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

Geraint I. Palmer, Jorge Martín-Pérez

Abstract—The future of networking promises to support time-sensitive applications that require ultra low latencies and reliabilities of 99.99%. 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 queuing 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 1.75 time units (e.g., milliseconds) for the 99.99\textsuperscript{th} 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 99.99\% reliable vehicular service and achieve accuracies above a 90%.

Index Terms—queuing theory, simulation, URLLC

1 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, 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%. That is, more than a 99% of the packets should meet latency constraints, which in many cases remain below values of 10 ms.

The aforementioned services lie within the category of Ultra-Reliable Low Latency Communications (URLLC). Regardless the best-effort nature of the Internet, during the last years the standardisation bodies and research community have proposed solutions to achieve URLLC. For wired connections, it is possible to achieve network latency guarantees using IEEE 802.1Q, and for wireless connections, technologies as 3GPP 5G networks and IEEE WiFi 6 still manage to meet URLLC requirements [6].

To achieve deterministic network (DetNet) latencies over wired technologies, IEEE 802.1Q proposes using time-gated queues, traffic shapers [7], synchronised buffers [8], and packet preemption [9]. Such techniques have been proven in the professional audio industry, and now the IETF DetNet [10] aims at extending them to URLLC use cases with wireless technologies as 5G, and WiFi6.

When the network provides 5G connectivity, the 5G core network (5GC) [11] provides URLLC using network slicing [12] and PHY enhancements in the Radio Access Network (RAN). Through network slicing it is possible to achieve URLLC and DetNet – as the IETF has studied [13] – imposing requirements [14] as traffic priorities or dedicated resource allocation. On top, the 5G reliability is boosted up via RAN PHY enhancements as packet replication, and mini-slot transmissions [15]. Additionally, the 5GC URLLC have been enhanced [16] through Quality of Service (QoS) monitoring via the Application Function (AF) exposure service [17], which periodically reports metrics [18] to evaluate the URLLC QoS, e.g. the packet delay budget.

But it’s not only the 3GPP dealing with reliable communications over wireless connections. As noted above, IEEE also aims at URLLC in domestic/non-cellular wireless access with IEEE 802.11ax [19] (commonly known as WiFi 6), which enhances the communication latency and predictability through the usage of Orthogonal Frequency Division Multiple Access (OFDMA), and the use of high frequencies around 6GHz. The underlying idea of the IEEE is to use both the frequency multiplexing and Time Sensitive Network (TSN) solutions to meet URLLC. Specifically, IEEE 802.11ax and the ongoing definition of IEEE 802.11be (WiFi 7) [20], attain URLLC using TSN tools as Time Synchronization 802.1AS [21] and Time-Aware Shaping [22].

It is a fact that standardisation efforts are devoting special attention to URLLC, with both wired and wireless access; and among network operators there is an increasing interest of predicting the behaviour of the infrastructure in
an automated fashion. Therefore, an IETF working group is proposing to use network Digital Twins (DTs) [23] to predict the network behaviour. The idea is to simulate the network behaviour using models that can tell whether the network infrastructure will satisfy the service demand and QoS.

Network DTs, together with 5G, IEEE 802.11, and IEEE 802.1 technologies can provide a detailed, yet compatible, frame to meet URLLC. However, on top of the communication technologies, it is necessary to characterize the latency induced by computation times. That is, once the traffic reaches the server, will it meet the latency reliability or not. Only when both the communication and computational segments meet reliable latency requirements we can tell that the network as a whole delivers URLLC.

In this paper we study how the traffic processing latency is distributed to determine whether a service meets URLLC. Specifically, we develop open-source simulation software, and also propose analytical approximations to characterise the processing latency of servers that dispatch the traffic processing to the least loaded CPU within a pool of R CPUs. As assumed by the state of the art [24]–[26], and alike Linux-based systems, we assume that each CPU utilises a processor sharing policy to attend the traffic processing.

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 from the sojourn time CDF in M/M/R-JSQ-PS systems,

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

2 AN M/M/R-JSQ-PS QUEUEING SYSTEM

This work is concerned with the sojourn time distribution $P(T \leq t)$ 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 $\Lambda$, and intended service times distributed exponentially with rate $\mu$. 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.

Fig. 1 illustrates this system.

3 RELATED WORK

In the networking community queueing theory is a well-established tool to assess the modelling of network infrastructure [27], [28]. 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 [27] 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 queuing 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 [27].

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 [27], and indications on how to derive its CDF [28].

But still, both M/M/1 and M/M/R systems may not be suitable to model networking components. Either because
the assumption of Poissonian arrivals is not suitable or because considering exponential service times is not realistic. To that end, the literature has devoted effort to derive the sojourn time CDF expressions of systems not satisfying such assumptions. For example [29, 30] 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 [31] shift the interest to systems that follow Markovian arrival processes (MAP), rather than Poissonian and provide closed-formulas for the sojourn time CDF in MAP/M/1 systems.

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 [32]. Hence, it is reasonable to model data centres as $M/G/R$ systems, as suggested by [28]. Namely, [28] motivates the study of $M/M/R$ systems as server farms for traffic processing, and the book leaves as exercise how to derive the sojourn time CDF of an $M/M/R$ system following the strategy used for $M/M/1$ systems. Nevertheless, $M/M/R$ systems do not mimic the behaviour of Linux based systems where each CPU shares the computing time using a processor sharing (PS) discipline, rather than the one-at-a-time processing of $M/M/R$, where packets wait in the queue until a server finishes processing a job. As such, [33] propose to model web server farms using $M/M/R$-JSQ-PS systems with jobs joining the CPU with the shortest queue (JSQ), and each CPU serving all it’s jobs simultaneously via a processor sharing discipline (here joining the ‘shortest queue’ implies joining the CPU with the smallest current load). 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.

The queuing theory literature has widely studied the sojourn time in different systems, and has managed to find out not only their average sojourn time but also the CDF. However, the latter has only been possible in some systems that do not capture the multiple CPUs with PS fashion 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% or similar guarantees of URLLC with almost negligible latencies in the order of 1-10 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 [34], with a number of commercial (e.g. Simul8 [35] and AnyLogic [36]) and open-source (e.g. SimPy [37], SimPy [38], and Ciw [39]) 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 4.

4 SIMULATION OF G/G/R-JSQ-PS

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 [39] 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 [40], and capacitated processor sharing as described in [41], and their combinations.

Ciw uses the event-scheduling approach to discrete event simulation [39]. Here time jumps from event to event in a discrete manner, while 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, for example the event of a customer beginning service will cause a future scheduled event of that customer finishing service. If the events are scheduled immediately, then they are called conditional or C-events, for example the event a customer joining a queue may immediately cause another event, that customer beginning service, if there was enough service capacity. In addition to scheduling events, events can cause future events to be re-scheduled for a later or earlier time. A B-event, and its scheduling and re-scheduling of future events, is called the B-phase; a C-event, and its scheduling and re-scheduling of future events is called the C-phase; and advancing the clock to the next B-event is called the A-phase. Fig. 2 gives illustrates this event scheduling process.

Processor sharing is implemented by manipulating the re-scheduling of future events in the following way. Upon arrival, a customer is given an arrival date $t_a$, and an intended service time $s$. They also observe the number of customers, including themselves, who are present at the processor-sharing server, $x$. 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 (1).

$$t_{end} = t_a + \frac{1}{x}(s - d)$$

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 $t_a < t < t_{end}$ 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 $x = x + 1$;
5 M/M/R-JSQ-PS SOJOURN TIME CDF APPROXIMATIONS

In order to find the sojourn time distribution of a joint-shortest-queue processor-sharing M/M/R-JSQ-PS queue, we follow an approach outlined in [33], called Single Queue Analysis (SQA). Here, rather than consider the whole M/M/R queue, we consider each server as its own M/M/1-PS queue, with state-dependent arrival rates dependent on the joint-shortest-queue mechanism. Let $\Lambda$ 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 $\lambda_n$ when there are $n$ customers already being served by that server. TABLE 1 summarizes the notation used throughout this paper.

Now, considering a single server as its own queue, we adapt the methodology developed in [31] to the joint-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 $\lambda_n$, gives the sojourn time CDF as:

$$\mathbb{P}(T \leq t) = 1 - \mathbb{P}(T > t) = 1 - W(t) = 1 - \sum_{n=0}^{\infty} A_n w_n(t) \quad (2)$$

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 $\lambda_n$, $A_n$, and $w_n$ for each $n$. Then combining these in (2) gives us six approximations of the sojourn time CDF for an M/M/R-JSQ-PS queue.

5.1 First approximation of $\lambda_n$

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

and if the event is a customer finishing service then set $x = x_* - 1$. At this point our original customer will have received $d = d + \frac{1}{x_*}(t - t_*)$ of their intended service. Now set $x_* = x$, $t_* = t$, and re-calculate their service end date using (1), 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 $x_*$ to change. This was implemented and released in Ciw v2.2.0, along with some processor sharing variations: limited processor sharing queues [40], 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 [41] 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 custom routing using inheritance of Ciw’s modules. An example is given in the documentation: [https://ciw.readthedocs.io/en/latest/Guides/behaviour/ps_routing.html](https://ciw.readthedocs.io/en/latest/Guides/behaviour/ps_routing.html) Sojourn time CDFs can then be calculated easily as all customer records are saved.
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 \( \pi_n \) is the proportion of arrivals a server will receive if they have \( n \) customers already present.

We find \( \pi_n \) 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 \( S = \{ (a_1, a_2, \ldots, a_R) \mid a_1, a_2, \ldots, a_R \in \mathbb{N}_0 \} \), where \( a_z \) denotes the number of customers with server \( z \). Order the states and let \( s_i \) be the \( i \)th state. Define the transition rate \( q_{i,j} \) from \( s_i \) to \( s_j \), for all \( i, j \), by (3):

\[
q_{i,j} = \begin{cases} 
\frac{\mu}{C(s_i, \min(s_i))} & \text{if } C(\delta, 0) = R - 1 \land C(\delta, 1) = 1; \\
\mu \Lambda & \text{if } \delta = C(\delta, 0) = R - 1 \land C(\delta, 1) = 1 \\
1 & \Lambda \subseteq Z(s_i, \min(s_i)) \\
0 & \text{otherwise}
\end{cases}
\]

where \( \delta = s_i - s_j; C(v, b) = \{ z \in v : z = b \} \) is a function that counts the number of occurrences of \( b \) in a vector \( v \); and \( Z(v, b) = \{ z : v_z = b \} \) is the set of indices in \( v \) where \( b \) occurs. Fig. 3 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 \( pQ = 0 \) with \( pe = 1 \), where \( Q \) is the transition matrix with entries \( q_{i,j} \) and \( e \) 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, \( \pi_n \), can be found using (4):

\[
\pi_n = \left( \sum_{s_j,0=n}^{\pi} \frac{p_j}{C(s_j, n)} \right) \left( \sum_{s_j,0=n}^{\pi} p_j \right)^{-1}
\]

where \( s_{j,0} \) represents the number of customers at the first server when in state \( j \).

### 5.2 Second approximation of \( \lambda_n \)

The authors of [33] provide numerical approximations for \( \lambda_0, \lambda_1, \lambda_2 \) in Section 5, given in (5), (6) and (7), and all other \( \lambda_n \) for \( n \geq 3 \) by (8).

\[
\begin{align*}
\lambda_0 &= \mu \left( k_a - k_b k_R - k_d k_e R \right) \\
\lambda_1 &= \mu \left( \rho^R - 1 + \frac{\mu(\rho^R+1)}{\lambda_0(1-\rho)} \right) \\
\lambda_2 &= \mu k_f k_g R \\
\lambda_n &= \mu \left( \frac{\Lambda}{n\mu} \right)^n
\end{align*}
\]

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

\[
\begin{align*}
k_a &= \frac{\rho}{(1-\rho)} \\
k_b &= -0.0263\rho^2 + 0.0054\rho + 0.1155 \\
k_c &= -6.2973\rho^4 + 14.3382\rho^3 - 12.3532\rho^2 + 6.2557\rho - 1.005 \\
k_d &= -226.1839\rho^2 + 342.3814\rho + 10.2851 \\
k_e &= -226.1839\rho^2 + 342.3814\rho + 10.2851 \\
k_f &= -0.29\rho^3 + 0.8822\rho^2 - 0.5349\rho + 1.0112 \\
k_g &= -0.1864\rho^2 + 1.195\rho - 0.016
\end{align*}
\]

### 5.3 First approximation of \( A_n \)

Using the same Markov chain defined in Section 5.1, \( A_n \) can be found by manipulating the steady-state probabilities \( p_{nr} \), given in (9):

\[
A_n = \sum_{\min(s_j=n)} p_j.
\]

### 5.4 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:

\[
A_n = \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu} A_0
\]

\[
A_0 = \left( 1 + \sum_{i=1}^{\infty} \prod_{j=0}^{i-1} \frac{\lambda_j}{\mu} \right)^{-1}
\]
\[ w_n(t) = e^{Dt} e \]  

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 \( \lambda_n \), \( D \) takes the form:

\[
D = \begin{pmatrix}
-(\lambda_0 + \mu) & \lambda_0 & 0 & 0 & \cdots \\
\frac{\mu}{2} & -(\lambda_0 + \mu) & \lambda_1 & 0 & \cdots \\
0 & \frac{\mu}{2} & -(\lambda_0 + \mu) & \lambda_2 & \cdots \\
0 & 0 & \frac{\mu}{2} & -(\lambda_0 + \mu) & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots 
\end{pmatrix} \tag{13}
\]

By constructing a truncated \( D \) explicitly, numerical methods, such as Padé’s method \[44\], are used to find the matrix exponential.

\subsection*{5.6 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 \( w_n(t) \).

\begin{lemma} \label{lem1}
If a server within an M/M/R-JSQ-PS system has \( n \) customers, its sojourn time CDF is

\[ \mathbb{P}(T > t \mid n) = w_n(t) = \sum_{i=0}^{\infty} \frac{(\lambda_0 + \mu)^i t^i}{i!} e^{-\lambda_n t} h_{n,i} \] \tag{14}

with \( h_{n,0} = 1 \) for all \( n \), \( h_{n-1,i} = 0 \) for all \( i \), and \( h_{n,i} \) satisfying

\[ h_{n,i+1} = \frac{n}{n+1} \frac{\lambda_n}{\lambda_0 + \mu} h_{n-1,i} + h_{n,i} \left( \frac{\lambda_n - \mu}{\lambda_0 + \mu} \right) - h_{n,i+1} \] \tag{15}

\end{lemma}

\begin{proof}
We mimic the proof presented in \[31\] Corollary 2, where authors derive the vector of sojourn time complementary CDF \( w(t) \), with \( w_n(t) = \mathbb{P}(T > t \mid n) \), using the differential equation \( \frac{d}{dt} w(t) = Dw(t) \). With explicit solution \( w(t) = e^{Dt} e \).

In the above explicit solution, \( D \) is considered the defective infinitesimal generator taking the form given in \[13\]. By applying the uniformisation technique \[44\], the explicit solution of the aforementioned differential equation is

\[ w_n(t) = \sum_{i=0}^{\infty} \frac{(\lambda_0 + \mu)^i t^i}{i!} e^{-(\lambda_0 + \mu) t} \left[ I + \frac{1}{\lambda_0 + \mu} T \right]^i e \] \tag{16}

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

\[ w_n(t) = \sum_{i=0}^{\infty} \frac{(\lambda_0 + \mu)^i t^i}{i!} e^{-(\lambda_0 + \mu) t} h_{n,i} \] \tag{17}

and the \( n^{th} \) element of \( w(t) \) is given by \[14\].
\end{proof}

\begin{table}[h]
\centering
\begin{tabular}{ccc}
\hline
\textbf{Method} & \textbf{\( \lambda_n \)} & \textbf{\( A_n \)} & \textbf{\( w_n(t) \)} \\
\hline
A & 3.1 & 3.3 & 3.5 \\
B & 3.1 & 3.3 & 3.5 \\
C & 3.1 & 3.3 & 3.5 \\
D & 3.1 & 3.3 & 3.5 \\
E & 3.1 & 3.3 & 3.5 \\
F & 3.1 & 3.3 & 3.5 \\
\hline
\end{tabular}
\caption{Summary of the six methods of calculating \( W(t) \).}
\end{table}

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 \[31\] replaces their static MAP with the state-dependent arrival rate \( \lambda_n \).

\subsection*{5.7 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.

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. 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 smooth out the stochastic nature of the DES by taking 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.

Each of the six sub-methods described in Section 5 have hyper-parameters than need to be chosen. Those that explicitly build an infinite Markov chain, that is methods \[5.1\] and \[5.3\] 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 \[5.4\] and \[5.6\] 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...
When we approximate $\lambda$ complexity of each approximation $M/M/R$ Markov chain with a server foresees using the steady-state probabilities $\pi_n$ that is, the probability of entering a state with a server with $L$ servers. The truncation of the Markov chain in (3). Namely, we limit the “last” considered state $S_i = (L_i - 1)e$ has $L_i - 1$ users in all the $R$ servers. The truncation $L_1$ should be carefully selected such that

$$\sum_{s_j: \max s_j \geq L_1} p_j < \varepsilon$$

that is, the probability of entering a state with a server with $L_1$ or more customers should remain below a tolerance $\varepsilon \in \mathbb{R}^+$. Fig. 4 illustrates how probability of having $L_1$ or more users at a server decreases as we increase the truncation limit $L_1$ and how this is effected by both $\rho$ and $R$. This data was obtained using the simulation described in Section 4.

6 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 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 $\lambda_n, A_n$ and $w_n(t)$.

6.1 First approximation of $\lambda$

Using (11) 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 $L_1^R$. For each entry $q_{i,j}$ of the transition matrix we make $\min(s_i), Z(\delta, b), C(\delta, b)$ operations, all of them of complexity $O(R)$. Hence, computing all entries of the transition matrix $Q$ takes $O(RL_1^R)$ operations.

To find the steady-state vector $p$ we solve $(Q|e)^T p = (p|e)^T$, where $Q$ 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 [45] gsv method leads to a cubic run-time complexity on the matrix size. Therefore, obtaining the steady-state probability takes $O(L_1^R)$. Note that it is the computation of $p$ that dominates the complexity of approximating $\lambda_n$, as creating the transition matrix $Q$ has $O(RL_1^R)$ complexity and computing $\pi_n$ has $O(L_1^R)$ complexity – see [4]. Hence, the first approximation of $\lambda_n$ has run-time complexity $O(L_1^R)$.

6.2 Second approximation of $\lambda_n$

In [8] we see that there is a power relationship between $n$ and $\lambda_n$, namely, $\lambda_n = \mu^* (\frac{1}{\mu^*})^n$. As computing a power has complexity $O(\log n)$, the second approximation of $\lambda_n$ has complexity $O(\log n)$.

6.3 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 [9]. Thus, the complexity of computing $A_n$ is $O(L_1^R)$, for we iterate over all the $L_1^R$ states and check whether each of them satisfies $\min s_j = n$.

6.4 Second approximation of $A_n$

Given the values of $\lambda_n$, first we compute the probability of joining the queue when there are 0 users $A_0$ in (11). As mentioned in Section 5.7 we truncate the infinite summations up to $L_2$. Hence, it takes $\sum_{i=1}^{L_2} i$ operations to compute $A_0$, and so is $O(L_2^2)$. Once $A_0$ is computed, we perform $O(L_1^R)$ operations to compute $A_n$ in (10). Therefore as a whole, the second approximation of $A_n$ has $O(L_2^2)$ complexity.

6.5 First approximation of $w_n(t)$

This approximation computes the exponential of the defective infinitesimal generator matrix $D$ – see (12). As mentioned in Section 5.7 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 $\leq 3$ terms at each row, its creation has complexity $O(L_2)$. With Padé’s method [43] we compute $T$ exponential with $O(L_2 \log L_2)$ complexity.

6.6 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 5.7 we truncate the infinite summation in (14) to $L_2$ iterations. At each summation iteration $i$, we perform $O(\log i)$ operations (the power operators), hence, computing the second approximation of $w_n(t)$ has complexity $O(L_2 \log L_2)$. 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 $h_{n,i}$. Similarly, we also
keep inside a hash table the factorial computations $i!$ at 14 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 $O(L_2^3R)$ complexity because they rely on the truncated Markov chain to derive $\lambda_{i\nu}$, which is the most demanding approximation. While methods C and F have an overall complexity of $O(L_2^3R)$ because the approximations dominate the computation of $W(t)$. TABLE 3 summarises the computational complexity of each method.

### 6.7 Simulation

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 $\Lambda$ 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 - 1. As join-shortest-queue systems should evenly share customers between servers, we expect there to be $M/R$ customers at that server. So per time unit, the expected number of operations for arrival events is $O\left(\Lambda \left(R + \frac{M}{R}\right)\right)$.

- **End services:** at steady state, due to work conservation and Burke’s theorem 46, there’s an average of $\Lambda$ 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 unit time, the expected number of operations for end service events is $O\left(\Lambda \frac{R}{M}\right)$.

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 $q_{\text{max}}$ be the maximum simulation time. Altogether, when in steady state, the expected number of operations for a simulation run is $O\left(q_{\text{max}} \left(\Lambda \left(R + \frac{M}{R}\right) + \frac{\Lambda M}{R}\right)\right) = O\left(q_{\text{max}} \left(\Lambda R^2 + \frac{2\Lambda^2}{\mu R}\right)\right)$.

### 7 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 47, or Earth-mover’s distance. This is given in 19, with a graphic interpretation given in Fig. 5. This measure goes from 0, representing equal CDFs, to 1, the maximum sojourn time CDFs using each method, for all $q_{\text{max}}$. We need $X$ samples to estimate a good CDF, then $q_{\text{max}}$ should be chosen such that $q_{\text{max}} = \frac{X}{n}$. 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 $q_{\text{max}} = \frac{X}{n} + q_{\text{warmup}}$, where $q_{\text{warmup}}$ is the warmup time, the time it takes to reach steady state. It is likely that $q_{\text{warmup}}$ would be affected by the system parameters.

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 affected by the system parameters themselves. This shows that for some specific cases and parameter sets, it might be worthwhile resorting to simulation after all.

![Fig. 5: Graphical interpretation of the Wasserstein distance between the actual and approximated CDFs.](image-url)

Although $q_{\text{max}}$ 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 $q_{\text{max}}$ should be chosen such that $q_{\text{max}} = \frac{X}{n}$. 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 $q_{\text{max}} = \frac{X}{n} + q_{\text{warmup}}$, where $q_{\text{warmup}}$ is the warmup time, the time it takes to reach steady state. It is likely that $q_{\text{warmup}}$ would be affected by the system parameters.

### TABLE 3: Complexity of each method.

| Method | $\lambda_n$ | $\lambda_s$ | $w_n(t)$ | Overall |
|--------|-------------|-------------|----------|---------|
| A      | $O(L_1^3R)$ | $O(L_1^3R)$ | $O(L_2 \log L_2)$ | $O(L_2^3R)$ |
| B      | $O(L_1^3R)$ | $O(L_1^3R)$ | $O(L_2 \log L_2)$ | $O(L_2^3R)$ |
| C      | $O(\log n)$ | $O(L_2^3R)$ | $O(L_2 \log L_2)$ | $O(L_2^3R)$ |
| D      | $O(L_1^3R)$ | $O(L_1^3R)$ | $O(L_2 \log L_2)$ | $O(L_2^3R)$ |
| E      | $O(L_1^3R)$ | $O(L_2^3R)$ | $O(L_2 \log L_2)$ | $O(L_2^3R)$ |
| F      | $O(\log n)$ | $O(L_2^3R)$ | $O(L_2 \log L_2)$ | $O(L_2^3R)$ |

For these experiments hyper-parameter choices are a fixed: $L_2 = 130$; $t_{\text{max}} = 182.32$; 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
Fig. 6: Accuracy of each method with increasing traffic intensity $\rho$ and number of CPUs $R$.

| $R$ | $L_1$ |
|-----|-------|
| 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 $R$.

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.

Figs. (a) (b) (c) show the obtained Wasserstein distance, for each method A to F respectively, for each value of $R$ and $\rho$.

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 dependent on the traffic intensity $\rho$, however the relationship between accuracy and $\rho$ 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 $\rho$ result in higher approximation error. This may be due to unstable approximation algorithms used to compute matrix exponential [45]. While the second approximation performs much better for low values of $\rho$, 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 $\lambda_n$ performs worse with higher $R$ than the first Markov chain approximation.

Fig. 7 shows which method was most accurate for each $R$, $\rho$ pair. From this we see that method D performed best for low values of $\rho$, while methods A and B perform best for middling to high values of $\rho$. Method E is the best performing methods for very high values of $\rho$, however from the plot in Fig. 6e 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.

8 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 $\rho$ we can use the approximation with highest accuracy (see Fig. 7) 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., 99.99% of the times. If the end-to-end latency requirement is of 100ms and the maximum network latency remains below 28ms, this means that the sojourn time should remain below 72ms in the 99.99% of the times. Therefore, the applicability of our methods A-F depend on their accuracy at the 99.99th percentile.

In Fig. 8, we illustrate the error, measured in scalable time units, achieved by the best approximation at the 99.99th percentile. In other words, if $T_{a,99.99}$ is the best method 99.99th percentile for the sojourn time, and $T_{99.99}$ is the simulated 99.99-percentile; then Fig. 8 illustrates $T_{a,99.99} - T_{99.99}$. To derive the simulated 99.99th percentile use to the simulation from Section 4.
Fig. 7: Most accurate method TABLE 2 for each $R$, $\rho$ pair.

Fig. 8: Sojourn time 99.99\textsuperscript{th} percentile error using the best method. Positive/negative means over/under-estimation, respectively.

As with the Wasserstein distance (see Figs. 6a-6f), Fig. 8 evidences that the 99.99\textsuperscript{th} error becomes more prominent as the load $\rho$ approaches 1 in the M/M/R-JSQ-PS system. In particular, the best method under-estimates the 99.99\textsuperscript{th} percentile of the sojourn time for the error falls towards negative values near -175 as $\rho$ approaches 1. Note that the maximum sojourn time in the experiments can pop up to $t_{\text{max}} = 182.32$ 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 99.99\textsuperscript{th} percentile error remains low. Namely, the error is of less than $t = 12$ time units with respect to the simulations when $R \geq 3$ CPUs. If the system has $R < 3$ CPUs, then the best method has erratic oscillations, indeed the 99.99\textsubscript{th} percentile is underestimated by $t = -56.6$ time units with $R = 1$ - see TABLE 3.

We have also analysed what is best method error for the 99\textsuperscript{th}, 99.9\textsuperscript{th}, and 99.99\textsuperscript{th} percentiles. The results are shown in Appendix A and they show the same pattern as the observed for the 99.99\textsuperscript{th} percentile in Fig. 8. That is, the best A-F method results in under-estimations of the sojourn time that get worse as $\rho$ approaches to 1. Moreover, the results from Fig. 7 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 99.99\textsuperscript{th} percentile of the sojourn time as long as $\rho \leq 0.85$; tends under-estimate the 99.99\textsuperscript{th} percentile; and is more stable for $R \geq 3$ CPUs.

### 8.1 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/K-JSQ-PS system, e.g., the 99.99\textsuperscript{th} percentile of the sojourn time. However, exponentially distributed service rates are often an unrealistic assumption, with uniformly distributed or deterministic 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.

To investigate, we calculate the compare sojourn time CDFs using exponentially distributed services and uniformly distributed and deterministic services, for various values of $R$ and $\rho$. All CDFs were obtained using the simulation using an exponential distribution with average service time $\frac{1}{\mu}$; a uniform distribution $U\left(\frac{1}{2\mu}, \frac{1}{\rho}\right)$; and deterministic service time of $\frac{1}{\rho}$ time units. In such a manner, all distributions share the same average service time, although their variances are not equal: the exponentially distributed times having the highest variance ($\frac{1}{\rho}$), followed by the uniform distributed times ($\frac{1}{12\mu^2}$), and then the deterministic with no variance.

We see that the CDFs obtained when modelling service times as exponentially distributed always lie below those obtained using uniformly distributed and deterministic service times. This is demonstrated in Fig. 9, which shows that the tail percentiles are always larger, or more pessimistic, when modelling exponential services as opposed to uniform and deterministic services.

Fig. 9 also evidences how near $\rho = 0.6$ when we use $R = 5$ or $R = 10$ CPUs the best method (dark green) 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 Fig. 7). Similarly, at high loads $\rho \geq 0.97$ the best method changes from C to 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 Fig. 9 respectively).
TABLE 5: Sojourn time 99.9th 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 |
|---------|-------|-------|-------|-------|--------|
| ρ = 0.10| -3.76 | 0.29  | 1.49  | 1.38  | 1.50   |
| ρ = 0.25| -0.68 | 2.38  | 2.58  | 2.94  | 3.04   |
| ρ = 0.50| -0.41 | 0.22  | 1.17  | 1.47  | 1.78   |
| ρ = 0.75| -6.56 | -7.15 | -4.51 | -3.04 | -1.87  |
| ρ = 0.85| -56.60| -8.83 | -11.87| -9.84 | -7.29  |
| ρ = 0.90| -109.06| -31.78| -24.69| -20.86| -16.63 |
| ρ = 0.95| -104.87| -89.57| -68.65| -54.94| -47.24 |
| ρ = 0.99| -180.58| -180.59| -180.60| -180.61| -180.62 |

Altogether, Fig. 9 shows that our best method (dark green) stays close to the sojourn time percentiles obtained in simulations (yellow) for loads ρ < 0.97; that our best method lies above the sojourn times provided by deterministic (red) and uniformly distributed (blue) service times; and that our best method largely underestimates the sojourn time percentile for ρ ≥ 0.97, resulting in even smaller percentiles than uniformly distributed and deterministic service times.

9 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 a round trip time (RTT), 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:

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

(20)

It might not be feasible to understand the CDF of the RTT F, but we might know the ηF-percentile of F, which we denote as f ηF. Then:

$$
\mathbb{P}(F + T \leq \tau) = \int_0^\tau \mathbb{P}((F \leq f) \cap (T \leq \tau - x)) \, dx
$$

(21)

$$
= \int_0^\tau \mathbb{P}(F \leq f) \mathbb{P}(T \leq \tau - x) \, dx
$$

(22)

$$
\geq \mathbb{P}(F \leq f) \mathbb{P}(T \leq \tau - f) \tag{23}
$$

$$
\geq \eta \mathbb{P}(T \leq \tau - f) \tag{24}
$$

(25)

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 > fηF, now there is still an opportunity for F + T ≤ τ depending on the value of T. Finally step 24 is from the definition of f ηF.
Together this gives the following inequality:

$$P(T \leq \tau - f^{\eta_T} \geq \frac{\eta}{\eta_T} = \eta_T$$ \quad (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 $\tau$ with reliability $\eta$. Otherwise, we have to increase the number of CPUs $R$ at the server that process the URLLC traffic.

Let's consider an URLLC service that has latency and reliability requirements of $\tau = 10$ ms and $\eta = 0.99$, respectively. If we know that the $\eta_T = 0.999$-percentile of the network round trip time is $f^0.999 = 5$ ms, the sojourn time $T$ of the server must satisfy $P(T \leq 10 - 5) \geq 0.99 \approx 0.9901$.

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 $\rho = 0.33$, and we should check which method best approximates the sojourn time CDF at such load. In Fig. 7 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 $P(T \leq 6.43) \geq 0.9901$. However we require that the sojourn time $T$ satisfies $T \leq \tau - f^{0.9999} = 5$ with reliability $0.9999 \approx 0.9901$. 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 $\Lambda$ 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 $\Lambda$, the current number of CPUs $R$ with their service rates $\mu$, the URLLC reliability $\eta$, the overall latency requirement $\tau$, and the network RTT value for the $\eta_T$-percentile $f^{\eta_T}$; Algorithm 1 gives the required number of CPUs for an M/M/R-JSQ-PS system. Namely, in line 5 the $\eta_T$-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.

9.1 Example: scaling an autonomous driving service

In this section we provide an example of how to use Algorithm 1 to scale an autonomous driving service, namely, an infrastructure assisted environment perception service. Such a service consists of processing video or sensor streams from vehicles to assist them in the detection and perception of road events. For example, the infrastructure can detect that a tree has fallen down from the video stream of a vehicle and inform other vehicles in the road of the event. According to [3], infrastructure assisted perception asks for an end-to-end service latency and reliability of $\tau \leq 100$ ms and $\eta = 0.9999$, respectively.

We consider the 5G-SA deployment in Fig. 10 to provide connectivity to connected vehicles. The setup was tested in the 5G-DIVE project [49] and consists of an Ericsson

![Fig. 10: 5G-SA setup][49] with local breakout to the Edge server.

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. Using the setup in Fig. 10 the maximum measured RTT was of $f^{1.0} = 28.095$ ms. Such RTT makes meeting the above service latency requirement of $\tau \leq 100$ ms feasible.

To generate the traffic demand $\Lambda$ 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 [50].

First we consider the highway. Fig. 11c (top) illustrates how the traffic demand $\Lambda$ fps evolves over four days in a road at highway road in Torino city. It is worth mentioning that the number of vehicles reported by our dataset is high enough to satisfy Palm-Khintchine theorem [32], hence, the Poissonian arrival assumption for our M/M/R-JSQ-PS system (Edge server) can be considered valid.

To process the autonomous driving traffic we consider that the road is equipped with an Edge server with $R = 10$ available CPUs performing video decoding of H.265/HEVC video streams [50] and object detection tasks. We use the results in [31], [32] to derive the service rate $\mu$ 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 min-

![Algorithm 1: URLLC server scaling][51]

- **Data:** $\Lambda, R, \mu, \eta, \tau, \eta_T, f^{\eta_T}$
- **Result:** $R, f^{\eta_T}$

1. **Function** sojourn_percentile($\eta'$, $R'$):
2. $\rho = \frac{\Lambda}{R'}$
3. method $\times$ = best_method($\rho$, $R'$)
4. $f^{\eta'} = \text{method}_X \cdot \text{percentile} (\eta')$
5. *return* $f^{\eta'}$
6. $R = 1$
7. $\eta_T = \frac{\eta}{\eta_T}$
8. $f^{\eta_T} = \text{sojourn} \cdot \text{percentile} (\eta_T, R)$
9. **while** $f^{\eta_T} + f^{\eta_T} > \tau$ **do**
10. $R = R + 1$
11. $f^{\eta_T} = \text{sojourn} \cdot \text{percentile} (\eta_T, R)$
12. **end**
The goal is to meet the 99.99-percentile delay of 100ms (middle rows) considering the 5G setup in Fig. 10. Via simulation of exponential, uniform, and deterministic service times, we then consider for Exponential, Uniform, and Deterministic intended service times. Here we investigate the difference in scaling of Exponential, Uniform, and Deterministic intended service times. In Section 8.1 we compared the percentiles obtained from the simulation with Algorithm 1. However, in the highway road (yellow top right), the difference in the scaling algorithm we meet the requirements of infrastructure assisted environment perception. In Fig. 11c 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 $R = 9$ in the two peak-hours foreseen each day, which correspond to the the time at which the citizens travel to and from work. Similarly, Fig. 11c (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 in the Fig. 11c) shows that the 99.99%-percentile of the sojourn time remains below $\tau \sim 100 = 71.905$ ms, because we invoked Algorithm 1 to scale considering the maximum network RTT of $f^{\Delta} = 28.095$ ms. Thus, Fig. 11c shows that by using the proposed approximations and scaling algorithm we meet the requirements of infrastructure assisted environment perception.

Similar analyses were carried out on an industrial road in Fig. 11b and a residential road, in Fig. 11a. Considering the highway analyses of Fig. 11c 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.

### 9.2 Impact of approximation errors

In Section 8.1 we compared the percentiles obtained from the best approximations, to those obtained through simulation of Exponential, Uniform, and Deterministic intended service times. Here we investigate the difference in scaling of the autonomous driving service in Torino if simulated Exponential, Uniform, and Deterministic intended service times were used. 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, and deterministic service times, that is we use the simulations depicted in Fig. 9.

We denote with $t^{\Delta \text{0.9999}}_{\text{sojourn}}$ the sojourn time percentile reported by Algorithm 1 when it uses the best method A-F. Similarly, we denote $t^{\Delta \text{0.9999}}_{\text{Exp}}$, $t^{\Delta \text{0.9999}}_{\text{U}}$, and $t^{\Delta \text{0.9999}}_{\text{Det}}$ the percentile reported by Algorithm 1 when it uses the simulation data for Exponential, Uniform, and Deterministic service times respectively. We then consider $\Delta t^{\Delta \text{0.9999}}$, the difference in the 99.99%-percentile of the sojourn time when we use the best method and simulations, respectively. A positive value of $\Delta t^{\Delta \text{0.9999}}$ 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 9.1 with increasing vehicular traffic: residential, industrial, and highway access roads with up to 240, 2184, and 4392 vehicles/hour respectively. As the highway road has more vehicular traffic $\Lambda$ [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.

Fig. 12 illustrates the empirical Probability Density Function (ePDF) $\Delta t^{\Delta \text{0.9999}}$ over four days when we use Algorithm 1 scaling, for the three road types and three 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 $\Delta t^{\Delta \text{0.9999}} = 0$. 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 $\Delta t^{\Delta \text{0.9999}} = 0$, which means that the best method underestimated the sojourn time percentile for the
10 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 both Uniform and Deterministic 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 \( O(L_{1/2}^3) \), and are useful to scale servers processing URLLC traffic under mid to high loads, for they yield errors of less than 3 time units in high percentiles as a 99.99%.

The proposed methods serve as conservative scaling approach thanks to the assumption of the exponential service times, yet achieving a scaling accuracy above a 90% in different scenarios with uniform and deterministic service times.

Acknowledgements
This work has been partially funded by European Union’s Horizon 2020 research and innovation programme under grant agreement No 101015956, and the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D 6G-EDGEDT and 6G-DATADRIVEN.

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. Fritsch, 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 CESICT 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405863150005921

[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. Barbonti, S. Baggioni, L. S. Mattos, 5G Robotic Telemedicine: Remote Transoral Laser Microsurgery on a Cadaveri,” IEEE Transactions on Medical Robotics and Bionics, vol. 2, no. 4, pp. 511–518, 2020.

[6] 3GPP, “Study on scenarios and requirements for next generation access technologies,” 3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.913, 4 2022.

[7] IEEE, “IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Services,” IEEE Std 802.1Qat-2009 (Amendment to IEEE Std 802.1Q-2005), pp. C1–72, 2010.
APPENDIX A

ERRORS FOR INCREASING RELIABILITIES

Fig. 14 shows the error for the best approximation A-F from the 99th up to the 99,999th percentile of the sojourn time. The sojourn time error is unitless, and it illustrates the increasing error as $\rho$ approaches 1, so as the increasing oscillations of the error for higher reliabilities, even with mid values of the number of CPUs like $R=4$ as explained in Section 8.

Fig. 14: The best method error for the 99th, 99.9th, and 99.999th percentile the sojourn time. Positive/negative mean over/under-estimation, respectively.

APPENDIX B

GETTING sojourn_percentile ($\eta'$, $R'$)

We provide an open-source implementation 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($\eta'$, $R'$) function used inside Algorithm 1, we first compute the load $\rho$ given a number of CPUs $R'$, and the arrival and service rates $\Lambda, \mu$; respectively. Second, we check Fig. 7 to know which is the best method for the given $(\rho, R')$ tuple, e.g., method-A. Third, we create an instance of method-A invoking $\text{jsq.MethodA}(\Lambda, \mu, R, L_1, L_2, \{t_0, t_1, \ldots\})$, 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 accessing property $sojourn\_time\_cdf$ of the method instance. This property holds a vector $[0, P_1, P_2, \ldots]$ 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 $\eta'$ percentile of the sojourn time as

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

APPENDIX C

CONSIDERED $\mu$ FOR AUTONOMOUS DRIVING

The scaling experiments presented in Section 9.1 consider an autonomous driving service, namely, an infrastructure assisted environment perception service [2]. Vehicles send an H.265/HEVC video stream that is processed in a remote server (modelled as an M/M/R-JSQ-PS system) to detect road events and inform the vehicle.

The vehicles video stream $\Lambda$ is expressed as frames/sec (fps), and is processed at a rate of $\mu$ fps in the server hosting the autonomous driving service. For the experiments in Section 9.1, we take into consideration the time that it takes to decode the video stream, and the time it takes for a Convolutional Neural Network (CNN) to detect events in a video frame. According to [51], [52] an Intel Xeon family CPU manages decode an HEVC video frame in 8 ms, and takes 0.37 ms to detect a road event in a video frame. Hence a single CPU within the considered M/M/R-JSQ-PS system offers a rate of $\mu = \frac{10^3}{8.37}$ fps for the considered infrastructure assisted environment perception service. Such value of $\mu$ is the one we used in the experiments presented in Section 9.1.

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 optimal resource allocation in networks, and since 2016 he participates in EU funded research projects in UC3M Telematics department.