measure this, Fig. 17(a) shows the cummulative CPU difference with respect to BiQui:

$$\sum_t |x(t)-x(t-1)|+|y(t)-y(t-1)| \Big/ \sum_t |x^*(t)-x^*(t-1)|+|y^*(t)-y^*(t-1)| \tag{19}$$

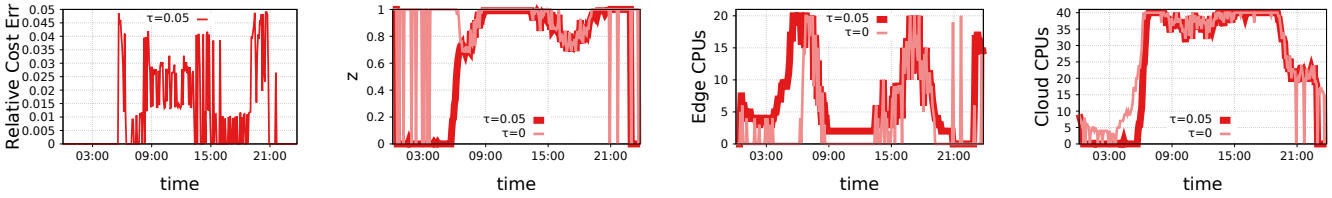
with $x^*(t), x(t)$ referring to the decisions of BiQui and Min-Swap with a given τ, respectively. In Fig. 17(a) we see that in one day Min-Swap ends up having $< 10\%$ of BiQui CPU swaps Moreover, larger values of τ decrease the accumulated CPU swaps during the day. Indeed, the difference of accumulated CPU swaps decrease as time progress.

The counter back of having $\tau > 0$ is to end up using more expensive solutions, *i.e.*, solutions with more CPUs. In



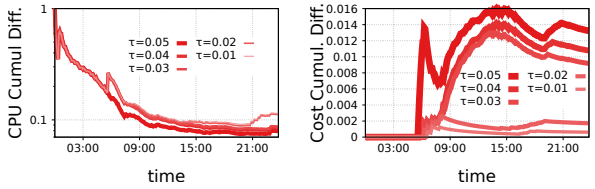
(a) Relative cost vs λ (b) Offloading z *vs.* λ (c) Activation x (edge) *vs.* λ (d) Activation y (cloud) *vs.* λ

Fig. 15: We vary the tolerance τ of the Min-Swap algorithm in a V2N service with reliability $P_G = 99.999\%$ and target delay $T = 100$ [ms]. Cloud and edge costs are equal. Propagation delays are inline with [18]: $D^e_{\text{prop}} = 18.2$ [ms] and $D^c_{\text{prop}} = 22.8$ [ms] for the propagation delay at the edge and cloud, respectively.



(a) Relative cost wrt. BiQui (b) Offloading z (c) Edge CPUs x (d) Cloud CPUs y

Fig. 16: Min-Swap ($\tau = 0.05$) and BiQui ($\tau = 0$) over real traces from Fig. 13. The experiment considers a V2N service with $T = 100$ [ms] target delay and $P_G = 99.999\%$ reliability. Following [18], we use $D^e_{\text{prop}} = 18.2$ [ms] and $D^c_{\text{prop}} = 22.8$ [ms] for the propagation delay at the edge and cloud, respectively.



(a) Cumulative CPU swaps (b) Cumul. cost difference

Fig. 17: Cummulative CPU and cost difference between Min-Swap and BiQui in Torino traces. We try Min-Swap with different tolerances τ.

Fig. 17(b) we measure the cost cumulative difference:

$$\sum_t K(x(t),y(t),z(t)) - K(x^*(t),y^*(t),z^*(t)) \Big/ \sum_t K(x^*(t),y^*(t),z^*(t)) \quad (20)$$

with $x^*(t), x(t)$ referring to the decisions of BiQui and Min-Swap with a given τ, respectively. Fig. 17(b) shows how Min-Swap ends the day having a cost excess that is a 0.14% larger than the cost of BiQui's decisions. Moreover, with tolerances $\tau \leq 0.02$ Min-Swap ends the day paying a 0.2% more than BiQui. That is, the smaller the τ, the smaller the extra cost.

It is worth mentioning the prominent peaks presented by Fig. 17(b) for $\tau > 0.02$ at 6:00, 15:00 and 21:00. The former occurs because at 6:00 Min-Swap with $\tau = 0.05$ suffers from an offloading ramp to the cloud – see Fig. 15(b) – and Min-Swap incurs into more expensive configurations. The increase at 15:00 and 21:00 are due to prominent swaps in the number of edge CPUs in the best solution – *i.e.*, with $\tau = 0$ in Fig. 15(c-d). Hence, the cost cumulative difference increases.

VIII. Related Work

Vehicles-to-anything (V2X) conveys communications among vehicles (V2V), pedestrians (V2P) and network or infrastructure (V2N), being a superset of all of them. Resource provisioning and offloading for V2V/V2P are also related to V2N, as both vehicles and phones have computing capacity in the network. We now provide an overview of the most recent works that are closely related to our study.

Traffic Offloading in V2X. Work [30] aims at minimizing the average response time in Internet of Vehicles (IoV) systems, via task offloading to the fog. Work [24] aims at revenue maximization through task offloading among multiple edge service providers. Work [31] assesses vehicle task offloading through Reinforcement Learning (RL) using an Asynchronous Advantage Actor-Critic (A3C) architecture, aiming to learn an offloading policy to minimize task execution time and resource usage while maximizing system performance. *All these works account for coarse aggregate metrics, e.g., average latency or data rate, thus not being able to adequately address the stringent reliability and latency requirements for safety-critical V2N services. We additionally consider (i) resource provisioning; and (ii) processing time of each task at the edge/cloud, guaranteeing URLLC constraints are met end-to-end.*

Resource Scaling and Allocation in V2X.

π-ROAD [32] is a static algorithm that performs RAN slicing for Emergency V2X services. The goal is to provide sufficient RAN resource blocks to each slice, ensuring that emergency V2X services receive the necessary resources for reliable and efficient communications during critical scenarios.

Work [33] studies dynamic radio and power resource allocation via a meta-reinforcement learning approach that combines Deep Q-networks (DQN) for discrete sub-band assignment with Deep Deterministic Policy Gradient (DDPG) for continuous power control, enabling efficient handling of hybrid action spaces.

Work [34] introduces a dynamic allocation of frequency

resources for LTE to Vehicle User Equipments (VUEs) and RSUs employing a two-stage Multi-Objective Discrete Particle Swarm Optimization (MDPSO) and precoding to reduce power consumption while adapting to real-time channel conditions.

Compared to these works, we explicitly address the stochastic nature of packet delays caused by queuing waiting times and service times that depend on the frame length, rigorously evaluating the 99.999-percentile of the sojourn time for V2N traffic—capturing the total time packets spend waiting in the queue at computing servers and being processed.

Joint task offloading and resource allocation for V2X. To the best of our knowledge, [35], [36] are the only works that jointly tackle the task offloading and resource allocation for V2X. However, they oversee the delay's stochastic nature, considering it as a ratio between the demand and computing resources [35], or ignore the V2X URLLC requirement [36]. *Our work considers the inherent stochasticity of queuing and processing delays, and guarantees delay and reliability constraints imposed by V2X.*

URLLC Service Provisioning. Some literature tackles URLLC services in contexts that are not related with V2X, *e.g.*, [37], [38]. Works [27], [28] leverage Stochastic Network Calculus (SNC) to infer the delay violation probability in URLLC, in order to scale the slice radio resources. Work [28] computes the amount of guaranteed radio resources for each URLLC service providing probabilistic delay guarantees, while [27] bounds the delay under constraints related to the target violation probability and the distribution of the traffic demand. *We account for the end-to-end delay of URLLC services and advocate to queuing theory, capturing the packetized nature of internet traffic and providing tight bounds even for very high reliability, such as the 99.999 delay percentile of the delay and approximate its distribution.*

Gamma-distributed service times. For multiple servers following a gamma distribution of order two, [39] studies the probabilities of having n tasks in a system. For a single server, [40] characterizes the steady-state queue distribution under batch service queue, while [41] characterizes the waiting time distribution. *However, none of these works study multi-server queueing systems whose service times are a mixture of Gamma distributions, i.e. the M/Γ/k system.*

$M/G/k$ **literature.** We discuss $M/G/k$ queues, as this is the mostly used type of queues. Existing works leverage deficit renewal equations [42] or difference-differential equations [43] to approximate the M/G/k waiting time distribution, leading to errors in the order of 10^{-2}, which do not suffice for URLLC services as tele-operared driving. These approaches generally assume steady-state conditions, which may overlook transient behaviors that are critical to low-latency applications requiring both reliability and responsiveness under fluctuating network loads. Thus, such methods often don't adapt to highly variable service distributions and may introduce significant inaccuracies in high-load or bursty arrival scenarios, which are typical in URLLC demands. *Rather, our approximation convolves the M/D/k waiting time with the gamma mixture service time, ensuring URLLC with errors lower than* 10^{-5}. *We leverage the deterministic structure of M/D/k queues, and we provide an analytically tractable, closed form, approximation that remains robust under diverse traffic conditions, ensuring the stringent accuracy and ultra-low latency that are required in real-time URLLC applications. Overall, our work extends the capabilities of M/G/k models to meet next-generation service requirements that demand error rates significantly lower than those achievable with traditional approaches.*

Latest Developments in V2X. The main focus has been enhancing reliability, reducing latency, and improving throughput to meet the stringent demands of vehicular networks. Work [44] presents IEEE 802.11bd as a significant advancement in vehicular communication, addressing key QoS challenges through enhancements in both the PHY and MAC layers. By introducing mechanisms such as LDPC coding and adaptive packet repetition the standard effectively mitigates issues related to latency, reliability, and congestion.

In parallel, work [45] presents a detailed analytical and simulation-based evaluation of 5G NR sidelink Mode 2 in unlicensed bands (SL-U), revealing that existing 3GPP access rules significantly degrade throughput due to strict transmission constraints at SL slot boundaries. The study proposes Markov-based models and a randomization strategy to mitigate collisions and enhance SL-U performance.

IX. Conclusion and Future work

We introduce the V2N Computation Offloading and CPU Activation (V2N-COCA) problem, aiming to minimize energy/monetary costs taking offloading and scaling decisions to process URLLC V2N traffic at the edge/cloud. To address the lack of closed-form expressions for the URLLC requirement, we use queuing theory to approximate tasks sojourn times at the servers. We rigorously validate our approximation, w.r.t. its accuracy and effectiveness when solving the V2N-COCA Problem. Based on its structural properties, we design BiQui, a provably asymptotically optimal and computationally efficient algorithm. Results over real-traces show that BiQui outperforms the state of the art, meeting the target delay 99.999% of the time. We also formulate the Min-Swap problem to minimize switching on/off CPUs as the load changes, yet bounding the extra the cost. To solve the Swap-Prevention problem we design Min-Swap, an algorithm that minimizes unnecessary CPU (de)activations constraint by a cost-tolerance parameter τ, with provably low complexity. Our evaluations on real-world vehicular traffic traces demonstrate that the integration of Min-Swap achieves superior resource stability with minimal additional cost, making it an indispensable tool for dynamically fluctuating environments. The results underline the versatility of our framework in ensuring cost-efficiency, reliability, and sustainability for V2N URLLC services.

We foresee four lines of future work. First, considering vehicles that use multiple V2N applications with diverse URLLC requirements. Second, accounting for a three-tier architecture with cloud, edge and in-vehicle processing. Third, minimizing transmission and propagation delays using optimal NR and transport scheduling, thus reducing CPU consumption. Fourth, considering non-linear task processing, parallel task processing and a pool of heterogeneous processing units.

ACKNOWLEDGEMENTS

L.E. Chatzieleftheriou is a Juan de la Cierva awardee (JDC2022-050266-I), funded by MCIU/AEI/10.13039/501100011033 and the European Union "NextGenerationEU"/PRTR, and MADQuantum-CM project, funded by the Regional Government of Madrid and the EU "NextGenerationEU"/PRTR. and is also partially supported by the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D SORUS project, and by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101192035 (AMAZING-6G).

REFERENCES

[1] L. E. Chatzieleftheriou, J. Pérez-Valero, J. Martín-Pérez *et al.*, "Sustainable Provision of URLLC Services for V2N: Analysis and Optimal Configuration," in *Proc. of International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, ser. MobiHoc '24. ACM, 2024, p. 161–170.

[2] J. Wang, J. Liu, and N. Kato, "Networking and Communications in Autonomous Driving: A Survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1243–1274, 2019.

[3] 5G Americas, "Vehicular connectivity: C-V2X & 5G," 2021.

[4] M. Noor-A-Rahim *et al.*, "6G for Vehicle-to-Everything (V2X) Communications: Enabling Technologies, Challenges, and Opportunities," 2022.

[5] "Nexar," https://www.getnexar.com/nexar-app/, accessed: 16/01/2024.

[6] H. Koumaras *et al.*, "Analysis of H. 264 video encoded traffic," in *Proc. of the 5th Internat. Network Conf. (INC2005)*, 2005, pp. 441–448.

[7] N. Zhang, Haijun Liu *et al.*, "Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges," *IEEE communications magazine*, vol. 55, no. 8, pp. 138–145, 2017.

[8] Z. Cao *et al.*, "Performance of hashing-based schemes for Internet load balancing," in *IEEE INFOCOM*, vol. 1, 2000, pp. 332–341 vol.1.

[9] L. Cominardi, L. M. Contreras, C. J. Bcrnardos *et al.*, "Understanding QoS Applicability in 5G Transport Networks," in *2018 IEEE Internat. Symp. on Broadband Multimedia Systems and Broadcasting*, 2018.

[10] M. Roitzsch *et al.*, "Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video," in *IEEE Internat. Real-Time Systems Symp. (RTSS'06)*. IEEE, 2006, pp. 271–280.

[11] A. C. Bavier *et al.*, "Predicting mpeg execution times," in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 1998, pp. 131–140.

[12] V. Gupta, M. Harchol-Balter, J. G. Dai, and B. Zwart, "On the inapproximability of M/G/K: why two moments of job size distribution are not enough," *Queueing Systems*, vol. 64, pp. 5–48, 2010.

[13] "Service requirements for the 5G system," 3GPP, Technical Specification (TS) 22.261.v19.3.0., June 2023.

[14] "Service requirements for enhanced v2x scenarios," 3GPP, Technical Specification (TS) 22.186.v18.0.1., June 2023.

[15] "Physical layer procedures for data. Tech. Report TS 38.214 v18.0.0. 2023," September 2023.

[16] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan *et al.*, "Worth their watts? - an empirical study of datacenter servers," in *Proceedings of the Internat. Symp. on High-Performance Computer Architecture*, 2010.

[17] "AWS EC2 calculator," https://calculator.aws/\#/addService/ec2-enhancement, 2023, [Online; accessed January 30, 2024].

[18] M. Xu *et al.*, "From cloud to edge: a first look at public edge platforms," in *Proc. of the 21st ACM Internet Measurement Conf.*, 2021, pp. 37–53.

[19] G. Franx, "A simple solution for the M/D/c waiting time distribution," *Operations Research Letters*, vol. 29, no. 5, pp. 221–229, 2001.

[20] H. C. Tijms, "Stochastic models: an algorithmic approach," 1994.

[21] J. Perez-Valero, J. Garcia-Reinoso, A. Banchs, P. Serrano *et al.*, "Performance trade-offs of auto scaling schemes for NFV with reliability requirements," *Computer Communications*, vol. 212, pp. 251–261, 2023.

[22] T. H. Cormen *et al.*, *Introduction to algorithms*. MIT press, 2022.

[23] "Traffic data," https://github.com/MartinPJorge/biqui/blob/master/data/traffic_torino_v02.csv, 2024.

[24] J. Ren, J. Liu, Y. Zhang, Z. Li, F. Lyu, Z. Wang, and Y. Zhang, "An Efficient Two-Layer Task Offloading Scheme for MEC System with Multiple Services Providers," in *IEEE INFOCOM*, 2022, pp. 1519–1528.

[25] N. Gans, G. Koole, and A. Mandelbaum, "Telephone Call Centers: Tutorial, Review, and Research Prospects," *Manufacturing & Service Operations Management*, vol. 5, no. 2, pp. 79–141, 2003.

[26] K. Xiong, S. Leng, C. Huang, C. Yuen, and Y. L. Guan, "Intelligent Task Offloading for Heterogeneous V2X Communications," *IEEE Trans. on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2226–2238, 2021.

[27] O. Adamuz-Hinojosa *et al.*, "A Stochastic Network Calculus (SNC)-Based Model for Planning B5G uRLLC RAN Slices," *IEEE Trans. on Wireless Communications*, vol. 22, no. 2, pp. 1250–1265, 2023.

[28] O. Adamuz-Hinojosa, L. Zanzi, V. Sciancalepore, A. Garcia-Saavedra, and X. Costa-Pérez, "Oranus: Latency-tailored orchestration via stochastic network calculus in 6g o-ran," in *IEEE INFOCOM*, 2024, pp. 61–70.

[29] Y. Jiang *et al.*, *Stochastic network calculus*. Springer, vol. 1.

[30] X. Wang, Z. Ning, and L. Wang, "Offloading in Internet of Vehicles: A Fog-Enabled Real-Time Traffic Management System," *IEEE Trans. on Industrial Informatics*, vol. 14, no. 10, pp. 4568–4578, 2018.

[31] Q. Qi, J. Wang, Z. Ma *et al.*, "Knowledge-Driven Service Offloading Decision for Vehicular Edge Computing: A Deep Reinforcement Learning Approach," *IEEE Trans. on Vehicular Technology*, vol. 68(5), 2019.

[32] A. Okic, L. Zanzi, V. Sciancalepore, A. Redondi, and X. Costa-Pérez, "π-ROAD: a Learn-as-You-Go Framework for On-Demand Emergency Slices in V2X Scenarios," in *IEEE INFOCOM*, 2021, pp. 1–10.

[33] Y. Yuan, G. Zheng, K.-K. Wong *et al.*, "Meta-reinforcement learning based resource allocation for dynamic v2x communications," *IEEE Trans. on Vehicular Technology*, vol. 70, no. 9, pp. 8964–8977, 2021.

[34] J. Shi, Z. Yang, H. Xu, M. Chen *et al.*, "Dynamic resource allocation for lte-based vehicle-to-infrastructure networks," *IEEE Trans. on Vehicular Technology*, vol. 68, no. 5, pp. 5017–5030, 2019.

[35] W. Feng, S. Lin, N. Zhang, G. Wang, B. Ai, and L. Cai, "Joint C-V2X Based Offloading and Resource Allocation in Multi-Tier Vehicular Edge Computing System," *IEEE JSAC*, vol. 41, no. 2, pp. 432–445, 2023.

[36] Q. Liu, R. Luo, H. Liang, and Q. Liu, "Energy-Efficient Joint Computation Offloading and Resource Allocation Strategy for ISAC-Aided 6G V2X Networks," *IEEE Trans. on Green Communications and Networking*, vol. 7, no. 1, pp. 413–423, 2023.

[37] A. Anand, G. De Veciana, and S. Shakkottai, "Joint scheduling of URLLC and eMBB traffic in 5G wireless networks," *IEEE/ACM Trans. on Networking*, vol. 28, no. 2, pp. 477–490, 2020.

[38] A. Azari, M. Ozger, and C. Cavdar, "Risk-aware resource allocation for URLLC: Challenges and strategies with machine learning," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 42–48, 2019.

[39] S. Shapiro, "The m-server queue with poisson input and gamma-distributed service of order two," *Operations Research*, vol. 14, no. 4, pp. 685–694, 1966.

[40] S. Pradhan, U. Gupta, and S. Samanta, "Queue-length distribution of a batch service queue with random capacity and batch size dependent service: M/g r y/1 m/gˆy_r/1," *Opsearch*, vol. 53, pp. 329–343, 2016.

[41] A. Ghosal, "Queues in series," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 24, no. 2, pp. 359–363, 1962.

[42] M. van Hoorn and H. Tijms, "Approximations for the waiting time distribution of the M/G/c queue," *Performance Evaluation*, vol. 2, no. 1, pp. 22–28, 1982.

[43] P. Hokstad, "Approximations for the M/G/m Queue," *Operations Research*, vol. 26, no. 3, pp. 510–523, 1978.

[44] C. Iliopoulos, A. Iossifides, C. H. Foh, and P. Chatzimisios, "Ieee 802.11 bd for next-generation v2x communications: From protocol to services," *IEEE Communications Standards Magazine*, 2025.

[45] V. Weerackody, H. Yin, and S. Roy, "Nr sidelink mode 2 in unlicensed bands: Throughput model & validation," *IEEE Trans. on Communications*, 2024.

[46] M. Fidler *et al.*, "A Guide to the Stochastic Network Calculus," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 92–105, 2015.

[47] S. M. Ross, "A first course in probability," 2014.

APPENDIX

A. Proof of Proposition 1

We only prove the increasing monotonicity w.r.t. decision x, *i.e.*, CPU activation at the edge. However, exactly the same arguments can be used to prove the monotonicity w.r.t. decisions y of CPU activation at the cloud. Fix $V = V_0$,

$y = y_0$ and $z = z_0$, and let $f(x) := K(V_0, x, y_0, z_0)$. By definition, $f(x)$ is monotone increasing in x *if and only if* $x_1 > x_2 \iff f(x_1) > f(x_2)$. Observe that the total service time $S_E = \lambda_E(z_0, V)s$ at the edge, *i.e.*, the aggregate over all activated CPUs at the edge, is independent of the CPU activation decisions, since it essentially depends on the amount of tasks that are sent there, *i.e.*, from the offloading decisions z_0, which we previously fixed. Similarly for the service time $S_C = \lambda_C(z_0, V)s$ at the cloud. Observing the linearity of Eq. (5) w.r.t. the number x of activated CPUs at the edge, we confirm that the monotonicity condition holds, which concludes the proof.

B. Proof of Proposition 2

(1) - (2): non-monotonicity/non-continuity w.r.t. z. We perform a proof by finding a non-continuous and non-monotone **counterexample**. Fig. 2 was produced by performing exhaustive searches, and can be thus treated as an oracle. Although this proof relies on a particular parametrization, in fact it suffices to show that the monotonicity and continuity properties do not hold in general. Also, we provide arguments that expose the characteristics of the the feasible region for any other instance of the problem, *i.e.*, for any parametrization. Let $z_1 = 0.672$, $z_2 = 0.685$, and $z_3 = 0.692$. Clearly, $z_1 < z_2 < z_3$. However, the feasible region found by exhaustive searches (Fig. 2) suggests that $K(V, x, y, z_1) < K(V, x, y, z_2)$, and $K(V, x, y, z_2) > K(V, x, y, z_3)$, which contradicts the definition of monotonicity, and thus concludes the proof for (1). Remark: by the monotonicity definition, all other parameters except of the offloading policy z should have been fixed, while in our example the number of cloud CPUs increases from $y_1 = 31$ to $y_2 = 32$ as we move offloading from $z_1 = 0.672$ to $z_2 = 0.685$. This jump is due to the URLLC requirement, which implies that an additional CPU needs to be activated in order to obtain a feasible solution. This concludes the proof for (2), and the description of our counterexample.

(3): continuity and linearity w.r.t. z between consecutive steps. The steps in the boundary occur due to the change in the number of minimum CPUs that are needed to ensure an URLLC. While being in between of two consecutive jumps in the boundary of the region, *i.e.*, when fixing the number x and y of edge or cloud activated CPUs, respectively, the subscription cost given by c_{0e} and c_{0c} in Eq. (5) is fixed. However, the total service time depends on the amount of tasks that are assigned, *i.e.*, by the offloading policy z. By combining eqs. (2), (3), and (5), we conclude the proof.

C. Proof of Proposition 4

The CDF of the sojourn time will allow the reliability requirement to be perfectly described, and thus the binary search in line 2 of Alg. 1 to find the optimal number y of CPUs to activate at the cloud. The next phase that BiQui continues with is walking down the feasibility region boundary. Since from Prop. 1 the objective function is monotone w.r.t. the CPU activation decisions, the optimal CPU activation configurations will lie in the boundary w.r.t. them. Combining this with the linearity of the feasibility region boundary w.r.t. the offloading decision z (third property in Prop. 2), and considering $z_{gran} \to 0$, we obtain the result.

D. Proof of Proposition 5

Given $x(t-1), y(t-1) > 0$, the projection of Problem 2 objective function at a fixed $y_0(t)$ is $f(x(t)) = |x(t) - x(t-1)| + \kappa$, where $\kappa = |y_0(t) - y(t-1)|$. For $x(t) \leq x(t-1)$, $f(x(t))$ is decreases with $x(t)$. While for $x(t) \geq x(t-1)$, $f(x(t))$ is decreases with $x(t)$. Hence, the projection $f(x(t))$ is not monotone, and thus the objective function from Problem 2 the same.

E. Proof of Proposition 6

Let $(x(t), y(t), z(t)) \in \Omega'(\tau_1)$. By definition, the cost of such configuration exceeds the optimal solution cost by less than a $\tau_1\%$, which is less than $\tau_2\%$ from the cost of the optimal solution. With this simple argument we trivially conclude that $\tau_1 \leq \tau_2 \Leftrightarrow \Omega'(\tau_1) \subseteq \Omega'(\tau_2)$.

F. Structural properties of Prob. 2 with CPU switching costs

In this appendix we discuss the structural properties of Problem 2 when the objective function considers CPU switching costs, i.e. when the objective function is $U(x(t), x(t-1), y(t), y(t-1))$ defined in (16). First, we prove that the objective function (16) is non-monotone. Then, we show that the feasibility region $\Omega(\tau)$ increases with the tolerance parameter τ despite the new objective function – i.e. Proposition 6 holds.

Proposition 7. *The objective function $U(x(t), x(t-1), y(t), y(t-1))$ with $\nu_x, \nu_y, \phi_x, \phi_y > 0$ is non-monotone w.r.t. the activation decisions $x(t), y(t)$.*

Proof. Fix $x(t-1), y(t), y(t-1)$. Then the objective function $U(x(t), x(t-1), y(t), y(t-1)) = U(x(t))$ becomes $\nu_x[x(t) - x(t-1)]^+ + \phi_x[x(t-1) - x(t)]^+ + \kappa$, with κ a constant. Take $x_0 \geq x(t-1)$ so the objective function becomes $U(x(t)) = \nu_x(x_0 - x(t-1)) + \kappa$. We can find a smaller value for the number of edge CPUs $x_1 \leq x_0$ such that the objective is larger, i.e. such that $U(x_1) \geq U(x_0)$. Concretely, if we take $x_1 \leq x(t-1)$ the objective function is $U(x_1) = \phi_x(x(t-1) - x_1) + \kappa$, and we can choose $x_1 < x(t-1) - \frac{\nu_x}{\phi_x}(x_0 - x(t-1))$ to guarantee $U(x_1) \geq U(x_0)$. Hereof, we conclude the projection of the objective function on the number of CPUs $U(x(t))$ is non-monotone, thus the four dimensional objective function $U(x(t), x(t-1), y(t), y(t-1))$. □

Lastly, it is worth remarking that replacing the objective function in Problem 2 by $U(x(t), x(t-1), y(t), y(t-1))$ has no impact on the feasibility region $\Omega(\tau)$. Hence, Proposition 6 still holds.

G. The SNC Benchmark

Stochastic Network Calculus (SNC) [29] defines an arrival curve $A(\tau,t) \in \mathbb{R}^+,\ \tau,t>0$ denoting the bits that arrive to a system in the time interval $(\tau,t]$. Similarly, SNC considers a service curve $S(\tau,t) \in \mathbb{R}^+,\ \tau,t>0$ denoting the bits that where served by a system in the time interval $(\tau,t]$. In case the arrival and service process are stationary the arrival and service curves are fully determined by $A(t), S(t),\ t>0$. With $A(t), S(t)$ referring to the bits that arrived and were served in t time units.

Following the lead of [27], we assume that the arrival and service curves are bounded by the affine envelopes $\alpha(t), \beta(t) \in \mathbb{R}^+$. In particular, we resort to the Exponentially Bounded Burstiness (EBB) and the Exponentially Bounded Fluctuation (EBF) approach taken by [27], [46]. That is, we consider affine envelopes $\alpha(t), \beta(t)$ satisfying

$$\mathbb{P}\left(A(t) > \alpha(t)\right) \leq \varepsilon_A, \qquad \mathbb{P}\left(S(t) < \beta(t)\right) \leq \varepsilon_S \tag{21}$$

with $\varepsilon_A, \varepsilon_S$ the overflow and deficit profiles.

Both envelopes are defined as $\alpha(t) = (\rho_A + \delta)t + b_A$ and $\beta(t) = (\rho_B - \delta)[t - \frac{b_S}{\rho_S}]^+$, with $[x]^+ = \max\{0, x\}$ and $\delta > 0$ the sample path argument [46]. In case the arrival and service curves satisfy (21), affine envelopes bound the delay as:

$$W = {}^{b_A + b_S}\!/_{\rho_S - \delta} \tag{22}$$

We now compute how should $b_A, b_S, \rho_S, \delta > 0$ must be to satisfy the overflow and deficit profiles in (21). In particular, proceed as indicated by [46] and [28, III.B].

First, we remark that the arrival curve $A(t)$ will denote the number of video processing tasks that arrived in t time units. Now, we bound the MGF of the arrival curve $A(t)$ using a rate-burst curve (ρ_A, σ_S)

$$M_A(\theta) = \mathbb{E}[e^{\theta A(t)}] = e^{\lambda t(e^\theta - 1)} \leq M_\alpha(\theta) = \mathbb{E}[e^{\theta\alpha(t)}] = e^{\theta(\rho_A t + \sigma_A)} \tag{23}$$

with the former equality given by the MGF of a Poisson distribution, and (23) holding if $\rho_A = \frac{\lambda(e^\theta - 1)}{\theta}, \sigma_A = 0$.

Now we follow the same approach for the service curve $S(t)$, which denotes the number of V2N tasks dispatched in t time units. According to (3), the service time of a V2N video task is proportional to the frame size l_i, which obeys a gamma distribution $\Gamma(\alpha_i, \beta_i)$. Consequently, the time to process a task is computed as $f_\Gamma(\alpha_i, \beta_i; l_i)s$, with $f_\Gamma(\alpha_i, \beta_i; l_i)$ the probability density function of the Gamma distribution for the task size l_i, and S the average V2N task processing time. Hence, given the MGF of the gamma distribution, we express the number of tasks processed in t time units is given by the service curve $S(t) = t - l_i s$, whose negative MGF is

$$\begin{aligned} M_S(-\theta) = \mathbb{E}[e^{-\theta(t - l_i s)}] &= e^{-\theta t} \int_0^\infty e^{\theta x s} f_\Gamma(\alpha_i, \beta_i; x)\, dx \\ &= e^{-\theta t}\left(1 - \frac{\theta s}{\beta_i}\right)^{-\alpha_i} = e^{\log\left(1 - \frac{\theta s}{\beta_i}\right)^{-\alpha_i} - \theta t}. \end{aligned} \tag{24}$$

As in (23) we bound the service curve $S(t)$ MGF through another rate-burst curve (ρ_B, σ_B). If the rate-burst curve is a lower bound of the service curve $S(t)$, then its negative MGF is an upper bound of $S(t)$ negative MGF. In other words, the deficit profile holds as long as

$$M_S(-\theta) = e^{\log\left(1 - \frac{\theta s}{\beta}\right)^{-\alpha} - \theta t} \leq M_\beta(-\theta) = e^{-\theta(\rho_S t - \sigma_S)},$$

which only occurs as long as $\rho_S = 1, \quad \sigma_S = \frac{-\alpha_i \log\left(1 - \frac{\theta s}{\beta_i}\right)}{\theta}$.

All left to compute the delay bound in (22) is to compute b_A, b_S. As [28] points out, the EBB and EBF models are connected with the Chernoff bound [47]. Consequently, we claim $b_A = \sigma_A - \frac{1}{\theta}\left(\log(\varepsilon_A) + \log(1 - e^{-\theta\delta})\right)$ and $b_S = \sigma_S - \frac{1}{\theta}\left(\log(\varepsilon_S) + \log(1 - e^{-\theta\delta})\right)$.

Altogether, we formulate an optimization problem to minimize the delay expression in (22). In particular, we seek the adequate $\delta, \theta > 0$ parameters to minimize the delay. The corresponding optimization problem stays as follows.

Problem 3 (SNC delay bound)**.** *Assume equal deficit and overflow violation probabilities $\varepsilon_A = \varepsilon_B = \frac{\varepsilon}{2}$. Given a Poisson arrival process or rate λ and a service time $l_i S$ with $l_i \sim f_\Gamma(\alpha_i, \beta_i)$, the SNC delay is given by the $\theta, \delta > 0$ setup minimizing:*

$$\min_{\theta,\delta} \frac{-\frac{\alpha}{\theta}\log\left(1 - \frac{\theta s}{\beta}\right) - \frac{2}{\theta}\left(\log\left(\frac{\varepsilon}{2}\right) + \log(1 - e^{-\theta\delta})\right)}{1 - \delta} \tag{25}$$

$$s.t.: \quad \theta, \delta, \rho_A > 0 \tag{26}$$

Note that Problem 3 assumes equal deficit and overflow probabilities as in [27], [28]. Although this is a common assumption, it is not true that the service curve is as prone as the arrival curve when it comes to violate the affine envelopes. Indeed, we conjecture that such assumption leads to the bad behaviour observed during our experiments in Section VII-C.

To find a solution for Problem 3 we need the objective function to be well defined, and the constraints to hold. Both translate into having $\theta \in \left(0, \frac{\beta}{s}\right)$ and $\delta \in \left(0, \frac{\rho_S}{2} - \frac{\lambda(e^\theta - 1)}{2\theta}\right)$.

We solve Problem 3 through an exhaustive search of (25) in which we take (1) $\theta' \in \Omega_\theta = \left\{\epsilon, \ldots, \frac{\beta}{S} - \epsilon\right\}$ with $|\Omega| = 100$; and then iterate over (2) $\delta' \in \Omega_\theta = \left\{\epsilon, \ldots, \frac{\rho_S}{2} - \frac{\lambda(e^\theta - 1)}{2\theta} - \epsilon\right\}$ with $|\Omega_\delta| = 100$, and $\epsilon = \frac{1}{100}$. Then, we select $\theta, \delta = \min_{\theta', \delta'}\{f(\theta', \delta')\}$ with f the objective function (25).

Livia Elena Chatzieleftheriou is a Marie Skłodowska-Curie Actions (MSCA) Postdoctoral fellow and a Juan de la Cierva awardee, currently working as a post-doctoral researcher with the IMDEA Networks Institute, and a part-time Lecturer with the University Carlos III of Madrid (UC3M). She holds an M.Sc. in applied mathematics and a Ph.D. in Computer Science. Her current research interests include online learning, optimization, and explainable AI for next-generation mobile networks.

Jesús Pérez-Valero received the Ph.D. degree from the Universidad Carlos III de Madrid (UC3M) in 2024. He is currently a Postdoctoral Researcher and lecturer with Universidad de Murcia (UMU). He has been involved in several projects funded by the European Commission through the SNS. His main research interests lie in the performance analysis and optimization of communication systems.

Jorge Martín Pérez obtained a B.Sc in mathematics, and a B.Sc in computer science, both at Universidad Autónoma de Madrid (UAM) in 2016. He obtained his M.Sc. and Ph.D in Telematics from Universidad Carlos III de Madrid (UC3M) in 2017 and 2021, respectively. His research focuses in applied math for communications. Since 2016 he has participated in national and EU funded research projects. He now works as associate professor at Universidad Politécnica de Madrid.

Pablo Serrano (M'09, SM'16) is an Associate Professor at the University Carlos III de Madrid. His research interests lie in the analysis of wireless networks and the design of network protocols and systems. He has over 100 scientific papers in peer-reviewed international journals and conferences and has served on the TPC of many conferences. He currently serves as Editor for IEEE Open Journal of the Communication Society.