Sustainable Provision of URLLC Services for V2N: Analysis and Optimal Configuration

Livia Elena Chatzieleftheriou
livia.chatzieleftheriou@imdea.org
IMDEA Networks Institute
Madrid, Spain

Jesus Perez-Valero
jesperez@pa.uc3m.es
Universidad Carlos III de Madrid
Madrid, Spain

Jorge Martín-Pérez
jorge.martin.perez@upm.es
Universidad Politécnica de Madrid
Madrid, Spain

Pablo Serrano
pablo@it.uc3m.es
Universidad Carlos III de Madrid
Madrid, Spain

ABSTRACT

The rising popularity of Vehicle-to-Network (V2N) applications is driven by the Ultra-Reliable Low-Latency Communications (URLLC) service offered by 5G. The availability of distributed resources could be leveraged to handle the enormous traffic arising from these applications, but introduces complexity in deciding where to steer traffic under the stringent delay requirements of URLLC. In this paper, we introduce the V2N Computation Offloading and CPU Activation (V2N-COCA) problem, which aims at finding the computation offloading and the edge/cloud CPU activation decisions that minimize the operational costs, both monetary and energetic, under stringent latency constraints. Some challenges are the proven non-monotonicity of the objective function w.r.t. offloading decisions, 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 with linear computational complexity w.r.t. computing resources algorithm for the V2N-COCA problem. We assess BiQui over real-world vehicular traffic traces, performing a sensitivity analysis and a stress-test. Results show that BiQui significantly outperforms state-of-the-art solutions, achieving optimal performance (found through exhaustive searches) in most of the scenarios.

CCS CONCEPTS

• **Networks** → **Network performance modeling**; **Network management**.

KEYWORDS

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

1 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 [30]. 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 [5], 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 [5], 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 [20].

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.

This work tackles the challenges above. In a nutshell, our contributions can be summarised as follows:

- We introduce the V2N Computation Offloading and CPU Activation (V2N-COCA) Problem, a novel problem that aims at minimising the operational costs (monetary, energetic) of distributed resources while ensuring the latency and reliability requirements of V2N URLLC services, by deciding which tasks to offload at the edge or at the cloud, and how many CPUs to activate. We rigorously study the problem's structural properties, proving, among others, the non-monotonicity of the objective function and the non-continuity of the feasible space boundary w.r.t the offloading decisions.

MobiHoc'24, October 14-17, 2024, Athens, Greece
2024. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- Motivated by experimental evidence, we propose an approximation of the sojourn time (*i.e.*, the sum of the waiting and service time), for which it currently does not exist a closed-form, challenging the URLLC requirement guarantee. Our approximation exploits the waiting time of $M/D/k$ queues. We thoroughly examine our proposal comparing against the optimal oracle found via exhaustive searches. We rigorously demonstrate that our approximation works perfectly when targeting vehicular applications based on tasks stemming from video frames, and we discuss why and how our proposal can be applied to different domains.
- We exploit the V2N-COCA Problem derived structural properties and its feasibility region, and design an efficient algorithm for its solution. We prove its correctness, its low computational complexity, and its asymptotic optimality.
- We evaluate our algorithm using real-world traces to evaluate its behaviour under realistic scenarios, while assessing its performance under a variety of conditions. We also perform an extensive sensitivity analysis on our system's parameters. The system parameterization is dictated by the related literature.

The rest of the paper is structured as follows: In §2 we discuss the related works. In §3 we present our system model. In §4 we introduce and analyse our optimization problem. In §5 we introduce and evaluate our proposal for the characterisation of the sojourn time of the tasks at the servers. In §6 we design and analyse our algorithm and its computing and approximability properties. In §7 we evaluate our algorithm, and in §8 we conclude the paper.

2 RELATED WORKS

Vehicles-to-anything (V2X) conveys communications among vehicles (V2V), with pedestrians (V2P) and the network/infrastructure (V2N), being a superset of all of them. Although our work tackles the latter, resource provisioning and offloading techniques for V2V and V2P are also of interest in V2N, as both vehicles and phones have computing capacity within the network. In the following we overview the existing literature regarding V2X offloading, V2X resource scaling/allocation, joint task offloading and resource allocation even in non-V2X contexts, URLLC service provisioning, and the $M/G/k$ literature that is related to waiting times.

Traffic Offloading in V2X Scenarios. The literature proposes offloading tasks to other vehicles [24] and edge premises [9] and [16], even leveraging Reinforcement Learning (RL) approaches [16, 23], and aim at minimizing the average waiting time [15] considering the channel quality [9]. *Our work also considers (i) the resource provisioning; and (ii) processing time of each task at the edge/cloud. Hence, guaranteeing URLLC constraints are met in an end-to-end fashion.*

Resource Scaling and Allocation in V2X Scenarios. Works as [21, 33] aim at accommodating enough radio resource blocks for V2X services, while [18] allocates enough resources in the V2X channel resorting to RL. However, such works [18, 21, 33] just analyze the radio link and oversee the tight 99.999-percentile delay requirement of URLLC services. *Our work accounts for queuing and processing delays, and allocates edge/cloud resources to ensure that queuing and processing delays meet the 99.999 percentile.*

Joint task offloading and resource allocation. To the best of our knowledge, [10, 27] and [19] are the only works that jointly tackle task offloading and resource allocation in V2X scenarios. However, they oversee the stochastic nature of the delay, considering it as a ratio between the demand and computing resources [10, 27], or ignoring the URLLC requirements of V2X services [19] . Only [22] keeps track of the stochastic nature of how queues grow to maintain them with a stable length, but considering only average metrics that do not capture reliability requirements. *Our work tackles the joint task offloading and allocation problem considering the inherent stochasticity of queuing and processing delays, and guaranteing delay and reliability constraints imposed by V2X.*

URLLC Service Provisioning. Works [6, 31] leverage Stochastic Network Calculus (SNC) to infer the delay violation probability in URLLC to: (*i*) scale the slice radio resources accordingly [6]; or (*ii*) decide where to process the tasks [31]. *Rather than just considering the radio network, our work accounts for the end-to-end delay of URLLC services and advocates to queuing theory instead of SNC. Hence, we capture the packetized nature of internet traffic and provide tighter bounds than SNC.*

M/G/k literature. Existing works [28] and [14] leverage deficit renewal equations and difference-differential equations, respectively, to approximate the M/G/k waiting time distribution. However, their approximations have errors in the order of 10^{-2}, which do not suffice for URLLC services as ToD. *Rather, we exploit the small variance of the gamma mixture to propose a new approximation that convolves the M/D/k waiting time with the gamma mixture service time and ensures URLLC with errors lower than* 10^{-5}.

3 SYSTEM MODEL

We illustrate in Fig. 1 the considered scenario: passing vehicles are provided wireless connectivity by Road Side Units (RSUs) deployed along the road or street. 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 [5]. Vehicles hold a CPU in which some computations can be performed, *e.g.*, to preprocess frames before actually performing heavier AI-related tasks [1]. 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.

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

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

Table 1: Notation Table

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 [5]. 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 [17], 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. 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 [7]. 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]. 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 [8], 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 $x \in \{1, \ldots, E\}$ and $y \in \{1, \ldots, C\}$ the number of CPUs to be activated at the edge and at the cloud, respectively, which correspond to our activation decisions. We

[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.

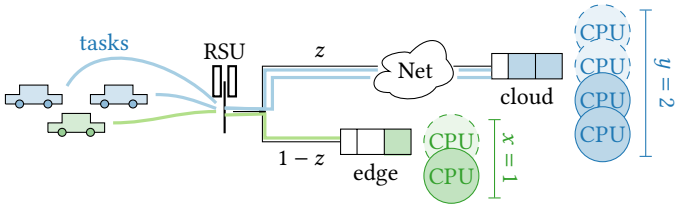


Figure 1: Vehicles produce tasks. We offload the task flows either to the cloud (with probability z) or to the edge (with probability $1 - z$). We process the frames by activating $x = 1$ CPU at the edge or $y = 2$ CPUs at the cloud. Maximum available CPUs: $E = 2$ (edge) and $C = 4$ (cloud).

assume that all CPUs have the same computational capacity K cycles per unit of time, and therefore what essentially distinguishes the edge from the cloud is the number of available CPUs [32, Sec. 2] and their distances to the vehicles (these assumptions could be relaxed as well). Assuming that the number of cycles required to process a task is proportional to its length [26] with constant c cycles/bit, the service time to process a task of length l_i equals

$$S(l_i) = l_i \frac{c}{K}. \tag{3}$$

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 [7], 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 cloud servers can be modeled as two different M/G/k systems: the edge with x active servers and arrival rate $\lambda_E = \lambda(1 - z)$, and the cloud with y active servers and arrival rate $\lambda_C = \lambda z$. 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 such as *Kingman's law*). In section 4 we present our approximation to characterize the CDF of the sojourn time.

Total delay experienced by a task. The total delay D_i experienced by a task i 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 activation and offloading decisions taken by the service provider, *i.e.*, variables x, y and z, respectively. Formally:

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

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

bounded and reliable transmission latency D_{tran} through a 3GPP-compliant NR deployment [4, Sec. 7] and Type B frame repetitions [3, Sec. 6.1.2.3.2]. Using a dedicated V2X slice for the service provider, propagation delays D_{prop} can also be bounded. Due to D_{prop} and D_{tran} being bounded, 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. 7.

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 [29] 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. Regarding the cost of the infrastructure usage, current pricing plans [2] 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:

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

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).

4 THE V2N COMPUTATION OFFLOADING AND CPU ACTIVATION PROBLEM

In this section we first 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).

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

$$s.t. \quad Eq.\ (1),$$

$$x \in \{0, 1, \ldots, E\}, \quad and \quad y \in \{0, 1, \ldots, C\}, \quad (7)$$

$$z \in [0, 1]. \quad (8)$$

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 $[0, 1]$.

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

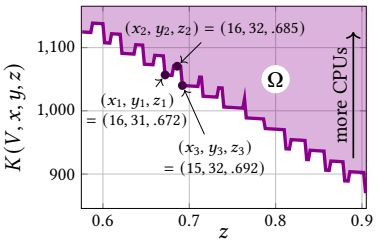


Figure 2: Total costs *vs.* offloading decision z for the feasibility region and boundary, depicted for an instance considering a ToD service [5] and real-world propagation delays [32].

V2N-COCA Problem's Structural Properties. We use these in Section 6, to design and analyse our computationally efficient and asymptotically optimal algorithm for its solution. 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.

Proof. 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. □

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 $\Omega := \left\{(x, y, z) : (1), (7), (8)\right\}$, *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.

Proof. **(1) - (2): non-monotonicity/non-continuity w.r.t. z.** We perform a proof by finding a **counterexample** for which the continuity and monotonicity properties do not hold. We want to stress that 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. In our counterexample, let $z_1 = 0.672$, $z_2 = 0.685$, and $z_3 = 0.692$. Clearly, it holds that $z_1 < z_2 < z_3$. However, from Fig. 2, which depicts the feasible region found by exhaustive searches, we observe 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). We remark that by the monotonicity definition, all other parameters except of the offloading policy z should have been fixed, while in our example the number of CPUs at the cloud increases from $y_1 = 31$ to $y_2 = 32$ as we move offloading from $z_1 = 0.672$ to $z_2 = 0.685$. The reason for this jump is 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. As identified above, the reason that these steps in the boundary occur is the change in the number of minimum CPUs that need to be activated in order 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 activated CPUs at the edge and at the cloud, respectively, the subscription cost given by the terms c_{0e} and c_{0c} in Eq. (5) is fixed. However, the total service time depends on the amount of tasks that are assigned at the edge and at the cloud, *i.e.*, by the offloading policy z. By combining eqs. (2), (3), and (5), we conclude the proof. □

Challenges 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 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)), and is the first of the problem's constraints, to ensure 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. The existing approximations for the M/G/k average waiting time do not suffice in URLLC, and we will compare against them in section 7. However, due to the safety concerns involved in V2N applications, it is of critical importance to know 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.

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 2: Parameters for frame length distribution l_i [17]

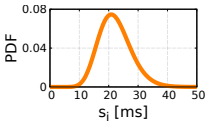


Figure 3: PDF of the service time s_i

5 OUR SOJOURN TIME APPROXIMATION

We start by presenting initial thoughts stemming from experimental evidence, we then discuss the structure of the URLLC requirement and our approximation proposal, and we validate it both in terms of accuracy and of impact w.r.t. solving the V2N-COCA Problem.

Remark: We focus on V2N computing tasks performed over video frames taken *e.g.*, from the front/back camera of autonomous vehicles [1], and we rely on evidence from the real-world traces in [17]. 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 that are arranged in a Group of Pictures (GOP) [17]. The frame length of each type $i \in \{I, B, P\}$ follows a Gamma distribution with parameters α_i (shape) and β_i (scale) in Table 2, and thus the frame length of the flow follows a mixture of them where each type i is weighted with weight w_i therein. Then, the average video frame length equals $l = w_I a_I \beta_I + w_B a_B \beta_B + w_P a_P \beta_P \approx 260$ kb. The number of cycles to process a video frame is proportional to its length, with a constant of approx. 21.42 cycles/bit [26]. 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 the case of 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. *The distribution f_{sojo} of the sojourn time D_{sojo} in an M/G/k queue, for video tasks [17], 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 [17] (i.e., a mixture of Gamma distributions). We formulate this as:*

$$f_{sojo} \coloneqq f_{W(M/G/k)} * f_{S(M/G/k)} \approx f_{W(M/D/k)} * f_{S(M/G/k)}. \quad (9)$$

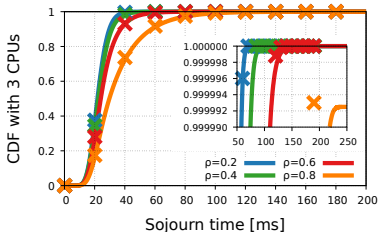
This proposition provides a closed-form expression to approximate with accuracy $\alpha = 0.01$ the sojourn time, *i.e.*, with a 99% confidence level in the estimation. The importance of this proposition is two-fold. First, it opens the way for solving Problem 1 by using existing results in the literature. Indeed, we can get $f_{W(M/D/k)}(t) = d/dt F_{W(M/D/k)}(t)$ using the following closed-form expression [11, 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 [11]. This supports dynamically scaling the computing resources as needed based on the computing load, thus resulting in minimisation of their operational costs. We now validate our proposition.

Kolmogorov-Smirnov test to rigorously validate Prop. 3. We compare the Cumulative Distribution Function (CDF) of the sojourn time obtained by convoluting the service times of $M/G/k$ queueing systems with the waiting times of (i) $M/G/k$ systems (obtained through exhaustive simulations), and (ii) $M/D/k$ systems (obtained using eq. (4.4) from [11]). We perform the comparison for different values of the intensity factor $I = \lambda/\mu$ (erlangs) and number of servers c, but due to space limitations we include only for $c = 3$ servers. We note that the values considered for c are in the order of magnitude of those required when experimenting with data from a real-trace dataset (see Section 7). 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 3: In all cases, the p-value remains above the significance level (α), leading to acceptance of H_0. In Fig. 4 we plot with lines our approximation, and with markers the exhaustive simulations, for $c = 3$ servers at the edge and cloud. We observe that our approximation of the CDF of the sojourn time that is obtained in the $M/G/k$ simulations, regardless of the load of the system, is both tight and conservative. We want to note that we have similar results for $c = 5$ and $c = 10$ in the system, but due to space limitations we omit them.



Take-away: Our approximation is tight & conservative regardless of the load ρ (colors) and number of servers.

Figure 4: Sojourn time of Proposition 3 approximation (lines) and $M/G/c$ simulations (markers).

c (servers)	I (Erlang) I=1	I=2	I=4	I=8
c = 10	(0,1)	(0,1)	(0,1)	(0,1)
c = 15	(0,1)	(0,1)	(0,1)	(0,1)
c = 20	(0,1)	(0,1)	(0,1)	(0,1)

Take-away: in all scenarios, H_0: "The CDF from analysis is slightly below from the CDF from simulations" is accepted.

Table 3: Comparison of (KS Statistics, P-value) parameters of the Kolmogorov-Smirnov test for different loads (I) and number of servers (c) with a significance level $\alpha = 0.01$.

The impact of the sojourn time approximation. We evaluate the effectiveness of our approximation of the sojourn time of M/G/k in Prop. 3, *vs.* the actual sojourn time of the M/G/k, in terms of its results when solving Problem 1. In Table 4 we present the results of different combinations of URLLC requirements expressed in terms of the maximum accepted delay T and the reliability P_G [5] (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 4 suggest that the optimal solution of the V2N-COCA problem found using our M/G/k approximation of Prop. 3 requires the same number of CPUs as when exhaustive M/G/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 Proposition 3 leads to near-optimal results and, given that is a closed formula, opens the way for the design of our efficient solution.

6 OUR JOINT OFFLOADING AND CPU ACTIVATION ALGORITHM

We now design **BiQui**[2]: an efficient **BI**nary search solution over the **Qu**e**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.

[2] https://github.com/MartinPJorge/biqui

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 4: Comparison (k_1, k_2) of the minimum CPUs required by Proposition 3 (k_1) and M/G/k simulations (k_2) to meet V2N service requirements upon different loads I.

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, and offloading policy z_0

1: Initialize: $z_0 = 1$ and $x_0 = 0$

2: Binary search on y using our $M/G/k$ 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 and $x \in \{0, 1, \dots, E-1\}$ **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: **end for**

6.1 Algorithm Design and Intuitive Explanation

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

Phase 1: Initialization. (lines 1). Initialize offloading policy as $z = 1$ and number of edge CPU activation as $x = 0$, motivated by the typically lower prices of the cloud servers, as compared to those specifically placed at the edge [32]. In case the opposite holds, the initialization is inverted as per $z = 0$ and $y = 0$.

Phase 2: Binary search. We perform a binary search in the number of available CPUs at the cloud (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 computational complexity in the worst case.

Phase 3: walking down the feasibility region. We exploit Prop. 1, *i.e.*, the cost function monotonicity w.r.t. CPU activation decisions, 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 the interval $[0, 1]$ and in steps of $z_{gran} \in (0, 1]$, *i.e.*, with a granularity z_{gran}. In Section 6.2 we detail the trade-off that emerges by this discretization choice, and prove BiQui's asymptotic optimality. Given these $1/z_{gran}$ values, we run a `for` loop (line 3) considering all sampled values of z. For each value we: (i) increase the activated CPUs at the edge until the reliability requirement is met (line 4), (ii) decrease the activated CPUs at the cloud while the reliability requirement is met (line 5). If the current configuration is better than the provisional one, we update the provisional one (line 6).

6.2 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 $O(1)$. Phase 2 has complexity $O(\log C)$ for the binary search (line 22). Phase 3 is a `for`-loop with $1/z_{gran}$ iterations. Although lines 4-5 are two `while`-loops nested within it, in practice the total number of times that the related commands will run is bounded from the number E and C of maximum 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 line 6, the complexity of the action to be taken should the condition be positive is $O(1)$, and it will be run $1/z_{gran}$ times. Thus, in total, for Phase 3 we have $O(E)+O(C)+O(1/z_{gran}) = O(E+C+1/z_{gran})$. That is, the total computational complexity of BiQui equals $O(\log C+C+E+1/z_{gran})$, *i.e.*, $O(C + E + 1/z_{gran})$.

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,*

$$\lim_{z_{gran} \to 0} K_{BiQui}(V, x, y, z) = K_{OPT}(V, x^*, y^*, z^*). \quad (10)$$

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

Proof. 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. □

7 PERFORMANCE EVALUATION

We assess BiQui's performance using real-world traffic traces, pricing plans currently used in the market by big industry players, and propagation delay setups drawn from previous related work on the topic. We consider reliability constraints imposed by 5G-Americas [5] for V2N applications.

7.1 Setting, Traces, and Benchmarks

Setting. In this section, unless otherwise specified, we consider $z_{\text{gran}} = 10^{-2}$ granularity, target delay $T_G = 100$ ms, and $P_G = 99.999$ % reliability [5]. We consider propagation delays from real-world providers [32]: edge Round Trip Time (RTT) $D^e_{\text{prop}} = 18.2$ ms and cloud RTT $D^c_{\text{prop}} = 22.8$ 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 [2]: $c_{0e} = 0.0052$, $c_{1e} = 1.363$ \$/h. Similarly, we consider the EC2 G4 instances price in Regional premises as reference for the cloud pricing, *i.e.*, we take $c_{0c} = 0.0052$, $c_{1c} = 0.94$ \$/h.

Real-world Traces. The traffic data [3] was recorded by 6 road probes located in streets in Torino, and comprises traffic flow measurements aggregated over 5 minutes. We present the geographical distribution in Fig. 5a, representing streets using points sized proportionally to their respective maximum traffic intensities. Fig. 5b depicts the traffic flows over the streets for one entire day. The aggregated demand achieves peaks $\lambda \leq 2.5$ pkt/ms (Fig. 5b), and we thus consider demands λ within $[0, 2.5]$.

Benchmarks. We compare BiQui performance against:

OPT: found through exhaustive search on (x, y, z). Although inefficient by means of running time and could not be used in practice in real systems, it shows the system's limits and allows us to compare BiQui against them.

AVG: finds optimal decisions (x, y, z) ensuring that the total ***avg*** delay of tasks remains below the delay constraint T [13, 25].

KNG: uses Kingman law of congestion to approximate the average waiting time in an M/G/k system [12].

SNC: adaptation of [31] that uses Stochastic Network Calculus (SNC), to capture the stochasticity of the service times for video processing tasks. We resort to the affine arrival/service curves [6] to bound the arrival/service excess/deficit.

OffAll: uses cloud as much as possible, then starts using edge.

LocAll: uses edge as much as possible, then starts using cloud.

7.2 Results

We evaluate BiQui against the benchmarks above by doing a sensitivity analysis on the problem's parameters, and over real traces.

The impact of varying RTTs. We consider high and low Round-Trip-Times (RTTs) for cloud servers. In Fig. 6 we depict normalized costs, offloading decisions z, number of activated CPUs at the edge and the cloud, *vs.* different generated computing load λ. Our main observations and insights per scheme are:

OPT: The max. supported computing load is up to 2.5 pkt/ms for low (Fig. 6a) and up to 0.75 pkt/ms for large (Fig. 6e) 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 limit BiQui, *i.e.*, for $\lambda \to 2.5$ for low RTTs (Fig. 6a) and for $\lambda \to 0.75$ for large RTTs (Fig. 6b). 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. 7, 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.*, [13, 25], which fail to capture the strict latency and reliability requirements of V2X services.

KNG: it only finds a solution under small RTT and $\lambda = 2.376$ (see Fig. 6a-d). Such artifact is caused due to the decreasing saw-tooth behaviour of the delay – see Fig. 7. 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. From Fig. 6a-d (low RTT), it eats up all CPUs with loads $\lambda \leq 0.71$, not finding feasible solutions for higher loads. From Fig. 6e-h (high RTT), it never finds feasible solutions. We conjecture that SNC provides rather loose bounds for the reliability P_G, hence pitfalls into resource over-provisioning.

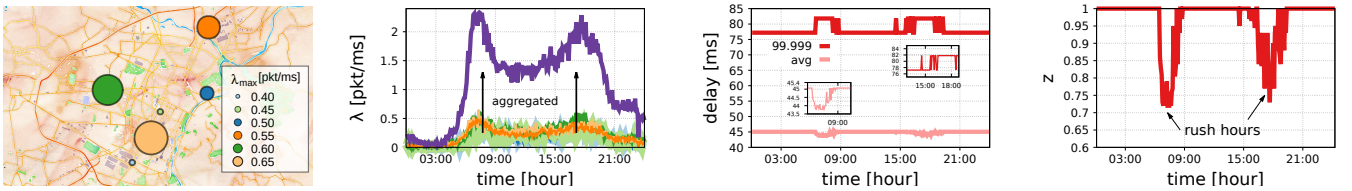
OffAll: its behaviour highly depends on RTTs. From Figs. 6a-d (low RTT) it matches OPT, as cloud is cheaper. However, in Figs. 6e-f (high RTT), feasible solutions are not possible due to delay violations stemming from high RTTs.

LocAll: is an optimal approach with large cloud RTT (as in Fig. 6e-h), for the only feasible solution is to locally process all tasks at the edge. Upon small cloud RTT (as in Fig. 6a-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, revealing its potential, while being relevant for more stringent scenarios that may be considered in future 6G applications. Table 5 compares BiQui decisions under the ToD target delay ($T = 100$ ms)[5], and $T = 77$ ms, upon different loads λ. As intuitively expected, in order to handle the tighter target delay, BiQui exploits the faster edge resources starting from lower computing loads λ (despite their higher costs). This consumes available resources earlier, and leads to unfeasible solutions for $T = 77$ ms and load $\lambda = 2.5$ pkt/ms.

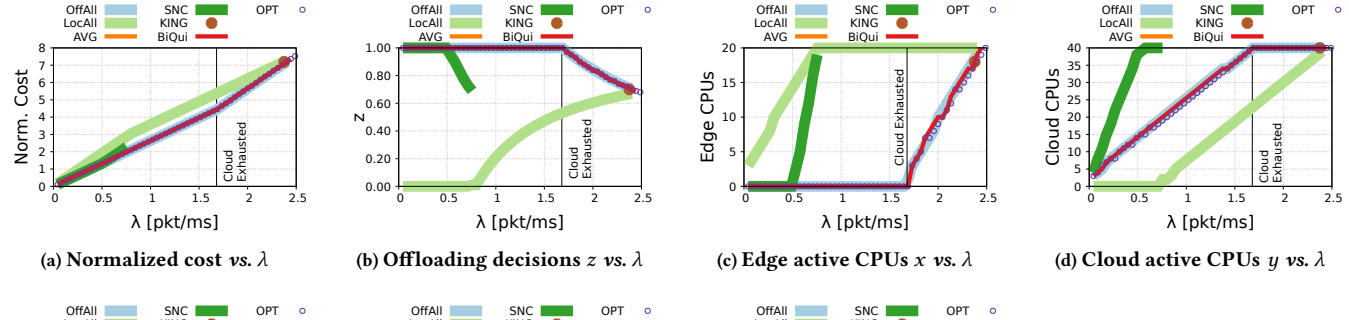
The impact of the granularity of the offloading decisions, z_{gran}. We use the same experimental setup as that for low cloud RTT. In Fig. 8 we depict BiQui for $z_{\text{gran}} = \{0.01, 0.1, 0.2, 0.3, 0.4\}$ *vs.* varying computing load λ. From Fig. 8b we observe that the offloading decision remains $z = 1$ when $\lambda \in [0, 1.73]$, *i.e.* until then BiQui offloads the load to the cloud (irrespectively of the granularity z_{gran}), thus producing the same solutions for all of these values of z_{gran}. We observe that BiQui matches OPT in Fig. 8. For loads $\lambda > 1.73$, both BiQui and OPT have consumed all the cloud CPUs,

[3] https://github.com/MartinPJorge/biqui/blob/master/data/traffic_torino_v02.csv



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

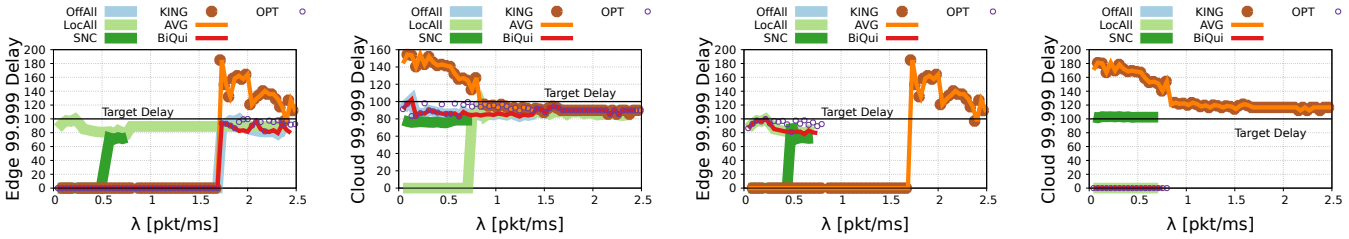
Figure 5: BiQui over real traces: The traffic demand (b) corresponds to that of five streets from Torino (a). Results show the delay experienced (c) by a ToD service [5] as BiQui changes the offloading decision (d) throughout a day.



(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

Figure 6: 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 [32]. Target delay: $T = 100$ ms. Reliability requirement: $P_G = 99.999\%$.

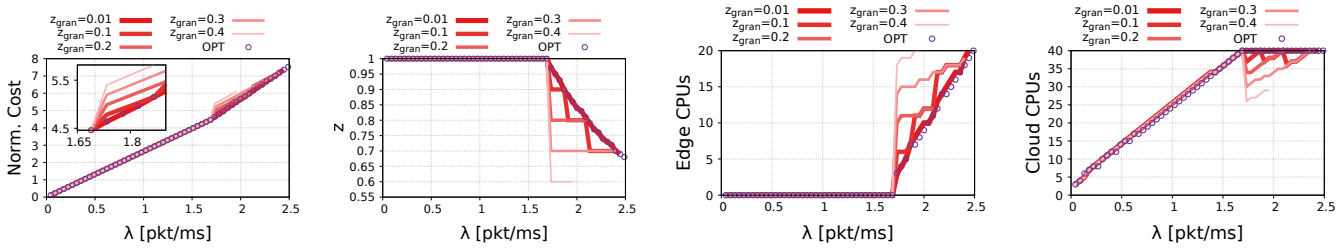


(a) Delay at edge - low cloud RTT (b) Delay at cloud - low cloud RTT (c) Delay at edge - high cloud RTT (d) Delay at cloud-high cloud RTT

Figure 7: 99.999-th percentile of the delay experienced by a ToD service [5] when it is processed at the edge and cloud. For the edge: $D^e_{\text{prop}} = 18.2$. For how cloud RTT: $D^c_{\text{prop}} = 22.8$ ms. For high cloud RTT: $D^c_{\text{prop}} = 49.1$ ms [32].

i.e., $x = C = 40$, thus requiring CPUs at the edge reducing the offloading decision ($z < 1$). If BiQui has *e.g.*, $z_{\text{gran}} = 0.1$ it will try to reduce the offloading down to $1 - z_{\text{gran}} = 0.9$ right after $\lambda = 1.73$ – see Fig. 8b. Such offloading reduction translates into turning on $x = 6$ CPUs at the edge and reduce the CPUs at the cloud to $y = 37$ – see Fig. 8e and Fig. 8d for $z_{\text{gran}} = 0.1$, respectively. However, a better solution would be to just turn on an additional "cheap" edge CPU right after $\lambda > 1.73$, as both OPT and BiQui do when it has a granularity $z_{\text{gran}} = 0.01$.

Fig. 8a highlights the cost deviation that BiQui experiences as we lower the granularity for $\lambda \in [1.65, 1.8]$. The reason behind such cost deviation is the excess of CPUs turned on at the edge due to excessive drops in the offloading z evidenced in Fig. 8b. Nevertheless, we observe that BiQui gets closer to the optimal as



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

Figure 8: Impact of BiQui granularity z_{gran}. Target delay $T = 100$ ms and reliability $P_G = 99.999\%$. We use $D^e_{\text{prop}} = 18.2$ (edge), $D^c_{\text{prop}} = 22,8$ ms (cloud) for propagation delays [32].

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 5: 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: $D^e_{\text{prop}} = 18.2, D^c_{\text{prop}} = 22.8$ ms [32].

we increase the granularity $z_{\text{gran}} \to 0$, confirming numerically Proposition 4 in section 6.2.

Varying the granularity of the offloading decisions also impacts BiQui's runtime, ranging from 6.3 to 25 ms for $z_{\text{gran}} = \{0.4, 0.3, 0.2, 0.1, 0.05\}$, and being 81ms and 119 ms for $z_{\text{gran}} = 0.01$ and low and high RTTs. Considering the constant flow of video frames in ToD services, at most only the first frame of the entire video flow may not meet the 100 ms requirement of ToD, and this only for the case that $z_{gran} = 0.01$ and high RTTs.

BiQui over a day. In Fig. 5c-d we plot an entire day, illustrating the delay experienced by tasks, and the portion of tasks z offloaded by BiQui, whose 99, 999th delay percentile remains below the target delay $T = 100$ ms. Fig. 5c also illustrates how the average task delay is half the 99.999 percentile, and that the task delay increase happens during peak hours (~8h and ~18h). Moreover, during rush hours, BiQui sends tasks to the edge to alleviate the saturated cloud – see how the offloading drops to $z = 0.75$ in Fig. 5d. The cloud saturation is reflected in the delay bumps experienced by tasks around the rush hours, for example, during 15:00-18:00 in Fig. 5c. Overall, BiQui manages the load oscillations due to the traffic rush hours leveraging the expensive edge resources when necessary.

8 CONCLUSION

We introduce the V2N Computation Offloading and CPU Activation (V2N-COCA) problem, aiming at minimizing operational costs (both monetary and energetic) while ensuring URLLC, by making decisions related to task offloading and CPU activation between the edge and the cloud. To overcome the non-existence of closed-form expressions to model the URLLC requirement, we resort to queuing theory expressions to approximate the service and waiting times of tasks in edge/cloud servers. We thoroughly and rigorously validate this approximation, in terms of both accuracy and effectiveness of solving the V2N-COCA Problem. Based on its structural properties, we design BiQui, a provably asymptotically optimal algorithm that is also computationally efficient (linear w.r.t. number of servers). Results show that BiQui outperforms the state of the art and meets the stringent V2N service requirements, meeting the target delay the 99.999% of the time. Future directions for research include exploring the performance of our approach in different contexts, such as aiming at the delay minimisation, to validate our intuition that BiQui will be an equally good solution for other cost functions.

ACKNOWLEDGMENTS

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. This work was supported by the Remote Driver project (TSI-065100-2022-003) funded by Spanish Ministry of Economic Affairs and Digital Transformation, 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.

REFERENCES

[1] [n. d.]. Nexar. https://www.getnexar.com/nexar-app/. Accessed: 16/01/2024.
[2] 2023. AWS EC2 calculator. https://calculator.aws/#/addService/ec2-enhancement. [Online; accessed January 30, 2024].
[3] 2023. *Physical layer procedures for data.* Technical Report TS 38.214 v18.0.0. 3GPP.
[4] 3GPP TS 22.261 2023. *Service requirements for the 5G system.* Technical Specification (TS) 22.261.v19.3.0. 3GPP.
[5] 5G Americas. 2021. Vehicular connectivity: C-V2X & 5G.
[6] Oscar Adamuz-Hinojosa, Lanfranco Zanzi, Vincenzo Sciancalepore, et al. 2024. ORANUS: Latency-tailored Orchestration via Stochastic Network Calculus in 6G O-RAN. arXiv:cs.NI/2401.03812 [ACCEPTED @INFOCOM2024].
[7] Zhiruo Cao, Zheng Wang, and E. Zegura. 2000. Performance of hashing-based schemes for Internet load balancing. In *IEEE INFOCOM*, Vol. 1. 332–341 vol.1.
[8] Luca Cominardi, Luis M. Contreras, Carlos J. Bcrnardos, et al. 2018. Understanding QoS Applicability in 5G Transport Networks. In *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 1–5.
[9] Penglin Dai, Kaiwen Hu, Xiao Wu, et al. 2021. Asynchronous Deep Reinforcement Learning for Data-Driven Task Offloading in MEC-Empowered Vehicular Networks. In *IEEE INFOCOM*. 1–10.
[10] Weiyang Feng, Siyu Lin, Ning Zhang, et al. 2023. Joint C-V2X Based Offloading and Resource Allocation in Multi-Tier Vehicular Edge Computing System. *IEEE Journal on Selected Areas in Communications* 41, 2 (2023), 432–445.
[11] G.J. Franx. 2001. A simple solution for the M/D/c waiting time distribution. *Operations Research Letters* 29, 5 (2001), 221–229.
[12] Varun Gupta, Mor Harchol-Balter, Jim G Dai, et al. 2010. On the inapproximability of M/G/K: why two moments of job size distribution are not enough. *Queueing Systems* 64 (2010), 5–48.
[13] Xingqiu He, Sheng Wang, Xiong Wang, et al. 2022. Age-Based Scheduling for Monitoring and Control Applications in Mobile Edge Computing Systems. In

IEEE INFOCOM. 1009–1018.

[14] Per Hokstad. 1978. Approximations for the M/G/m Queue. *Operations Research* 26, 3 (1978), 510–523. https://doi.org/10.1287/opre.26.3.510

[15] Yuyu Hu, Taiping Cui, Xiaoge Huang, et al. 2019. Task Offloading Based on Lyapunov Optimization for MEC-assisted Platooning. In *IEEE WCSP*. 1–5.

[16] Hongchang Ke, Jian Wang, Lingyue Deng, et al. 2020. Deep Reinforcement Learning-Based Adaptive Computation Offloading for MEC in Heterogeneous Vehicular Networks. *IEEE Transactions on Vehicular Technology* 69, 7 (2020), 7916–7929. https://doi.org/10.1109/TVT.2020.2993849

[17] H Koumaras, C Skianis, G Gardikis, et al. 2005. Analysis of H. 264 video encoded traffic. In *Proceedings of the 5th International Network Conference (INC2005)*. 441–448.

[18] Anitha Saravana Kumar, Lian Zhao, and Xavier Fernando. 2022. Multi-Agent Deep Reinforcement Learning-Empowered Channel Allocation in Vehicular Networks. *IEEE Transactions on Vehicular Technology* 71, 2 (2022), 1726–1736.

[19] Qian Liu, Rui Luo, Hairong Liang, et al. 2023. Energy-Efficient Joint Computation Offloading and Resource Allocation Strategy for ISAC-Aided 6G V2X Networks. *IEEE Transactions on Green Communications and Networking* 7, 1 (2023), 413–423. https://doi.org/10.1109/TGCN.2023.3234263

[20] Md. Noor-A-Rahim, Zilong Liu, Haeyoung Lee, et al. 2022. 6G for Vehicle-to-Everything (V2X) Communications: Enabling Technologies, Challenges, and Opportunities. *Proc. IEEE* 110, 6 (2022), 712–734.

[21] Armin Okic, Lanfranco Zanzi, Vincenzo Sciancalepore, et al. 2021. π-ROAD: a Learn-as-You-Go Framework for On-Demand Emergency Slices in V2X Scenarios. In *IEEE INFOCOM*. 1–10.

[22] Tao Ouyang, Kongyange Zhao, Xiaoxi Zhang, et al. 2023. Dynamic Edge-centric Resource Provisioning for Online and Offline Services Co-location. In *IEEE INFOCOM*. 1–10.

[23] Qi Qi, Jingyu Wang, Zhanyu Ma, et al. 2019. Knowledge-Driven Service Offloading Decision for Vehicular Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Transactions on Vehicular Technology* 68, 5 (2019), 4192–4203.

[24] Iftikhar Rasheed. 2022. Dynamic mode selection and resource allocation approach for 5G-vehicle-to-everything (V2X) communication using asynchronous federated deep reinforcement learning method. *Vehicular Communications* 38 (2022), 100532.

[25] Ju Ren, Jiani Liu, Yongmin Zhang, et al. 2022. An Efficient Two-Layer Task Offloading Scheme for MEC System with Multiple Services Providers. In *IEEE INFOCOM*. 1519–1528.

[26] Michael Roitzsch and Martin Pohlack. 2006. Principles for the prediction of video decoding times applied to mpeg-1/2 and mpeg-4 part 2 video. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. IEEE, 271–280.

[27] Tuyen X. Tran and Dario Pompili. 2019. Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks. *IEEE Transactions on Vehicular Technology* 68, 1 (2019), 856–868.

[28] M.H van Hoorn and H.C Tijms. 1982. Approximations for the waiting time distribution of the M/G/c queue. *Performance Evaluation* 2, 1 (1982), 22–28. https://doi.org/10.1016/0166-5316(82)90018-9

[29] A. Vasan, A. Sivasubramaniam, V. Shimpi, et al. 2010. Worth their watts? - an empirical study of datacenter servers. In *Proceedings of the Sixteenth International Symposium on High-Performance Computer Architecture (HPCA 2010)*. Bangalore, India.

[30] Jiadai Wang, Jiajia Liu, and Nei Kato. 2019. Networking and Communications in Autonomous Driving: A Survey. *IEEE Communications Surveys Tutorials* 21, 2 (2019), 1243–1274.

[31] Kai Xiong, Supeng Leng, Chongwen Huang, et al. 2021. Intelligent Task Offloading for Heterogeneous V2X Communications. *IEEE Transactions on Intelligent Transportation Systems* 22, 4 (2021), 2226–2238. https://doi.org/10.1109/TITS.2020.3015210

[32] Mengwei Xu, Zhe Fu, Xiao Ma, et al. 2021. From Cloud to Edge: A First Look at Public Edge Platforms. In *ACM IMC (IMC '21)*. Association for Computing Machinery, New York, NY, USA, 37–53.

[33] Tinghan Yang, Rongqing Zhang, Xiang Cheng, et al. 2016. A graph coloring resource sharing scheme for full-duplex cellular-VANET heterogeneous networks. In *2016 International Conference on Computing, Networking and Communications (ICNC)*. 1–5.