Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version 11 January, 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.011100

Modelling M/M/R-JSQ-PS Sojourn Time Distributions for URLLC Services

Geraint I. Palmer[1], Jorge Martín-Pérez[2]

[1]School of Mathematics, Cardiff University,UK
[2]ETSI Telecomunicación, Universidad Politécnica de Madrid, Spain

CORRESPONDING AUTHOR: Geraint I. Palmer (e-mail: palmergi1@cardiff.ac.uk).

This work was partially supported by José Castillejo mobility grant of the Spanish Ministry of Science, Innovation and Universities (grant no. CAS23/00455); Remote Driver Project under Grant TSI-065100-2022-003 of the Spanish Ministry of Economic Affairs and Digital Transformation; and European Union's Horizon Europe research and innovation program under Grant Agreement No. 101097122 (ACROSS).

ABSTRACT The future of networking promises to support time-sensitive applications that require ultra low latencies and reliabilities of 99.999%. Recent advances in cellular and WiFi connections enhance the network to meet high reliability and ultra low latencies. However, the aforementioned services require that the server processing time ensures low latencies with high reliability, otherwise the end-to-end performance is not met. To that end, in this paper we use queueing theory to model the sojourn time distribution for ultra reliable low latency constrained services of M/M/R-JSQ-PS systems: Markovian queues with R CPUs following a join-shortest-queue processor sharing discipline (for example Linux systems). We develop open-source simulation software, and develop and compare six analytical approximations for the sojourn time distribution. The proposed approximations yield Wasserstein distances below 2 time units, and upon medium loads incur into errors of less than 4.7 time units (e.g., milliseconds) for the 99.999th percentile sojourn time. Moreover, the proposed approximations are stable regardless the number of CPUs and stay close to the simulations regardless the service time distribution. To show the applicability of our approximations, we leverage on a real world vehicular dataset to scale a 99.999% reliable vehicular service.

INDEX TERMS queuing theory, simulation, URLLC

I. Introduction

RECENT advances in the networking community aim at a better control over infrastructure behaviour. Although the Internet was designed to provide a best-effort delivery [1] recent use cases require guarantees regarding the service latency, no matter whether the network is congested or not. Recent use cases such as vehicle to everything (V2X) [2], Industry 4.0 [3], drones control [4], or remote surgery [5] require that the Internet delivers packets with reliabilities above a 99.999%. That is, more than a 99.999% of the packets should meet latency constraints in the order of few milliseconds.

The aforementioned services lie within the category of Ultra-Reliable Low Latency Communications (URLLC). 5G and WiFi 6 are key enabler technologies, for their fast and reliable transmission mechanisms ensure wireless reliability (above a 99.999%) and transmission latencies in the order of few milliseconds. Namely, 5G provides URLLC using techniques such as mini-slot transmissions [6] to reduce the transmission time and Type B repetitions [7] to increase the probability of successful packet delivery in the radio transmissions. While IEEE 802.11 (WiFi) provides URLLC using techniques as traffic shapers [8], synchronised buffers [9], packet preemption [10], 802.1AS time sinchronization [11] Time-Aware Shaping [12], and Orthogonal Frequency Division Multiple Access (OFDMA).

Through the aforementioned techniques it is possible to ensure fast and reliable communications in V2X use cases [2], [13]. For example, a remote controller can provide Teleoperated Support (TeSo) to a vehicle with 5G connectivity. The vehicle sends a video stream using 5G and it is played in the monitor of the remote controller, who takes control over the vehicle to e.g. press the brakes.

To support URLLC in V2X services as TeSo, the end-to-end delay must be smaller than 20 ms with 99.999% reliability. Hence, the 99.999% of the time the (i) communication between the vehicle and the 5G antenna; and (ii) video processing/decoding at the remote control centre must take less than 20 ms. Works such as [14]–[16] propose solutions to perform resource allocation in the (i) radio communication between the vehicle and the 5G antenna to achieve URLLC. However, the aforementioned works oversee the (ii) processing time when computing the end-to-end delay of the service.

This work uses queueing theory to model the processing time of URLLC services (as V2X traffic). In particular, we focus on modelling the processing time in a server with multiple CPUs as processor sharing discipline, which is the case of Linux-based systems and the considered systems in network provisioning [17]–[19]. Existing works in the literature do consider these kinds if systems, see the related work in Section II, however they have not specified 99.999-percentiles of the sojourn time cumulative distribution function (CDFs).

To fill this gap, the present paper proposes six approximations for the CDF of the sojourn time in M/M/R-JSQ-PS systems. M/M/R-JSQ-PS systems follow a join shortest queue (JSQ) policy to dispatch incoming traffic among CPUs. We study JSQ for it is a popular in server farms [20] and it is how the CFS Linux scheduler [21] behaves when all server CPUs are dedicated to an application. The proposed approximations, with some leveraging [20], are useful to know how much time is consumed processing URLLC traffic, and hence to understand whether tasks' processing time meet the delay requirements with e.g. a 99.999% reliability. To validate the proposed approximations, we develop open-source simulation software based on discrete event simulation.

The contributions of our work are summarised as follows:

- We build a discrete event simulation for G/G/R-JSQ-PS systems;
- We propose six analytical approximations for the sojourn time cumulative distribution function (CDF) of M/M/R-JSQ-PS systems;
- We derive a run-time complexity analysis to obtain the sojourn time CDF using both the simulation and analytical approximations;
- We study which approximation is more accurate depending on the system load and number of CPUs;
- We use the proposed approximations to decide the scaling of an URLLC use case using a real-world vehicular dataset; and
- We explain how to reuse our results to derive scaling decisions in a network infrastructure with PS servers processing traffic for URLLC.

In terms of Wasserstein distance, the proposed analytical approximations deviate less than a 2 out of 182 time units from the sojourn time CDF in M/M/R-JSQ-PS systems, and we show that that serves as an upper bound to take scaling decisions for servers processing URLLC traffic, while achieving accuracies above a 99.999%.

The paper is structured as follows: in Section II we go over the related work about the sojourn time CDF in queueing systems. Then, in Section III we introduce the considered system that we study in this paper. In Section IV we detail the analytical approximations that we propose for the sojourn time CDF of M/M/R-JSQ-PS systems, and in Section V we study the run-time complexity of the proposed approximations. Later, in Section VI we discuss the development of the G/G/R-JSQ-PS simulation, and give some measure of its complexity for comparison with the proposed approximations. Then, in Section VII we use the simulation as a baseline for comparing the accuracies of the approximations derived earlier. Finally, in Section IX we use the derived expressions to scale an URLLC vehicular service, Section X offers discussion of the work and its potential and limitations, and in Section XI we conclude our work and point out future research directions.

II. Related work

In the networking community queueing theory is a well-established tool to assess the modelling of network infrastructure [22], [23]. The packet-based nature of the Internet, so as the buffering and processor sharing nature of servers, make it a useful theoretical tool to derive insights on the behaviour of the network. Recent URLLC services and their urgent need for communication guarantees can benefit from the theoretical results of queueing theory in order to adequately provision the network.

The fundamental results of queueing theory [22] give closed-form formulas for the sojourn time (waiting plus service time) of M/M/1 systems, i.e. systems with 1 server that has exponential service time to dispatch customers arriving according to a Poisson distribution and queue up before they are served. Namely, it is possible to find both the average and CDF for the sojourn time of M/M/1 systems, with the latter having also an exponential distribution [22].

However, the internet traffic is typically dispatched in parallel by multiple servers or CPUs within a server. Hence, it is better resorting to M/M/R systems with up to R servers/CPUs that attend customers in parallel. For such systems, the queueing theory fundamentals also give closed-form expressions for the average sojourn time [22], and indications on how to derive its CDF [23].

But still, both M/M/1 and M/M/R systems may not be suitable to model networking components, as the assumption of exponential service times may not be realistic. To that end, the literature has devoted effort to derive the sojourn time CDF expressions of systems not satisfying such assumptions. For example [24], [25] provide expressions for the sojourn time CDFs of M/D/1 and M/G/1 systems, respectively. However both works provide the sojourn time CDF expression in the form of the Laplace-Stieltjes transform, i.e. a non-closed expression of the sojourn time CDF.

Other works such as [26] generalise the Possion arrival process to a Markovian arrival processes (MAP) and provide closed-formulas for the sojourn time CDF in MAP/M/1 systems. However in general, making the assumption of Poissonian arrivals is fair as long as there is a considerable amount of independent flows, as the Palm–Khintchine theorem states [27]. Hence, it is reasonable to model data centres as M/G/R systems, as suggested by [23], where server farms for traffic processing are used as motivation for studying M/M/R systems.

Nevertheless, M/M/R systems with first-come-first-serve service discipline do not mimic the behaviour of Linux based systems, where each CPU shares the computing time using a processor sharing (PS) discipline, where packets wait in the queue until a server finishes processing a job. Under processor sharing jobs all receive service concurrently, but their instantaneous service rate changes with the number of concurrent jobs being served. They therefore experience slower service rates the more jobs are present. A survey on the modelling of queues with PS discipline is given in [28].

Given a number of parallel PS servers, a number of authors have given consideration to the scheduling or distribution of arriving jobs amongst the servers, for example [29]–[32]. In particular, [20], [33], [34] consider join-shortest-queue, or load-balancing schemes. For example, in [20] the research resorts to single queue analysis (SQA) to provide insights on how the traffic intensity changes depending on the queue occupation at each CPU, so as the average number of jobs at each CPU; and in [34] they aims to minimize the average number of users at all servers. However, none specify what is e.g. the 99.999-percentile of the sojourn time (that is the queueing and processing time).

Although the queueing theory literature has widely studied the sojourn time in different systems, it has not yet managed to find out exact sojourn time CDFs in systems with the multiple PS CPUs of Linux based servers. To the best of our knowledge, the existing literature does not provide expressions to compute the sojourn time CDF in PS multi-processor systems that are close to those servers that will process URLLC traffic. Therefore this paper contributes to the related work by proposing six approximations for the sojourn time CDF of M/M/R-JSQ-PS systems. The proposed approximations are useful to check whether the URLLC traffic processing will meet the 99.999% or similar guarantees of URLLC with almost negligible latencies in the order of few milliseconds.

When exact analytical expressions or closed form approximations for CDFs are intractable, researchers often run stochastic simulations of the system. Discrete-event simulation is a standard technique for the task [35], with a number of commercial (e.g. Simul8 [36] and AnyLogic [37]) and open-source (e.g. Simmer [38], SimPy [39], and Ciw [40]) software options. However, to the authors' knowledge, prior to the work of this paper the listed options do not offer straightforward out-of-the-box ways to simulate processor sharing servers, requiring bespoke code or modifications. Therefore, a major contribution of this paper is the extension of the Ciw software to be able to simulate various kinds of processor sharing queues. This work is described in Section VI.

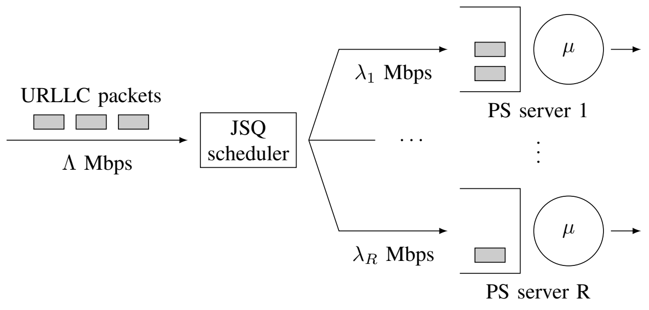


FIGURE 1: M/M/R-JSQ-PS system processing URLLC packets.

III. An M/M/R-JSQ-PS queueing system

This work is concerned with the sojourn time distribution $\mathbb{P}(T \leq t)$ of customers in an M/M/R-JSQ-PS system, that is a system with R parallel processor sharing queues, with overall Poisson arrival rate Λ, and intended service times distributed exponentially with rate μ. Customers join the processor sharing queue that has the least amount of customers.

Processor-sharing is a queueing discipline where all customers are served simultaneously, but the service load is shared between the customers. That is, if a customer is expecting to receive a service time s, then the rate at which that service is given is s/n when there are n customers present. Therefore if there are n customers present throughout the customer's service, then it will last sn time units. A key feature is that n can vary during that customer's service.

FIGURE 1 illustrates this system.

IV. M/M/R-JSQ-PS sojourn time CDF approximations

In order to find the sojourn time distribution of a join-shortest-queue processor-sharing M/M/R-JSQ-PS queue, we follow an approach outlined in [20], called Single Queue Analysis (SQA). Here, rather than considering the whole M/M/R queue, we consider each server as its own M_n/M/1-PS queue, with state-dependent arrival rates dependent on the join-shortest-queue mechanism, which relies on the state of all R parallel M_n/M/1-PS queues. Let Λ denote the overall arrival rate to the M/M/R-JSQ-PS queue, then for each PS server their effective state-dependent arrival rate is λ_n when there are n customers already being served by that server. Note that the probability of there being n customers at the server depends on how the JSQ mechanism operates given the other queues. TABLE 1 summarizes the notation used throughout this paper.

TABLE 1: Notation table

Symbol	Definition
T	random variable for the customer sojourn time
Λ	overall arrival rate.
μ	intended service rate
R	number of parallel processor sharing servers
ρ	traffic intensity $\rho = \frac{\Lambda}{R\mu}$
λ_n	arrival rate experienced by a server with n customers
W	complementary sojourn time CDF, $W(t) = \mathbb{P}(T > t)$
w_n	W with n customers, $w_n(t) = \mathbb{P}(T > t \mid n)$
A_n	probability of joining a server with n customers
π_n	portion of arrivals a server receives with n customers
$C(\mathbf{v}, b)$	number of occurrences of b in the vector $\mathbf{v}$
$Z(\mathbf{v}, b)$	set of indices in $\mathbf{v}$ where b occurs
Q	system transition matrix, with entries $q_{i,j}$
p_j	probability of being in state j
D	defective infinitesimal generator
L_1	maximum number of customers at a server
L_2	maximum number of customers at the system
q_{max}	maximum runtime of the simulation
q_{warmup}	warmup time used in the simulation
t_{max}	largest value of T calculated
$\Omega(G, H)$	Wasserstein distance between CDF G and CDF H
F	random variable representing the RTT
τ	overall (sojourn time plus RTT, $T + F$) latency target
η	overall (sojourn time plus RTT, $T + F$) reliability
η_T	sojourn time reliability
η_F	RTT reliability
t^{η_T}	η_T percentile of T
f^{η_F}	η_F percentile of F

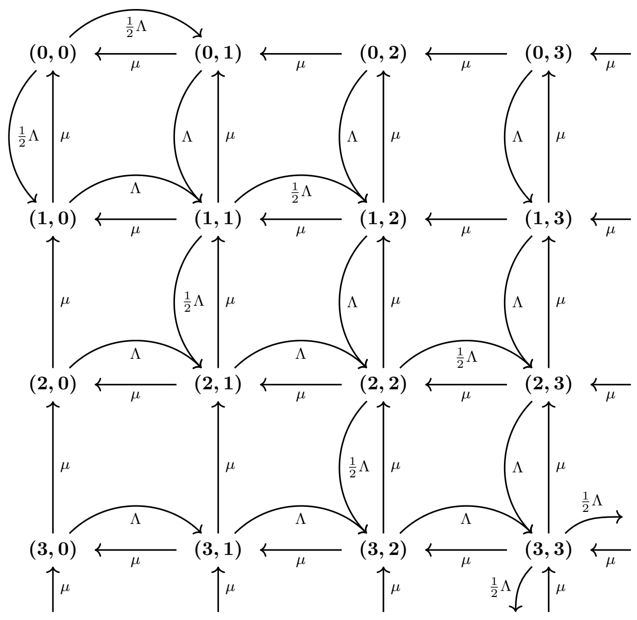


FIGURE 2: Transition state diagram of the M/M/R-JSQ-PS system when $R = 2$.

Now, considering a single server as its own queue, we adapt the methodology developed in [26] to the join-shortest-queue situation. In that paper Theorem 1 gives the sojourn time CDF of a single MAP/M/1-PS queue. A small adaptation, now considering an generic MAP process state-dependent Markovian arrivals λ_n, gives the sojourn time CDF as:

$$\mathbb{P}(T \leq t) = 1 - \mathbb{P}(T > t) = 1 - W(t) = 1 - \sum_{n=0}^{\infty} A_n w_n(t) \tag{1}$$

where A_n is the probability of an arriving customer joining the queue when there are n customers already present, and $w_n(t)$ is the conditional probability that the sojourn time is greater than t given that there are n customers already present at arrival.

We study two approximations each for finding the λ_n, A_n, and w_n for each n. Then combining these in (1) gives us six approximations of the sojourn time CDF for an M/M/R-JSQ-PS queue. The second approximations for λ_n, A_n are the ones proposed in [20] and we present them to have a self-contained section. Lastly, the second approximation of w_n follows the proof strategy from [20] using the defective infinitesimal generator we propose. Therefore, our contributions are the first approximations for λ_n, A_n so as the first and second approximation for w_n.

A. First approximation of λ_n

Note first that the arrival rate for each single queue being dependent on the number of customers already present in that queue is a valid assumption: the arrival rates to each individual queue when there are n customers already present depends on the probability of n being the smallest number of customers present in all R of the queues. This however is not straightforward to calculate in isolation of the other R queues, therefore we resort to approximations.

First we note that $\lambda_n = \pi_n \Lambda$, where π_n is the proportion of arrivals a server will receive if they have n customers already present.

We find π_n by constructing a truncated Markov chain of the M/M/R-JSQ-PS system. Define the state space of the non-truncated Markov chain by $S = \{(a_1, a_2, \dots, a_R) \; \forall \; a_1, a_2, \dots, a_R \in \mathbb{N}_0\}$, where a_z denotes the number of customers with server z. Let $\mathbf{s}_i \in S$ denote the ith state of the Markov chain, which indicates the number of customers at each server. For example, the ith state being $\mathbf{s}_i = (2, 3, \dots, 4)$ to indicate the first, second and last CPU have 2, 3 and 4 customers; respectively. Define the transition rate $q_{i,j}$ from $\mathbf{s}_i$ to $\mathbf{s}_j$, for all i, j, by (2):

$$q_{i,j} = \begin{cases} \mu & \text{if } C(\delta,0) = R-1 \wedge C(\delta,-1) = 1; \\ \frac{\Lambda}{C(\mathbf{s}_i,\min(\mathbf{s}_i))} & \text{if } \delta = C(\delta,0) = R-1 \wedge C(\delta,1) = 1 \\ & \wedge\ Z(\delta,1) \subseteq Z(\mathbf{s}_i,\min(\mathbf{s}_i)); \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $\delta = \mathbf{s}_i - \mathbf{s}_j$; $C(\mathbf{v}, b) = |\{z \in \mathbf{v} : z = b\}|$ is a function that counts the number of occurrences of b in a vector $\mathbf{v}$; and $Z(\mathbf{v}, b) = \{z : \mathbf{v}_z = b\}$ is the set of indices in $\mathbf{v}$ where b occurs. FIGURE 2 is a representation of the Markov chain when $R = 2$. When $R = 1$ this reduces to an M/M/1 (or equivalently M/M/1-PS) system, and it becomes difficult to represent this system when $R > 2$.

Steady-state probabilities can be found numerically by truncating the Markov chain, that is choosing an appropriate L_1 such that $a_z < L_1$ for all servers z, and solving $\mathbf{p}Q = \mathbf{0}$ with $\mathbf{pe} = 1$, where Q is the transition matrix with entries $q_{i,j}$ and $\mathbf{e}$ is the vector of ones.

Once all p_i are found, the proportion of arrivals a server will receive if they have n customers already present, π_n, can be found using (3):

$$\pi_n = \left(\sum_{\substack{\mathbf{s}_{i,0}=n \\ \min \mathbf{s}_i = n}} \frac{p_i}{C(\mathbf{s}_i, n)} \right) \left(\sum_{\mathbf{s}_{i,0}=n} p_i \right)^{-1} \tag{3}$$

where $\mathbf{s}_{i,0}$ represents the number of customers at the first server when in state i. In the aforementioned example – with $\mathbf{s}_i = (2, 3, \dots, 4)$ – the first server has $\mathbf{s}_{i,0} = 2$ CPUs.

B. Second approximation of λ_n

The authors of [20] provide numerical approximations for $\lambda_0, \lambda_1, \lambda_2$ in [20, Section 5], given in (4), (5) and (6), and all other λ_n for $n \geq 3$ by (7).

$$\lambda_0 = \mu \left(k_a - k_b k_c^R - k_d k_e^R \right) \tag{4}$$

$$\lambda_1 = \frac{\mu \left(\rho^R - 1 + \frac{\mu(\rho - \rho^{R+1})}{\lambda_0 (1-\rho)} \right)}{\frac{\lambda_2}{\mu} - \rho^R + 1} \tag{5}$$

$$\lambda_2 = \mu k_f k_g^R \tag{6}$$

$$\lambda_n = \mu \left(\frac{\Lambda}{n\mu} \right)^n \tag{7}$$

with k_a, k_b, k_c, k_d, k_e, k_f and k_g defined by:

$$\begin{aligned} k_a &= \frac{\rho}{(1-\rho)} \\ k_b &= \frac{-0.0263\rho^2 + 0.0054\rho + 0.1155}{\rho^2 - 1.939\rho + 0.9534} \\ k_c &= -6.2973\rho^4 + 14.3382\rho^3 - 12.3532\rho^2 + 6.2557\rho - 1.005 \\ k_d &= \frac{-226.1839\rho^2 + 342.3814\rho + 10.2851}{\rho^3 - 146.2751\rho^2 - 481.1256\rho + 599.9166)} \\ k_e &= 0.4462\rho^3 - 1.8317\rho^2 + 2.4376\rho - 0.0512 \\ k_f &= -0.29\rho^3 + 0.8822\rho^2 - 0.5349\rho + 1.0112 \\ k_g &= -0.1864\rho^2 + 1.195\rho - 0.016 \end{aligned}$$

C. First approximation of A_n

Using the same Markov chain defined in Section A, A_n can be found by manipulating the steady-state probabilities p_i, given in (8):

$$A_n = \sum_{\min \mathbf{s}_i = n} p_i. \tag{8}$$

D. Second approximation of A_n

From the SQA we can consider each PS server to be its own M/M/1-PS queue with state-dependent arrival rates. This gives a birth-death process, where A_n is the probability of that system being in state n. Thus we have:

$$A_n = \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu} A_0 \tag{9}$$

$$A_0 = \left(1 + \sum_{i=1}^{\infty} \prod_{j=0}^{i-1} \frac{\lambda_j}{\mu} \right)^{-1}. \tag{10}$$

E. First approximation of $w_n(t)$

In [26], for a general MAP, $w_n(t)$ is given explicitly by:

$$w_n(t) = e^{Dt}\mathbf{e} \tag{11}$$

where D is considered to be a defective infinitesimal generator that defines the sojourn time of a customer arriving in state n. Simplifying the MAP to be state-dependent arrival λ_n, D takes the from:

$$D = \begin{pmatrix} -(\lambda_0+\mu) & \lambda_0 & 0 & 0 & \dots \\ \frac{1}{2}\mu & -(\lambda_1+\mu) & \lambda_1 & 0 & \dots \\ 0 & \frac{2}{3}\mu & -(\lambda_2+\mu) & \lambda_2 & \dots \\ 0 & 0 & \frac{3}{4}\mu & -(\lambda_3+\mu) & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{12}$$

By constructing a truncated D explicitly, numerical methods, such as Padé's method [41], are used to find the matrix exponential.

F. Second approximation of $w_n(t)$

As constructing D explicitly and numerically computing a matrix exponential can be computationally inefficient, in Lemma 1 we give a recurrent relation for finding $w_n(t)$.

Lemma 1. *If a server within an M/M/R-JSQ-PS system has n customers, its sojourn time CDF is*

$$\mathbb{P}(T > t \mid n) = w_n(t) = \sum_{i=0}^{\infty} \frac{(\lambda_0+\mu)^i t^i}{i!} e^{-(\lambda_0+\mu)t} h_{n,i} \tag{13}$$

with $h_{n,0} = 1$ for all n, $h_{-1,i} = 0$ for all i, and $h_{n,i}$ satisfying

$$h_{n,i+1} = \frac{n}{n+1} \frac{\mu}{\lambda_0+\mu} h_{n-1,i} + h_{n,i}\left(1 - \frac{\lambda_n+\mu}{\lambda_0+\mu}\right) + \frac{\lambda_n}{\lambda_0+\mu} h_{n+1,i} \tag{14}$$

Proof:
We mimic the proof presented in [26, Corollary 2], where authors derive the vector of sojourn time complementary CDF $\mathbf{w}(t)$, with $\mathbf{w}_n(t) = \mathbb{P}(T > t \mid n)$, using the differential equation $\frac{d}{dt}\mathbf{w}(t) = D\mathbf{w}(t)$. With explicit solution $\mathbf{w}(t) = e^{Dt}\mathbf{e}$.

In the above explicit solution, D is considered the defective infinitesimal generator taking the form given in (12). By applying the uniformisation technique [42], the explicit solution of the aforementioned differential equation is

$$\mathbf{w_n}(t) = \sum_{i=0}^{\infty} \frac{(\lambda_0+\mu)^i t^i}{i!} e^{-(\lambda_0+\mu)t} \left[I + \frac{1}{\lambda_0+\mu} T\right]^i \mathbf{e} \tag{15}$$

with I the identity matrix. To ease the computation of the matrix to the power of i, (i.e., $[\cdot]^i$) the following vector is defined $\mathbf{h}_{n,i} = \left[I + \frac{1}{\lambda_0+\mu}T\right]^i \mathbf{e}$. And it leads to the recursion $\mathbf{h}_{n,i+1} = \left[I + \frac{1}{\lambda_0+\mu}T\right]\mathbf{h}_{n,i}$, with $\mathbf{h}_{n,0} = \mathbf{e}, \forall n$. As a result, $\mathbf{w}(t)$ is defined as

$$\mathbf{w_n}(t) = \sum_{i=0}^{\infty} \frac{(\lambda_0+\mu)^i t^i}{i!} e^{-(\lambda_0+\mu)t} \mathbf{h}_{n,i} \tag{16}$$

and the nth element of $\mathbf{w}(t)$ is given by (13). ■

This gives $w_n(t)$ in a form which, for a sufficiently large value, L_2, in place of infinity, can be found recursively. This naive adaptation of [26] replaces their static MAP with the state-dependent arrival rate λ_n.

G. Summary & Considerations

In this work we implement and test six different methods of approximating the complementary sojourn time CDF of an M/M/R-JSQ-PS system, $W(t)$. TABLE 2 summarises the methodology.

TABLE 2: Summary of the six methods of calculating $W(t)$.

Method	λ_n	A_n	$w_n(t)$
A	A	C	E
B	A	D	E
C	B	D	E
D	A	C	F
E	A	D	F
F	B	D	F

Choices of model hyper-parameters, those that concern only the methodology and not the system that is itself being modelled, can effect both the accuracy and run-time (or computational complexity) of the model, and choices are usually a compromise between the two. Each of the six sub-methods described in Section IV have hyper-parameters than need to be chosen. Those that explicitly build an infinite Markov chain, that is methods A and C, need to truncate the Markov chain using a limit L_1, so that numerical methods can be used on a finite Markov chain. The limit L_1 corresponds to the maximum number of customers each PS server will receive. Thus these Markov chains will have L_1^R states, and so its construction requires defining L_1^{2R} transitions. The larger the L_1 the more accurate the model, as there would be a smaller probability of a server receiving more than L_1 customers, however larger limits have longer run-times and larger memory consumption.

Other sub-methods, methods D and F contain infinite sums. For these, a sufficiently large cut-off, L_2 is required to truncate these sums for numerical computation. This L_2 corresponds to the overall maximum number of customers that can be present, and so can be chosen to much larger than L_1. Similarly, method E requires the construction of a matrix, where each state corresponds the the overall number of customers, and so L_2 is also be used to truncate this matrix.

H. Markov chain truncation

When we approximate λ_n and A_n using Section A and Section C, respectively, we truncate the transition matrix Q of the Markov chain in (2). Namely, we limit the “last” considered state $S_i = (L_1 - 1)\mathbf{e}$ has $L_1 - 1$ users in all the R servers. The truncation L_1 should be carefully selected such that

$$\sum_{\mathbf{s}_j:\ \max \mathbf{s}_j \geq L_1} p_j < \varepsilon \tag{17}$$

that is, the probability of entering a state with a server with L_1 or more customers should remain below a tolerance $\varepsilon \in \mathbb{R}^+$

FIGURE 3 illustrates how the probability of having L_1or more users at a server decreases as we increase the truncation limit L_1 and how this is effected by both ρ and R. This data was obtained using the simulation described in Section VI.

V. Complexity analysis

It is of paramount importance to consider the run-time complexity of each approximation $\lambda_n, A_n, w_n(t)$, as a network operator may require fast operational decisions to satisfy the

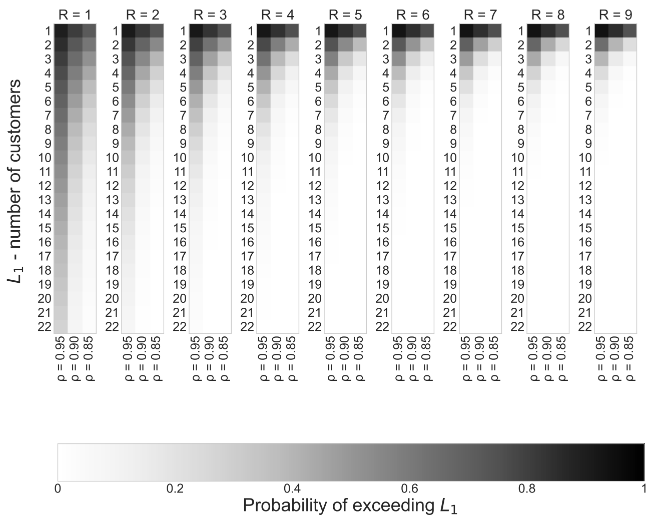


FIGURE 3: Probability of having L_1 or more customers at some server (17) with different loads $\rho = 0.85, 0.90, 0.95$ and available servers $R = 1, \ldots, 9$.

URLLC. If the approximation run-time is not fast enough, the operator would not be able to update the scaling and routing strategies upon demand changes in time. Therefore, in the following we analyse the run-time complexity of each approximation for λ_n, A_n and $w_n(t)$.

A. First approximation of λ_n

Using (3) this approximation finds the portion of arrivals that a server foresees using the steady-state probabilities p_i of the M/M/R-JSQ-R Markov chain with L_1^R states and transition matrix Q of size L_1^{2R}. For each entry $q_{i,j}$ of the transition matrix we make $\min(\mathbf{s}_i), Z(\delta, b), C(\delta, b)$ operations, all of them of complexity $\mathcal{O}(R)$. Hence, computing all entries of the transition matrix Q takes $\mathcal{O}(RL_1^{2R})$ operations.

To find the steady-state vector $\mathbf{p}$ we solve $(\tilde{Q}|\mathbf{e})^T\mathbf{p} = (\mathbf{p}|\mathbf{e})^T$, where $\tilde{Q}$ is the transition matrix Q less one row. This is a linear system with a matrix of size $L_1^R \times L_1^R$. Finding such solution with the LAPACK [43] gesv method leads to a cubic run-time complexity on the matrix size. Therefore, obtaining the steady-state probability takes $\mathcal{O}(L_1^{3R})$. Note that it is the computation of $\mathbf{p}$ that dominates the complexity of approximating λ_n, as creating the transition matrix Q has $\mathcal{O}(RL_1^R)$ complexity and computing π_n has $\mathcal{O}(L_1^R)$ complexity – see (3). Hence, the first approximation of λ_n has run-time complexity $\mathcal{O}(L_1^{3R})$.

B. Second approximation of λ_n

In (7) we see that there is a power relationship between n and λ_n, namely, $\lambda_n = \mu(\frac{\Lambda}{n\mu})^n$. As computing a power has complexity $\mathcal{O}(\log n)$, the second approximation of λ_n has complexity $\mathcal{O}(\log n)$.

C. First approximation of A_n

Once we compute the Markov chain steady-state probabilities p_n, this method only performs a summation over such probabilities (8). Thus, the complexity of computing A_n is $\mathcal{O}(L_1^R)$, for we iterate over all the L_1^R states and check whether each of them satisfies $\min \mathbf{s}_j = n$.

D. Second approximation of A_n

Given the values of λ_n, first we compute the probability of joining the queue when there are 0 users A_0 in (10). As mentioned in Section G, we truncate the infinite summations up to L_2. Hence, it takes $\sum_i^{L_2} i$ operations to compute A_0, and so is $\mathcal{O}(L_2^2)$. Once A_0 is computed, we perform $\mathcal{O}(L_2)$ operations to compute A_n in (9). Therefore as a whole, the second approximation of A_n has $\mathcal{O}(L_2^2)$ complexity.

E. First approximation of $w_n(t)$

This approximation computes the exponential of the defective infinitesimal generator matrix D – see (11). As mentioned in Section G, we also truncate the D matrix up to L_2 elements in its diagonal such that D is an $L_2 \times L_2$ matrix. As D is diagonal with ≤ 3 terms at each row, its creation has complexity $\mathcal{O}(L_2)$. With Padé's method [41] we compute T exponential with $\mathcal{O}(L_2 \log L_2)$ complexity.

F. Second approximation of $w_n(t)$

Using the recurrent formula of Lemma 1 we can check the complexity of this second approximation of $w_n(t)$. As mentioned in Section G, we truncate the infinite summation in (13) to L_2 iterations. At each summation iteration i, we perform $\mathcal{O}(\log i)$ operations (the power operators), hence, computing the second approximation of $w_n(t)$ has complexity $\mathcal{O}(L_2 \log L_2)$. Note that we compute $h_{n,i}$ incrementally thanks to the recursive approach, hence, such computation does not dominate the approximation complexity as $h_{n,i+1}$ reuses already computed values of $h_{*,i}$. Similarly, we also keep inside a hash table the factorial computations $i!$ at (13) denominator to ease the computational burden.

Depending on which Method we use – see TABLE 2 – we will get different run-time complexities. Namely, methods A,B,D, and E have an $\mathcal{O}(L_1^{3R})$ complexity because they rely on the truncated Markov chain to derive λ_n, which is the most demanding approximation. While methods C and F have an overall complexity of $\mathcal{O}(L_2^2)$ because the B and F approximations dominate the computation of $W(t)$. TABLE 3 summarises the computational complexity of each method.

VI. Simulation of G/G/R-JSQ-PS

Simulation offers an alternative method to find sojourn time CDFs, and a baseline to compare the approximations' accuracies. Generally simulation models are less efficient than analytical or approximation methods, especially in cases where a large number of simulated events need to occur before a good approximation of the required measures

TABLE 3: Complexity of each method.

Method	λ_n	A_n	$w_n(t)$	Overall
A	$\mathcal{O}(L_1^{3R})$	$\mathcal{O}(L_1^{R})$	$\mathcal{O}(L_2 \log L_2)$	$\mathcal{O}(L_1^{3R})$
B	$\mathcal{O}(L_1^{3R})$	$\mathcal{O}(L_2^{2})$	$\mathcal{O}(L_2 \log L_2)$	$\mathcal{O}(L_1^{3R})$
C	$\mathcal{O}(\log n)$	$\mathcal{O}(L_2^{2})$	$\mathcal{O}(L_2 \log L_2)$	$\mathcal{O}(L_2^{2})$
D	$\mathcal{O}(L_1^{3R})$	$\mathcal{O}(L_1^{R})$	$\mathcal{O}(L_2 \log L_2)$	$\mathcal{O}(L_1^{3R})$
E	$\mathcal{O}(L_1^{3R})$	$\mathcal{O}(L_2^{2})$	$\mathcal{O}(L_2 \log L_2)$	$\mathcal{O}(L_1^{3R})$
F	$\mathcal{O}(\log n)$	$\mathcal{O}(L_2^{2})$	$\mathcal{O}(L_2 \log L_2)$	$\mathcal{O}(L_2^{2})$

are found. This is the case here with the G/G/R-JSQ-PS simulation, where under high loads the simulation may halt due to the experienced service time of customers approaching infinity. However, for reasonable traffic intensities, simulation can provide an alternative baseline to measure the approximation accuracies. Here we outline implementation details and complexity analysis for the simulation.

A. Implementation

In discrete event simulation a virtual representation of a queueing system is created, and 'run' by sampling a number of basic random variables such as arrival dates of customers and intended service times, which interact with one another and the system to emulate the behaviour of the queueing system under consideration. Given a long enough runtime and/or a large enough number of trials, observed statistics will converge to exact values due to the law of large numbers. However due to their stochastic nature convergence may be slow, and depending on the complexity of the system, can be computationally expensive. Here the Ciw library [40] is used, an open-source Python library for conducting discrete event simulation. A key contribution of this work is the adaption of the library to include processor-sharing capabilities, which were included in release v2.2.0: these capabilities include standard processor sharing, limited processor sharing as described in [44], and capacitated processor sharing as described in [45], and their combinations.

Ciw uses the event-scheduling approach to discrete event simulation [40]. Events in the sense of a discrete event simulation correspond to individual calculations or manipulations to the simulated individuals of the system, for example a customer joining a queue, a customer sampling an intended service time. In this implementation events are 'scheduled' to occur at some point in the future, and time jumps from event to event in a discrete manner. The scheduling here is an artifact of the implementation only: a random event needs to be scheduled, by sampling a random time in the future for it to occur. Events themselves can cause any number of other events to be scheduled, either immediately or at some point in the future. If they are scheduled for the future, then they are called **B**-events, and if they are scheduled immediately, then they are called conditional or **C**-events, Events can also cause future events to be re-scheduled for a later or earlier time. The event-scheduling approach proceeds by implementing any **B**-events, called the **B**-phase; then implementing any **C**-events, called the **C**-phase; and then advancing the clock to the next **B**-event, called the **A**-phase. FIGURE 4 gives illustrates this event scheduling process.

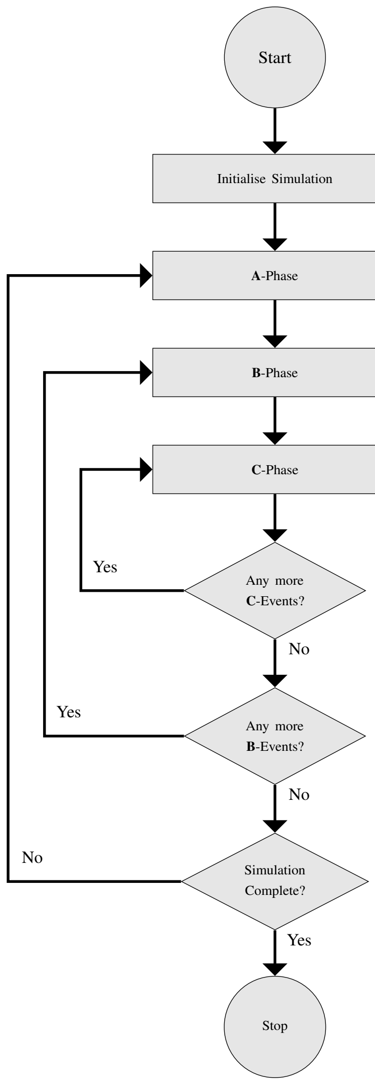


FIGURE 4: Flow diagram of the event scheduling approach used by Ciw, taken from [46].

Processor sharing is implemented by manipulating the rescheduling of future events in the following way. Upon arrival, a customer is given an arrival date $t_\star$, and an intended service time s. They also observe the number of customers, including themselves, who are present at the processor-sharing server, $x_\star$. At this point they have already received $d = 0$ of their intended service time. Given that nothing else changes, this customer will finish service at date t_{end} calculated from (18).

$$t_{\text{end}} = t_\star + \frac{1}{x_\star}(s - d) \tag{18}$$

Therefore this is the date that will be scheduled for that customer to finish service. Now, say and event happens at some t such that $t_\star < t < t_{\text{end}}$, and that event is either an arrival to the server, or another customer finishing service with the server. If the event is an arrival, set $x = x_\star + 1$; and if the event is a customer finishing service then set $x = x_\star - 1$. At this point our original customer will have received $d = d + \frac{1}{x_\star}(t - t_\star)$ of their intended service. Now set $x_\star = x$, $t_\star = t$, and re-calculate their service end date using (18), and then re-schedule their finish service event.

This re-scheduling process is to be performed for every customer in service at any **B**- or **C**- event that causes $x_\star$ to change. This was implemented and released in Ciw v2.2.0, along with some processor sharing variations: limited processor sharing queues [44], a generalisation of a processor sharing queue, in which only a given number of customers may share the service load at any one time; and capacitated processor sharing queues [45] with a switching parameter, where the service discipline switches to from FIFO to processor sharing if the number of customers exceeds this parameter.

The join-shortest-queue processor sharing system considered in this paper is implemented by combining this processor sharing capability with routing objects. An example is given in the documentation: https://ciw.readthedocs.io/en/latest/Guides/Routing/join_shortest_queue.html. Sojourn time CDFs can then be calculated easily as all customer records are saved.

B. Considerations

For the Ciw simulation there are three hyper-parameters to consider: the maximum simulation time, the warm up time, and the number of trials. The larger the number of trials, the more we can smoothe out the stochastic nature of the DES by take averages of the key performance indicators of each trial, however the more trials take longer to run. The warm-up time is a proportion of the maximum simulation time where results are not collected. This filtering of results ensures that key performance indicators are not collected before the simulation reaches steady-state, and therefore and not dependent on the starting conditions of the simulation. The larger the warm-up time, the higher the chance that the collected results are in steady state (this is highly dependent on other model parameters), although this means less results to collect and so more uncertainty. A larger maximum simulation time does both, ensures that there are enough results to decrease uncertainty, and increases the chance that steady-state is reached, however this also increases run-times.

C. Simulation complexity

Events, and more importantly the number of events in a run of the simulation are random. Therefore we cannot have a true complexity analysis, but we can say something about the order of expected number of operations. In this section we consider the average time complexity of the $M/M/R-JSQ-PS$ system.

We will consider number of operations per unit of simulation time when in steady state. Assuming there are M customers in the system at steady-state, there are two types of **B**-events that can take place in a given time unit, arrivals, and customers ending service.

- *Arrivals*: there's an average of Λ arrivals per time unit. At each arrival we need to check R servers to see which is least busy. Then once a server is chosen, we need to go through each customer at that server and re-schedule their end service dates - (18). As join-shortest-queue systems should evenly share customers between servers, we expect there to be $\frac{M}{R}$ customers at that server. So per time unit, the expected number of operations for arrival events is $\mathcal{O}\left(\Lambda\left(R + \frac{M}{R}\right)\right)$.
- *End services*: at steady state, due to work conservation and Burke's theorem [47], there's an average of Λ services ending per time unit. At each end service we need to go through each customer at that server and re-schedule their end service dates. So per time unit, the expected number of operations for end service events is $\mathcal{O}(\Lambda\frac{M}{R})$.

It is difficult to find a closed expression for M, hence the need for simulation and approximations. However a naive estimate for the average number of customers M is the traffic intensity, $M \approx \rho = \frac{\Lambda}{\mu R}$. Let $q_{\max}$ be the maximum simulation time. Altogether, when in steady state, the expected number of operations for a simulation run is $\mathcal{O}\left(q_{\max}\left(\Lambda\left(R + \frac{M}{R}\right) + \Lambda\frac{M}{R}\right)\right) = \mathcal{O}\left(q_{\max}\left(\Lambda R^2 + \frac{2\Lambda^2}{\mu R^2}\right)\right)$.

Although $q_{\max}$ is a user chosen hyper-parameter, and increases the expected number of operations linearly, it is useful to consider if it's choice should be influenced by other system parameters. Consider that, when in steady state, increasing the simulation time increases the number of sojourn time samples we have to estimate the CDF. Say we need X samples to estimate a good CDF, then $q_{\max}$ should be chosen such that $q_{\max} = \frac{X}{\Lambda}$. As X is independent of any other parameter, it can be considered a constant. However, this is assuming a steady state. We should actually choose $q_{\max} = \frac{X}{\Lambda} + q_{\text{warmup}}$, where q_{warmup} is the warmup time, the time it takes to reach steady state. It is likely that q_{warmup} would be effected by the system parameters. In practice, the factor X, and thus the maximum simulation time $q_{\max}$, needs to be quite large in order to gain a good approximation of the sojourn time CDF. As a demonstration, consider FIGURE 5 which shows, for a given system ($\Lambda = 10$, $\rho = 0.85$, $R = 2$) the simulation's running approximation of the 99th percentile of the sojourn time CDF, as the simulation time progresses. Here it takes roughly $X = 75000$ before a good approximation is reached; taking around 15 minutes to run, as opposed to the few seconds it took for approximation method B to run.

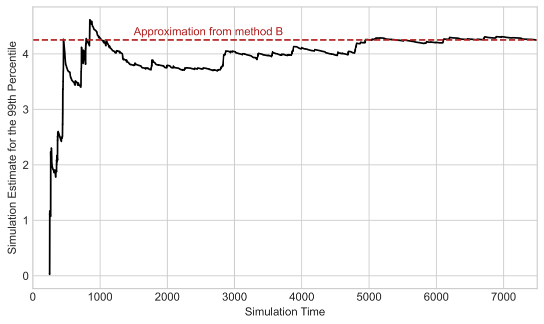


FIGURE 5: Demonstrating the simulation time required for a good approximation of the CDF.

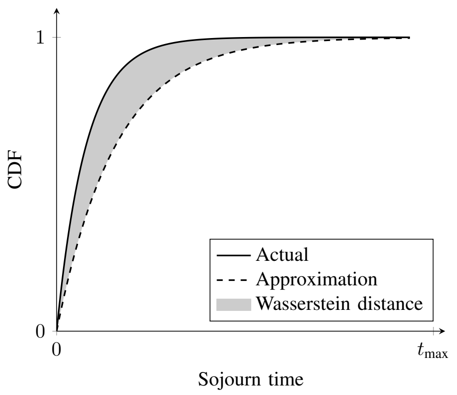


FIGURE 6: Graphical interpretation of the Wasserstein distance between the actual and approximated CDFs.

It is interesting to note that the six approximations' time complexities, and the expected time complexity for the simulation, are affected by different parameters. The approximations are affected by the hyper-parameters L_1 and L_2, along with R, however the simulation is effected by the system parameters themselves. This shows that for some specific cases and parameter sets, it might be worthwhile resorting to simulation after all.

VII. Approximations' accuracy

We perform a computational experiment to compare the six methods against one another under various circumstances. With a fixed choice of $\mu = 1$ we calculate the sojourn time CDFs using each method, for all $1 \leq R \leq 10$, and all $\rho \in (0, 1)$ in steps of 0.01. CDFs are compared against the simulation CDF using the Wasserstein distance [48], or Earth-mover's distance. This is given in (19), with a graphic interpretation given in FIGURE 6. This measure goes from 0, representing equal CDFs, to $t_{\max}$, the maximum sojourn time calculated, representing the largest possible difference between the CDFs. In practice this is calculated numerically by taking Riemann sums with $\Delta = 0.01$ time units.

R	L_1
1	22
2	22
3	22
4	13
5	7
6	5
7	4
8	3
9	3
10	2

TABLE 4: Choice of Markov chain limit L_1 for each R.

$$\Omega(G, H) = \int_{-\infty}^{+\infty} |G(t) - H(t)| dt \tag{19}$$

For these experiments hyper-parameter choices are a fixed: $L_2 = 130$; $t_{\max} = 182.32$; a maximum simulation time of 160000 time units and a warm-up time of 8000. The choice of the Markov chain limit L_1 is dependent on R, it is chosen to be both large enough that the probability of exceeding this is small, and small enough so that the number of defined transitions is manageable, we choose $(L_1+1)^{2R} < 10 \times 10^{10}$. For each R our choice of L_1 is given in TABLE 4.

FIGURES. 7a-7f show the obtained Wasserstein distance, for each method A to F respectively, for each value of R and ρ.

First it is important to note the scale of the y-axis on these plots, they range from 0 to 2; while the Wasserstein distance has the potential to range from 0 to 182.32. Therefore, wherever the Wasserstein distance falls within the plot's range, we can note that these are not bad approximations overall. We can see that all methods are highly dependant on the traffic intensity ρ, however the relationship between accuracy and ρ is different for the methods that use the first approximation of w_n (methods A, B and C), and those that use the second approximation of w_n (methods D, E and F). For the first approximation, low and high values of the load ρ result in higher approximation error. This may be due to unstable approximation algorithms used to compute matrix exponential [49]. While the second approximation performs much better for low values of ρ, middling values perform much worse here. In addition, we see that the second approximation is more dependant of R, with lower values of R performing better. Similarly, this dependence on R is more pronounced in methods C and F, suggesting that the second approximation of λ_n performs worse with higher R than the first Markov chain approximation.

FIGURE 8 shows which method was most accurate for each R, ρ pair. From this we see that method D performed best for low values of ρ, while methods A and B perform best for middling to high values of ρ. Method E is the best performing methods for very high values of ρ, however from

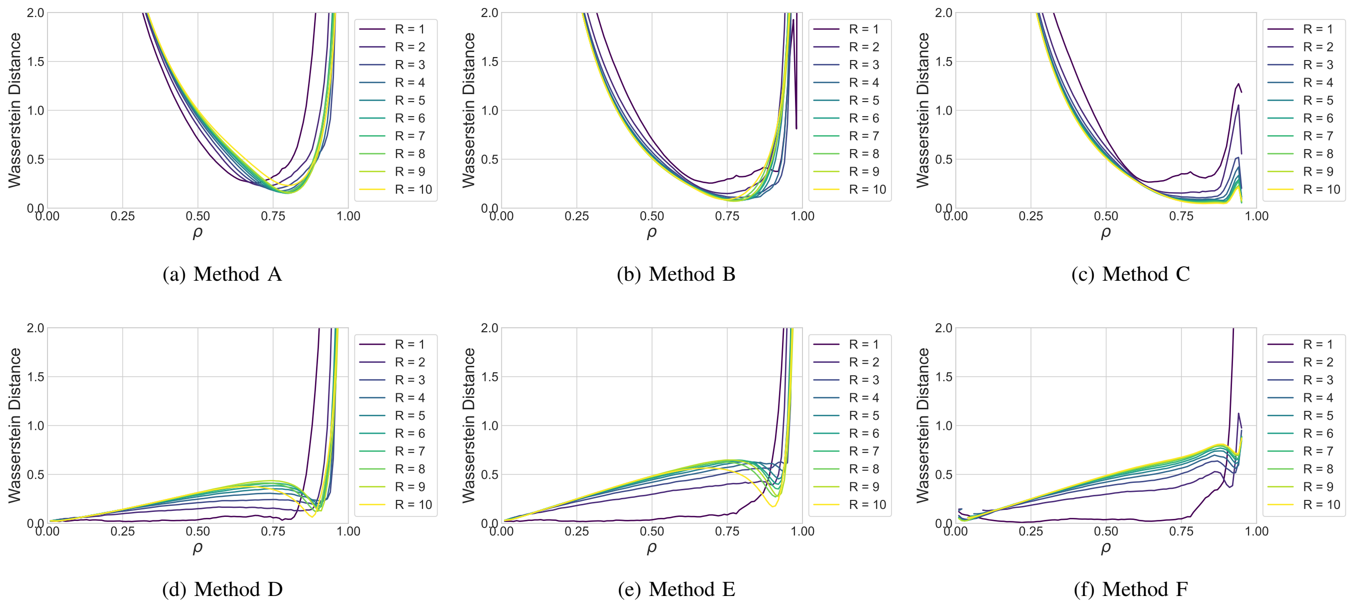


FIGURE 7: Accuracy of each method with increasing traffic intensity ρ and number of CPUs R.

the plot in FIGURE 7e we know that these are still not good approximations of the CDF. Interestingly, when $R = 1$, that is when there is no join-shortest-queue behaviour happening, method F performs consistently.

VIII. Behaviour in high reliabilities

In the prior section we have seen that methods A-F yield an accurate approximation of the sojourn time CDF, namely that the Wasserstein distance remains reasonably small. Depending on the load conditions ρ we can use the approximation with highest accuracy (see FIGURE 8) to achieve accurate sojourn time CDF approximations.

However, URLLC services ask for end-to-end latencies with high reliabilities such as 99%, 99.9%, 99.99%, or 99.999%. This means that the network latency plus processing latency of a service (that is the sojourn time) should be met, e.g., 99.999% of the times. If the end-to-end latency requirement is of 20ms and the maximum network latency remains below 1ms[1], this means that the sojourn time should remain below 19ms in the 99.999% of the times. Therefore, the applicability of our methods A-F depend on their accuracy at the 99.999th percentile.

In FIGURE 9 we illustrate the error, measured in scalable time units, achieved by the best approximation at the 99.999th percentile. In other words, if $T_{a,99.999}$ is the best method 99.999th percentile for the sojourn time, and $T_{99.999}$ is the simulated 99.999-percentile; then FIGURE 9 illustrates $T_{a,99.999} - T_{99.999}$. To derive the simulated 99.999th percentile we use the simulation from Section VI.

As with the Wasserstein distance (see FIGURES 7a-7f), FIGURE 9 evidences that the 99.999th error becomes more prominent as the load ρ approaches to 1 in the M/M/R-JSQ-PS system. In particular, the best method under-estimates the 99.999th percentile of the sojourn time for the error falls towards negative values near -180 as ρ approaches to 1. Note that the maximum sojourn time in the experiments can pop up to $t_{\max} = 182.32$ time units, hence, the error is notoriously large towards the highest load $\rho \approx 1$.

Nevertheless, for not so high loads $\rho \leq 0.85$ the 99.999th percentile error remains low. Namely, the error is of less than $t = 17.78$ time units with respect to the simulations when $R \geq 3$ CPUs – see TABLE 5. For high loads $\rho > 0.85$ our accuracy measure is bad. This is likely due to the fact that upon large loads the PS fashion increases the customers' experienced service time to infinity, which is something impossible to simulate. Therefore, the baseline measure would no longer be accurate. As such, for loads above $\rho > 0.85$ we cannot comment on the accuracy of the approximations.

If the system has $R < 3$ CPUs, then the best method has erratic oscillations, indeed the 99.999th percentile is underestimated by $t = -67.15$ time units with $R = 1$ – see TABLE 5.

We have also analysed what is best method error for the 99th, 99.9th, and 99.99th percentiles. The results are shown in Appendix A and they show the same pattern as the observed for the 99.999th percentile in FIGURE 9. That is, the best A-F method results in under-estimations of the sojourn time that get worse as ρ approaches to 1. Moreover, the results from FIGURE 14 in Appendix A shows that the error oscillations start to become more prominent with higher reliabilities and mid values of CPUs.

Overall, the best method gives accurate estimations for the 99.999th percentile of the sojourn time as long as $\rho \leq 0.85$;

[1]NR PUSCH transmissions take less than 1ms using mini-slots [50].

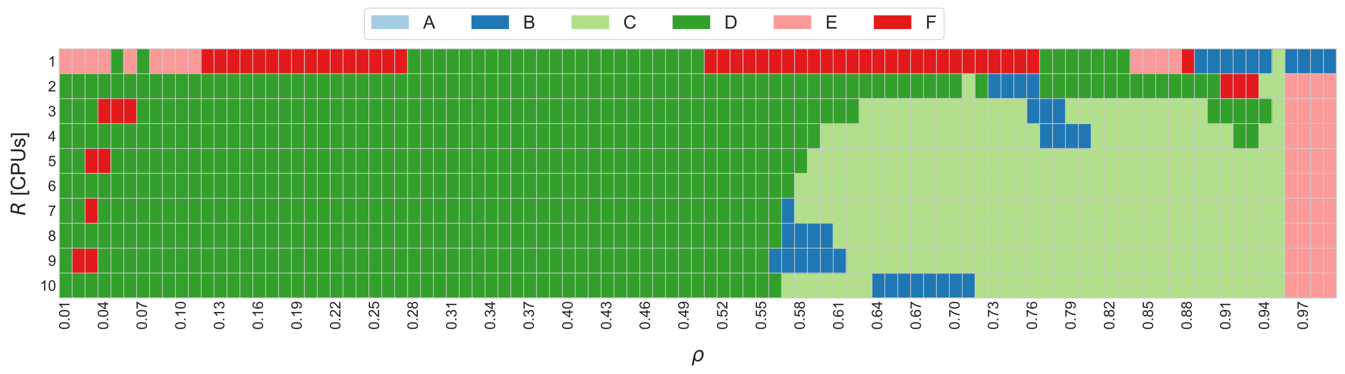


FIGURE 8: Most accurate method TABLE 2 for each R, ρ pair.

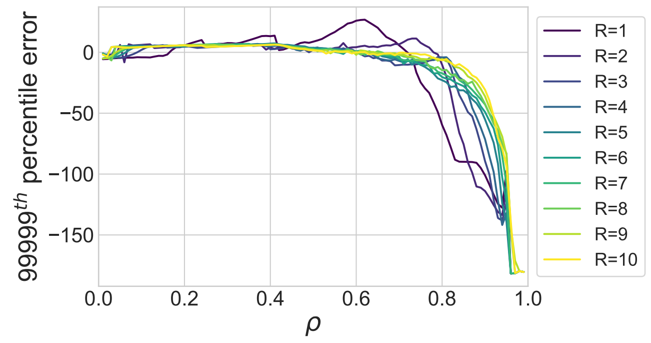


FIGURE 9: Sojourn time 99.999th percentile error using the best method. Positive/negative means over/under-estimation, respectively.

TABLE 5: Sojourn time 99.999th percentile errors using the best method with increasing number of CPUs R and load ρ. Positive/negative mean over/under-estimation, respectively.

	$R=1$	$R=3$	$R=5$	$R=7$	$R=10$
$\rho = 0.10$	-4.11	0.88	2.03	1.92	1.92
$\rho = 0.25$	-1.30	2.02	3.82	3.92	3.47
$\rho = 0.50$	-4.70	-1.05	1.64	1.77	1.97
$\rho = 0.75$	-17.71	-6.74	-8.15	-5.25	-2.78
$\rho = 0.85$	-67.15	-14.46	-17.78	-15.67	-10.08
$\rho = 0.90$	-104.75	-51.17	-38.83	-32.18	-25.52
$\rho = 0.95$	-100.83	-126.16	-102.52	-79.20	-71.42
$\rho = 0.99$	-180.50	-180.51	-180.52	-180.53	-180.54

tends under-estimate the 99.999th percentile; and is more stable for $R \geq 3$ CPUs.

A. Comparison with non-exponential service times

So far we have seen that the methods A-F perform sufficiently well to estimate high reliabilities of an M/M/R-JSQ-PS system, e.g., the 99.999th percentile of the sojourn time. However, exponentially distributed service rates are often an unrealistic assumption, with deterministic, or uniform, lognormal, and gamma distributed service times being more realistic for services with a bounded number of operations. Here we explore the use of Markovian models to model other service distributions.

TABLE 6: Summary of the service time distributions compared in FIGURE 10.

Distribution	Mean	Variance
Expon(μ)	$\frac{1}{\mu}$	$\frac{1}{\mu^2}$
Det $= \frac{1}{\mu}$	$\frac{1}{\mu}$	0
$U\left(\frac{1}{2\mu}, \frac{3}{2\mu}\right)$	$\frac{1}{\mu}$	$\frac{1}{12\mu^2}$
Lognormal $\left(\frac{1}{\mu} - \frac{\ln(2)^2}{2}, \ln(2)\right)$	$\frac{1}{\mu}$	$\frac{1}{\mu^2}$
Gamma $\left(\frac{1}{2}, \frac{2}{\mu}\right)$	$\frac{1}{\mu}$	$\frac{2}{\mu}$

To investigate, we calculate and compare sojourn time CDFs using exponentially distributed services and deterministic, uniform, lognormal, and gamma services for various values of R and ρ. A summary of the service time distributions used is given in Table 6. In such a manner, all distributions share the same average service time, although their variances vary.

We see that the CDFs obtained when modelling service times as exponentially distributed always lie below those obtained using the service time distributions with equal or smaller variances. This is demonstrated in FIGURE 10, which shows that the tail percentiles are always larger, or more pessimistic, when modelling exponential services. However for the Gamma distributed service times, with twice the variance of the exponential service times, using the exponential approximations no longer gives a bound on the percentiles.

FIGURE 10 also evidences how near $\rho = 0.6$ when we use $R = 5$ or $R = 10$ CPUs the best method (black) gets closer to the percentiles of the simulated results (yellow). This behaviour is because after $\rho = 0.6$ the best method changes from method D to method C (see FIGURE 8). Similarly, at high loads $\rho \geq 0.97$ the best method changes from C to

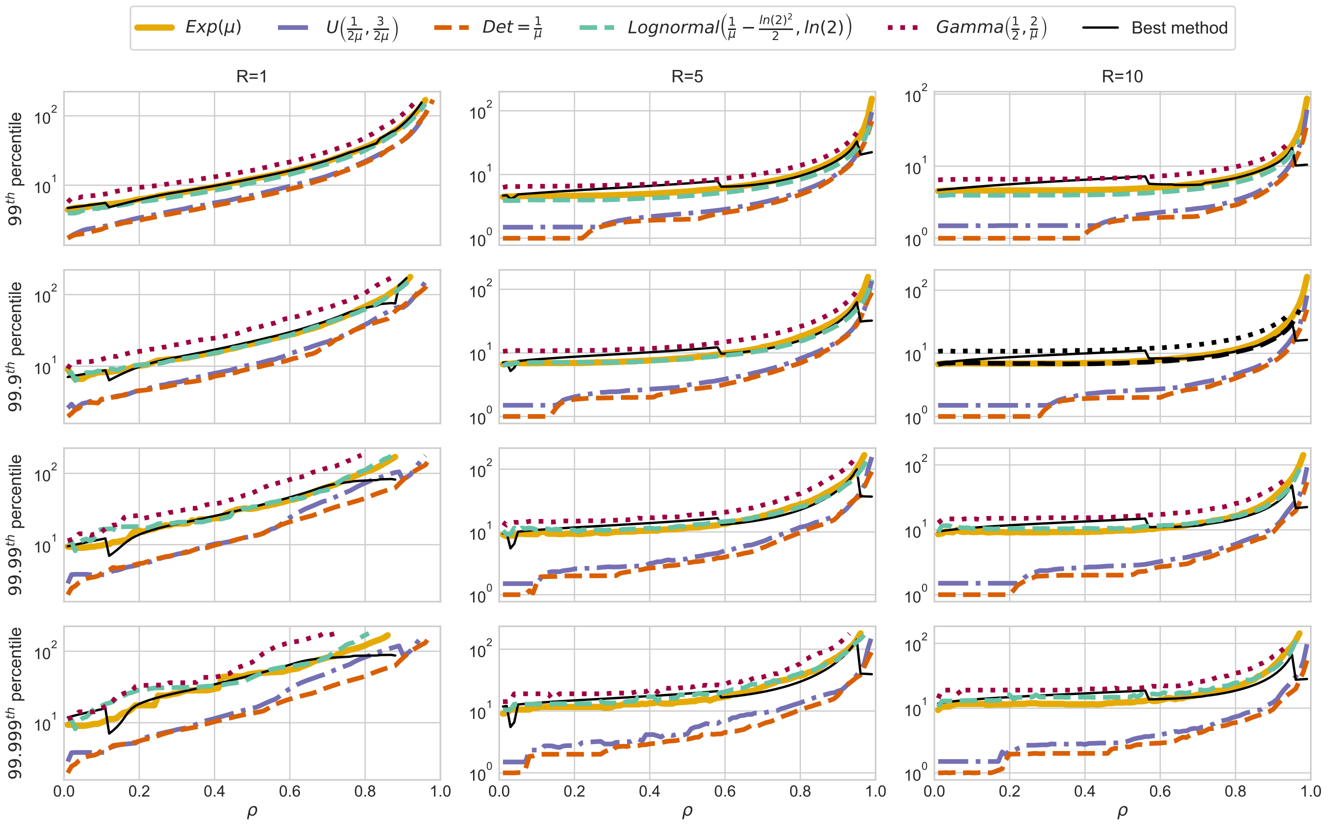


FIGURE 10: The 99th, 99.9th, 99.99th and 99.999th percentiles of the sojourn time distributions, on a log scale, when service times are modelled as exponentially distributed, uniformly distributed, deterministic, lognormal and gamma distributed.

E, thus, the sudden change in the sojourn time percentiles. Namely, the sojourn time percentiles at such high loads differs significantly from the values obtained in the simulation (yellow). The erratic values of the best method for $\rho \geq 0.97$ even goes below the sojourn time percentiles obtained for uniform and deterministic service times (blue and red lines in FIGURE 10, respectively).

Altogether, FIGURE 10 shows that our best method (black): (i) stays close to the sojourn time percentiles obtained in simulations (yellow) for loads $\rho < 0.97$; (ii) lies above the sojourn times provided by deterministic (red) and uniformly distributed (blue) service times; (iii) lies below the sojourn times provided by the gamma (wine) service times; and (iv) close to lognormal (green) service times. Lastly, FIGURE 10 shows (v) our best method largely underestimates the sojourn time percentile for $\rho \geq 0.97$, resulting in even smaller percentiles than uniformly distributed and deterministic service times.

IX. Scaling an URLLC service

In this section we explain how to use the proposed methods to scale the number of CPUs at a server processing an URLLC service with latency target τ and reliability requirement η. We assume that the URLLC traffic is sent through a network that introduces an end-to-end delay denoted by the random variable F, and retain the previous notation of the sojourn time (server processing time) as the random variable T. Overall, we have to ensure that:

$$\mathbb{P}(F+T \leq \tau) \geq \eta. \tag{20}$$

It might not be feasible to understand the CDF of the network end-to-end delay F, but we might know the η_F-percentile of F, which we denote as f^{η_F}. Then:

$$\begin{aligned}
\mathbb{P}(F+T \leq \tau) &= \int_0^\tau \mathbb{P}\left((F \leq x) \cap (T \leq \tau - x)\right) dx && (21)\\
&= \int_0^\tau \mathbb{P}(F \leq x)\mathbb{P}(T \leq \tau - x) dx && (22)\\
&\geq \mathbb{P}(F \leq f^{\eta_F})\mathbb{P}(T \leq \tau - f^{\eta_F}) && (23)\\
&= \eta_F \mathbb{P}(T \leq \tau - f^{\eta_F}) && (24)\\
&\geq \eta && (25)
\end{aligned}$$

Here step 21 is derived from the convolution of the two random variables F and T; and step 22 is from assuming the independence of F and T. Step 23 takes one slice of the previous integral, and so would be smaller by definition. To see this, consider the case where $F > f^{\eta_F}$, now there is still an opportunity for $F + T \leq \tau$ depending on the value of T. Finally step 24 is from the definition of f^{η_F}.

Together this gives the following inequality:

$$\mathbb{P}(T \leq \tau - f^{\eta_F}) \geq \frac{\eta}{\eta_F} = \eta_T \tag{26}$$

Thus we wish to choose the number of CPUs R such that T satisfies (26), which implies that (20) will be satisfied. If the above holds, we ensure that the URLLC service will meet a latency τ with reliability η. Otherwise, we have to increase the number of CPUs R at the server that process the URLLC traffic.

Lets consider an URLLC service that has latency and reliability requirements of $\tau = 10$ ms and $\eta = 0.99$, respectively. If we know that the $\eta_F = 0.9999$-percentile of the network end-to-end delay is $f^{0.9999} = 5$ ms, the sojourn time T of the server must satisfy $\mathbb{P}(T \leq 10 - 5) \geq \frac{0.99}{0.9999} \approx 0.9901$.

At a given time of the day the URLLC users send $\Lambda = 100$ packets/sec, and the server has $R = 3$ CPUs with processing rates of $\mu = 50$ packets/sec. Therefore, the server foresees a load $\rho \approx 0.33$, and we should check which method best approximates the sojourn time CDF at such load. In FIGURE 8 we see that method D achieves the highest accuracy at the tuple $(\rho, R) = (0.33, 3)$, hence, we use method D approximation for the 0.9901-percentile of the sojourn time – see Appendix B for further details.

According to method D, with a load $\rho \approx 0.33$ and $R = 3$ CPUs the sojourn time is 6.43 ms, i.e., we have $\mathbb{P}(T \leq 6.43) \geq 0.9901$. However we require that the sojourn time T satisfies $T \leq \tau - f^{0.9999} = 5$ with reliability $\frac{0.99}{0.9999} = 0.9901$. In other words with the currant load and 3 CPUs the sojourn time exceeds the required latency percentile by 1.43 ms.

Following the above example, we should increase the number of CPUs until (26) is satisfied. Equivalently, as the URLLC demand traffic Λ decreases, we should decrease the number of CPUs to the minimum number that satisfies (26). Algorithm 1 details the above procedure.

Given the traffic rate Λ, the current number of CPUs R with their service rates μ, the URLLC reliability η, the overall latency requirement τ, and the network end-to-end delay for the η_F-percentile f^{η_F}; Algorithm 1 gives the required number of CPUs for an M/M/R-JSQ-PS system. Namely, in line 8 the η_T-percentile of the sojourn time is computed, according to the best method – according to (26) and checks if this is sufficient or a CPU increase is needed. Moreover, in line 9 we increase CPUs until the load remains smaller than an 85%. Hereof, we prevent the accuracy degradation of our approximations at high loads – as discussed in Section. VIII.

Algorithm 1: URLLC server scaling

```
   Data: $\Lambda, R, \mu, \eta, \tau, \eta_F, f^{\eta_F}$
   Result: $R, t^{\eta_T}$
1  Function sojourn_percentile($\eta', R'$):
2      $\rho = \frac{\Lambda}{R'\mu}$
3      method_X = best_method($\rho, R'$)
4      $t^{\eta'}$ = method_X.percentile($\eta'$)
5  return $t^{\eta'}$
6  $R = 1$
7  $\eta_T = \frac{\eta}{\eta_F}$
8  $t^{\eta_T}$ = sojourn_percentile($\eta_T, R$)
9  while $f^{\eta_F} + t^{\eta_T} > \tau$ and $\rho > 0.85$ do
10     $R = R + 1$
11     $t^{\eta_T}$ = sojourn_percentile($\eta_T, R$)
12 end
```

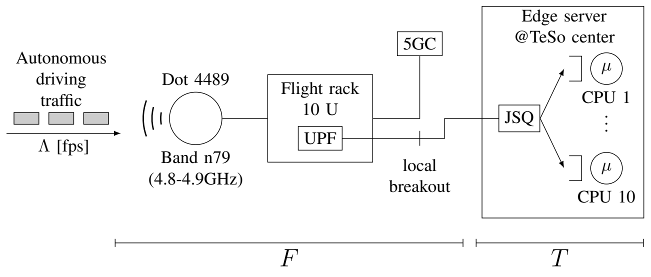


FIGURE 11: 5G-SA setup [51] with local breakout to the Edge server.

A. Example: scaling Teleoperated Support (TeSo) service

In this section we provide an example of how to use Algorithm 1 to scale an autonomous driving service, namely, the Teleoperated Support (TeSo) service [13]. Such service consists of a vehicle sending video or sensor streams to a remote controller that takes control over the vehicle in risky situations. For example, if a car accident happens the vehicle may require that a remote driver with expertise takes control until the vehicle leaves the road. The video stream is decoded and played at the remote driver location (the TeSo center) and the driver sends instructions as e.g. pressing the brakes. According to [13], TeSo asks for an end-to-end service latency and reliability of $\tau \leq 20$ ms and $\eta = 0.99999$, respectively. We consider an M/M/R-JSQ-PS server that decodes the video stream sent in the NR PUSCH, and here the PUSCH transmission time and video decoding must take less than 20 ms the 99.999% of the time. Otherwise, the remote driver may not receive some frames or receive them too late to react to e.g. a car crash.

We consider the 5G-SA deployment in FIGURE 11 to provide connectivity to connected vehicles. The setup was tested in the 5G-DIVE project [51] and consists of an Ericsson Dot 4489 Radio Unit that was connected to a Flight rack connected to a 5G core (5GC) in a remote location. To prevent the network traffic going to the remote 5GC, the UPF was deployed within the Flight rack with a local breakout that sends latency-sensitive traffic to an Edge server. With a preemptive priority queue for URLLC traffic at the UPF, TeSo traffic does not suffer from queueing delay [52, 1.3.2]. Consequently, the end-to-end delay F is governed by the uplink NR PUSCH transmissions.

TABLE 7: NR setup used in TeSo scaling

Parameter	Value
Numerology	$\mu = 2$
SNR	5,10,15 dB
Carrier Freq.	4.8 GHz
Channel size	138 RB
PUSCH/PDSCH mapping	Type A
MCS	256 QAM
HARQ RV seq	[0,2,3,1]
LDPC decoding algo	Norm. min-sum
Tx Layers	2
Tx Antennas	8
Rx Antennas	2
Propagation Channel	CDL-C [53]
Delay Spread	300 ns
Max Doppler Shift	5 Hz
MIMO setup	4x2 array

TABLE 8: Scaling performance at peak demand for (5,10,15) dB SNR.

Area	Peak CPUs	Peak rejection	Peak 99.999-delay (ms)
Residential	(2,2,2)	(0%, 0%, 0%)	(19.82, 19.57, 19.57)
Industrial	(7,7,7)	(13%, 0%, 0%)	(19.92, 19.93, 19.93)
Highway	(7,10,10)	(53%, 5%, 5%)	(19.92, 19.95, 19.95)

To obtain the transmission time in the NR PUSCH we use Matlab simulations with the parameters specified in TABLE 7. For fast PUSCH transmissions we consider numerology $\mu = 2$ to have mini-slots of 0.25 ms. The simulations consider SNR levels of 5, 10 and 15 dB to account for different channel quality conditions. For the throughput increases with the SNR, so it does the accepted demand of TeSo. Moreover, the SNR impacts the probability of successful decoding and the number of HARQ repetitions until the frames are successfully transmitted. Namely, having one repetition leads to a PUSCH transmission time of two mini-slots, i.e. of 0.5 ms. In our simulations we compute the worst transmission delay in the NR PUSCH for each SNR, i.e. we compute $f^{1.0}$ for 5, 10 and 15 dB. Recall we consider an UPF with an URLLC priority queue, hence the worst end-to-end delay $f^{1.0}$ corresponds to the worst transmission delay (accounting repetitions) in the NR PUSCH.

To generate the traffic demand Λ [fps] we use a real-world dataset of the traffic flow of three roads in Torino city, representing three different load intensities - a highway, an industrial area, and a residential area. For each, we assume a future scenario with 10% of all traffic being connected autonomous vehicles, each of them sending an H.265/HEVC video stream to the 5G network – inline with the standard [54].

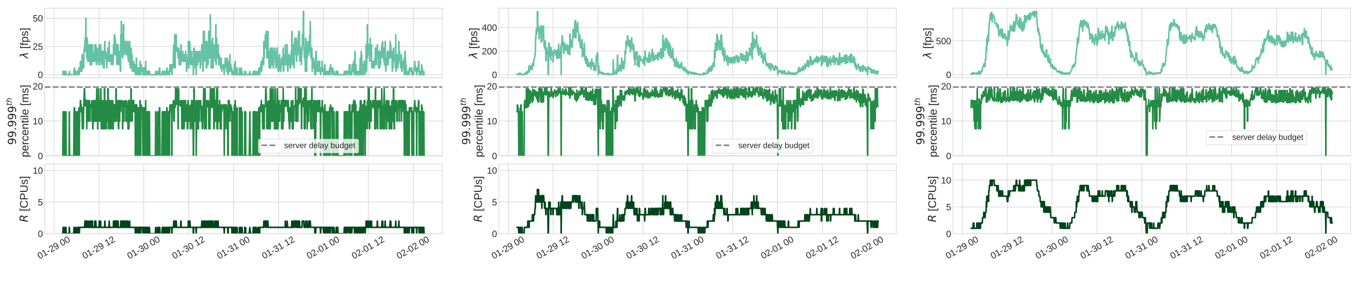
In TABLE 8 we show how the SNR impacts the number of CPUs, rejected demand and highest 99.999-delay percentile (i.e. $F + T$) achieved during peak demand in residential, industrial and highway roads. Upon small SNR of 5 dB, the NR rejects the 53% of the demand of the highway at peak hours for the 138 RBs are not enough to accommodate the peak demand during rush hours upon mild channel conditions. Even with good channel conditions (15 dB) the highway rejects a 5% of the incoming demand for the RBs are not enough, which pins the paramount importance of RAN provisioning tackled in the literature [14]–[16]. Results from TABLE 8 show that 10 CPUs are enough to attend the admitted demand of peak hours, and they satisfy the 20 ms end-to-end delay required in TeSo. Indeed, results show that Algorithm 1 meets the end-to-end delay requirement the 99.999% of the time during peak hours. Moreover, TABLE 8 highlights the 99.999-percentile of the end-to-end delay is the same for 10 and 15 dB. This is because, PUSCH transmissions do not require repetitions upon such SNR levels. For 5 dB PUSCH transmissions require at most of an additional repetition. However, for the peak demand occurs at a different point in time the 99.999-percentile is not necessarily adding 0.25 ms to the peak 99.999 delay percentile with 10 and 15 dB – see TABLE 8.

FIGURE 12a, FIGURE 12b and FIGURE 12c illustrate the scaling performance of Algorithm 1 when the NR offers 15 dB of SNR at residential, industrial and highway roads in Torino; respectively. The top row illustrates the traffic demand attended by NR with 138 RBs during four days in residential, industrial and highway roads. We remark the attended demand is high enough to satisfy the Palm-Khintchine theorem [27], hence, the Poissonian arrival assumption for our M/M/R-JSQ-PS system (Edge server) can be considered valid.

To process the TeSo traffic we consider the remote operation center where the expert provides support has an Edge server with $R = 10$ available CPUs performing video decoding of H.265/HEVC video streams [54]. We use the results in [55] to derive the service rate μ [fps] of each CPU. For further details, we refer the reader to Appendix C.

Using Algorithm 1 we perform a reactive scaling that updates the number of CPUs at the Edge server each 5 minutes, which is the frequency at which we receive updates of road traffic in our data set. Thus, each 5 minutes we invoke Algorithm 1 (with the fixed parameters $\eta = 0.99999$, $\tau = 20$ ms, $\eta_F = 1.0$, and $f^{1.0}$ obtained through Matlab NR PUSCH simulations) with the new value of Λ [fps].

In FIGURE 12c we see the effect of applying Algorithm 1 over four days in an highway road. The results show that the number of CPUs increase up to $R = 10$ in the two peak-hours foreseen each day, which correspond to the the time at which the citizens travel to and from work. Similarly, FIGURE 12c (bottom) shows that the number of CPUs drop down to $R = 1$ in the night hours due to the absence of traffic (barely 1 vehicle passing). Indeed, FIGURE 12c shows that the 99.999-percentile of the sojourn time remains below $\tau - f^{1.0} = 19.70$ ms, because we invoked Algorithm 1 to



(a) Residential road (b) Industrial road (c) Highway road

FIGURE 12: Algorithm 1 performing scaling (bottom rows) on different roads as the autonomous driving traffic increases (top rows). The goal is to meet the 99.999-percentile delay of 20ms (middle rows) considering the 5G setup in FIGURE 11 and a SNR of 15 dB.

scale considering the maximum network delay due to NR PUCSH transmissions is $f^{1.0} = 0.25$ ms (the duration of one mini-slot with NR numerology $\mu = 2$). Thus, FIGURE 12c shows that by using the proposed approximations and scaling algorithm we meet the requirements of TeSo [13].

Similar analyses were carried out on an industrial road in FIGURE 12b, and a residential road, in FIGURE 12a. Considering the highway analyses of FIGURE 12c as a heavy traffic load, these represent medium and low traffic loads respectively. It is interesting to note here that the frequency of needing to switch CPUs on and off is related to the size of the load.

B. Impact of approximation errors

In Section A we compared the percentiles obtained from the best approximations, to those obtained through simulation of Exponential, Uniform, Deterministic, Lognormal and Gamma distributed intended service times. Here we investigate the difference in scaling of the autonomous driving service in Torino if simulated intended service times followed these distributions. Namely, rather than using the best method in the `sojourn_percentile()` function of Algorithm 1, at lines 8 and 11; we use the sojourn time percentile obtained via simulation of exponential, uniform, deterministic, lognormal, and gamma service times, that is we use the simulations depicted in FIGURE 10.

We denote with $t_m^{0.99999}$ the sojourn time percentile reported by Algorithm 1 when it uses the best method A-F. Similarly, we denote $t_{\text{Exp}}^{0.99999}$, $t_{\text{U}}^{0.99999}$, $t_{\text{Det}}^{0.99999}$, $t_{\text{Log}}^{0.99999}$ and $t_{\text{Gam}}^{0.99999}$ the percentile reported by Algorithm 1 when it uses the simulation data for Exponential, Uniform, Deterministic, Lognormal and Gamma service times respectively. We then consider $\Delta t^{0.99999}$, the difference in the 99.999-percentile of the sojourn time when we use the best method and simulations, respectively. A positive value of $\Delta t^{0.99999}$ represents an overestimation by the approximation methods, while a negative value represents an underestimation.

In the experiments we considered the three different roads studied in Section A, with increasing vehicular traffic: residential, industrial, and highway roads with up to 240, 2184, and 4392 vehicles/hour respectively. As the highway

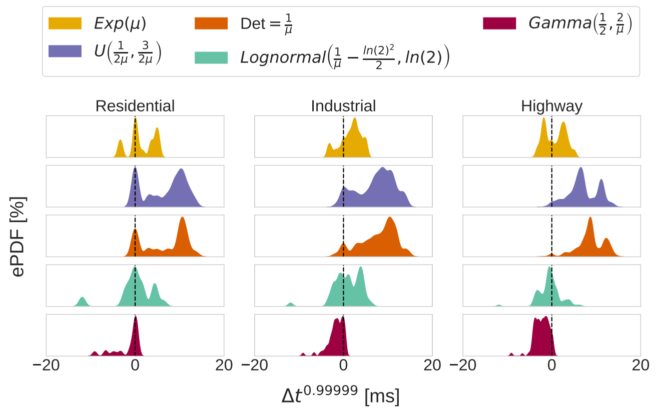


FIGURE 13: Difference of the 99.999-percentile sojourn time $\Delta t^{0.99999}$ between our best method and other service time distributions at different roads. $\Delta t^{0.99999} = 0$ means no difference, $\Delta t^{0.99999} > 0$ means that our best method estimated a higher delay than the distribution. We consider a 15 dB SNR in the 5G connection.

road has more vehicular traffic Λ [fps], there are more chances of having a high load $\rho \approx 1$ at the Edge server, thus, of having more approximation errors. Conversely, the experiment in the residential area is the least likely to achieve high loads because of the smaller amount of vehicular traffic.

FIGURE 13 illustrates the empirical Probability Density Function (ePDF) $\Delta t^{0.99999}$ over four days when we use Algorithm 1 scaling, for the three road types and five intended service time distributions.

Comparing the scaling of the best method against the exponential service time (yellow), we see that the ePDF is centred around $\Delta t^{0.99999} = 0$. Hence, our best methods accurately approximate the sojourn time percentile in Algorithm 1. However, in the highway road (yellow top right) we notice that the mode of the obtained ePDF is shifted to the left of $\Delta t^{0.99999} = 0$, which means that the best method underestimated the sojourn time percentile for the exponential service times. This is because upon high loads – i.e., as ρ approaches to 1 – the best method suffers from large errors, as observed in FIGURE 10 and TABLE 5.

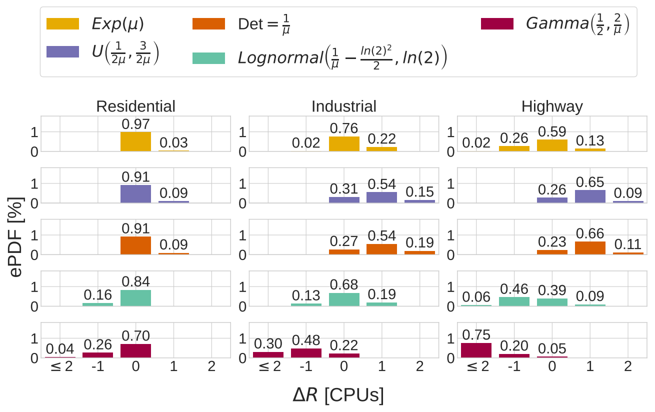


FIGURE 14: CPU difference ΔR between our best method and other service time distributions at different roads. $\Delta R = 0$ means no difference, $\Delta R = -1$ or $\Delta R = 1$ means that our best method uses one CPU less or more, respectively. We consider a 15 dB SNR in the 5G connection.

However, when services follow a uniform distribution (blue) or are deterministic (orange), in agreement with the results in FIGURE 10, we see that using the best method in the scaling of Algorithm 1 results into overestimating the sojourn time percentile. That is the modes of the obtained ePDFs are shifted to the right of $\Delta t^{0.9999} = 0$. And again, the high loads ρ seen in the highway road result in a slight increase in the percentage of time with the scaling of Algorithm 1 underestimating the sojourn time $\Delta t^{0.99999} < 0$.

For both the lognormal and gamma distributed service times (green and wine in FIGURE 13), we observe that our approximation does not suffer that much over-estimation of the 99.999-percentile of sojourn time as with the uniform and deterministic service times. Indeed, for the chosen lognormal distribution has same variance as the exponential, the sojourn time error is centered around $\Delta t^{0.99999} = 0$. Whilst for the gamma distribution has a larger variance, our best method underestimates the 99.999-percentile of the sojourn time – see how the ePDF is skewed to the left ($\Delta t^{0.99999} < 0$) in FIGURE 13 fifth row.

Under- and over-estimating the sojourn time percentile can lead to under- and over-provisioning the number of CPUs R, respectively. In FIGURE 14 we depict how such error in the estimation impacts to the number of CPUs. In particular, we show ΔR, the difference in required CPUs given by Algorithm 1 when using the best approximation to when using simulated distributions. Note that $\Delta R > 0$ denotes over-provisioning due to the best method m, while $\Delta R < 0$ means under-provisioning of CPUs.

From the results in FIGURE 14 we conclude that in residential areas with small load ρ, there is no over-provisioning with distributions of smaller variance than the exponential service time – i.e. deterministic and uniform distributions. However, if the service time follows a distribution with equal or larger variance – as the lognormal and gamma distributions – our Algorithm 1 incurs into under-provisioning. For higher loads ρ, as the experienced in industrial and highway scenarios, scaling with our best method leads to over-provisioning for the uniform and deterministic distribution, while it tends to under-provision for service times following the lognormal or gamma distribution. Note this is aligned with the skewness of the ePDF for the 99.999-percentile error – i.e. in FIGURE 14 the ePDF is skewed to the right for deterministic and uniform distribution and skewed to the left for the lognormal and gamma distributions.

Overall, FIGURE 13 and FIGURE 14 provide three main conclusions. First, they show that using the best method of Section IV in the scaling Algorithm 1 results in perfect scaling upon low loads the 70% of the time – see how $\Delta R = 0$ in FIGURE 14 for the residential road. Second, results show that upon higher loads (ρ approaches to 1) in highway and access roads, our best method achieves perfect scaling more than a 23% of the time, except for gamma distributed service times. Third, scaling with the best method over-provisions when there is smaller variance and under-provisions when there is higher variance than the exponential distribution.

X. Discussion

The approximations proposed in Section IV are useful to perform CPU scaling in servers attending URLLC traffic. Thanks to the CDF approximation, we can tell whether e.g. $R = 2$ CPUs are enough to meet small processing delays the 99.999% of the time. In the following we discuss practical considerations regarding our proposed approximations.

Prevent 85% loads. Results from Section VIII evidence we cannot comment on the accuracy of the approximations upon very high loads $\rho > 0.85$. Therefore, we suggest system administrators to avoid having a server with loads above an 85% to prevent inaccurate provisioning. For example, as the load approaches to the 85%, an additional CPUs should be turned on. Otherwise, scaling with our best approximation leads to over-provisioning non-exponential service times – see FIGURE 14.

Approximations applicability. Our approximations are useful to meet URLLC in existing solutions for task offloading offloading [56] and resource provisioning (CPUs) in both edge [57] & Radio Access Network (RAN) [58], [59]. For example, our approximations could be used in the feasibility checks performed in solutions found by [56, Algorithm 1] and [59, Algorithm 1]. Hence the obtained solutions would guarantee URLLC in task offloading and CPU provisioning. Another application of our approximations is in the context of online optimization, the framework used in [58] to perform live provisioning of CPUs/GPUs in the RAN. By using our approximations within the loss function, solutions will satisfy URLLC requirements. Moreover, methods as stochastic gradient descend [60] are applicable by numerically computing the derivative of our approximations.

Accuracy vs. runtime. Our approximations' accuracy depend on the granularity of the states considered in the Markov chain. The larger the number of CPUs R and customers considered L_1, the larger the runtime – see TABLE 3. Given the probability of having L_1 users for different loads – see FIGURE 3 – we recommend using the R, L_1 combinations devised in TABLE 4 for an adequate accuracy and runtime trade-off.

XI. Conclusions

In this paper we:

i) present a generic open-source discrete event simulation software for G/G/R-JSQ-PS systems;
ii) derive and compare six analytical approximations for the sojourn time CDF of M/M/R-JSQ-PS systems, and analyse their run time complexities;
iii) investigate the applicability of M/M/R-JSQ-PS models to M/G/R-JSQ-PS systems under Uniform, Deterministic, Lognormal and Gamma distributed intended service times; and
iv) apply these approximations and simulations to the scaling of a URLLC service for automated vehicles on three different roads in Torino.

The proposed methods have polynomial time complexities $\mathcal{O}(L_1^{3R})$, and are useful to scale servers processing URLLC traffic under mid to high loads, for they yield errors of less than 4.17 time units in high percentiles as a 99.999%. Moreover, the proposed methods serve as conservative scaling approach as long as the service time tail and variance is smaller than the exponential distribution.

In future work we plan to use the Laplace-Stieltjes transform together with SQA to aim at closed formed expressions of sojourn time CDF. Additionally, given the advent of online convex optimization [60] in networking provisioning [58], we plan to use our CDF approximations as loss function to leverage existing algorithms (as the FTRL) and make CPU provisioning with regret guarantees. As a result, we will provide online QoS guarantees for URLLC live provisioning.

REFERENCES

[1] J. Kempf, IAB, and R. Austein, "The Rise of the Middle and the Future of End-to-End: Reflections on the Evolution of the Internet Architecture," RFC 3724, Mar. 2004. [Online]. Available: https://www.rfc-editor.org/info/rfc3724

[2] 5G Americas, "Vehicular connectivity: C-V2X & 5G," 5G Americas, White Paper, September 2021.

[3] D. Aschenbrenner, M. Fritscher, F. Sittner, M. Krauß, and K. Schilling, "Teleoperation of an Industrial Robot in an Active Production Line," *IFAC-PapersOnLine*, vol. 48, no. 10, pp. 159–164, 2015, 2nd IFAC Conference on Embedded Systems, Computer Intelligence and Telematics CESCIT 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896315009921

[4] A. Albanese, V. Sciancalepore, and X. Costa-Perez, "SARDO: An Automated Search-and-Rescue Drone-based Solution for Victims Localization," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[5] A. Acemoglu, J. Krieglstein, D. G. Caldwell, F. Mora, L. Guastini, M. Trimarchi, A. Vinciguerra, A. L. C. Carobbio, J. Hysenbelli, M. Delsanto, O. Barboni, S. Baggioni, G. Peretti, and L. S. Mattos, "5G Robotic Telesurgery: Remote Transoral Laser Microsurgeries on a Cadaver," *IEEE Transactions on Medical Robotics and Bionics*, vol. 2, no. 4, pp. 511–518, 2020.

[6] 3GPP, "Release 15 Description; Summary of Rel-15 Work Items," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 21.915, 10 2019.

[7] ——, "Physical layer procedures for data," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.214, 03 2023.

[8] IEEE, "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. C1–72, 2010.

[9] ——, "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 29: Cyclic Queuing and Forwarding," *IEEE 802.1Qch-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd(TM)-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, IEEE Std 802.1Qbz-2016, and IEEE Std 802.1Qci-2017)*, pp. 1–30, 2017.

[10] ——, "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption," *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, pp. 1–52, 2016.

[11] ——, "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[12] ——, "IEEE/ISO/IEC International Standard - Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 1Q: Bridges and bridged networks - AMENDMENT 7: Cyclic queuing and forwarding," *ISO/IEC/IEEE 8802-1Q:2016/Amd.7:2019(E)*, pp. 1–34, 2019.

[13] 3GPP, "Study on enhancement of 3GPP Support for 5G V2X Services," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 22.886, 03 2016.

[14] Q. Qi *et al.*, "Knowledge-Driven Service Offloading Decision for Vehicular Edge Computing: A Deep Reinforcement Learning Approach," *IEEE Trans. on Vehicular Technology*, vol. 68, no. 5, pp. 4192–4203, 2019.

[15] A. Okic *et al.*, "π-ROAD: a Learn-as-You-Go Framework for On-Demand Emergency Slices in V2X Scenarios," in *IEEE INFOCOM*, 2021, pp. 1–10.

[16] A. Kumar *et al.*, "Multi-Agent Deep Reinforcement Learning-Empowered Channel Allocation in Vehicular Networks," 2022.

[17] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE INFOCOM*, 2015.

[18] F. B. Jemaa, G. Pujolle, and M. Pariente, "QoS-aware VNF placement optimization in edge-central carrier cloud architecture," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–7.

[19] D. B. Oljira, K.-J. Grinnemo, J. Taheri, and A. Brunstrom, "A model for QoS-aware VNF placement and provisioning," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2017, pp. 1–7.

[20] V. Gupta, M. Harchol Balter, K. Sigman, and W. Whitt, "Analysis of join-the-shortest-queue routing for web server farms," *Performance Evaluation*, vol. 64, no. 9, pp. 1062–1081, 2007, performance 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166531607000624

[21] J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, and A. Fedorova, "The Linux scheduler: a decade of wasted cores," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2901318.2901326

[22] L. Kleinrock, *Queueing Systems, Volume 2: Computer applications*. Wiley Online Library, 1977.

[23] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[24] R. Egorova, B. Zwart, and O. Boxma, "Sojourn time tails in the M/D/1 processor sharing queue," *Probability in the Engineering and Informational Sciences*, vol. 20, no. 3, p. 429–446, 2006.

[25] T. J. Ott, "The Sojourn-Time Distribution in the M/G/1 Queue with Processor Sharing," *Journal of Applied Probability*, vol. 21, no. 2,

pp. 360–378, 1984. [Online]. Available: http://www.jstor.org/stable/3213646
[26] H. Masuyama and T. Takine, "Sojourn time distribution in a MAP/M/1 processor-sharing queue," *Operations Research Letters*, vol. 31, no. 5, pp. 406–412, 2003.
[27] C. Palm, "Variation in intensity in telephone conversations," *Applied Probability Trust*, pp. 1–89, 1943.
[28] S. F. Yashkov and A. Yashkova, "Processor sharing: A survey of the mathematical theory," *Automation and Remote Control*, vol. 68, pp. 1662–1731, 2007.
[29] F. Bonomi, "On job assignment for a parallel system of processor sharing queues," *IEEE Transactions on Computers*, vol. 39, no. 7, pp. 858–869, 1990.
[30] G. Hoekstra, R. Van Der Mei, and S. Bhulai, "Optimal job splitting in parallel processor sharing queues," *Stochastic models*, vol. 28, no. 1, pp. 144–166, 2012.
[31] A. Mukhopadhyay and R. R. Mazumdar, "Randomized routing schemes for large processor sharing systems with multiple service rates," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 1, pp. 555–556, 2014.
[32] J. Fu, B. Moran, J. Guo, E. W. Wong, and M. Zukerman, "Asymptotically optimal job assignment for energy-efficient processor-sharing server farms," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 4008–4023, 2016.
[33] L. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 5, pp. 491–496, 1985.
[34] E. Altman, U. Ayesta, and B. J. Prabhu, "Load balancing in processor sharing systems," *Telecommunication Systems*, vol. 47, pp. 35–48, 2011.
[35] S. Robinson, *Simulation: the practice of model development and use*. Palgrave Macmillan, 2014.
[36] SIMUL8 Corporation., "Simul8," https://www.simul8.com/, 2022.
[37] The AnyLogic Company, "Anylogic," https://www.anylogic.com/, 2022.
[38] I. Ucar, B. Smeets, and A. Azcorra, "simmer: Discrete-event simulation for r," *Journal of Statistical Software*, vol. 90, no. 2, p. 1–30, 2019. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v090i02
[39] Team SimPy, "Simpy," https://simpy.readthedocs.io/, 2022.
[40] G. I. Palmer, V. A. Knight, P. R. Harper, and A. L. Hawa, "Ciw: An open-source discrete event simulation library," *Journal of Simulation*, vol. 13, no. 1, pp. 68–82, 2019.
[41] M. Arioli, B. Codenotti, and C. Fassino, "The padé method for computing the matrix exponential," *Linear Algebra and its Applications*, vol. 240, pp. 111–130, 1996. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0024379594001901
[42] P. Schweitzer, "Stochastic Models, an Algorithmic Approach, by Henk C. Tijms (Chichester: Wiley, 1994), 375 pages, paperback." *Probability in the Engineering and Informational Sciences*, vol. 10, no. 3, pp. 463–464, 1996.
[43] J. Demmel, "LAPACK: a portable linear algebra library for supercomputers," in *IEEE Control Systems Society Workshop on Computer-Aided Control System Design*, 1989, pp. 1–7.
[44] J. Zhang, J. Dai, and B. Zwart, "Law of large number limits of limited processor-sharing queues," *Mathematics of Operations Research*, vol. 34, no. 4, pp. 937–970, 2009.
[45] X. Li, *Radio Access Network Dimensioning for 3G UMTS*. Springer, 2011.
[46] G. Palmer, "Modelling deadlock in queueing systems," Ph.D. dissertation, Cardiff University, 2018.
[47] P. J. Burke, "The output of a queuing system," *Operations research*, vol. 4, no. 6, pp. 699–704, 1956.
[48] H. Mostafaei and S. Kordnourie, "Probability metrics and their applications," *Applied Mathematical Sciences*, vol. 5, no. 4, pp. 181–192, 2011.
[49] C. Moler and C. Van Loan, "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later," *SIAM review*, vol. 45, no. 1, pp. 3–49, 2003.
[50] The Third Generation Partnership Project (3GPP), "Physical layer procedures for data," 3GPP, Tech. Rep. TS 38.214 v17.10.0, July 2024.
[51] 5G-DIVE, "KPI and Performance Evaluation of 5G-DIVE Platform in Vertical Field Trials," 5G-DIVE, Tech. Rep. deliverable D3.3, 01 2022.
[52] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.
[53] 3GPP, "Technical Specification Group Radio Access Network; Study on channel model for frequencies from 0.5 to 100 GHz," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.901, 03 2024.
[54] ETSI, "5G; Vehicle-to-everything (V2X); Media handling and interaction," European Telecommunications Standards Institute (ETSI), Technical Report (TR) 26.985.v16.0.0, November 2020.
[55] M. Alvarez-Mesa, C. C. Chi, B. Juurlink, V. George, and T. Schierl, "Parallel video decoding in the emerging HEVC standard," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 1545–1548. [Online]. Available: https://doi.org/10.1109/ICASSP.2012.6288186
[56] D. Van Huynh, V.-D. Nguyen, S. R. Khosravirad, V. Sharma, O. A. Dobre, H. Shin, and T. Q. Duong, "URLLC Edge Networks With Joint Optimal User Association, Task Offloading and Resource Allocation: A Digital Twin Approach ," *IEEE Transactions on Communications*, vol. 70, no. 11, pp. 7669–7682, 2022.
[57] M. Adeppady, A. Conte, P. Giaccone, H. Karl, and C. F. Chiasserini, "Dynamic Management of Constrained Computing Resources for Serverless Services," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2024.
[58] F. Aslan, G. Iosifidis, J. A. Ayala-Romero, A. Garcia-Saavedra, and X. Costa-Perez, "Fair Resource Allocation in Virtualized O-RAN Platforms," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 1, Feb. 2024. [Online]. Available: https://doi.org/10.1145/3639043
[59] F. Mungari, C. Puligheddu, A. Garcia-Saavedra, and C. F. Chiasserini, "O-RAN Intelligence Orchestration Framework for Quality-Driven Xapp Deployment and Sharing," *IEEE Transactions on Mobile Computing*, pp. 1–17, 2025.
[60] E. Hazan, "Introduction to Online Convex Optimization," 2023. [Online]. Available: https://arxiv.org/abs/1909.05207

Appendix A
Errors for increasing reliabilities

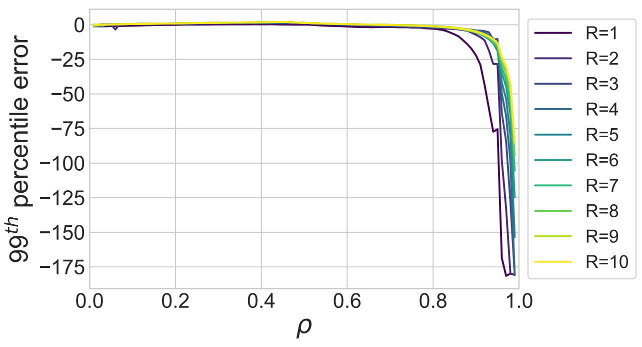
Fig. 14 shows the error for the best approximation A-F from the 99th up to the 99.99th percentile of the sojourn time. The sojourn time error is unitless, and it illustrates the increasing error as ρ approaches 1, so as the increasing oscilacions of the error for higher reliabilities, even with mid values of the number of CPUs like $R = 4$ – as explained in Section VIII.

Appendix B
Getting `sojourn_percentile`(η', R')

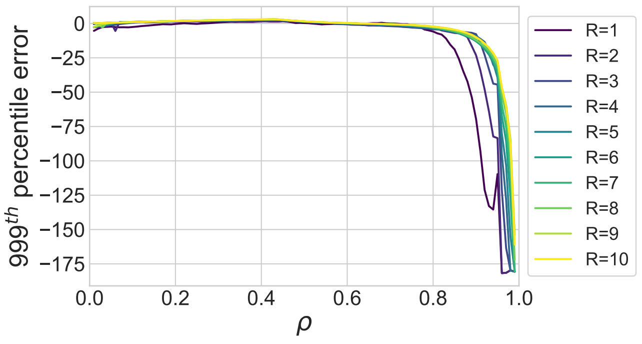
We provide an open-source implementation[2] of the proposed approximation methods A-F. Every method is implemented in Python and yields the sojourn time CDF for a given number of CPUs R. Additionally, it is possible to specify the truncation limits for the maximum number of customers considered at each CPU L_1, and the maximum number of customers at the system L_2.

In order to obtain the result of the `sojourn_percentile`(η', R') function used inside Algorithm 1, we first compute the load ρ given a number of CPUs R', and the arrival and service rates Λ, μ; respectively. Second, we check Fig. 8 to know which is the best method for the given (ρ, R') tuple, e.g., method-A. Third, we create an instance of method-A invoking `jsq.MethodA`$(\Lambda, \mu, R, L_1, L_2, \{t_0, t_1, \ldots\})$, with $\{t_0, t_1, \ldots\}$ being the discrete time points at which we compute the CDF. Then, we obtain the CDF of method-A by

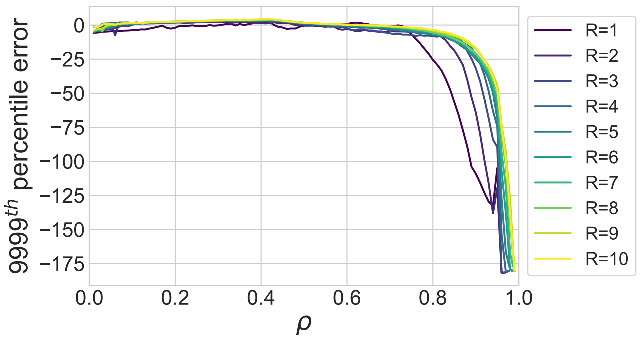
[2] https://github.com/geraintpalmer/mmr-jsq-ps/



(a)



(b)



(c)

FIGURE 14: The best method error for the 99th, 99.9th, and 99.99th percentile the sojourn time. Positive/negative mean over/under-estimation, respectively.

accessing property `sojourn_time_cdf` of the method instance. This property holds a vector $\{P_0, P_1, \ldots\}$ that represents the CDF computed by method-A. In particular, each element represents $P_i = \mathbb{P}(T \leq t_i)$. Finally, we obtain the η' percentile of the sojourn time as

$$t^{\eta'} = \arg\min\{t_i : \mathbb{P}(T \leq t_i) > \eta'\} \tag{27}$$

Appendix C
Considered μ for autonomous driving

The scaling experiments presented in Section A consider an autonomous driving service, namely, a Teleoperated Support (TeSo) service [13]. Vehicles send an H.265/HEVC video stream that is decoded in a remote server (modelled as an M/M/R-JSQ-PS system). The decoded video is reproduced at a monitor used by the remote driver to see the vehicle environment and take actions as e.g. press the brakes.

The vehicles video stream Λ is expressed as frames/sec (fps), and is processed at a rate of μ fps in the server hosting the autonomous driving service. For the experiments in Section A we take into consideration the time that it takes to decode the video stream. According to [55] an Intel Xeon family CPU manages decode an HEVC video frame in 8 ms. Hence a single CPU within the considered M/M/R-JSQ-PS system offers a rate of $\mu = \frac{10^3}{8}$ fps for the considered infrastructure assisted environment perception service. Such value of μ is the one we used in the experiments presented in Section IX.

Geraint I. Palmer graduated from Aberystwyth University in 2013 with a BSc in mathematics, and then moved to Cardiff University to obtain his MSc in operational research and applied statistics in 2014, and his PhD in applied stochastic modelling in 2018, for which he won the OR Society's Doctoral Award. He now works as a lecturer at Cardiff University where his research is in operational research, queueing models and discrete event simulation.

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 assistant professor at Universidad Politécnica de Madrid.